

Learning jQuery

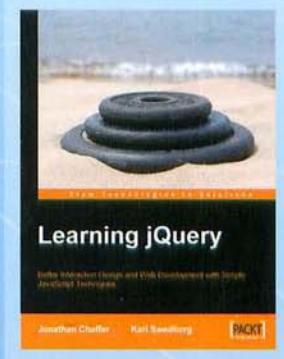
Better Interaction Design and Web Development
with Simple JavaScript Techniques

jQuery基础教程

[美] Jonathan Chaffer 著
Karl Swedberg

李松峰 李炜 等译

- 全球第一部jQuery著作
- Amazon全五星盛誉
- jQuery官方网站推荐



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序员设计丛书 Web 开发系列

Learning jQuery

Better Interaction Design and Web Development
with Simple JavaScript Techniques

jQuery 基础教程

[美] Jonathan Chaffer 著
Karl Swedberg

李松峰 李炜 等译

人民邮电出版社

北京

www.TopSage.com

图书在版编目 (CIP) 数据

jQuery 基础教程 / (美) 查弗 (Chaffer, J.), (美) 斯威德伯格 (Swedberg, K.) 著; 李松峰等译. —北京: 人民邮电出版社, 2008.7 (2008.11 重印)

(图灵程序设计丛书)
书名原文: Learning jQuery: Better Interaction Design and Web Development with Simple JavaScript Techniques
ISBN 978-7-115-18110-7

I. j... II. ①查...②斯...③李... III. JAVA 语言 - 程序设计 - 教材 IV. TP312

中国版本图书馆CIP数据核字 (2008) 第067863号

内 容 提 要

本书以通俗易懂的方式介绍了 jQuery 的基本概念, 主要包括 jQuery 的选择符、事件、特效、DOM 操作、AJAX、表格操作、表单函数、拖曳与旋转和插件等内容, 最后几章以实例操作为主, 在前面内容的基础上, 提出了常见的客户端实际问题并给出了其解决方案。

本书是一本注重理论与实践结合的基础教程, 适合 Web 开发人员阅读和参考。

图灵程序设计丛书

jQuery基础教程

◆ 著 [美] Jonathan Chaffer Karl Swedberg

译 李松峰 李 炜 等

责任编辑 傅志红

执行编辑 杨 爽

◆ 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

◆ 开本: 800×1000 1/16

印张: 19.5

字数: 458 千字 2008 年 7 月第 1 版

印数: 4 001 - 5 000 册 2008 年 11 月河北第 2 次印刷

著作权合同登记号 图字: 01-2008-2029号

ISBN 978-7-115-18110-7/TP

定价: 45.00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

Copyright © Packt Publishing 2007. First published in the English language under the title *Learning jQuery : Better Interaction Design and Web Development with Simple JavaScript Techniques.*

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

本书中文简体字版由Packt Publishing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

说起我与jQuery结缘，还要感谢2006年发生在台湾海峡的地震。2006年12月26日，中国南海台湾附近发生7.2级地震，数分钟后又发生了6.7级地震。受强烈地震影响，中美海缆等多条国际海底通信光缆发生中断，造成附近国家和地区的国际和地区性通信受到严重影响。2007年1月29日，电信网通宣布，经过20多天的抢修，受地震影响中断的国际通信业务已全部恢复。在此期间，中国雅虎在邮箱主页顶部发布了一个由于海缆中断可能会造成邮件收发有问题的通告。当时，通告是在页面加载完成大约1s后，以渐变和动画形式出现在页面顶部的——跟jQuery官方网站首页那个“*The quick and dirty*”的演示效果很相似。而且，通告显示了大约几秒钟后又以动画形式自动消失，整个页面好像什么都没有发生过一样。这个动画效果深深地吸引了我。以前，我也试着写过像卓越亚马逊网站首页“所有20类商品”按钮的鼠标悬停动画（可以在<http://www.amazon.com.cn/>上面看到这个动画效果），但使用了几十行代码，如今这个更酷的效果是怎么实现的呢？于是，我怀着强烈的好奇心开始查看它的源代码（这要感谢JavaScript天生的开源特性）。惊奇地发现，这个效果仅用了寥寥几行代码！惊讶之余，溯本求源，最后“认识”了精巧而美妙的jQuery，特别是它优雅的方法连缀能力，更令我如获至宝、兴奋不已！后来我查了很多jQuery的资料，发现它的文档没有汉化，就用一周的休息时间翻译了它的API（1.1版）文档。这份汉化文档在jQuery中文资料匮乏的时候为广大jQuery网友提供了一点帮助，也获得了大家的认可和好评。

JavaScript库和框架解决的问题，无非就是（跨浏览器的）DOM操作、事件处理、样式更换和外部通信（AJAX）。但jQuery独特的集合对象、隐式迭代、方法连缀、自定义选择符和事件方法，加之只有不到20 KB的超轻巧和执行速度超快，赢得了众多JavaScript开发者的青睐。

jQuery不仅支持各式各样的CSS选择符表达式，而且还支持XPath和自定义的选择符表达式，这一点在JavaScript库和框架领域中无出其右者，使开发者找到要操作的元素或集合简单得难以置信；它细腻灵巧而又富有弹性的事件处理机制，包括事件注册、触发和自定义，特别是令JavaScript的Guru级人物都喜不自禁的hover()方法，使它在JavaScript库和框架之林中独树一帜、个性十足；它在操作DOM文档时的大处着眼、小处着手，提供的丰富而实用的各种遍历和操作DOM结构及元素的方法，令人耳目一新，简直“直逼每个JavaScript爱好者的心理防线”，那种令人怦然心动的感觉，历久弥新；它处理AJAX请求和响应的简洁明快、它的简单易用、它超级方便的扩展机制、它丰富的插件支持（Interface等）、它背后的强大社区……所有这些，引无数JavaScript高手竞折腰！

事实上，因特网上的JavaScript库和框架数以百千计，为什么唯独jQuery对我们这些爱好者有如此大的吸引力呢？就是因为jQuery采取了与其他库和框架截然不同的理念，处处匠心独运，别出心裁——具体细节，请参考第1章。

本书作为第一本全面、深入介绍jQuery库的图书，可以说是应运而生的。书中包含了jQuery教程、jQuery实例和JavaScript最佳实践。jQuery教程部分是本书第2章至第6章，分别介绍了jQuery中的选择符、事件处理、DOM操作、动画效果和AJAX方法。其中，第3章、第4章、第5章结尾，特别归纳了相应方法及适用情形，既简明又实用。jQuery实例部分是本书第7章、第8章、第9章，分别围绕Web开发中最常见的表格、表单和动画效果，详尽地探讨了使用jQuery的方方面面。这几章的实例，深入讨论诸多Web开发问题，深入浅出、娓娓道来，时不时令人拍案叫绝、感叹很多百思不得其解的问题，其实只有一层窗户纸！第10章介绍了jQuery强大的扩展能力，介绍了扩展jQuery或者编写自己的jQuery插件的方法。这一章深入到jQuery核心，把整个库的架构全部展现给了读者，并向读者揭示出jQuery库中的“陷阱”和“关键”，令人有豁然开朗、恍然大悟之感。

现代JavaScript开发的一个基准点就是最佳实践。为了让读者不走弯路、不浪费宝贵的时间，本书在介绍通过jQuery进行JavaScript开发的过程中，实践了“渐进增强”和“平稳退化”这两个不唐突（unobtrusive）的JavaScript开发原则。把抽象的概念形象化、具体化，字里行间，渗透着作者对这些先进理念的阐发与启示。

值得一提的是，本书附录C是名符其实的“压轴好戏”。这么举重若轻、浅显易懂地讨论JavaScript闭包，在译者看来还是头一次。几个精心设计的例子，读者跟着走下来，不知不觉中就能领略到JavaScript这一高级特性的精髓所在（也许没有说得那么容易）。

书是人类进步的阶梯，这话一点不假，但“尽信书不如无书”。要想学习jQuery不能不看jQuery的图书，但是，只看是不管用的，还要动手实践——打开文本编辑器和浏览器，亲手写jQuery代码！书中很多地方讲的只是要点，而动手实践才能收获书中没有讲到的东西。

本书由李松峰负责翻译，参加翻译工作的还有李炜、秦绪文、李丽、程宝杰、宋连海、付荣艳、封耀杰、贾爱华、左艳坡、熊俊芹、刘英、宋会敏等。

最后，也是最重要的，我要感谢在翻译此书过程中，傅志红老师给我提供的帮助和建议。感谢武卫东老师、刘江老师对译稿的指点。感谢本书的责任编辑杨爽对译稿的认真审核和修改，如果不是她创造性地与我沟通，本书恐怕要留下不少遗憾。此外，还要感谢图灵俱乐部“明月星光”网友的热心建言，他的建议解决了一些我在翻译过程中遇到的问题。不过，囿于个人水平和能力，翻译中的错误和不当之处在所难免。如果读者发现了书中的问题，请在我的个人网站 <http://www.cn-cuckoo.com> 中指出，或者将电子邮件发送到lsf.email@yahoo.com.cn。

译者
2008年2月于北京

前　　言

jQuery是一个强大的JavaScript库。无论你具有什么编程背景，都可以通过它来增强自己的网站。

由John Resig创建的jQuery是一个开源项目，其核心团队由富有献身精神的顶尖JavaScript开发人员组成。jQuery在一个紧凑的文件中提供了丰富多样的特性、简单易学的语法和稳健的跨平台兼容性。此外，百余种为扩展jQuery功能而开发的插件，更使得它几乎成为适用于各类客户端脚本编程的必备工具。

本书以通俗易懂的方式介绍了jQuery的基本概念，通过学习本书，即使曾经因编写JavaScript而受过挫折的人，也能够掌握为网页添加交互和动态效果的技术。本书将引导读者跨越AJAX、事件、效果及高级JavaScript语言特性中的各种陷阱。

本书网站<http://book.learningjquery.com>中，包含书中各章的在线示例。

本书内容

本书的第一部分是jQuery简介，用来帮助读者对jQuery有个大概的了解。第1章的内容主要涉及如何下载和设置jQuery库，同时也会指导你使用jQuery编写第一个脚本。

本书的第二部分将深入讨论jQuery库的各个主要方面。第2章讲述如何取得我们想要的一切。通过jQuery中的选择符表达式，你可以在页面中的任何地方找到想要的元素。这一章将使用各种选择符表达式为页面中的不同元素添加样式，其中一些是通过纯CSS方式做不到的。

第3章讲述如何“扣动扳机”。本章介绍如何通过jQuery的事件处理机制，在浏览器发生事件时触发行为。同时，还会介绍jQuery的独家秘笈——以不唐突的方式添加事件（甚至在页面加载完成之前）。

第4章讲述如何增加操作的艺术感。这一章介绍通过jQuery实现动画的技术，从中我们能够体会到隐藏、显示和移动页面元素时那种轻松自如的感觉。

第5章讲述如何通过指令改变页面。本章讲述的是动态修改HTML文档结构的技术。

第6章讲述如何让你的网站跻身主流行列。在学习完本章后，你也可以做到不用像过去那样刷新页面而访问服务器端功能。

本书的第三部分与前两部分不同。这一部分主要以实例为主，即在前几章学习的基础上，创建常见问题的稳健jQuery解决方案。第7章将讲述排序、筛选和为信息添加样式并创建优美实用的数据布局。

第8章以客户端数据验证为主题。届时，将设计一个具有适应能力的表单布局，还会实现基于客户端与服务器通信的交互式表单功能，例如自动完成。

第9章介绍如何在小窗口内显示页面元素来增强它们的美观和实用性。其中，动态显示和隐藏信息的方式既可是自动的，也可是用户控制的。

第10章讲述jQuery令人印象深刻的扩展能力。读者将在理解和掌握3个优秀jQuery插件的基础上，从头开始构建自己的插件。

附录A提供了很多与jQuery、JavaScript以及通常的Web开发有关的内容丰富的网站信息。

附录B推荐了一些有用第三方程序和实用工具，用于在个人的开发环境中编辑和调试jQuery代码。

附录C讨论JavaScript语言的常见壁垒之一——闭包。学了本附录，你将会依赖闭包的强大威力而不是害怕它的副作用。

本书读者对象

本书适合想在自己的设计中添加交互元素的Web设计者，也适合想在自己的Web应用中创建最佳用户界面的开发者。

读者需要具备基本的HTML和CSS知识，并且应该熟悉JavaScript语法。但是，不需要有jQuery的知识，也不必拥有其他JavaScript库的使用经验。

本书约定

在本书中，读者会发现针对不同信息类型的文本样式。下面是这些样式的示例和解释。

书中的代码有3种样式。正文中提到的代码如下所示：“总之，`\$(document).ready(function() {`和`\$('.addClass('emphasized');`对我们修改诗歌文本的外观已经足够了。”

代码如下所示：

```
\$(document).ready(function() {
  \$('.span:contains(language)').addClass('emphasized');
});
```

当需要读者特别注意代码块中的某一部分时，相关的代码行或项将以粗体显示：

```
$(document).ready(function() {  
    $('a[@href$=".pdf"]') .addClass('pdflink');  
});
```

新术语及重要词汇将以粗体字显示。对于在屏幕上看到的文字（例如在菜单或对话框中）在正文中会表示为：下一步是单击“*All*”按钮运行那些测试。

说明 重要的注意事项以说明体例给出。

提示 提示和技巧以提示体例给出。

读者反馈

我们始终欢迎来自读者的反馈意见。我们想知道读者对本书的看法，读者喜欢哪些内容或不喜欢哪些内容。读者真正深有感触的反馈，对于我们开发图书产品至关重要。如有反馈意见，请将电子邮件发送到contact@turingbook.com，不要忘记在邮件标题中注明你要评论的书名。

客户支持

为了让你的付出得到最大的回报，请注意以下信息。

本书的示例代码下载

访问<http://www.packtpub.com/support>，然后从图书列表中选择本书，以便下载本书的示例代码及其他资源。在选择了本书之后，请根据提示完成下载。

下载的文件中包含使用说明。

勘误

虽然我们会全力确保本书内容的准确性，但错误仍在所难免。如果你发现了本书中的错误（包括文字和代码错误），而且愿意向我们提交这些错误，我们会十分感激。这样一来，不仅可以减少其他读者的疑虑，也有助于本书后续版本的改进。要提交你发现的错误，请访问<http://www.packtpub.com/support>，选择你的图书，单击Submit Errata链接，然后输入勘误信息。经过验证之后，你提交的勘误信息就会添加到已有的勘误列表中。现有的勘误信息也可以通过访问<http://www.packtpub.com/support>并选择你的图书查看到。

疑难解答

如果你对本书的某些方面有疑问，请将电子邮件发送到contact@turingbook.com或questions@packtpub.com，我们会尽力解决。

目 录

第1章 jQuery入门	1
1.1 jQuery能做什么	1
1.2 jQuery为什么如此出色	2
1.3 第一个jQuery文档	3
1.3.1 下载jQuery	3
1.3.2 设置HTML文档	4
1.3.3 编写jQuery代码	6
1.4 小结	9
第2章 选择符——取得你想要的一切	10
2.1 DOM	10
2.2 工厂函数\$()	11
2.3 CSS选择符	11
2.4 XPath选择符	14
2.5 自定义选择符	16
2.6 DOM遍历方法	18
2.7 访问DOM元素	22
2.8 小结	22
第3章 事件——扣动扳机	23
3.1 在页面加载后执行任务	23
3.1.1 代码执行的时机选择	23
3.1.2 基于一个页面执行多个脚本	24
3.1.3 缩短代码的简写方式	25
3.2 简单的事件	25
3.2.1 简单的样式转换器	26
3.2.2 简写的事情	32
3.3 复合事件	33
3.3.1 显示和隐藏高级特性	33
3.3.2 突出显示可单击的项	34
3.3.3 事件的旅程	36
3.3.4 事件冒泡的副作用	37
3.4 限制和终止事件	38
3.4.1 阻止事件冒泡	38
3.4.2 移除事件处理程序	40
3.5 模仿用户操作	42
3.6 小结	43
第4章 效果——为操作添加艺术性	44
4.1 修改内联CSS	44
4.2 基本的隐藏和显示	47
4.3 效果和速度	49
4.4 多重效果	50
4.4.1 构建具有动画效果的show()	51
4.4.2 创建一种自定义的动画效果	51
4.4.3 理解数字的含义	53
4.4.4 改进自定义动画效果	54
4.5 并发与排队效果	55
4.5.1 处理一组元素	55
4.5.2 处理多组元素	57
4.6 简单概括	60
4.7 小结	60
第5章 DOM操作——基于命令改变页面	61
5.1 操作属性	61
5.2 插入新元素	64
5.3 移动元素	66
5.3.1 标注、编号和链接到上下文	69
5.3.2 插入脚注	71
5.4 包装元素	72

5.5 复制元素.....	72	7.8.1 筛选选项.....	150
5.5.1 复制的深度.....	73	7.8.2 同其他代码整合.....	152
5.5.2 通过复制创建突出引用.....	74	7.9 完成的代码.....	155
5.6 DOM 操作方法的简单归纳.....	79	7.10 小结.....	158
5.7 小结.....	80	第8章 构建功能型表单.....	159
第6章 AJAX——让网站与时俱进.....	81	8.1 渐进增强的表单设计.....	159
6.1 基于请求加载数据.....	81	8.1.1 图标符号.....	161
6.1.1 追加 HTML.....	83	8.1.2 必填字段的提示信息.....	162
6.1.2 操作 JavaScript 对象.....	85	8.1.3 根据条件显示的字段.....	166
6.1.3 加载 XML 文档.....	91	8.2 表单验证.....	168
6.2 选择数据格式.....	94	8.2.1 即时反馈.....	168
6.3 向服务器传递数据.....	95	8.2.2 最终检查.....	173
6.3.1 执行 GET 请求.....	96	8.3 复选框操作.....	174
6.3.2 执行 POST 请求.....	99	8.4 完成的代码.....	177
6.3.3 序列化表单.....	100	8.5 字段的占位符文本.....	180
6.4 关注请求.....	102	8.6 AJAX 自动完成.....	182
6.5 AJAX 和事件.....	105	8.6.1 服务器端代码.....	182
6.5.1 限定事件绑定函数的作用域.....	106	8.6.2 浏览器端脚本.....	183
6.5.2 利用事件冒泡.....	106	8.6.3 填充搜索字段.....	184
6.6 安全限制.....	107	8.6.4 键盘导航.....	184
6.7 小结.....	108	8.6.5 自动完成与实时搜索.....	188
第7章 表格操作.....	109	8.7 完成的代码.....	189
7.1 排序.....	110	8.8 输入掩码.....	191
7.1.1 服务器端排序.....	110	8.8.1 购物车表格结构.....	191
7.1.2 JavaScript 排序.....	111	8.8.2 拒绝非数字输入.....	194
7.2 分页.....	123	8.9 数字计算.....	194
7.2.1 服务器端分页.....	124	8.9.1 解析和格式化货币值.....	195
7.2.2 JavaScript 分页.....	125	8.9.2 处理小数位.....	196
7.3 完成的代码.....	129	8.9.3 其他计算.....	198
7.4 高级行条纹效果.....	132	8.10 删除商品.....	200
7.4.1 三色交替模式.....	135	8.11 修改送货信息.....	204
7.4.2 三行一组交替.....	137	8.12 完成的代码.....	207
7.5 突出显示行.....	141	8.13 小结.....	209
7.6 工具提示条.....	143	第9章 滑移和翻转.....	210
7.7 折叠和扩展.....	148	9.1 标题翻转效果.....	210
7.8 筛选.....	150	9.1.1 设置页面.....	210

9.1.2 取得新闻源.....	212	10.2 流行的插件.....	251
9.1.3 设置翻转效果.....	214	10.2.1 Dimensions	251
9.1.4 标题翻转函数.....	215	10.2.2 Form.....	253
9.1.5 悬停时暂停.....	217	10.2.3 Interface	255
9.1.6 从不同的域中取得新闻源	219	10.3 查找插件文档.....	259
9.1.7 附加的内部渐变效果.....	221	10.4 开发新插件.....	261
9.2 图像传送带.....	223	10.4.1 添加新的全局函数.....	261
9.2.1 设置页面.....	223	10.4.2 添加 jQuery 对象方法.....	263
9.2.2 通过 JavaScript 修改样式	225	10.4.3 DOM 遍历方法	264
9.2.3 通过单击滑移图像.....	226	10.4.4 添加新的简写方法.....	267
9.2.4 添加滑移效果.....	228	10.4.5 维护多事件日志.....	268
9.2.5 显示操作图标.....	229	10.4.6 添加选择符表达式.....	270
9.3 放大图像.....	232	10.4.7 创建缓动样式.....	272
9.3.1 隐藏大幅封面.....	234	10.4.8 做个好公民.....	274
9.3.2 更有价值的标记.....	236	10.5 小结	275
9.3.3 为封面放大添加动画效果	237		
9.4 完成的代码.....	244		
9.5 小结	249		
第 10 章 插件	250		
10.1 使用插件.....	250	附录 A 在线资源	277
		附录 B 开发工具	283
		附录 C JavaScript 闭包	287



jQuery入门

*Up on the buzzer
Quick on the start
Let's go! Let's go! Let's go!^①*
—Devo[®],
“Let's Go”

今天的万维网是一个动态的环境，Web用户对网站的设计和功能都提出了高要求。为了构建有吸引力的交互式网站，开发者们借助于像jQuery这样的JavaScript库，实现了常见任务的自动化和复杂任务的简单化。jQuery库广受欢迎的一个原因，就是它对种类繁多的开发任务都能游刃有余地提供帮助。

由于jQuery的功能如此丰富多样，找到合适的切入点似乎都成了一项挑战。不过，这个库的设计秉承了一致性与对称性原则，它的大部分概念都是从HTML和CSS（Cascading Style Sheet，层叠样式表）的结构中借用而来的。鉴于很多Web开发人员对这两种技术比对JavaScript更有经验，所以编程经验不多的设计者能够快速学会使用该库。实际上，在本书开篇第1章中，只需3行代码就能编写一个有用的jQuery程序。另一方面，经验丰富的程序设计人员也会受益于这种概念上的一致性，通过学习后面的更高级内容，你会感受到这一点。

但是，在举例说明如何使用这个库之前，我们应该首先讨论一下选择它的理由。

1.1 jQuery能做什么

jQuery库为Web脚本编程提供了通用的抽象层，使得它几乎适用于任何脚本编程的情形。由于它容易扩展而且不断有新插件面世增强它的功能，所以一本书根本无法涵盖它所有可能的用途和功能。抛开这些不谈，仅就其核心特性而言，jQuery能够满足下列需求：

① 本书各章都以著名“后朋克乐队Devo（退化）”的一首歌的歌词开头，歌词大意与相应章的名称或内容有微妙的对应。要了解“后朋克”的更多信息，请参考<http://baike.baidu.com/view/638729.htm>。要了解有关“朋克”的更多信息，请参考<http://baike.baidu.com/view/966.htm>。——译者注

- **取得页面中的元素。**如果不使用JavaScript库，遍历**DOM**（Document Object Model，文档对象模型）树，以及查找HTML文档结构中某个特殊的部分，必须编写很多行代码。jQuery为准确地获取需要检查或操纵的文档元素，提供了可靠而富有效率的选择符机制。
- **修改页面的外观。**CSS虽然为影响文档呈现的方式提供了一种强大的手段，但当所有浏览器不完全支持相同的标准时，单纯使用CSS就会显得力不从心。jQuery可以弥补这一不足，它提供了跨浏览器的标准解决方案。而且，即使在页面已经呈现之后，jQuery仍然能够改变文档中某个部分的类或者个别的样式属性。
- **改变页面的内容。**jQuery能够影响的范围并不局限于简单的外观变化，使用少量的代码，jQuery就能改变文档的内容。可以改变文本、插入或翻转图像、对列表重新排序，甚至，对HTML文档的整个结构都能重写和扩充——所有这些只需一个简单易用的API。
- **响应用户的页面操作。**即使是最强大和最精心设计的行为，如果我们无法控制它何时发生，那它也毫无用处。jQuery提供了截取形形色色的页面事件（比如用户单击一个链接）的适当方式，而不需要使用事件处理程序搞乱HTML代码。此外，它的事件处理API也消除了经常困扰Web开发人员的浏览器不一致性。
- **为页面添加动态效果。**为了实现某种交互式行为，设计者也必须向用户提供视觉上的反馈。jQuery中内置的一批淡入、擦除之类的效果，以及制作新效果的工具包，为此提供了便利。
- **无需刷新页面即可从服务器获取信息。**这种编程模式就是众所周知的AJAX（Asynchronous JavaScript and XML，异步JavaScript和XML），它能辅助Web开发人员创建出反应灵敏、功能丰富的网站。jQuery通过消除这一过程中的浏览器特定的复杂性，使开发人员得以专注于服务器端的功能设计。
- **简化常见的JavaScript任务。**除了这些完全针对文档的特性之外，jQuery也提供了对基本的JavaScript结构^①（例如迭代和数组操作等）的增强。

1.2 jQuery为什么如此出色

随着近年来人们对动态HTML兴趣的复苏，催生了一大批JavaScript框架。有的特别专注于上述任务中的一项或两项，有的则试图以预打包的形式囊括各种可能的行为和动态效果。为了在维持上述各种特性的同时仍然保持紧凑，jQuery采用了如下策略：

- **利用CSS的优势。**通过将查找页面元素的机制构建于CSS选择符之上，jQuery继承了简明清晰地表达文档结构的方式。由于进行专业Web开发的一个必要条件是掌握CSS语法，因而jQuery成为希望向页面中添加行为的设计者们的切入点。

^① 本书在提到JavaScript中的函数、对象、语句、变量时，经常使用“结构”泛指。——译者注

- **支持扩展。**为了避免特性蠕变（feature creep）^①，jQuery将特殊情况下的用途归入插件当中。创建新插件的方法很简单，而且拥有完备的文档说明，这促进了大量有创意和有实用价值的模块的开发。甚至在下载的基本jQuery库文件当中，多数特性在内部都是通过插件架构实现的。而且，如有必要，可以移除这些内部插件，从而生成更小的库文件。
- **抽象浏览器不一致性。**Web开发领域中一个令人遗憾的事实是，每种浏览器对颁布的标准都有自己的一套不一致的实现方案。任何Web应用程序中都会包含一个用于处理这些平台间特性差异的重要组成部分。虽然不断发展的浏览器前景使得为某些高级特性提供浏览器中立的完美的基础代码（code base）不大可能，但jQuery添加一个抽象层来标准化常见的任务，从而有效地减少了代码量，同时，也极大地简化了这些任务。
- **总是面向集合。**当我们指示jQuery“找到带有‘collapsible’类的全部元素，然后隐藏它们”时，不需要循环遍历每一个返回的元素。相反，`.hide()`之类的方法被设计成自动操作对象集合，而不是单独的对象。这种称作隐式迭代（implicit iteration）的技术，使得大量的循环结构变得不再必要，从而大幅地减少代码量。
- **将多重操作集于一行。**为了避免过度使用临时变量或不必要的代码重复，jQuery在其大多数方法中采用了一种被称作连缀（chaining）^②的编程模式。这种模式意味着基于一个对象进行的大多数操作的结果，都会返回这个对象自身，以便于为该对象应用下一次操作。

这些策略不仅确保了jQuery包的小型化——压缩文件约20 KB，同时，也为我们使用这个库的自定义代码保持简洁提供了技术保障。

jQuery库的适用性一方面归因于其设计理念，另一方面则得益于围绕这个开源项目涌现出的活跃社区的促进作用。jQuery用户聚集到一起，不仅会讨论插件的开发，也会讨论如何增强核心库。附录A中提供了很多jQuery开发人员可用的社区资源的详细信息。

除了为工程师提供灵活且稳健的系统之外，jQuery的最终产品对所有人都是免费的。而且，这个开源项目还具有**GNU Public License**（适合包含在很多其他开源项目中）和**MIT License**（便于在专有的软件中使用jQuery）的双重许可。

1.3 第一个 jQuery 文档

在了解了jQuery能够为我们提供的丰富特性之后，我们可以看一看这个库的实际应用了。

1.3.1 下载 jQuery

jQuery官方网站(<http://jquery.com/>)始终都包含与该库有关的最新代码和新闻的第一手资源。

① 术语feature creep也有人译为特性蔓延，指软件应用开发中过分强调新的功能以至于损害了其他的设计目标，例如简洁性、轻巧性、稳定性及错误出现率等。——译者注
 ② 术语chaining可译为链接，但为避免与人们耳熟能详的超级链接混淆（如常见的“单击链接”等），所以才译为更贴切的连缀。——译者注

为了开始学习，我们需要从官方网站上面下载一个jQuery库文件。官方网站在任何时候都会提供几种不同版本的jQuery库，但其中最适合我们的是该库最新的未压缩版。

使用jQuery库不需安装，只要把下载的库文件放到网站上的一个公共位置即可。因为JavaScript是一种解释型语言，所以使用它不必进行编译或者构建。无论什么时候，当我们想在某个页面上使用jQuery时，只需在相关的HTML文档中简单地引用该库文件的位置。

1.3.2 设置HTML文档

本书多数jQuery应用示例都包含以下3个部分：HTML文档、为该文档添加样式的CSS文件和为该文档添加行为的JavaScript文件。在本书的第一个例子中，我们使用一个包含图书内容提要的页面，同时，该页面中的很多部分都添加了相应的类。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <meta http-equiv="Content-Type" content="text/html;
                                         charset=utf-8"/>
        <title>Through the Looking-Glass</title>
        <link rel="stylesheet" href="alice.css" type="text/css"
              media="screen" />
        <script src="jquery.js" type="text/javascript"></script>
        <script src="alice.js" type="text/javascript"></script>
    </head>
    <body>
        <div id="container">
            <h1>Through the Looking-Glass</h1>
            <div class="author">by Lewis Carroll</div>
            <div class="chapter" id="chapter-1">
                <h2 class="chapter-title">1. Looking-Glass House</h2>
                <p>There was a book lying near Alice on the table, and while
                   she sat watching the White King (for she was still a
                   little anxious about him, and had the ink all ready to
                   throw over him, in case he fainted again), she turned over
                   the leaves, to find some part that she could read, <span
                   class="spoken">"&mdash;for it's all in some language I
                   don't know,"</span> she said to herself.</p>
                <p>It was like this.</p>
                <div class="poem">
                    <h3 class="poem-title">YKCOWREBBAJ</h3>
                    <div class="poem-stanza">
                        <div>sevot yhtils eht dna ,gillirb sawT'</div>
                        <div>;ebaw eht ni elbmig dna eryg diD</div>
                        <div>,sevogorob eht erew ysmim lla</div>
                        <div>.ebargtuo shtar emom eht dnA</div>
                    </div>
                </div>
            </div>
```

```

<p>She puzzled over this for some time, but at last a bright
thought struck her. <span class="spoken">"Why, it's a
Looking-glass book, of course! And if I hold it up to a
glass, the words will all go the right way again."</span></p>
<p>This was the poem that Alice read.</p>
<div class="poem">
    <h3 class="poem-title">JABBERWOCKY</h3>
    <div class="poem-stanza">
        <div>'Twas brillig, and the slithy toves</div>
        <div>Did gyre and gimble in the wabe;</div>
        <div>All mimsy were the borogoves,</div>
        <div>And the mome raths outgrabe.</div>
    </div>
    </div>
</div>
</body>
</html>

```

说明 这个示例文档在服务器上的实际布局并不重要。读者可以根据自己选中的组织结构，调整一个文件中指向另一个文件的引用。在本书多数例子中，都将使用相对路径引用文件（`../images/foo.png`），不使用绝对路径（`/images/foo.png`）。这样，我们不需要使用 Web 服务器就能在本地运行代码。

文档中，紧随常规的 HTML 开头代码之后的是加载样式表文件的代码。在这个例子中，我们使用了一个简单的样式表。

```

body {
    font: 62.5% Arial, Verdana, sans-serif;
}
h1 {
    font-size: 2.5em;
    margin-bottom: 0;
}
h2 {
    font-size: 1.3em;
    margin-bottom: .5em;
}
h3 {
    font-size: 1.1em;
    margin-bottom: 0;
}
.poem {
    margin: 0 2em;
}
.emphasized {
    font-style: italic;
    border: 1px solid #888;
    padding: 0.5em;
}

```

在引用样式表文件的代码之后，是包含JavaScript文件的代码。这里要注意的是，引用jQuery库文件的`<script>`标签，必须放在引用自定义脚本文件的`<script>`标签之前。否则，在我们编写的代码中将引用不到jQuery框架。

说明 在本书其他章中，我们将只展示HTML和CSS文件的相关部分。书中提到文件的完整版可以在本书配套网站<http://book.learningjquery.com>或者出版社网站<http://www.packtpub.com/support>中找到。

现在，这个例子页面的外观如图1-1所示。

The screenshot shows a web page with the title "Through the Looking-Glass" by Lewis Carroll. Below the title is a short text snippet. Then, there is a section titled "1. Looking-Glass House" which contains a poem. The poem starts with "YKCOWREBBAJ" and includes several lines of text. Below the poem, there is another snippet of text. The page has a light gray background with a large watermark "大家网" and a logo "TopSage.com" visible.

图 1-1

接下来，我们就使用jQuery为页面中的诗歌文本添加一种新样式。

提示 这个例子是为了展示jQuery的简单用法而有意设计的。在现实应用中，为页面中的文本添加样式可以通过纯CSS的方式来实现。

1.3.3 编写jQuery代码

我们自定义的代码应该放在第2个、在HTML中使用`<script src="alice.js" type="text/javascript"></script>`引入的空JavaScript文件中。对这个例子而言，我们只需编写3行代码：

```
$ (document) . ready(function() {
    $('.poem-stanza') . addClass('emphasized');
});
```

1. 查找诗歌文本

jQuery中基本的操作就是选择文档中的某一部分。这是通过`$()`结构来完成的。通常，该结构需要一个字符串参数，参数中可以包含任何CSS选择符表达式。在这个例子中，我们想要找到应用了`poem-stanza`类的所有文档部分，因此选择符非常简单。不过，在本书其他章中，我们还会介绍很多更复杂的选择符表达式。在第2章中，我们要讨论的就是查找文档部分的不同方式。

这里用到的`$()`函数实际上是jQuery对象的一个制造工厂。jQuery对象，是我们从现在开始就要打交道的基本的构建块。jQuery对象中会封装零个或多个DOM元素，并允许我们以多种不同的方式与这些DOM元素进行交互。在这个例子中，我们希望修改页面中这些部分的外观，而为了完成这个任务，需要改变应用到诗歌文本的类。

2. 加入新类

本例中`.addClass()`方法的作用是不言而喻的，即它会将一个CSS类应用到我们选择的页面部分。该方法唯一的参数就是要添加的类名。`.addClass()`方法及其反方法`.removeClass()`，为我们探索jQuery支持的各种选择符表达式提供了便利。现在，这个例子只是简单地添加了`emphasized`类，而我们的样式表中为这个类定义的是带边框的斜体文本样式。

我们注意到，无需迭代操作就能为所有诗歌中的节^①添加这个类。前面我们提到过，jQuery在`.addClass()`等方法中使用了隐式迭代机制，因此一次函数调用就可以完成对所有选择的文档部分的修改。

3. 执行代码

综合起来，`$()`和`.addClass()`对我们修改诗歌中文本的外观已经够用了。但是，如果将这段代码单独插入文档的头部，不会有任何效果。通常，JavaScript代码在浏览器初次遇到它们时就会执行，而在浏览器处理头部时，HTML还不会呈现样式。因此，我们需要将代码延迟到DOM可用时再执行。

控制JavaScript代码何时执行的传统机制是在事件处理程序中调用代码。有许多针对用户发起的事件（例如鼠标单击或敲击键盘）的处理程序。在没有jQuery的情况下，我们需要依靠`onload`处理程序，它会在页面（连同其中包含的所有图像）呈现完成后触发。为了在`onload`事件中执行我们的代码，需要先把代码放到一个函数中：

```
function emphasizePoemStanzas() {
    $('.poem-stanza').addClass('emphasized');
}
```

然后，需要修改HTML的`<body>`标签，将这个函数附加给事件：

^① 即类为`.poem-stanza`的文档部分。——译者注

```
<body onload="emphasizePoemStanzas();">
```

这样，当页面加载完成后，我们的代码就会执行。

可是，这种方法存在很多缺点。为了添加行为，我们修改了HTML代码。这种结构与功能紧密耦合的做法，会导致代码混乱。很多页面都可能需要重复调用相同的函数，或者，在处理鼠标单击之类事件的情况下，页面中每个元素的实例也可能需要重复调用相同的函数。这样，添加新的行为将涉及两个不同位置上的改动，不仅增加了出错的可能性，也使设计者与编程人员之间的并行工作流程趋于复杂化。

为了避开这个缺陷，jQuery允许我们使用`$(document).ready()`结构预定DOM加载完成后（不必等待图像加载完成）触发的函数调用。在上面函数定义的基础上，我们可以这样编写代码：

```
$(document).ready(emphasizePoemStanzas);
```

这种技术不需要对HTML进行任何修改。所有行为完全从JavaScript文件内部添加。在第3章中，我们将学习在功能与HTML结构分离的基础上，响应其他类型的用户操作。

不过，由于定义的函数`emphasizePoemStanzas()`马上会用到，并且实际上只使用一次，所以，这种写法仍然有点浪费。同时，这也意味着我们在函数的全局命名空间中使用了一个标识符，还必须记住不能再次使用这个标识符，而这仅仅是为一时方便。同其他编程语言一样，JavaScript也有一种解决这种低效率做法的方式，叫做匿名函数（有时候也称为**lambda函数**）。现在，我们就把代码改回初始状态：

```
$(document).ready(function() {
  $('.poem-stanza').addClass('emphasized');
});
```

通过使用不带函数名称的`function`关键字，我们在实际需要它的地方（而不是提前）定义了一个函数。这样在消除混乱的同时，也把我们带回了3行JavaScript代码的状态。在jQuery中，这种习惯用法非常方便，因为很多方法都需要一个几乎不会再重用的函数作为参数。

当在另一个函数的主体内使用这种语法定义了一个匿名函数时，就创建了一个闭包。闭包是一种高级而强大的概念，但由于它可能会带来意想不到的后果和内存使用上的问题，所以，在大量使用嵌套的函数定义之前，必须要透彻地理解它的原理。我们将在附录C中对这个主题进行充分的讨论。

4. 最终结果

在编写好JavaScript代码之后，现在的页面外观会变成如图1-2所示。

由于JavaScript插入了`emphasized`类，页面中的两节诗歌文本变成了斜体，并且被包含在了方框中。

Through the Looking-Glass

by Lewis Carroll

1. Looking-Glass House

There was a book lying near Alice on the table, and while she sat watching the White King (for she was still a little anxious about him, and had the ink all ready to throw over him, in case he fainted again), she turned over the leaves, to find some part that she could read,—"for it's all in some language I don't know," she said to herself.

It was like this.

YKCOWREBAJ

```
sevot yhtis eht dna ,gillirb sawT
,ebaw eht ni elbmig dna eryg dlD
,sevogorob eht erew ysmim lla
.ebargtuo shtar emom eht dna
```

She puzzled over this for some time, but at last a bright thought struck her. "Why, it's a Looking-glass book, of course! And if I hold it up to a glass, the words will all go the right way again."

This was the poem that Alice read.

JABBERWOCKY

```
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
```

图 1-2

1.4 小结

经过对本章的学习，我们对开发者选择使用JavaScript框架，而不是从零开始编写代码（即使是对最基本的任务）的原因有了一个概念。同时，也理解了jQuery作为一个框架，都有哪些值得称道的地方，以及我们选择它而不是别的框架的理由。而且，我们也大体上知道了jQuery能够简化哪些任务。

在本章中，我们学习了怎样设置jQuery，以便网页中的JavaScript使用它；学习了使用\$()工厂函数查找具有给定类的页面部分；学习了调用.addClass()为页面的这些部分应用额外的样式；还学习了调用\$(document).ready()基于页面加载来执行这些代码。

本章中举的这个示范如何使用jQuery的简单例子，在现实中并不是很有用。在下一章中，我们会在此基础上继续探索jQuery中高级的选择符使用方式，并介绍这一技术的实际应用。

选择符——取得你想要的一切

*She's just the girl
She's just the girl
The girl you want*
—Devo,
"Girl U Want"

jQuery利用了CSS和Xpath选择符的能力，让我们在DOM中快捷而轻松地获取元素或元素组。在本章中，我们将探索一些CSS和XPath选择符，以及jQuery独有的自定义选择符。此外，还将介绍一下jQuery的DOM遍历方法，这些方法为取得目标元素提供了更大的灵活性。

2.1 DOM

jQuery最强大的方面之一就是它能够简化DOM遍历任务。文档对象模型与家谱有几分类似。同其他标记语言一样，HTML也使用这个模型来描述页面中元素之间的关系。当我们提到这些关系时，将使用与描述家庭关系时相同的术语，即父、子等。通过一个简单的例子，可以帮助我们理解将文档比喻为家谱：

```
<html>
  <head>
    <title>the title</title>
  </head>
  <body>
    <div>
      <p>This is a paragraph.</p>
      <p>This is another paragraph.</p>
      <p>This is yet another paragraph.</p>
    </div>
  </body>
</html>
```

这里，`<html>`是其他所有元素的祖先元素，换句话说，其他所有元素都是`<html>`的后代元素。`<head>`和`<body>`元素是`<html>`的子元素。因此除了作为`<head>`和`<body>`的祖先元素之外，`<html>`也是它们的父元素。而`<p>`元素则是`<div>`的子元素（也是后代元素），是`<body>`和`<html>`

的后代元素，是其他

元素的同辈元素。要了解如何使用第三方软件形象化地展示DOM中的家族树结构，请参考附录B。

在开始本章的内容之前，还要提醒大家注意的另一个重要的问题，就是我们通过各种选择符和方法取得的结果集合实际上都是一个jQuery对象。当我们想要实际地操纵在页面中找到的元素时，通过jQuery对象会非常简单。既可以轻松地为jQuery对象绑定事件，或者添加漂亮的效果，也可以将多重修改或效果通过jQuery对象连缀到一起。然而，jQuery对象与常规的DOM元素不同，而且也没有必要为实现某些任务给纯DOM元素添加相同的方法和属性。在本章的最后一部分中，我们会介绍如何访问包装在jQuery对象中的DOM元素。

2.2 工厂函数\$()

在jQuery中，无论我们使用哪种类型的选择符（不管是CSS、XPath，还是自定义的选择符），都要从一个美元符号和一对圆括号开始：\$()。

在第1章中我们曾经提到过，\$()函数会消除使用for循环访问一组元素的需求，因为放到圆括号中的任何元素都将自动执行循环遍历，并且会被保存到一个jQuery对象中。可以在\$()函数的圆括号中使用的参数几乎没有什么限制。比较常用的一些例子如下。

- 标签名：\$('p')会取得文档中所有的段落。
- ID：\$('#some-id')会取得文档中具有对应的some-id ID的一个元素。
- 类：\$('.some-class')会取得文档中带有some-class类的所有元素。

提示 让jQuery与其他JavaScript库有效地协同

在jQuery中，美元符号\$只不过是对标识符jQuery的一种简写方式。由于\$()在JavaScript库中很常见，所以，如果在一个页面中使用了几个这样的库，那么就会导致冲突。在这种情况下，可以在我们自定义的jQuery代码中，通过将每个\$的实例替换成jQuery来避免这种冲突。第10章还会介绍对这个问题的其他解决方案。

在介绍了基本的情况之后，下面我们就开始探索选择符的一些更强大的用途。

2.3 CSS 选择符

jQuery支持CSS规范1到规范3中的大多数选择符，具体内容可以参考W3C（World Wide Web Consortium，万维网联盟）网站<http://www.w3.org/Style/CSS/#specs>。这种对CSS选择符的支持，使得开发者在增强自己的网站时，不必为哪种浏览器（特别是IE 6及更低版本）可能会不理解高级的选择符而担心，只要该浏览器启用了JavaScript就没有问题。

说明 负责任的jQuery开发者应该在编写自己的程序时，始终坚持渐进增强（progressive enhancement）和平稳退化（graceful degradation）的理念，做到在JavaScript禁用时，页面仍然能够与启用JavaScript时一样准确地呈现，即使没有那么美观。贯穿本书，我们还将继续探讨这些理念。

为了学习在jQuery中如何使用CSS选择符，我们选择了一个很多网站中都会有的通常用于导航的结构——嵌套的无序列表。

```
<ul id="selected-plays">
  <li>Comedies
    <ul>
      <li><a href="http://www.mysite.com/asyoulikeit/">
          As You Like It</a></li>
      <li>All's Well That Ends Well</li>
      <li>A Midsummer Night's Dream</li>
      <li>Twelfth Night</li>
    </ul>
  </li>
  <li>Tragedies
    <ul>
      <li><a href="hamlet.pdf">Hamlet</a></li>
      <li>Macbeth</li>
      <li>Romeo and Juliet</li>
    </ul>
  </li>
  <li>Histories
    <ul>
      <li>Henry IV (<a href="mailto:henryiv@king.co.uk">email</a>)
        <ul>
          <li>Part I</li>
          <li>Part II</li>
        </ul>
      <li><a href="http://www.shakespeare.co.uk/henryv.htm">
          Henry V</a></li>
        <li>Richard III</li>
      </ul>
    </li>
  </ul>
```



我们注意到，其中第一个``具有一个值为`selected-plays`的ID，但``标签则全都没有与之关联的类。在没有应用任何样式的情况下，这个列表的外观如图2-1所示。

图2-1中的嵌套列表按照我们期望的方式显示——一组带符号的列表项垂直排列，并且每个列表都按照各自的级别进行了缩进。

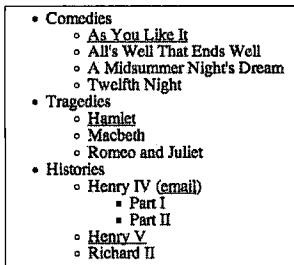


图 2-1

基于列表项的级别添加样式

假设我们想让顶级的项，而且只有顶级的项水平排列。那么我们可以先在样式表中定义一个 `horizontal` 类：

```
.horizontal {
  float: left;
  list-style: none;
  margin: 10px;
}
```

这个 `horizontal` 类会将元素浮动到它后面元素的左侧，如果这个元素是一个列表项，那么会移除其项目符号，最后，再为该元素的每一边各添加10像素的外边距。

这里，我们没有直接在HTML中添加 `horizontal` 类，而只是将它动态地添加给位于顶级的列表项 `Comedies`、`Tragedies` 和 `Histories`，以便示范jQuery中选择符的用法：

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
});
```

我们在第1章讨论过，当在jQuery代码中使用 `$(document).ready()` 结构时，位于其中的所有代码都会在DOM加载后立即执行。

第2行代码使用子元素组合符 (`>`) 将 `horizontal` 类只添加到位于顶级的项中。实际上，位于 `$()` 函数中的选择符的含义是，查找ID为 `selected-plays` 的元素 (`#selected-plays`) 的子元素 (`>`) 中所有的列表项 (`li`)。

随着这个类的应用，现在的嵌套列表如图2-2所示。

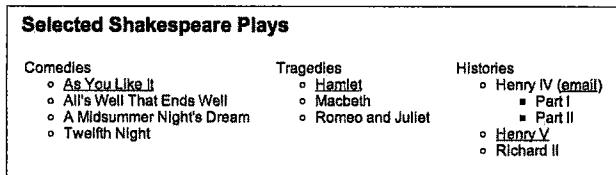


图 2-2

要为其他项（非顶级的项）添加样式，有很多种方式。因为已经为顶级项添加了horizontal类，所以取得全部非顶级项的一种方式，就是使用否定式伪类选择符来识别没有horizontal类的所有列表项。注意下面添加的第3行代码：

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');
});
```

这一次取得的每个列表项（li）：

- (1) 是ID为selected-plays的元素（#selected-plays）的后代元素。
- (2) 没有horizontal类（:not(.horizontal)）。

在为这些列表项添加了sub-level类之后，它们取得了在样式表规则.sub-level {background-color: #ffc;}中定义的浅黄的背景颜色。此时的嵌套列表如图2-3所示。

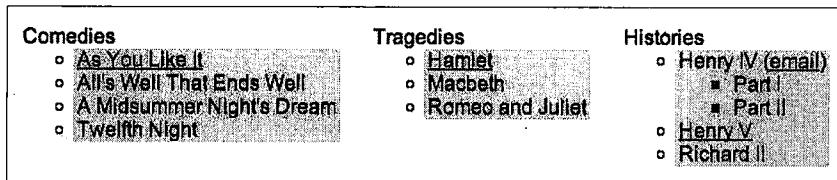


图 2-3

2.4 XPath 选择符

XPath（XML Path Language， XML路径语言）是在XML文档中识别不同元素或者元素值的一种语言，与CSS在HTML文档中识别元素的方式类似。jQuery库支持一组基本的XPath选择符，如果愿意也可以将它们与CSS选择符一同使用。而且，在jQuery中，无论对什么类型的文档都可以使用XPath和CSS选择符。

在涉及属性选择符（attribute selector）时，jQuery使用了XPath中的惯例来标识属性，即将属性前置一个@符号并放在一对方括号中。也就是说，jQuery没有采用CSS中不够灵活的属性选择符语法。例如，要选择所有带title属性的链接，可以使用下面的代码：

```
$('.a[@title]')
```

此外，方括号在XPath语法中还有另外一种用途，即在不带前置@符号的情况下，可以用来指定包含在另一个元素中的元素。例如，我们可以通过下面的选择符表达式，取得包含一个ol元素的所有div元素^①：

^① 请读者注意，属性选择符方括号中的字符串，仅作为筛选标准存在，用来辅助选择目标对象；方括号左侧的元素才是属性选择符真正要选择的目标对象。——译者注

```
$('div[ol]')
```

为链接添加样式

属性选择符允许以类似正则表达式的语法来标识字符串的开始 (^) 和结尾 (\$)。而且，也可以使用星号 (*) 表示字符串中的任意位置。

假设我们想以不同的文本颜色来显示不同类型的链接。那么，首先要在样式表中定义如下样式：

```
a {
    color: #00f; /* 普通链接显示为蓝色 */
a.mailto {
    color: #f00; /* 电子邮件链接显示为红色 */
}
a.pdflink {
    color: #090; /* 指向PDF文件的链接显示为绿色 */
}
a.mysite {
    text-decoration: none; /* 移除内部链接的下划线 */
    border-bottom: 1px dotted #00f;
}
```

然后，可以使用jQuery为符合条件的链接添加3个类：mailto、pdflink和mysite。

要取得所有电子邮件链接，需要构造一个选择符，用来寻找所有带href属性 ([@href]) 且以mailto开头 (^="mailto:") 的锚元素 (a)。结果如下所示：

```
$(document).ready(function() {
    $('a[@href^="mailto:"]').addClass('mailto');
});
```

要取得所有指向PDF文件的链接，需要使用美元符号 (\$) 而不是脱字符号 (^)，来取得所有带href属性并以.pdf结尾的链接，相应的代码如下所示：

```
$(document).ready(function() {
    $('a[@href^="mailto:"]').addClass('mailto');
    $('a[@href$=".pdf"]').addClass('pdflink');
});
```

最后，要取得所有内部链接，即到mysite.com中其他页面的链接，则需要使用星号：

```
$(document).ready(function() {
    $('a[@href^="mailto:"]').addClass('mailto');
    $('a[@href$=".pdf"]').addClass('pdflink');
    $('a[@href*="mysite.com"]').addClass('mysite');
});
```

这里，mysite.com可以出现在href属性值中的任何位置上。如果我们也想选择指向mysite.com

内部所有子域名的链接，使用这个选择符特别关键^①。

随着这3个类被应用到3种类型的链接中，我们应该看到下列应用的样式。

- 带点划线的蓝色文本：`As You Like It`
- 绿色文本：`Hamlet`
- 红色文本：`email`

图2-4是添加这些样式之后的一幅屏幕截图。



图 2-4

2.5 自定义选择符

除了CSS和XPath选择符之外，jQuery还添加了独有的完全不同的自定义选择符。可以说，jQuery中的多数自定义选择符都可以让我们基于某个标准选出特定的元素。自定义选择符的语法与CSS中的伪类选择符语法相同，即选择符以一个冒号（:）开头。例如，我们想要从匹配的带有horizontal类的div集合中，选择第2个项，那么应该使用下面的代码：

```
$('div.horizontal:eq(1)')
```

注意，因为JavaScript数组采用从0开始的编号方式，所以`eq(1)`取得的是集合中的第2个元素。而CSS则是从1开始的，因此CSS选择符`'div:nth-child(1)'`取得的是作为其父元素第1个子元素的所有div。

交替地为表格行添加样式

jQuery库中的两个十分有用的自定义选择符是`:odd`和`:even`。下面，我们就来看一看如何通过这两个选择符为表格添加基本的条纹样式，针对下面的表格：

```
<table>
<tr>
  <td>As You Like It</td>
```

^① 这里指的应该是使用顶级域名mysite.com，而不是使用具体的子域名www.mysite.com。否则，会将排除www之外的任何子域名，如blog.mysite.com等。——译者注

```

<td>Comedy</td>
</tr>
<tr>
  <td>All's Well that Ends Well</td>
  <td>Comedy</td>
</tr>
<tr>
  <td>Hamlet</td>
  <td>Tragedy</td>
</tr>
<tr>
  <td>Macbeth</td>
  <td>Tragedy</td>
</tr>
<tr>
  <td>Romeo and Juliet</td>
  <td>Tragedy</td>
</tr>
<tr>
  <td>Henry IV, Part I</td>
  <td>History</td>
</tr>
<tr>
  <td>Henry V</td>
  <td>History</td>
</tr>
</table>

```

我们可以向样式表中添加两个类，一个用于为奇数行添加样式，另一个用于为偶数行添加样式：

```

.odd {
  background-color: #ffc; /* 奇数行的背景颜色为浅黄色 */
}
.even {
  background-color: #cef; /* 偶数行的背景颜色为浅蓝色 */
}

```

最后，编写jQuery代码，将这两个类添加到表格行（`<tr>`标签）中：

```

$(document).ready(function() {
  $('tr:odd').addClass('odd');
  $('tr:even').addClass('even');
});

```

如此简单的一点代码，就能产生如图2-5所示的表格。

乍一看，表格行的背景颜色似乎与我们的意图相反。但是，与`:eq()`选择符一样，`:odd()`和`:even()`选择符使用的都是JavaScript本身的从0开始的编号方式。因此，表格的第一行编号为0（偶数），而表格的第二行编号是1（奇数），然后依此类推。

As You Like It	Comedy
All's Well that Ends Well	Comedy
Hamlet	Tragedy
Macbeth	Tragedy
Romeo and Juliet	Tragedy
Henry IV, Part I	History
Henry V	History

图 2-5

不过，要注意的是，如果一个页面上存在多个表格，我们则真有可能会看到意料之外的结果。例如，因为这个表格中的最后一行拥有浅蓝色背景，所以下一个表格的第一行就会带有浅黄色的背景。我们将在第7章介绍解决这个问题的方法。

下面，我们介绍最后一个自定义选择符。假设出于某种原因，我们希望突出显示提到任何一种Henry游戏的所有表格单元。为此，我们所要做的就是在样式表中添加一个声明了粗体和红色文本的类 (`.highlight {font-weight:bold; color: #f00;}`)，然后向jQuery代码中添加一行代码，其中使用的是`:contains()`选择符。

```
$ (document).ready(function() {
  $('tr:odd').addClass('odd');
  $('tr:even').addClass('even');
  $('td:contains("Henry")').addClass('highlight');
});
```

这样，在我们可爱的条纹表格中，就能够看到突出显示的Henry游戏了，如图2-6所示。

As You Like It	Comedy
All's Well that Ends Well	Comedy
Hamlet	Tragedy
Macbeth	Tragedy
Romeo and Juliet	Tragedy
Henry IV, Part I	History
Henry V	History

图 2-6

诚然，不使用jQuery（或任何客户端编程语言）也可以通过其他方式实现这种突出显示效果。然而，jQuery加上CSS，在内容由程序动态生成，而我们又无权改动HTML和服务器端代码的情况下，对这种样式化操作提供了优秀的替换方案。

2.6 DOM 遍历方法

利用前面介绍的jQuery选择符取得一组元素，就像是我们在DOM树中纵横遍历再经过筛选得

到结果一样。如果只有这一种取得元素的方式，那我们选择的余地也是很有限的（尽管坦诚地讲，选择符表达式本身还是很强的，特别是与常规的DOM脚本编程相比）。很多情况下，取得某个元素的父元素或者祖先元素都是基本的操作。而这正是jQuery的DOM遍历方法的用武之地。有了这些方法，我们可以轻而易举地在DOM树中上、下、左、右地自由漫步。

其中一些方法与选择符表达式有异曲同工之妙。例如，这行用于添加odd类的代码`$('tr:odd').addClass('odd');`，可以通过`.filter()`方法重写成下面这样：

```
$('tr').filter(':odd').addClass('odd');
```

而且，这两种取得元素的方式在很大程度上可以互为补充。下面，我们仍以条纹式表格为例，看看通过这些方法如何实现同样的效果。

首先，表格中要使用标题行，因此需要在表格中添加带有两个`<th>`元素（不是`<td>`元素）的另一个`<tr>`元素：

```
[...]
<tr>
  <th>Title</th>
  <th>Category</th>
</tr>
[...]
```

提示 也可以通过更语化的方式来标记表格，即将标题行放在`<thead>`和`</thead>`之间，而将其他的行放在`<tbody>`和`</tbody>`之间。不过，出于简化例子的考虑，我们在此没有使用更多的标记。

我们会为标题行添加不同于其他行的样式，使它带有粗体和黄色的背景（而不是浅蓝色）。如果不修改，那么这一行就是那样了。

其次，我们客户看了看这个页面，很喜欢这些条纹。不过，客户想让红色的粗体文本出现在Henry行的类别（category）单元格中，而不是出现在标题（title）单元格中。

1. 为标题行添加样式

为标题行添加不同样式的任务，可以通过找到`<th>`标签并取得它们的父元素来完成。而其他行则可以通过组合使用CSS、XPath以及自定义选择符过滤`<tr>`元素来选择并添加样式。所需的代码如下所示：

```
$(document).ready(function() {
  $('th').parent().addClass('table-heading');
  $('tr:not([th]):even').addClass('even');
  $('tr:not([th]):odd').addClass('odd');
  $('td:contains("Henry")').addClass('highlight');
});
```

对于标题行，我们通过`parent()`方法取得了一个普通的父元素，没有在圆括号中包含任何参数。因为我们知道这个父元素就是`<tr>`，而且结果集合中只有这一个元素。虽然我们会认为由于存在两个`<th>`元素，所以`<tr>`中最终会包含两个`table-heading`类，但jQuery很聪明，它不会向元素中重复添加同一个类名。

对于表体中的行，我们首先排除了后代元素中包含`<th>`的`<tr>`元素，进而，又应用了`:odd`或`:even`过滤器。选择符的次序在这里很重要，如果我们在那里使用的是`$('tr:odd:not([th])')`，而不是`$('tr:not([th]):odd')`，那么最终的表格会完全不同。

2. 为Category单元格添加样式

为了给包含Henry的单元格之后的单元格添加样式，我们可以在已经写出的选择符基础上，向其中添加`.next()`方法：

```
$ (document).ready(function() {
    $('the').parent().addClass('table-heading');
    $('tr:not([th]):even').addClass('even');
    $('tr:not([th]):odd').addClass('odd');
    $('td:contains("Henry")').next().addClass('highlight');
});
```

在添加了`table-heading`类和将`highlight`类应用到Category列的单元格之后，表格的外观如图2-7所示。

Title	Category
As You Like It	Comedy
All's Well that Ends Well	Comedy
Hamlet	Tragedy
Macbeth	Tragedy
Romeo and Juliet	Tragedy
Henry IV, Part I	History
Henry V	History

图 2-7

其中，`.next()`方法只取得最接近的下一个同辈元素。那么，要是表格中包含更多的列怎么办呢？比如，如果还有一个`Year Published`列，那么只要所在行的`Title`列中包含`Henry`，我们希望该列中的文本也突出显示。换句话说，对于一个单元格中包含`Henry`的行，我们想要取得该行中的其他所有单元格。通过组合不同的选择符表达式和jQuery方法，有如下几种方式可以达到我们的目的。

(1) 取得包含Henry的单元格，然后取得该单元格的所有同辈元素（不仅仅是下一个同辈元素）。最后添加类：

```
$('.td:contains("Henry")').siblings().addClass('highlight');
```

(2) 取得包含Henry的单元格，再取得它的父元素，然后找到该元素中包含的所有编号大于0（0是第1单元格）的单元格。最后添加类：

```
$('.td:contains("Henry")').parent().find('td:gt(0)')
    .addClass('highlight');
```

(3) 取得包含Henry的单元格，再取得它的父元素，找到该元素中包含的所有单元格，然后过滤这些单元格排除包含Henry的那一个。最后添加类：

```
$('.td:contains("Henry")').parent().find('td').not(':
    contains("Henry")').addClass('highlight');
```

(4) 取得包含Henry的单元格，再取得它的父元素，找到该元素包含的子元素中的第2个单元格，添加类；取消最后一次`.find()`，再查找该元素包含的子元素中的第3个单元格，添加类：

```
$('.td:contains("Henry")').parent().find('td:eq(1)').addClass(
    'highlight').end().find('td:eq(2)').addClass('highlight');
```

所有这些方法都会得到相同的结果，如图2-8所示。

Title	Category	Year Published
As You Like It	Comedy	
All's Well that Ends Well	Comedy	1601
Hamlet	Tragedy	1604
Macbeth	Tragedy	1606
Romeo and Juliet	Tragedy	1595
Henry IV, Part I	History	1596
Henry V	History	1599

图 2-8

为了代码清晰起见，以上组合选择符表达式和方法的这几种方式并不全都是我们所推荐的。实际上，第4种转弯抹角的方式几乎达到了荒谬的程度。不过，通过这些方式应该能够体现出jQuery的DOM遍历方式的确极具灵活性。

3. 连缀

以上几种方式同时也示范了jQuery的连缀能力。在jQuery中，可以通过一行代码取得多个元素集合并对这些元素集合执行多次操作。而且，为了获得更佳的可读性，也可以把一行代码分成多行。例如，上一节中的第4种方式可以写成7行代码，而且每行代码都可以包含自己的注释，但本质上它们仍然是一行代码：

```
$('td:contains("Henry")')          //取得包含“Henry”的所有单元格
.parent()                           //取得它的父元素
.find('td: eq(1)')                 //在父元素中查找第2个单元格
.addClass('highlight')             //为该单元格添加“highlight”类
.end()                             //恢复到包含“Henry”的单元格的父元素
.find('td: eq(2)')                 //在父元素中查找第3个单元格
.addClass('highlight');           //为该单元格添加“highlight”类
```

连缀就像是一口气说出一大段话——虽然效率很高，但对别人来说可能会难于理解。而将它分开放到多行并添加明确的注释，从长远来看则可以节省更多的时间。

2.7 访问DOM元素

所有选择符表达式和多数jQuery方法都返回一个jQuery对象，而这通常都是我们所希望的，因为jQuery对象能够提供隐式迭代和连缀能力。

尽管如此，我们仍然有需要在代码中直接访问DOM元素的时候。例如，可能需要为另一个JavaScript库提供一组元素的结果集合。或者，可能不得不访问某个元素的标签名。对于这些少见但合理的情形，jQuery提供了`.get()`方法。要访问jQuery对象引用的第一个DOM元素，可以使用`.get(0)`。如果需要在循环中使用DOM元素，那么应该使用`.get(index)`。因而，如果想知道带有`id="my-element"`属性的元素的标签名，应该使用如下代码：

```
var myTag = $('#my-element').get(0).tagName;
```

为了进一步简化这些代码，jQuery还为`.get()`方法提供了一种简写方式。比如，可以将`$('#my-element').get(0)`简写为`$('#my-element')[0]`，也就是说，可以在选择符后面直接使用方括号。显然，这种语法与访问DOM元素数组很相似，而使用方括号就好像剥掉jQuery的包装并直接取得这些元素。

2.8 小结

通过本章介绍的技术，应该掌握了如何使用CSS选择符为嵌套列表中的顶级和非顶级项分别添加样式，如何使用XPath属性选择符为不同类型的链接应用不同的样式，如何使用自定义的jQuery选择符`:odd`和`:even`为表格添加条纹效果，以及如何使用连缀的jQuery方法突出显示某个表格单元中的文本。

到现在为止，我们使用了`$(document).ready()`事件为一组匹配的元素添加类。在下一章中，我们将探索基于用户发起的事件来添加类的技术。

事件——扣动扳机

Getting bigger, pull the trigger

—Devo,

“Puppet Boy”

JavaScript内置了一些对用户的交互和其他事件给予响应的方式。为了使页面具有动态性和响应性，就需要利用这种能力，以便在适当的时候，将我们学过的jQuery技术派上用场。虽然使用普通的JavaScript也可以做到这一点，但jQuery增强并扩展了基本的事件处理机制。它不仅提供了更加优雅的事件处理语法，而且也极大地增强了事件处理机制。

3.1 在页面加载后执行任务

我们已经看到如何让jQuery响应网页的加载事件，`$(document).ready()`事件处理程序可以用来触发函数中的代码，但对这个过程还有待于进行深入分析。

3.1.1 代码执行的时机选择

在第1章中，我们知道了`$(document).ready()`是jQuery中响应JavaScript内置的`onload`事件并执行任务的一种典型方式。虽然`$(document).ready()`和`onload`具有类似的效果，但是，它们在触发操作的时间上存在着微妙的差异。

当一个文档完全下载到浏览器中时，会触发`window.onload`事件。这意味着页面上的全部元素对JavaScript而言都是可以访问的，这种情况对编写功能性的代码非常有利，因为无需考虑加载的次序。

另一方面，通过`$(document).ready()`注册的事件处理程序，则会在DOM完全就绪并可以使用时调用。虽然这也意味着所有元素对脚本而言都是可以访问的，但是，却不意味着所有关联的文件都已经下载完毕。换句话说，当HTML下载完成并解析为DOM树之后，代码就可以运行。

举一个例子，假设有一个表现图库的页面，这种页面中可能会包含许多大型图像，我们可以通过jQuery隐藏、显示或以其他方式操纵这些图像。如果我们通过`onload`事件设置界面，那么用

户在能够使用这个页面之前，必须要等到每一幅图像都下载完成。更糟糕的是，如果行为尚未添加给那些具有默认行为的元素（例如链接），那么用户的交互可能会导致意想不到的结果。然而，当我们使用`$(document).ready()`进行设置时，这个界面就会更早地准备好可用的正确行为。

说明 使用`$(document).ready()`一般来说都要优于使用`onload`事件处理程序，但必须要明确的一点是，因为支持文件可能还没有加载完成，所以类似图像的高度和宽度这样的属性此时则不一定有效。如果需要访问这些属性，可能就得选择实现一个`onload`事件处理程序（或者更类似于jQuery中`.load()`^①的等效方法）。这两种机制能够和平共存。

3.1.2 基于一个页面执行多个脚本

通过JavaScript（而不是指直接在HTML中添加处理程序属性）注册事件处理程序的传统机制是，把一个函数指定给DOM元素的对应属性。例如，假设我们已经定义了如下函数：

```
function doStuff() {
    // 执行某种任务……
}
```

那么，我们既可以在HTML标记中指定该函数：

```
<body onload="doStuff();">
```

也可以在JavaScript代码中指定该函数：

```
window.onload = doStuff;
```

这两种方式都会导致在页面加载完成后执行这个函数。但第2种方式的优点在于，它能使行为更清晰地从标记中分离出来。现在，假设我们又定义了第2个函数：

```
function doOtherStuff() {
    // 执行另外一种任务……
}
```

我们也可以将它指定为基于页面的加载来运行：

```
window.onload = doOtherStuff;
```

然而，这次指定的函数会取代刚才指定的第1个函数。因为`.onload`属性一次只能保存对一个函数的引用，所以不能在现有的行为基础上再增加新行为。

对于这种情况下，通过`$(document).ready()`机制能够得到很好的处理。每次调用这个方法^②都会向内部的行为队列中添加一个新函数，当页面加载完成后，所有函数都将得到执行。而

① `.load()`方法的用途是在每一个匹配元素的`load`事件中绑定一个处理程序。——译者注

② 即每次调用`$(document).ready()`方法。——译者注

且，这些函数会按照注册它们的顺序依次执行^①。

说明 公平地讲，jQuery并不是解决这个问题的唯一方法。我们可以编写一个JavaScript函数，用它构造一个调用现有的onload事件处理程序的新函数，然后再调用一个传入的事件处理程序。这种在Simon Willison的addLoadEvent()中使用的手段^②，可以避免\$(document).ready()这类对抗性处理程序之间的冲突，但是却不具有我们刚才所讨论的那些优点。

3.1.3 缩短代码的简写方式

前面提到的\$(document).ready()结构，实际上是在基于document这个DOM元素构建而成的jQuery对象上，调用了.ready()方法。因为这是一个常用方法，所以\$()函数为我们提供了一种简写方式。当调用这个函数而不传递参数时，该函数的行为就像是传递了document参数。也就是说，对于：

```
$(document).ready(function() {
    // 这里是代码……
});
```

也可以简写成：

```
$(().ready(function() {
    // 这里是代码……
});
```

此外，这个工厂函数也可以接受另一个函数作为参数。此时，jQuery会在内部执行一次对.ready()的隐含调用，因此，使用下面的代码也可以得到相同的结果：

```
$(function() {
    // 这里是代码……
});
```

虽然这几种语法都更短一些，但程序设计者们通常更喜欢使用较长的形式，因为较长的形式能够更清楚地表示代码在做什么。

3.2 简单的事件

除了页面加载之外，我们也想在其他很多时候完成某个任务。正如JavaScript可以让我们通过<body onload="">或者window.onload来截取页面加载事件一样，它对用户发起的事件也提供了相似的“挂钩（hook）”。例如，鼠标单击（onclick）、表单被修改（onchange）以及窗口大

① 通过window.onload虽然也可以注册多个函数，但却不能保证按顺序执行。——译者注

② 关于这个addLoadEvent()方法，读者可以参考<http://simonwillison.net/2004/May/26/addLoadEvent/>。——译者注

小变化（onresize）等。在这些情况下，如果直接在DOM中为元素指定行为，那么这些挂钩也会与我们讨论的onload一样具有类似的缺点。为此，jQuery也为处理这些事件提供了一种改进的方式。

3.2.1 简单的样式转换器

为了说明某些事件处理技术，我们假设希望一个页面能够基于用户的输入呈现出不同的样式。也就是说，允许用户通过单击按钮在正常视图、将文本限制在窄列中的视图和适合打印的大字内容区视图之间进行切换。

在现实当中，一个优秀的Web公民应该在这里遵守渐进增强的原则。页面上的样式转换器应该能够在JavaScript无效时隐藏起来，甚至更好的是，能够通过链接到当前页面的替代版本而仍然起到应有的作用^①。出于简化例子的考虑，我们假设所有用户都启用了JavaScript。

用于样式转换器的HTML标记如下所示：

```
<div id="switcher">
  <h3>Style Switcher</h3>
  <div class="button selected" id="switcher-normal">Normal</div>
  <div class="button" id="switcher-narrow">Narrow Column</div>
  <div class="button" id="switcher-large">Large Print</div>
</div>
```

在与页面中其他HTML标记和基本的CSS组合以后，我们可以看到如图3-1所示的页面外观。

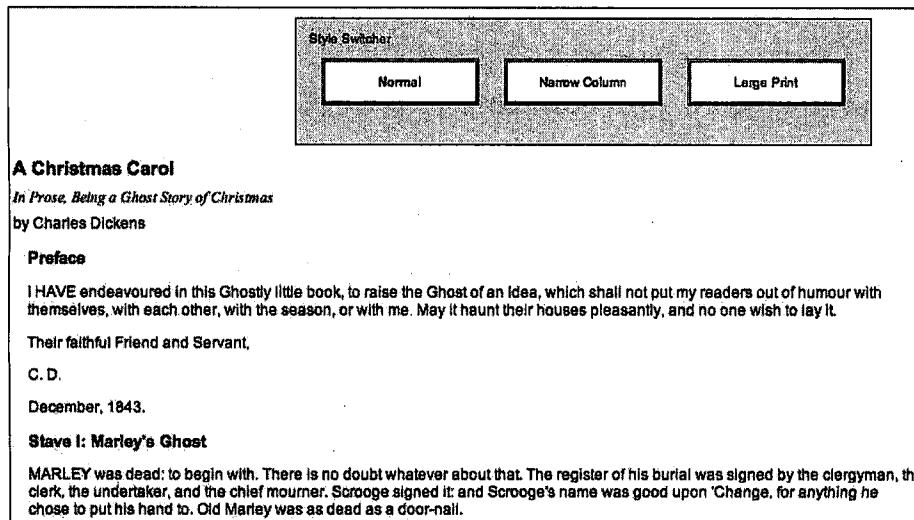


图 3-1

^① 即在用户禁用或不能使用JavaScript的情况下提供链接，指向包含另外两种视图的页面。——译者注

首先，我们来编写Large Print按钮的功能。此时，需要一点CSS代码来实现页面的替换视图：

```
body.large .chapter {
    font-size: 1.5em;
}
```

然后，我们的目标就是为body标签应用large类。这样会导致样式表对页面进行重新格式化。显然，添加类的语句应该如下所示：

```
$( 'body' ).addClass( 'large' );
```

但是，我们希望这条语句在用户单击按钮时执行。为此，我们需要引入.bind()方法。通过这个方法，可以指定任何JavaScript事件，并为该事件添加一种行为。此时，事件是click，而行为则是由上面的一行代码构成的函数：

```
$(document).ready(function() {
    $('#switcher-large').bind('click', function() {
        $('body').addClass('large');
    });
});
```

现在，当单击Large Print按钮时，就会运行函数中的代码，而页面的外观将如图3-2所示。

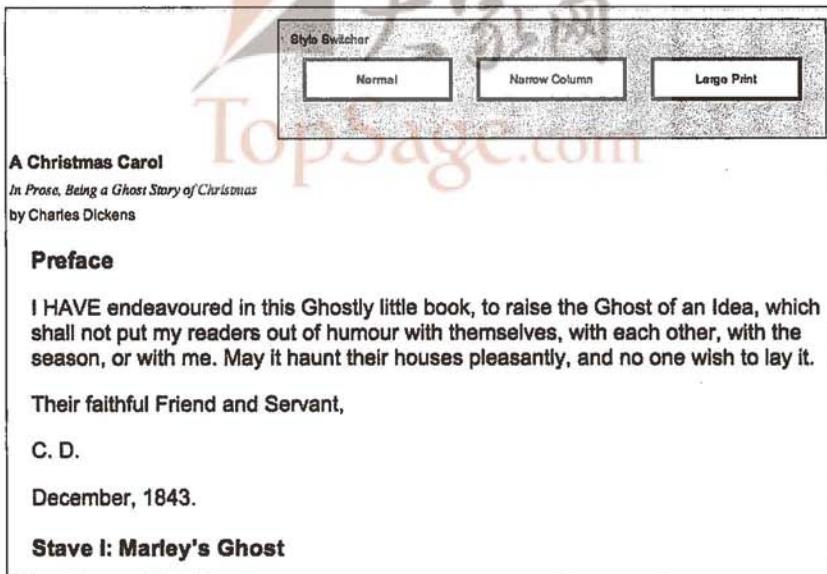


图 3-2

这里的全部操作就是绑定了一个事件。我们前面介绍的.ready()方法的优点在此也同样适用。多次调用.bind()也没有任何问题，即可以按照需要为同一个事件追加更多的行为。

但是，这还不是完成上述任务的最优雅或者说最有效的方式。随着本章内容的展开，我们会对刚才的代码加以扩展和改进，使其达到足以令我们自豪的水平。

1. 启用其他按钮

现在，Large Print按钮就像是广告一样地开始生效了。接下来，我们要以类似的方式处理其他两个按钮，让它们也都执行各自的任务。这个过程很简单，即分别使用`.bind()`为它们添加一个单击处理程序，同时视情况移除或添加类。完成之后的代码如下所示：

```
$ (document).ready(function() {
    $('#switcher-normal').bind('click', function() {
        $('body').removeClass('narrow');
        $('body').removeClass('large');
    });
    $('#switcher-narrow').bind('click', function() {
        $('body').addClass('narrow');
        $('body').removeClass('large');
    });
    $('#switcher-large').bind('click', function() {
        $('body').removeClass('narrow');
        $('body').addClass('large');
    });
});
```

以下是配套的narrow类的CSS规则：

```
body.narrow .chapter {
    width: 400px;
}
```

现在，如果单击Narrow Column按钮，随着相应的CSS生效，页面的外观会如图3-3所示。

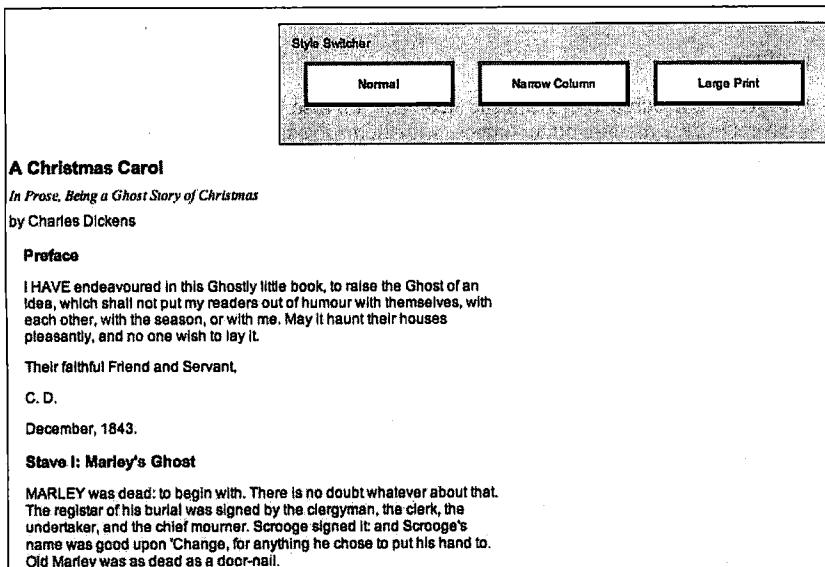


图 3-3

2. 事件处理程序的环境

虽然样式转换器的功能很正常，但我们并没有就哪个按钮处于当前在用状态对用户给出反馈。为此，我们的方法是在按钮被单击时，为它应用selected类，同时从其他按钮上移除这个类。selected类只是为按钮文本添加了粗体样式：

```
.selected {
    font-weight: bold;
}
```

为了实现类的变换，可以按照前面的做法，通过ID来引用每个按钮，然后再视情况为它们应用或移除类。不过，这一次我们要探索一种更优雅也更具有弹性的解决方案，这个方案利用了事件处理程序运行的环境。

当触发任何事件处理程序时，关键字this引用的都是携带相应行为的DOM元素。前面我们谈到过，`$()`函数可以将一个DOM元素作为参数，而this关键字正是这个功能有效的关键原因之一^①。通过在事件处理程序中使用`$(this)`，可以为相应的元素创建一个jQuery对象，然后就如同使用CSS选择符找到该元素一样对它进行操作。

知道了这些之后，我们可以编写出下面的代码：

```
$(this).addClass('selected');
```

把这行代码放到那3个事件处理程序中，就可以在按钮被单击时为按钮添加selected类。要从其他按钮中移除这个类，可以利用jQuery的隐式迭代特性，并编写如下代码：

```
$('#switcher .button').removeClass('selected');
```

这行代码会移除样式转换器中每个按钮的selected类。因此，按照正确的次序放置它们，就可以得到如下代码：

```
$(document).ready(function() {
    $('#switcher-normal').bind('click', function() {
        $('body').removeClass('narrow');
        $('body').removeClass('large');
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
    });
    $('#switcher-narrow').bind('click', function() {
        $('body').addClass('narrow');
        $('body').removeClass('large');
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
    });
});
```

^① 即允许`$()`函数传递DOM元素，也是为了更方便地将引用DOM元素的this转换为jQuery对象。——译者注

```
$('#switcher-large').bind('click', function() {
    $('body').removeClass('narrow');
    $('body').addClass('large');
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
});
});
```

这样，样式转换器就会如图3-4所示，对用户给出适当的反馈了。

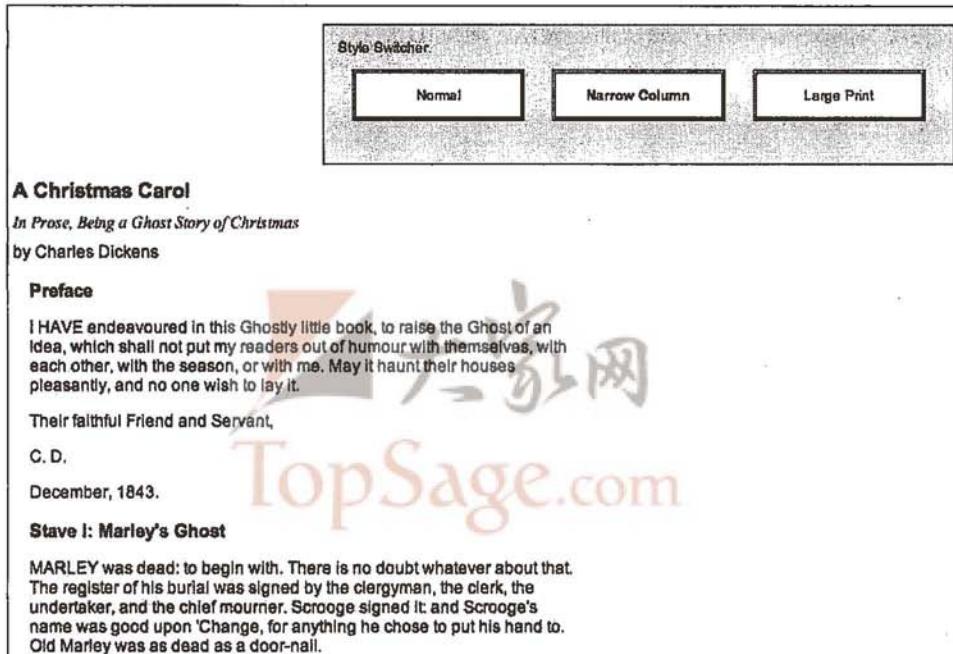


图 3-4

利用处理程序的环境将语句通用化，可以使代码更有效率。因为针对3个按钮的突出显示代码都一样，所以，我们可以把它提取到一个单独的处理程序中，结果如下列代码所示：

```
$(document).ready(function() {
    $('#switcher-normal').bind('click', function() {
        $('body').removeClass('narrow').removeClass('large');
    });
    $('#switcher-narrow').bind('click', function() {
        $('body').addClass('narrow').removeClass('large');
    });
    $('#switcher-large').bind('click', function() {
        $('body').removeClass('narrow').addClass('large');
    });
});
```

```

$( '#switcher .button' ).bind('click', function() {
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
});
}
);

```

这一步优化利用了我们讨论过的3种jQuery特性。第一，在通过对.bind()的一次调用为每个按钮都绑定相同的单击事件处理程序时，隐式迭代机制再次发挥了作用。第二，行为队列机制让我们在同一个单击事件上绑定了两个函数，而且第2个函数不会覆盖第1个函数。最后，我们使用jQuery的连缀能力将每次添加和移除类的操作压缩到了一行代码中。

3. 进一步合并

下面，再看一看绑定到每个按钮的行为。其中，.removeClass()方法的参数是可选的，即当省略参数时，该方法会移除元素中所有的类。利用这一点，可以把代码再改进得更简单一些：

```

$(document).ready(function() {
    $('#switcher-normal').bind('click', function() {
        $('body').removeClass();
    });
    $('#switcher-narrow').bind('click', function() {
        $('body').removeClass().addClass('narrow');
    });
    $('#switcher-large').bind('click', function() {
        $('body').removeClass().addClass('large');
    });

    $('#switcher .button').bind('click', function() {
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
    });
});
}
);

```

此时，在每个按钮的处理程序中仍然会执行某些相同的代码。这些代码也可以轻而易举地提取到通用的按钮单击处理程序中：

```

$(document).ready(function() {
    $('#switcher .button').bind('click', function() {
        $('body').removeClass();
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
    });

    $('#switcher-narrow').bind('click', function() {
        $('body').addClass('narrow');
    });
    $('#switcher-large').bind('click', function() {

```

```

        $('body').addClass('large');
    });
});

```

这里要注意的是，必须把通用的处理程序转移到特殊的处理程序上方，因为`.removeClass()`需要先于`.addClass()`执行。而之所以能够做到这一点，是因为jQuery总是按照我们注册的顺序来触发事件处理程序。

说明 我们在这里能够完全移除所有类，是因为现在的HTML是由我们控制的。当为了重用而编写代码时（例如编写插件），必须考虑到已经存在的所有类并保证它们原封不动。

最后，可以通过再次利用事件的执行环境来完全消除特殊的处理程序。因为环境关键字`this`引用的是DOM元素，而不是jQuery对象，所以可以使用原生的DOM属性来确定被单击元素的ID。因而，就可以对所有按钮都绑定相同的处理程序，然后在这个处理程序内部针对按钮执行不同的操作：

```

$(document).ready(function() {
    $('#switcher .button').bind('click', function() {
        $('body').removeClass();
        if (this.id == 'switcher-narrow') {
            $('body').addClass('narrow');
        }
        else if (this.id == 'switcher-large') {
            $('body').addClass('large');
        }
        $('#switcher .button').removeClass('selected');
        $(this).addClass('selected');
    });
});

```

3.2.2 简写的事件

鉴于为某个事件（例如简单的单击事件）绑定处理程序极为常用，jQuery提供了一种简化事件操作的方式——**简写事件方法**，简写事件方法的原理与对应的`.bind()`调用相同，但可以减少一定的代码输入量。

例如，不使用`.bind()`而使用`.click()`可以将前面的样式转换器程序重写为如下所示：

```

$(document).ready(function() {
    $('#switcher .button').click(function() {
        $('body').removeClass();
        if (this.id == 'switcher-narrow') {
            $('body').addClass('narrow');
        }
        else if (this.id == 'switcher-large') {

```

```

        $('body').addClass('large');
    }
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
});
);

```

3.3 复合事件

jQuery中的多数事件处理方法都会直接响应JavaScript的本地事件。但是，也有少数出于跨浏览器优化和方便性考虑而添加的自定义处理程序。其中，我们前面详细讨论过的`.ready()`方法就是其中之一。另外，`.toggle()`和`.hover()`方法则是另外两个自定义的事件处理程序。对于后两个方法，由于它们截取组合的用户操作，并且以多个函数作为响应，因此被称为复合事件处理程序。

3.3.1 显示和隐藏高级特性

假设我们想在不需要时隐藏样式转换器。隐藏高级特性^①的一种便捷方式，就是使它们可以折叠。因此，我们要实现的效果是在标签^②上单击能够隐藏所有按钮，最后只剩一个标签；而再次单击标签则会恢复这些按钮。为了隐藏按钮，我们还需要另外一个类：

```

.hidden {
    display: none;
}

```

为实现这个功能，可以把当前的按钮状态保存在一个变量中，每当标签被单击时，通过检查这个变量的值就能知道应该向这些按钮中添加，还是要从这些按钮中移除`.hidden`类。此外，也可以直接检查这个类在一个按钮上是否存在，然后再决定做什么。jQuery提供了能够替我们完成所有这些内部处理的`.toggle()`方法。实际上，jQuery定义了两个`.toggle()`方法，要了解另一个同名的效果（effect）方法，请参见<http://docs.jquery.com/Effects/toggle>。

我们这里所说的`.toggle()`方法接受两个参数，而且这两个参数都是函数。第1次在元素上单击会调用第1个函数，第2次单击则会触发第2个函数。此后，这两个函数会随着每一次单击交替执行。有了`.toggle()`方法，实现我们的可折叠的样式转换器就非常简单了：

```

$(document).ready(function() {
    $('#switcher h3').toggle(function() {
        $('#switcher .button').addClass('hidden');
    }, function() {
        $('#switcher .button').removeClass('hidden');
    });
});

```

^① 这里所谓的高级特性就是指为页面提供样式切换能力的样式转换器。——译者注

^② 这里将`#switcher h3`选择的h3标题元素称为标签。下同。——译者注

在第1次单击后，所有按钮都会隐藏起来，如图3-5所示。

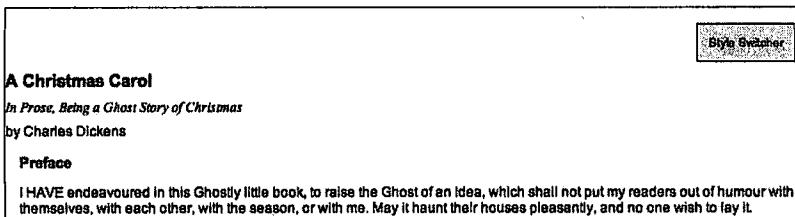


图 3-5

而第2次单击则又恢复了它们的可见性，如图3-6所示。

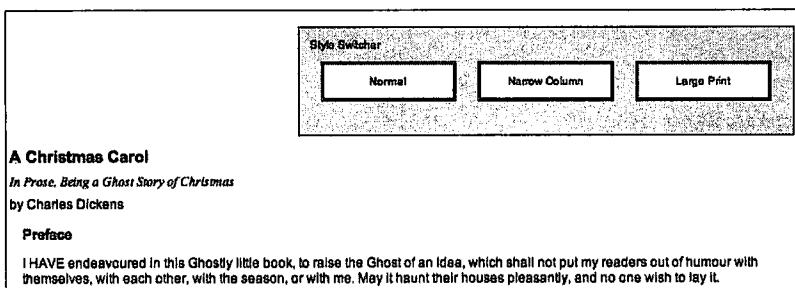


图 3-6

同样，这里我们依靠的仍然是jQuery的隐式迭代能力，即一次就能隐藏所有按钮，而不需要使用包装元素^①。

在这种特殊的情况下，jQuery也为我们要完成的折叠功能提供了另外一种机制。即，在为元素应用或从元素移除类之前，可以使用`.toggleClass()`方法自动检查该类是否存在：

```
$(document).ready(function() {
    $('#switcher h3').click(function() {
        $('#switcher .button').toggleClass('hidden');
    });
});
```

在这个例子中，或许`.toggleClass()`是一种更优雅的解决方案，但对于交替执行两种不同的操作而言，`.toggle()`则是适用性更强的方式。

3.3.2 突出显示可单击的项

在说明基于通常不可单击的页面元素^②处理单击事件的能力时，我们构思的界面中已经给出

① 即不需要在这3个按钮外部再添加额外的标签（如`<div>`）。如果没有隐式迭代机制，那么想一次隐藏3个按钮，一种常见的方法就是隐藏包含这3个按钮的包装元素。——译者注

② 因为在这个例子的标记中，按钮是通过`<div>`元素来表现的，而`<div>`元素就是通常不可单击的页面元素。

——译者注

了一些提示——按钮实际上都是活动的。为了改进界面，我们可以为按钮添加一种翻转状态，以便清楚地表明它们能与鼠标进行某种方式的交互：

```
#switcher .hover {
  cursor: pointer;
  background-color: #afa;
}
```

CSS规范加入了一个名叫`:hover`的伪类选择符，这个选择符可以让样式表在用户鼠标悬停在某个元素上时，影响元素的外观。在IE 6中，这种能力仅限制在链接元素上^①，因而我们不能在跨浏览器的程序中将这个选择符用于其他元素。但是，jQuery则可以让我们在鼠标指针进入元素和离开元素时执行任意操作。

同`.toggle()`方法一样，`.hover()`方法也接受两个函数参数。但在这里，第1个函数会在鼠标指针进入被选择的元素时执行，而第2个函数会在指针离开该元素时触发。我们可以在这些时候修改应用到按钮上的类，从而实现翻转效果：

```
$(document).ready(function() {
  $('#switcher .button').hover(function() {
    $(this).addClass('hover');
  }, function() {
    $(this).removeClass('hover');
  });
});
```

这里，我们再次使用隐式迭代和事件环境实现了简洁的代码。现在，当鼠标悬停在任何按钮上时，我们都能看到如图3-7中所示的应用了类之后的效果。

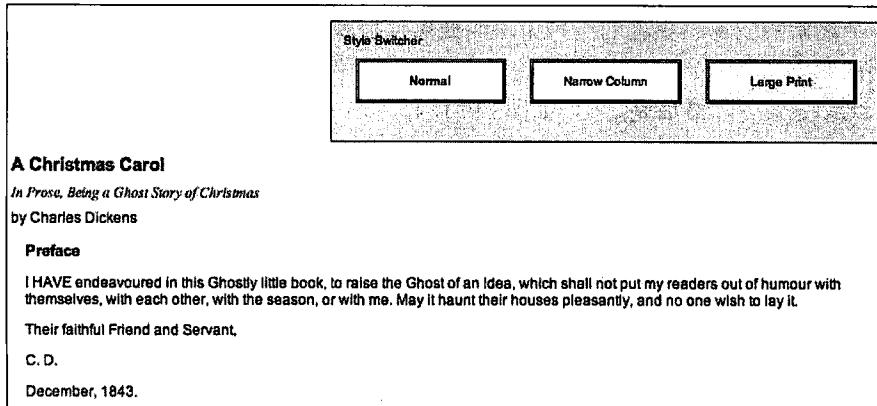


图 3-7

^① 即在IE 6中，不支持对链接之外的元素（比如`<div>`）使用`:hover`伪类选择符。这就限制了`:hover`的能力。

——译者注

而且，使用`.hover()`也意味着可以避免JavaScript中的事件传播（event propagation）导致的头痛问题。要理解事件传播的含义，首先必须搞清楚JavaScript如何决定由哪个元素来处理给定的事件。

3.3.3 事件的旅程

当页面上发生一个事件时，每个层次上的DOM元素都有机会处理这个事件。以下面的页面模型为例：

```
<div class="foo">
  <span class="bar"><a href="http://www.example.com/">The quick brown
    fox jumps over the lazy dog.</a></span>
  <p>How razorback-jumping frogs can level six piqued gymnasts!</p>
</div>
```

当在浏览器中形象化地呈现这些由嵌套的代码构成的元素时，我们看到的效果如图3-8所示。

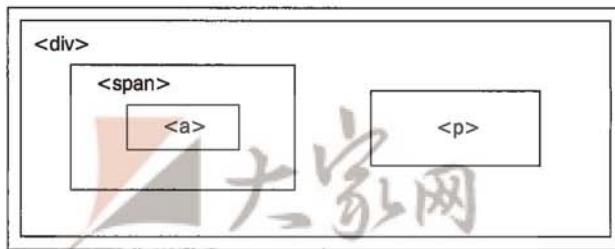


图 3-8

举例来说，如果单击了页面中的锚元素，那么`<div>`、``和`<a>`全都应该得到响应这次单击的机会。毕竟，这3个元素同时都处于用户鼠标的指针之下。

允许多个元素响应单击事件的一种策略叫做事件捕获^①。在事件捕获的过程中，事件首先会交给最外层的元素，接着再交给更具体的元素。在这个例子中，意味着单击事件首先会传递给`<div>`，然后是``，最后是`<a>`，如图3-9所示。

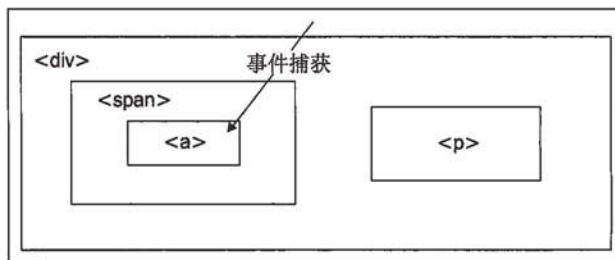


图 3-9

^① 事件捕获和下文中的事件冒泡是“浏览器大战”时期分别由Netscape和微软提出的两种相反的事件传播模型。

——译者注

提示 从技术上说，在浏览器对事件捕获的实现中，每个的元素都会注册并侦听发生在它们后代元素中的事件。这里提供的近似情况非常接近于我们的需求。

另一种相反的策略叫做事件冒泡。即当事件发生时，会首先发送给最具体的元素，在这个元素获得响应机会之后，事件会向上冒泡到更一般的元素。在我们的例子中，`<a>`会首先处理事件，然后按照顺序依次是``和`<div>`，如图3-10所示。

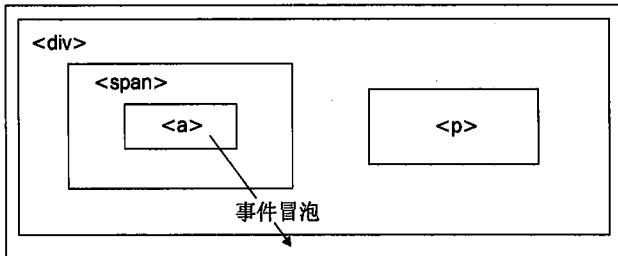


图 3-10

毫不奇怪，不同的浏览器开发者们最初采用的是不同的事件传播模型。因而，最终出台的DOM标准规定应该同时使用这两种策略：首先，事件要从一般到具体进行捕获，然后，事件再通过冒泡返回DOM树的顶层。而事件处理程序可以注册到这个过程中的任何一部分^①。

并不是所有浏览器都为了与新标准保持一致而进行了更新。而且，对于那些支持捕获的浏览器来说，通常必须明确启用才行^②。为了提供跨浏览器的一致性，jQuery始终会在模型的冒泡阶段注册事件处理程序^③。因此，我们总是可以假定最具体的元素会首先获得响应事件的机会。

3.3.4 事件冒泡的副作用

事件冒泡可能会导致始料不及的行为，特别是在错误的元素响应`mouseover`或`mouseout`事件的情况下。假设在我们的例子中，为`<div>`添加了一个`mouseout`事件处理程序。当用户的鼠标指针退出这个`<div>`时，会按照预期运行`mouseout`处理程序。因为这个过程发生顶层元素上，所以其他元素不会取得这个事件。但是，当指针从`<a>`元素上退出时，`<a>`元素也会取得一个`mouseout`事件。然后，这个事件会向上起泡到``和`<div>`，从而触发上述的事件处理程序。这种起泡序列很可能不是我们所希望的。因为，对于样式转换器例子中的按钮来说，这可能会意

① 即可以把事件处理程序注册到事件捕获阶段，也可以注册到事件冒泡阶段。——译者注

② 即在Firefox等支持标准事件模型（即同时支持捕获和冒泡）的浏览器中，要把事件处理程序注册到捕获阶段，必须在标准的事件注册方法`addEventListener()`中将最后一个参数设置为`true`。——译者注

③ 这是为了与IE保持一致——IE的事件注册方法`attachEvent()`只能将事件处理程序注册到冒泡阶段。为了在支持标准事件模型的浏览器中也把事件处理程序注册到冒泡阶段，就要将`addEventListener()`中的最后一个参数设置为`false`。另外，这样一来，就相当于以IE的事件模型代替了W3C的标准。——译者注

味着突出显示会过早地消失^①。

不过，`.hover()`方法能够聪明地处理这些冒泡问题，当我们使用该方法添加事件时，可以不必考虑由于错误的元素取得`mouseover`或`mouseout`事件而导致的问题。这就使得`.hover()`成为绑定个别鼠标事件的一种有吸引力的替代方案。

3.4 限制和终止事件

刚才介绍的`mouseout`事件的例子，说明有必要对事件的作用范围进行限制。当使用`.click()`^②处理这种特殊例子时，需要考虑其他一些从空间（阻止事件被发送到某些元素）或时间（阻止事件在某个时间发送）上限制事件的情况。

3.4.1 阻止事件冒泡

我们在前面已经举例说明事件冒泡可能会导致问题的一种情形。为了展示一种`.hover()`也帮不上我们的情况^③，需要改变前面实现的折叠行为。

假设我们希望增大触发样式转换器折叠或扩展的可单击区域。一种方案就是将事件处理程序从标签移至包含它的`<div>`元素：

```
$ (document).ready(function() {
  $('#switcher').click(function() {
    $('#switcher .button').toggleClass('hidden');
  });
});
```

这种改变会使样式转换器的整个区域都可以通过单击切换其可见性。而这样一来就造成了一个问题，即单击按钮会在修改内容区的样式之后折叠样式转换器。导致这个问题的原因就是事件冒泡，即事件首先通过按钮处理，然后又沿着DOM树向上传递，直至到达`<div id="switcher">`，最终隐藏按钮。

要解决这个问题，必须访问事件对象。事件对象是一种JavaScript结构，它会在元素获得处理事件的机会时被传递给相应的事件处理程序。这个对象中包含着与事件有关的信息（例如事件发生时的鼠标指针位置），也提供了可以用来影响事件在DOM中传递进程的一些方法。

为了在处理程序中使用事件对象，需要为函数添加一个参数：

① 这里所谓的突出显示过早地消失，是假设在表现按钮的`<div>`元素中也包含一个`<a>`元素。这样，当鼠标指针离开`<a>`但尚未离开`<div>`（按钮）时，由于事件冒泡，`<div>`会继`<a>`之后也取得`mouseout`事件，因而会导致按钮上的突出显示提前消失。——译者注

② 原文`.hover()`有误，应该是`.click()`。——译者注

③ 这种情况是指为元素的单击事件注册处理程序，而不是像前面那样为悬停事件注册处理程序。所以，这种情况下只能使用`.click()`方法，而不能使用`.hover()`方法。——译者注

```
$(document).ready(function() {
    $('#switcher').click(function(event) {
        $('#switcher .button').toggleClass('hidden');
    });
});
```

1. 事件目标

现在，事件处理程序中的变量`event`保存着事件对象。而`event.target`属性保存着发生事件的目标元素。这个属性是DOM API中规定的，但是没有被所有浏览器实现^①。jQuery对这个事件对象进行了必要的扩展，从而在任何浏览器中都能够使用这个属性。通过`.target`，可以确定DOM中首先接收到事件的元素（即实际被单击的元素）。而且，我们知道`this`引用的是处理事件的DOM元素，所以可以编写下列代码：

```
$(document).ready(function() {
    $('#switcher').click(function(event) {
        if (event.target == this) {
            $('#switcher .button').toggleClass('hidden');
        }
    });
});
```

此时的代码确保了被单击的元素是`<div id="switcher">`^②，而不是其他后代元素。现在，单击按钮不会再折叠样式转换器，而单击边框则会触发折叠操作。但是，单击标签同样什么也不会发生，因为它也是一个后代元素。实际上，我们可以不把检查代码放在这里，而是通过修改按钮的行为来达到目标^③。

2. 停止事件传播

事件对象还提供了一个`.stopPropagation()`方法，该方法可以完全阻止事件冒泡。与`.target`类似，这个方法也是一种纯JavaScript特性，但在跨浏览器的环境中则无法安全地使用^④。不过，只要我们通过jQuery来注册所有的事件处理程序，就可以放心地使用这个方法。

下面，我们会删除刚才添加的检查语句`event.target == this`，并在按钮的单击处理程序中添加一些代码：

```
$(document).ready(function() {
    $('#switcher .button').click(function(event) {
```

① 这里指IE没有按照标准把事件对象传递给事件处理程序。实际上，IE把事件对象保存在了全局属性`window.event`中，而下文中提到的事件目标，则可以通过`window.event.srcElement`属性访问到。但在jQuery中，则不必考虑这些浏览器间的不一致性。——译者注

② 即只有在`<div id="switcher">`被单击时才会执行样式转换器的折叠操作。——译者注

③ 即达到单击标签和div元素都可以折叠，但单击按钮不会折叠的目标。——译者注

④ 这里指在IE中要阻止事件冒泡，需要将事件对象的`cancelBubble`属性设置为`false`。——译者注

```

$('body').removeClass();
if (this.id == 'switcher-narrow') {
    $('body').addClass('narrow');
}
else if (this.id == 'switcher-large') {
    $('body').addClass('large');
}
$('#switcher .button').removeClass('selected');
$(this).addClass('selected');
event.stopPropagation();
});
});

```

同以前一样，需要为用作单击处理程序的函数添加一个参数，以便访问事件对象。然后，通过简单地调用`event.stopPropagation()`就可以避免其他所有DOM元素响应这个事件。这样一来，单击按钮的事件会被按钮处理，而且只会被按钮处理。单击样式转换器的其他地方则可以折叠和扩展整个区域。

3. 默认操作

如果我们把单击事件处理程序注册到一个锚元素，而不是一个外层的`<div>`上，那么就要面对另外一个问题：当用户单击链接时，浏览器会加载一个新页面。这种行为与我们讨论的事件处理程序不是同一个概念，它是单击锚元素的默认操作。类似地，当用户在编辑完表单后按下回车键时，会触发表单的`submit`事件，在此事件发生后，表单提交才会真正发生。

如果我们不希望执行这种默认操作，那么在事件对象上调用`.stopPropagation()`方法也无济于事，因为默认操作不是在正常的事件传播流中发生的。在这种情况下，`.preventDefault()`方法则可以在触发默认操作之前终止事件^①。

提示 当在事件的环境中完成了某些验证之后，通常会用到`.preventDefault()`。例如，在表单提交期间，我们会对用户是否填写了必填字段进行检查，如果用户没有填写相应字段，那么就需要阻止默认操作。我们将在第8章详细讨论表单验证。

事件传播和默认操作是相互独立的两套机制，在二者任何一方发生时，都可以终止另一方。如果想要同时停止事件传播和默认操作，可以在事件处理程序中返回`false`，这是对在事件对象上同时调用`.stopPropagation()`和`.preventDefault()`的一种简写方式。

3.4.2 移除事件处理程序

有时候，我们需要停用以前注册的一个事件处理程序。可能是因为页面的状态发生了变化，

^① 在IE中，要预防默认操作发生，需要将事件对象的`returnValue`属性设置为`false`。不过，在使用jQuery注册事件处理程序时不必考虑浏览器，只需使用文中提到的标准方法即可。——译者注

导致相应的操作不再有必要。处理这种情形的一种典型做法，就是在事件处理程序中使用条件语句。但是，如果能够完全移除处理程序显然更有效率。

假设我们希望样式转换器在页面没有使用正常样式的情况下保持扩展状态。那么，可以在样式转换器中的按钮被单击时，调用`.unbind()`方法来移除相应的事件处理程序。为此，首先要给作为处理程序的函数起个名字，以便可以多次调用这个处理程序：

```
$(document).ready(function() {
    var toggleStyleSwitcher = function() {
        $('#switcher .button').toggleClass('hidden');
    };

    $('#switcher').click(toggleStyleSwitcher);
});
```

我们注意到，这里使用了另一种定义函数的语法，即没有使用前置的`function`关键字，而是将一个匿名创建的函数指定给了一个局部变量。选择使用这种语法形式，可以使事件处理程序与其他函数定义在格式上更加接近；无论使用哪种语法，它们的功能都是等价的。

在函数有了名字之后，我们就可以在必要时移除它：

```
$(document).ready(function() {
    var toggleStyleSwitcher = function() {
        $('#switcher .button').toggleClass('hidden');
    };

    $('#switcher').click(toggleStyleSwitcher);

    $('#switcher-narrow, #switcher-large').click(function() {
        $('#switcher').unbind('click', toggleStyleSwitcher);
    });
});
```

`.unbind()`方法将一种事件类型作为其第1个参数，将要移除的函数作为第2个参数。这里，如果省略函数也可以得到相同的结果，因为`.unbind()`的默认行为就是移除为指定事件注册的所有处理程序。不过，更具体也意味着更安全，这样我们就不必担心是否有其他代码也为这个元素的同一个事件绑定了行为。

这些代码可以保证在单击任何按钮之后，样式转换器不会再发生折叠。然而，其中还缺少当样式返回正常状态时恢复这一行为的代码。为此，还需要为Normal按钮添加另一个行为：

```
$(document).ready(function() {
    var toggleStyleSwitcher = function() {
        $('#switcher .button').toggleClass('hidden');
    };
});
```

```

$( '#switcher' ).click( toggleStyleSwitcher );

$( '#switcher-normal' ).click( function() {
    $( '#switcher' ).click( toggleStyleSwitcher );
});

$( '#switcher-narrow, #switcher-large' ).click( function() {
    $( '#switcher' ).unbind('click', toggleStyleSwitcher);
});
}
);

```

这样，切换样式转换器的行为当文档加载后会被绑定；当单击Narrow Column或Large Print按钮时会解除绑定；而当此后再单击Normal按钮时，又会恢复绑定。

但是，我们在这里回避了一个潜在的问题。我们知道，当通过jQuery为一个事件绑定处理程序时，先前绑定的处理程序仍然有效。这就意味着，如果连续单击了两次Normal按钮，则可能会触发两次切换行为。没错，如果我们在这个例子中全都使用匿名函数，那么这个问题确实存在。但是，因为我们为函数起了名字，而且在代码中使用的都是同一个函数，所以相应的行为只会被绑定一次。也就是说，如果已经存在了一个事件处理程序，`.bind()`方法不会为元素再重复添加同一个事件处理程序。

在jQuery 1.0中，反绑定(`unbinding`)事件处理程序也可以使用简写的事件方法，就和对应的绑定操作一样。例如，`.unclick()`就是`.unbind('click')`的简写方法。但由于这种措施很少有人使用，为了减少多余的库代码，也为了降低API的复杂程度，1.1版中删除了这些简写的事件方法。

说明 重新引入被删除的简写事件方法也很简单。我们将在第10章将这个主题作为扩展jQuery功能的一个例子进行探讨。

对于只需触发一次，随后要立即解除绑定的情况也有一种简写方法为`.one()`，这个简写方法的用法如下：

```

$(document).ready(function() {
    $( '#switcher' ).one('click', toggleStyleSwitcher);
});

```

这样会使切换操作只发生一次，之后就再也不会发生。

3.5 模仿用户操作

有时候，即使某个事件没有真正发生，但如果能执行绑定到该事件的代码将会很方便。例如，假设我们想让样式转换器在一开始时处于折叠状态。那么，可以通过样式来隐藏按钮，或者在`$(document).ready()`处理程序中调用`.hide()`方法。不过，还有一种方法，就是模拟单击样式转换器，以触发我们设定的折叠机制。

通过`.trigger()`方法就可以完成模拟事件的操作：

```
$(document).ready(function() {
    $('#switcher').trigger('click');
});
```

这样，随着页面加载完成，样式转换器也会被折叠起来，就好像是被单击了一样。结果如图3-11所示。

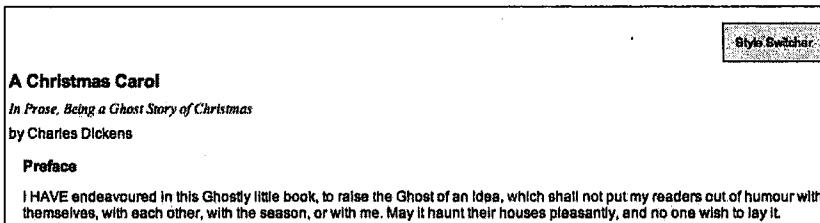


图 3-11

这里要注意的是，当我们通过jQuery以这种方式触发事件时，不会发生事件传播；只会执行直接添加到元素的处理程序。而且，为保证代码正常运行，必须在`$('#switcher')`上面，而不是`('#switcher h3')`上面执行触发器`(.trigger())`，因为前者是携带该行为的元素。

`.trigger()`方法提供了一组与`.bind()`方法相同的简写方法。当使用这些方法而不带参数时，结果将是触发操作而不是绑定行为：

```
$(document).ready(function() {
    $('#switcher').click();
});
```

3.6 小结

在本章中，我们讨论的能力主要实现以下功能。

- 通过`.bind()`或`.click()`响应用户在页面元素上的单击操作，改变应用于页面的样式。
- 即使将处理程序绑定到几个元素，都可以通过事件环境基于被单击的页面元素执行不同的操作。
- 通过使用`.toggle()`交替地扩展和折叠页面元素。
- 通过使用`.hover()`突出显示位于鼠标指针下方的页面元素。
- 通过`.stopPropagation()`和`.preventDefault()`来影响哪个元素能够响应事件。
- 调用`.unbind()`移除停用的事件处理程序。
- 通过`.trigger()`引发执行绑定的事件处理程序。

总之，我们可以使用这些能力构建交互性很好的页面。在下一章中，我们会学习如何在交互期间为用户提供视觉上的反馈。

效果——为操作添加艺术性

*Move it up and down now
Move it all around now*

*—Devo,
“Gut Feeling”*

如果说行（action）胜于言，那么在JavaScript的世界里，效果则会让操作（action）更胜一筹。通过jQuery，我们不仅能够轻松地为页面操作添加简单的视觉效果，甚至能创建更精致的动画。

jQuery效果确实能增添艺术性，一个元素逐渐滑入视野而不是突然出现时，带给人的美感是不言而喻的。此外，当页面发生变化时，通过效果吸引用户的注意力，则会显著增强页面的可用性（在AJAX应用程序中尤其常见）。在本章中，我们就来探索一些这样的效果，并将这些效果以有趣的方式组合起来。

4.1 修改内联CSS

在接触漂亮的jQuery效果之前，有必要先简单地谈一谈CSS。在前几章中，为了修改文档的外观，我们都是先在单独的样式表中为类定义好样式，然后再通过jQuery来添加或者移除这些类。一般而言，这都是为HTML应用CSS的首选方式，因为这种方式不会影响样式表负责处理页面表现的角色。但是，在有些情况下，可能我们要使用的样式没有在样式表中定义，或者通过样式表定义不是那么容易。针对这种情况，jQuery提供了`.css()`方法。

这个方法集获取方法（getter）和设置方法（setter）于一体。为取得某个样式属性的值，可以为这个方法传递一个字符串形式的属性名，比如`.css('backgroundColor')`。对于由多个单词构成的属性名，jQuery既可以解释连字符版的CSS表示法（如`background-color`），也可以解释驼峰大小写形式（camel-cased）的DOM表示法（如`backgroundColor`）。在设置样式属性时，`.css()`方法能够接受的参数有两种，一种是为它传递一个单独的样式属性和值，另一种是为它传递一个由属性-值对构成的映射^①（map）：

^① 此处的“映射”为JavaScript中的对象字面量。下同。——译者注

- .css('property', 'value')
- .css({property1: 'value1', 'property-2': 'value2'})

有经验的JavaScript开发者会将这些jQuery映射看成是JavaScript对象字面量。

说明 一般来说，数字值不需要加引号而字符串值需要加引号。但是，当使用映射表示法时，如果属性名使用驼峰大小写形式的DOM表示法，则可以省略引号。

使用.css()的方式与我们前面使用.addClass()的方式相同——将它连缀到选择符后面并绑定到某个事件。为此，我们仍以样式转换器为例，但这次使用的HTML稍有不同：

```
<div id="switcher">
  <div class="label">Style Switcher</div>
  <div class="button" id="switcher-large">Increase Text Size</div>
  <div class="button" id="switcher-small">Decrease Text size</div>
</div>
<div class="speech">
  <p>Four score and seven years ago our fathers brought forth on
    this continent a new nation, conceived in liberty, and dedicated
    to the proposition that all men are created equal.</p>
</div>
```

在通过链接的样式表为这个文档添加了一些基本样式规则之后，初始的页面如图4-1所示。



图 4-1

在这个版本的样式转换器中，包含两个以div元素表现的按钮。单击switcher-large div会增大speech div中文本的字体大小，而单击switcher-small div则会减小字体大小。

如果每次都增大或减小为预定的值，那么仍然可以使用.addClass()方法。但是，这次假设我们希望每单击一次按钮，文本的字体大小就会持续地递增或者递减。虽然为每次单击定义一个单独的类，然后迭代这些类也是可能的，但更简单明了的方法是每次都以当前字体大小乘以一个设定的值，得到新的字体大小。

同以前一样，我们代码仍然是从\$(document).ready()和\$('#switcher-large').click()事件处理程序开始：

```
$(document).ready(function() {
  $('#switcher-large').click(function() {
  });
});
```

接着，通过`$('.div.speech').css('fontSize')`可以轻而易举地取得当前的字体大小。不过，由于返回的值中既包含数字值也包含度量单位，所以需要把这两部分保存到各自的变量中，在乘出新的字体大小后，再重新加上单位。同样，在需要多次使用某个jQuery对象时，最好也把这个对象保存到一个变量中。

```
$(document).ready(function() {
  $('#switcher-large').click(function() {
    var $speech = $('.div.speech');
    var currentSize = $speech.css('fontSize');
    var num = parseFloat(currentSize, 10);
    var unit = currentSize.slice(-2);
  });
});
```

`.click()`中的第一行用于把`speech` div保存到一个变量中。

我们注意到，变量名`$speech`中有一个`$`。因为在JavaScript中是一个合法的字符，所以通过它我们可以提醒自己变量中保存的是一个jQuery对象。下一行用于保存`speech` div中的字体大小，比如`12px`。

接下来的两行中使用了`parseFloat()`和`.slice()`。其中，`parseFloat()`函数会在一个字符串中从左到右地查找一个浮点（十进制）数。例如，它会将字符串`12`转换成数字`12`。另外，它还会去掉末尾的非数字字符，因此`12px`就变成了`12`。如果字符串本身以一个非数字开头，那么`parseFloat()`会返回`Nan`，即`Not a Number`（非数字）。函数的第2个参数用以确保将这个数译码为十进制数，而不是八进制或其他表示法。

`.slice()`方法返回字符串中从指定的字符开始的一个子字符串。因为这里使用的度量单位`(px)`是两个字符，所以我们指定子字符串应该从倒数第2个字符开始。

至此，所剩的就是将`num`乘以`1.4`，然后再连接两个已分解的变量`num`和`unit`来设置字体大小了：

```
$(document).ready(function() {
  $('#switcher-large').click(function() {
    var $speech = $('.div.speech');
    var currentSize = $speech.css('fontSize');
    var num = parseFloat(currentSize, 10);
    var unit = currentSize.slice(-2);
    num *= 1.4;
    $speech.css('fontSize', num + unit);
  });
});
```

说明 其中，等式`num *= 1.4`是`num = num * 1.4`的简写形式。同样，对于其他基本的数学运算，也可使用这种简写形式，比如：加，`num += 1.4`；减，`num -= 1.4`；除，`num /= 1.4`和取模（除法的余数），`num %= 1.4`。

现在，当用户单击switcher-large div时，speech div中的文本就会像图4-2中所示的那样变大。

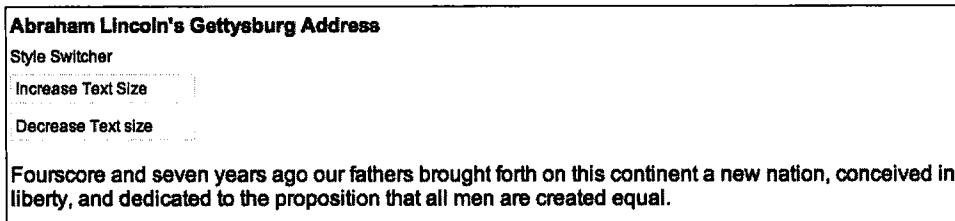


图 4-2

再单击一次，文本会变得更大。

要通过单击switcher-small div减小字体大小，需要使用除法而不是乘法，即`num /= 1.4`。同样，更好的方案是把对这两个div的单击操作，通过button类组合到一个`.click()`处理程序中。在设置完变量后，再根据用户单击的div的ID来决定使用乘法还是除法。下面就是相应的代码：

```
$(document).ready(function() {
    $('div.button').click(function() {
        var $speech = $('div.speech');
        var currentSize = $speech.css('fontSize');
        var num = parseFloat(currentSize, 10);
        var unit = currentSize.slice(-2);
        if (this.id == 'switcher-large') {
            num *= 1.4;
        } else if (this.id == 'switcher-small') {
            num /= 1.4;
        }
        $speech.css('fontSize', num + unit);
    });
});
```

根据第3章学习的内容，我们知道可以访问由`this`引用的DOM元素的`id`属性，因而就有了`if`和`else if`语句中的代码。这里，如果仅测试属性的值，使用`this`显然要比创建jQuery对象更有效。

4.2 基本的隐藏和显示

基本的`.hide()`和`.show()`方法不带任何参数。可以把它们想象成类似`.css('display', 'string')`方法的简写方式，其中`string`是适当的显示值。不错，这两个方法的作用就是立即隐藏或显示匹配的元素集合，不带任何动画效果。

其中，`.hide()`方法会将匹配的元素集合的内联`style`属性设置为`display:none`。但它的聪明之处是，它能够在把`display`的值变成`none`之前，记住原先的`display`值，通常是`block`或

inline。恰好相反，`.show()`方法会将匹配的元素集合的`display`属性，恢复为应用`display:none`之前的可见属性。

`.show()`和`.hide()`的这种特性，使得它们非常适合隐藏那些默认的`display`属性在样式表中被修改的元素。例如，在默认情况下，``元素具有`display:block`属性，但是，为了构建水平的导航菜单，它们可能会被修改成`display:inline`。而在类似这样的``元素上面使用`.show()`方法，不会简单地把它重置为默认的`display:block`，因为那样会导致把``元素放到单独的一行中；相反，`.show()`方法会把它恢复为先前的`display:inline`状态，从而维持水平的菜单设计。

要示范这两个方法，最明显的一个例子就是在前面的HTML中，为段落末尾加上一个省略号，然后再加上一个段落：

```

<div id="switcher">
  <div class="label">Style Switcher</div>
  <div class="button" id="switcher-large">
    Increase Text Size</div>
  <div class="button" id="switcher-small">
    Decrease Text size</div>      </div>
<div class="speech">
  <p>Four score and seven years ago our fathers brought forth on
    this continent a new nation, conceived in liberty, and dedicated
    to the proposition that all men are created equal.
  <span class="more">...</span></p>
  <p>Now we are engaged in a great civil war, testing whether that
    nation, or any nation so conceived and so dedicated, can long
    endure. We are met on a great battlefield of that war. We have come
    to dedicate a portion of that field as a final resting-place for
    those who here gave their lives that the nation might live. It is
    altogether fitting and proper that we should do this. But, in a
    larger sense, we cannot dedicate, we cannot consecrate, we cannot
    hallow, this ground.</p>
</div>

```

当DOM就绪时，隐藏第2个段落：

```

$(document).ready(function() {
  $('p:eq(1)').hide();
});

```

现在的讲话（speech）文本看起来如图4-3所示。

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal...

图 4-3

然后，当用户单击第1段末尾的省略号（...）时，就会隐藏省略号同时显示第2个段落：

```

$(document).ready(function() {
  $('p:eq(1)').hide();
  $('span.more').click(function() {
    $('p:eq(1)').show();
    $(this).hide();
  });
});

```

此时的讲话文本如图4-4所示。

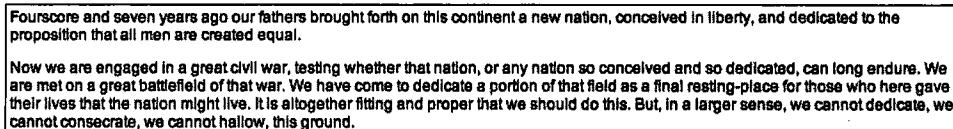


图 4-4

虽然`.hide()`和`.show()`方法简单实用，但它们没有那么花哨。为了增添更多的艺术感，我们可以为它们指定速度。

4.3 效果和速度

当在`.show()`或`.hide()`中指定一个速度参数时，就会产生动画效果，即效果会在一个特定的时间段内发生。例如`.hide('speed')`方法，会同时减少元素的高度、宽度和不透明度，直至这3个属性的值都达到0，与此同时会为该元素应用CSS规则`display:none`。而`.show('speed')`方法则会从上到下增大元素的高度，从左到右增大元素的宽度，同时从0到1增加元素的不透明度，直至其内容完全可见。

1. 指定显示速度

对于jQuery提供的任何效果，都可以指定3种速度参数：`slow`、`normal`和`fast`。使用`.show('slow')`会在0.6秒内完成效果，`.show('normal')`是0.4秒，而`.show('fast')`则是0.2秒。要指定更精确的速度，可以使用毫秒数值，例如`.show(850)`。注意，与字符串表示的速度参数名称不同，数值不需要使用引号。

下面，我们就为《林肯盖茨堡演说辞》(Lincoln's Gettysburg Address) 的第2段指定显示速度：

```

$(document).ready(function() {
  $('p:eq(1)').hide();
  $('span.more').click(function() {
    $('p:eq(1)').show('slow');
    $(this).hide();
  });
});

```

如果能够在效果完成大约一半时捕获段落的外观，那么应该看到类似图4-5所示的结果。

Fourscore and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.
Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield

图 4-5

2. 淡入和淡出

如果想在显示整个段落时，只是逐渐地增大其不透明度，那么可以使用`.fadeIn('slow')`方法：

```
$ (document).ready(function() {
  $('p:eq(1)').hide();
  $('span.more').click(function() {
    $('p:eq(1)').fadeIn('slow');
    $(this).hide();
  });
});
```

这一次如果捕获到段落显示到一半时的外观，则会变成如图4-6所示。

Fourscore and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.
Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field as a final resting-place for those who here gave their lives that the nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate, we cannot consecrate, we cannot hallow, this ground.

图 4-6

最近两次效果的差别在于，`.fadeIn()`会在一开始设置段落的尺寸，以便内容能够简单地逐渐显示出来。类似地，要逐渐减少不透明度，可以使用`.fadeOut()`。

4.4 多重效果

在绑定到jQuery核心的效果中，只有`show()`和`hide()`会同时修改多个样式属性——高度、宽度和不透明度。其他效果则只修改一种属性。

- `fadeIn()` 和 `fadeOut()`：不透明度
- `fadeTo()`：不透明度
- `slideDown()` 和 `slideUp()`：高度

不过，jQuery也提供了一个强大的`animate()`方法，通过该方法可以创建包含多重效果的自定义动画。`animate()`方法接受以下4个参数。

- (1) 一个包含样式属性及值的映射——与本章前面讨论的`.css()`方法中的映射类似。
- (2) 可选的速度参数——既可以是前面提到的预置的字符串，也可是一个毫秒数值。
- (3) 可选的缓动（easing）类型——将在第10章中讨论的一个高级选项。
- (4) 可选的回调函数——将在本章后面讨论。

把其中3个参数放到一起，结果如下所示：

```
.animate({param1: 'value1', param2: 'value2'}, speed, function() {
    alert('The animation is finished.');
});
```

4.4.1 构建具有动画效果的 show()

我们再看一看让《林肯盖茨堡演说辞》第2段逐渐显示的代码：

```
$(document).ready(function() {
    $('p:eq(1)').hide();
    $('span.more').click(function() {
        $('p:eq(1)').show('slow');
        $(this).hide();
    });
});
```

还记得吧，.show('slow')会同时修改宽度、高度和不透明度属性。因此，事实上它只是.animate()方法的一种内置了特定样式属性的简写形式。如果想通过.animate()得到同样的效果，那么相应的代码如下所示：

```
$(document).ready(function() {
    $('p:eq(1)').hide();
    $('span.more').click(function() {
        $('p:eq(1)').animate({height: 'show', width: 'show',
            opacity: 'show'}, 'slow');
        $(this).hide();
    });
});
```

显然，.animate()拥有它自己的一些简写的参数值！这里，我们使用简写的show将高度、宽度等恢复到了它们被隐藏之前的值。在此，也可以使用hide、toggle或其他任意数字值。

4.4.2 创建一种自定义的动画效果

通过.animate()方法，不仅能够控制其他效果方法中使用的样式属性，而且，也可以控制其他属性，比如left和top。通过使用额外的属性，能够创建更加精致新颖的效果。比如说，可以在一个元素的高度增加到50像素的同时，将它从页面的左侧移动到页面右侧。

下面，我们就为样式转换按钮实现这一效果。图4-7是移动它们之前的屏幕截图。

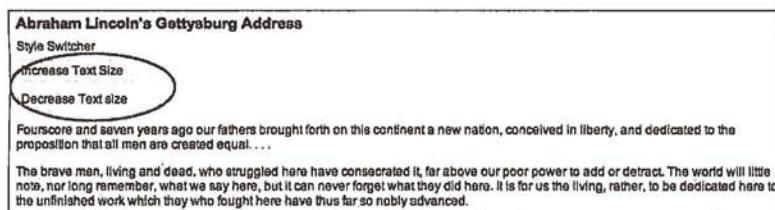


图 4-7

我们要把这两个按钮向右移动，同时增加它们的高度，而触发这个动画效果的操作是单击链接上方的**Style Switcher**文本。于是，相应的代码应该如下所示：

```
$(document).ready(function() {
    $('div.label').click(function() {
        $('div.button').animate({left: 650, height: 38}, 'slow');
    });
});
```

这些代码可以增加按钮的高度，但此时还不能改变它们的位置。所以，下面还需要修改CSS，以便能够改变它们的位置。

通过CSS设置定位

在使用`.animate()`方法时，必须明确CSS对我们要改变的元素所施加的限制。例如，在元素的CSS定位没有设置成`relative`或`absolute`的情况下，调整`left`属性对于匹配的元素毫无作用。所有块级元素默认的CSS定位属性都是`static`，这个值精确地表明：在改变元素的定位属性之前试图移动它们，它们只会保持静止不动。

说明 要了解与绝对（`absolute`）和相对（`relative`）定位有关的更多信息，请参考Joe Gillespie的文章“*Absolutely Relative*”（http://www.wpdfd.com/issues/78/absolutely_relative/）。

打开样式表，我们注意到其中已经为`<div id="switcher">`容器和个别的按钮设置了相对的定位：

```
#switcher {
    position: relative;
}
.button {
    position: relative;
    width: 140px;
    padding: 5px;
    border: 1px solid #e3e3e3;
    margin: .5em 0;
}
```

在有了CSS的定位支持之后，单击**Style Switcher**，最终完成的效果将如图4-8所示。

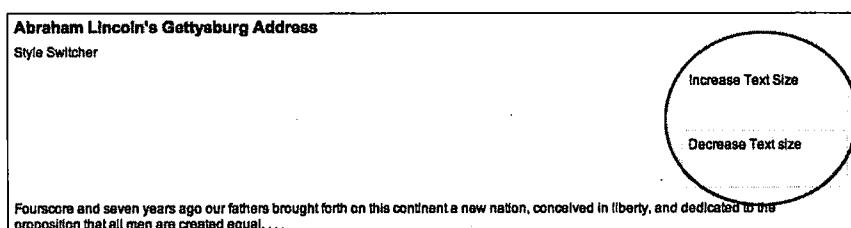


图 4-8

4.4.3 理解数字的含义

通过仔细观察代码，可以发现两个值，`left`属性的650和`height`属性的38：

```
$('.div.button').animate({left: 650, height: 38}, 'slow');
```

为什么用两个数？为什么不给左侧位置指定750或800？而且，更奇怪的是，为什么不给高度指定50？

说起左侧位置，我们只能承认——这个值完全是估计的！也就是说，我们只能猜测页面有多宽。尽管我们也知道，对于那些使用宽屏和具有高分辨率显示器的人，结果可能是按钮会停留在页面中间的某个地方。

另一方面，38像素的高度则是我们有目的地指定的。因为这两个按钮，除设置了`position: relative`之外，还应用了`padding: 5px`和`border: 1px solid #e3e3e3`。但由于`height`属性的值中不包含内边距和边框，所以为了使按钮达到50像素的总高度，需要从中减去上、下内边距和上、下边框的宽度值。虽然我们在样式表中使用了简写的`padding`和`border`属性，但它与我们分别为4条边设置5像素的内边距和1像素的边框宽度，效果是一样的，比如：

```
.button {
    position: relative;
    width: 140px;
    /* 设置每一边的内边距…… */
    padding-top: 5px;
    padding-right: 5px;
    padding-bottom: 5px;
    padding-left: 5px;

    /* 设置每一边的边框宽度…… */
    border-top-width: 1px;
    border-right-width: 1px;
    border-bottom-width: 1px;
    border-left-width: 1px;
    border-style: solid;
    border-color: #e3e3e3;
    margin: .5em 0;
}
```

也就是说，我们要计算下面这个算式：

$$\text{height} - (\text{padding-top} + \text{padding-bottom}) - (\text{border-top-width} + \text{border-bottom-width})$$

代入相应的值，就会得到：

$$50 - (5 + 5) - (1 + 1)$$

而结果呢，当然是38！

以上计算的依据是W3C在CSS规范中提出的盒模型。有关该模型的完整规范可以在<http://www.w3.org/TR/REC-CSS2/box.html>中找到。

4.4.4 改进自定义动画效果

现在，回到我们在自定义动画效果中遇到的左侧位置的问题上。如果能够把这两个按钮移动到将它们的右边与下方段落的右边对齐（从视觉上看只是近似，因为此时的段落不是右对齐），那就非常理想了。下面就是解决这个问题的步骤。

- (1) 取得段落的宽度。
- (2) 取得按钮的宽度，包括其左、右内边距和边框的宽度。
- (3) 从段落的宽度中减去按钮的宽度。
- (4) 使用计算的结果（以变量的形式）作为.animate()方法的left值。

为了计算按钮的总宽度，同时保证代码的可读性，需要设置很多变量。以下就是新的改进之后的代码：

```
$ (document).ready(function() {
    $('div.label').click(function() {
        //取得所有的宽度……
        var paraWidth = $('div.speech p').width();
        var $button = $('div.button');
        var buttonWidth = $button.width();
        var paddingRight = $button.css('paddingRight');
        var paddingLeft = $button.css('paddingLeft');
        var borderRightWidth = $button.css('borderRightWidth');
        var borderLeftWidth = $button.css('borderLeftWidth');

        //计算总宽度……
        var totalButtonWidth = parseInt(
            buttonWidth, 10) + parseInt(paddingRight, 10) + parseInt(
            paddingLeft, 10) + parseInt(borderRightWidth, 10) +
            parseInt(borderLeftWidth, 10);
        var rightSide = paraWidth - totalButtonWidth;
        $button.animate({left: rightSide, height: 38}, 'slow');
    });
});
```

现在，按钮与段落沿着右边准确地对齐了，如图4-9所示。

在前面的代码中，为了将变量的数值加到一起，频繁地使用了JavaScript的parseInt()函数，这个函数与parseFloat()的作用类似，但是，它返回的是一个整数，而不是一个浮点数。由于通过.css()方法返回的每个值后面都带有px单位，例如paddingRight的值是5px，因此要想完成加法或减法运算，就必须从这些变量中删除px，只留下实际的数字。但是，要注意一点，这种情况下只有使用像素值是最安全的，因为IE可能会错误地解释其他度量单位。

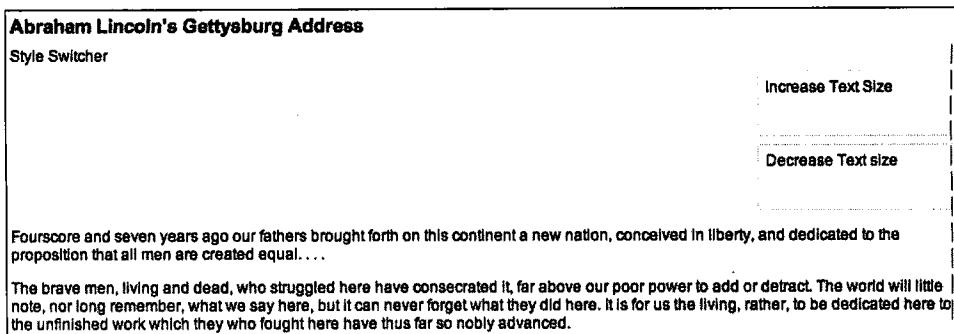


图 4-9

不过，jQuery也提供了与`.css('width')`返回相同数值的简写方法`width()`，这个方法返回的值不带度量单位。

4.5 并发与排队效果

通过刚才的例子，可以看出`.animate()`方法在为一组特定的元素创建并发效果时非常有用。然而，有的时候我们需要的是排队效果，即让效果一个接一个地发生。

4.5.1 处理一组元素

当为同一组元素应用多重效果时，可以通过连缀这些效果轻易地实现排队。为了示范排队效果，我们来看一看向右移动并扩大转换器的那个简单一点的例子：

```
$(document).ready(function() {
    $('#div.label').click(function() {
        $('#div.button').animate({left: 650, height: 38}, 'slow');
    });
});
```

我们知道，目前这两个动画效果（`left:650`和`height:38`）事实上是同时发生的。要排队这些效果，我们可以简单地将它们拆开并连缀起来：

```
$(document).ready(function() {
    $('#div.label').click(function() {
        $('#div.button')
            .animate({left: 650}, 'slow')
            .animate({height: 38}, 'slow');
    });
});
```

现在，按钮首先会向右移动650像素，然后高度再增长到50像素（38加上下外边距及边框）。如果我们想先增加高度，只需将代码中的这两个动画效果颠倒位置即可。

虽然连缀允许我们把这两个`.animate()`方法放在同一行，但为了更好的可读性，这里故意

将它们分开放在了各自的一行中。

通过使用连缀，可以对其他任何jQuery效果进行排队，而并不限于.animate()方法。比如说，我们可以按照下列顺序对按钮上的效果进行排队：

- (1) 将它们的不透明度减退为0.5，使它们呈半透明状。
- (2) 将它们向右移动650像素。
- (3) 将它们渐增回完全不透明。
- (4) 通过将它们滑到上方最终隐藏它们。

我们所要做的，就是在代码中按照相同的顺序连缀这些效果：

```
$(document).ready(function() {
    $('div.label').click(function() {
        $('div.button')
            .fadeTo('slow', 0.5)
            .animate({left: 650}, 'slow')
            .fadeTo('slow', 1.0)
            .slideUp('slow');
    });
});
```

当对一组元素的效果进行排队时，需要注意的最后一个关键问题是，排队不适用于其他的非效果方法，例如.css()。为示范这个问题，我们假设需要在动画的末尾改变按钮的背景颜色，而不是将它们向上滑出。那么，我们可以试一试下面的代码：

```
$(document).ready(function() {
    $('div.label').click(function() {
        $('div.button')
            .fadeTo('slow', 0.5)
            .animate({left: 650}, 'slow')
            .fadeTo('slow', 1.0)
            .css('backgroundColor', '#f00');
    });
});
```

然而，即使将改变背景的代码放在了方法链的末尾，它也会随着单击事件的发生而立即执行。要是把.css()从连缀的方法中拿出来，再重复使用一次选择符表达式会怎么样呢？

```
$(document).ready(function() {
    $('div.label').click(function() {
        $('div.button')
            .fadeTo('slow', 0.5)
            .animate({left: 650}, 'slow')
            .fadeTo('slow', 1.0);
        $('div.button').css('backgroundColor', '#f00');
    });
});
```

一切照旧——当单击转换器标签时，按钮的背景颜色仍然会立即改变。

那么，应该怎样排队非效果方法呢？在下面讨论多组元素的效果时，我们再来回答这个问题。

4.5.2 处理多组元素

与一组元素的情况不同，当为不同组的元素应用效果时，这些效果几乎会同时发生。为了示范这种并发的效果，我们可以在向上滑出一个段落时，向下滑入另一个段落。首先，把《林肯葛底斯堡演说辞》剩余的部分添加到HTML中，并把它们分成两段：

```
<div id="switcher">
  <div class="label">Style Switcher</div>
  <div class="button" id="switcher-large">Increase Text Size</div>
  <div class="button" id="switcher-small">Decrease Text size</div>
</div>
<div class="speech">
  <p>Fourscore and seven years ago our fathers brought forth on this
  continent a new nation, conceived in liberty, and dedicated to the
  proposition that all men are created equal.<span class="more"> . . .
  .</span></p>
  <p>Now we are engaged in a great civil war, testing whether that
  nation, or any nation so conceived and so dedicated, can long
  endure. We are met on a great battlefield of that war. We have come
  to dedicate a portion of that field as a final resting-place for
  those who here gave their lives that the nation might live. It is
  altogether fitting and proper that we should do this. But, in a
  larger sense, we cannot dedicate, we cannot consecrate, we cannot
  hallow, this ground.</p>
  <p>The brave men, living and dead, who struggled here have
  consecrated it, far above our poor power to add or detract. The
  world will little note, nor long remember, what we say here, but it
  can never forget what they did here. It is for us the living,
  rather, to be dedicated here to the unfinished work which they who
  fought here have thus far so nobly advanced.</p>
  <p>It is rather for us to be here dedicated to the great task
  remaining before us—that from these honored dead we take
  increased devotion to that cause for which they gave the last full
  measure of devotion—that we here highly resolve that these
  dead shall not have died in vain—that this nation, under God,
  shall have a new birth of freedom and that government of the
  people, by the people, for the people, shall not perish from the
  earth.</p>
</div>
```

接着，为了更清楚地看到效果发生期间的变化，我们为第3段和第4段分别添加浅蓝色和淡紫色的背景。同时，在DOM就绪时立即隐藏第4段：

```
$(document).ready(function() {
  $('p:eq(3)').css('backgroundColor', '#fcf').hide();
  $('p:eq(2)').css('backgroundColor', '#cff');
});
```

最后，为第3段添加`.click()`方法，以便单击它时会将第3段向上滑出视图，同时将第4段向下滑入视图：

```
$(document).ready(function() {
    $('p: eq(3)').css('backgroundColor', '#fcf').hide();
    $('p: eq(2)').css('backgroundColor', '#cff').click(function() {
        $(this).slideUp('slow').next().slideDown('slow');
    });
});
```

通过截取到的这两个滑动效果变化过程中的屏幕截图，如图4-10所示，可以证实，它们确实是同时发生的。

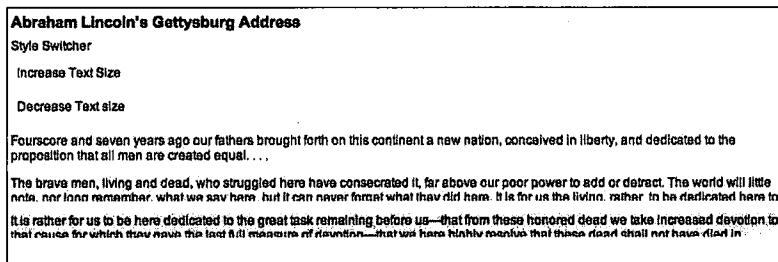


图 4-10

原来可见的浅蓝色段落，正处于向上滑到一半的状态；与此同时，原来隐藏的淡紫色段落，正处于向下滑到一半的状态。

回调函数

为了对不同元素上的效果实现排队，jQuery提供了回调函数。同我们在事件处理程序中看到的一样，回调函数就是作为方法的参数传递的一个普通函数。在效果方法中，它们是方法的最后一个参数。

当使用回调函数排队两个滑动效果时，可以在第3个段落滑上之前，先将第4个段落滑下。首先，我们看一看怎样通过回调函数设置`.slideDown()`方法：

```
$(document).ready(function() {
    $('p: eq(3)').css('backgroundColor', '#fcf').hide();
    $('p: eq(2)').css('backgroundColor', '#cff').click(function() {
        $(this).next().slideDown('slow', function() {
            // 这里的slideUp()将在slideDown结束后开始
        });
    });
});
```

不过，这里我们需要注意的是，必须搞清楚要滑上的到底是哪个段落。因为回调函数位于`.slideDown()`方法中，所以`$(this)`的环境已经发生了改变。现在，`$(this)`已经不再是指向`.click()`的第3个段落了——由于`.slideDown()`方法是通过`$(this).next()`调用的，所以该方法中的一切现在都将`$(this)`视为下一个同辈元素，即第4个段落。因而，如果在回调函数中放入`$(this).slideUp('slow')`，那么我们最终还会把刚刚显示出来的段落给隐藏起来。

可靠地引用`$(this)`的一种简单方法，就是在`.click()`方法内部把它保存到一个变量中，比如`var $thisPara = $(this)`。

这样，无论是在回调函数的外部还是内部，`$thisPara`引用的都是第3个段落。下面是使用了新变量之后的代码：

```
$(document).ready(function() {
    $('p:eq(3)')
        .css('backgroundColor', '#fcf')
        .hide();
    $('p:eq(2)')
        .css('backgroundColor', '#cff')
        .click(function() {
            var $thisPara = $(this);
            $thisPara.next().slideDown('slow',function() {
                $thisPara.slideUp('slow');
            });
        });
});
```

在`.slideDown()`的回调函数内部使用`$thisPara`会创建一个闭包。我们将在附录C中详细地讨论闭包。

这次效果中途的屏幕截图4-11显示，第3段和第4段同时都是可见的，而且，第4段已经完成下滑，第3段刚要开始上滑。

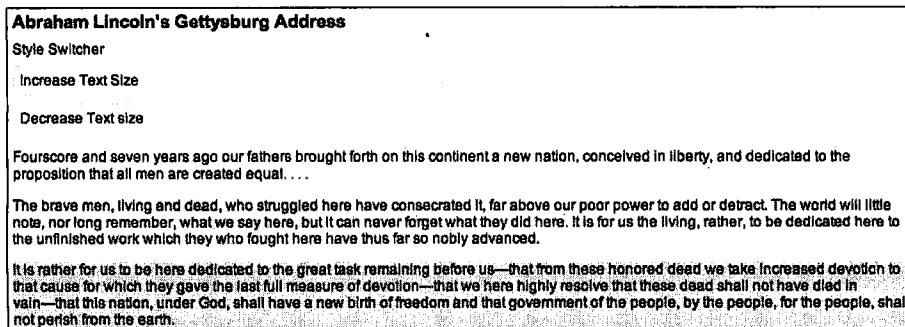


图 4-11

既然讨论了回调函数，那么就可以回过头来解决本章前面要在一系列效果之后改变背景颜色的问题了。这次，我们不采取前面没有成功的连缀`.css()`的方式，而是把它放到最后一个效果

的回调函数中：

```
$(document).ready(function() {
    $('div.label').click(function() {
        $('div.button')
            .fadeTo('slow', 0.5)
            .animate({left: 650}, 'slow')
            .fadeTo('slow', 1.0, function() {
                $(this).css('backgroundColor', '#f00');
            });
    });
});
```

这样，我们就实现了在按钮淡出到50%的不透明度，慢慢地向右移动650像素并淡入回100%的不透明度之后，再将按钮的背景变红的效果。

4.6 简单概括

随着在应用效果时需要考虑的变化的增多，要记住这些效果是同时发生还是按顺序发生会变得越来越困难。因此，下面简单的概括可能会对你有所帮助。

(1) 一组元素上的效果：

- ◆ 当在一个.animate()方法中以多个属性的方式应用时，是同时发生的。
- ◆ 当以方法连缀的形式应用时，是按顺序发生的（排队效果）。

(2) 多组元素上的效果：

- ◆ 默认情况下是同时发生的。
- ◆ 当在事件处理程序的回调函数中应用时，是按顺序发生的（排队效果）。

4.7 小结

通过使用本章介绍的效果方法，应该能够使用.css()来渐进地增大或减小文本的大小。也应该能够以不同的方式应用多种效果，以便逐渐地隐藏和显示页面元素。此外，还应该能够通过许多方式，同时地或相继地为多个元素实现动画效果。

在本书前面4章中，所有例子都只涉及了操作硬编码到页面的HTML中的元素。在第5章中，我们会探索使用jQuery来创建新元素的方式，以及如何把它们插入到选择的DOM结构中。

DOM操作——基于命令 改变页面

*Something's changed
Everything's rearranged
—Devo,
“Let's Talk”*

就像魔术师能够无中生有地变出一束花来，jQuery能够在网页中创建元素、属性和文本——酷似变魔术。别急，还没说完呢！通过jQuery，也能瞬间把这些东西变得无影无踪。而且，我们还可以拿起那束花，把它变成`<div class="magic" id="flowers-to-dove">dove</div>`。

5.1 操作属性

在本书前4章里，我们经常使用`.addClass()`和`.removeClass()`方法来示范如何改变页面上元素的外观。实际上，这两个方法所做的，是在操作`class`属性（或者，用DOM脚本编程的说法，是`className`属性）。即`.addClass()`方法创建或增加这个属性，而`.removeClass()`则删除或缩短该属性。而具备了这两种操作的`.toggleClass()`方法能够交替地添加和移除一个类。这样，我们就具有了处理类的一种有效而可靠的方式。

不过，`class`只是需要访问和改变的属性中的一种，其他属性还有`id`、`rel`及`href`等。对于其他属性，jQuery提供了`.attr()`和`.removeAttr()`方法。甚至，可以使用`.attr()`和`.removeAttr()`来分别代替相应的`.class()`方法——如果我们想给自己找点麻烦的话（但我们不想）。

1. 非`class`属性

有些属性如果不通过jQuery不容易操作，而且，通过jQuery还可以一次修改多个属性，同我们在第4章中使用`.css()`方法修改多个CSS属性的方式类似。

例如，可以同时为链接设置`id`、`rel`和`title`属性。首先来看一看我们例子中的HTML代码：

```

<h1 id="f-title">Flatland: A Romance of Many Dimensions</h1>
<div id="f-author">by Edwin A. Abbott</div>
<h2>Part 1, Section 3</h2>
<h3 id="f-subtitle">Concerning the Inhabitants of Flatland</h3>
<div id="excerpt">an excerpt</div>

<div class="chapter">
  <p class="square">Our Professional Men and Gentlemen are Squares
    (to which class I myself belong) and Five-Sided Figures or <a
    href="http://en.wikipedia.org/wiki/Pentagon">Pentagons</a>.
  </p>

  <p class="nobility hexagon">Next above these come the Nobility, of
    whom there are several degrees, beginning at Six-Sided Figures,
    or <a href="http://en.wikipedia.org/wiki/Hexagon">Hexagons</a>,
    and from thence rising in the number of their sides till they
    receive the honourable title of <a href="http://en.wikipedia.org/
    wiki/Polygon">Polygonal</a>, or many-Sided. Finally when the
    number of the sides becomes so numerous, and the sides themselves
    so small, that the figure cannot be distinguished from a <a
    href="http://en.wikipedia.org/wiki/Circle">circle</a>, he is
    included in the Circular or Priestly order; and this is the
    highest class of all.
  </p>

  <p><span class="pull-quote">It is a <span class="drop">Law of
    Nature</span> with us that a male child shall have <strong>one
    more side</strong> than his father</span>, so that each
    generation shall rise (as a rule) one step in the scale of
    development and nobility. Thus the son of a Square is a Pentagon;
    the son of a Pentagon, a Hexagon; and so on.
  </p>

  <!-- 省略的代码-->
</div>

```

对于以上HTML，我们可以循环遍历`<div class="chapter">`中的每个链接，并逐个为它们添加属性。如果只想为所有链接设置一个公共的属性值，那么在`$(document).ready`处理程序中通过一行代码即可完成这一操作：

```

$(document).ready(function() {
  $('div.chapter a').attr({'rel': 'external'});
});

```

但是，对于任何给定的文档，如果要保证JavaScript代码有效，那么每个`id`属性的值必须唯一。要为每个链接设置唯一的`id`，必须放弃单行解决方案，转而使用jQuery的`.each()`方法。

```

$(document).ready(function() {
  $('div.chapter a').each(function(index) {
    $(this).attr({
      'rel': 'external',
      'id': 'wikilink-' + index
    });
  });
});

```

jQuery的`.each()`方法类似一个迭代器，它实际上是`for`循环的一种更简洁的形式。当我们想要为选择符匹配的元素集合应用复杂的代码，而使用隐式迭代语法无法胜任时，可以利用`.each()`方法。在上面的例子中，我们为`.each()`方法的匿名函数传递了一个`index`参数，该参数可以添加到每个`id`值中。这个`index`参数类似一个计数器，对第1个链接它的值是0，然后对每个后续的链接它的值会递增1。也就是说，将`id`设置为`'wikilink-' + index`，对于第1个链接而言，其`id`值就是`wikilink-0`，对于第2个链接就是`wikilink-1`，然后依此类推。

而且，我们希望通过`title`属性来邀请人们到Wikipedia上面学习有关术语的更多内容。在前面的HTML代码中，虽然所有链接都指向了Wikipedia，但更可取的方式仍然是使用再具体一点的选择符表达式，只选择`href`属性中包含wikipedia的链接，以防我们今后再向HTML中添加其他非Wikipedia的链接：

```
$(document).ready(function() {
    $('div.chapter a[@href*=wikipedia]').each(function(index) {
        var $thisLink = $(this);
        $thisLink.attr({
            'rel': 'external',
            'id': 'wikilink-' + index,
            'title': 'learn more about ' + $thisLink.text() + ' at Wikipedia'
        });
    });
});
```

这里有必要提一下，我们把`$(this)`保存到名为`$thisLink`的变量中，原因就是需要多次使用它。

在设置了全部3个属性之后，以第1个链接为例，该链接现在的代码应该如下所示：

```
<a href="http://en.wikipedia.org/wiki/Pentagon" rel="external"
    id="wikilink-0" title="learn more about Pentagons at Wikipedia">
    Pentagons
</a>
```

2. 深入理解`$()`工厂函数

从本书开始到现在，我们一直在使用`$()`函数来访问文档中的元素。在某种意义上说，这个函数在jQuery库中处于最核心的位置，因为无论是在添加效果、事件，还是为匹配的元素集合添加属性时，都离不开它。

然而，除了选择元素之外，`$()`函数的圆括号内还有另外一个玄机——这个强大的特性使得`$()`函数不仅能够改变页面的视觉外观，更能改变页面中实际的内容。只要在这对圆括号中放入一组HTML元素，就能轻而易举地改变整个DOM结构。

再次重申，无论什么时候都不应该忘记，我们添加的所有功能、视觉效果或者文本性的信息，只有在可以使用（并启用了）JavaScript的Web浏览器中才能正常有效。但是，重要的信息应该对

所有人都是可以访问的，而不应该只针对使用了正确的软件的人。

在FAQ页面中，一个常见的功能是出现在每一对“问题-答案”后面的Back to top（返回页面顶部）链接。通常，这些链接并没有语义上的价值，因而可以合理地通过JavaScript来生成它们，将它们作为访问者所浏览页面的一个增强的子功能。在我们的例子中，需要为每个段落后面添加一个Back to top链接，而且，也需要添加作为Back to top链接返回目标的锚。首先，我们来创建新元素：

```
$(document).ready(function() {
    $('<a href="#top">back to top</a>');
    $('<a id="top"></a>');
});
```

图5-1是此时页面的外观。

The screenshot shows a portion of the 'Flatland: A Romance of Many Dimensions' page by Edwin A. Abbott. The visible text includes:

- Flatland: A Romance of Many Dimensions**
- by Edwin A. Abbott
- Part 1, Section 3**
- Concerning the Inhabitants of Flatland**
- an excerpt*
- Our Professional Men and Gentlemen are Squares (to which class I myself belong) and Five-Sided Figures or Pentagons.
- Next above these come the Nobility, of whom there are several degrees, beginning at Six-Sided Figures, or Hexagons, and from thence rising in the number of their sides till they receive the honourable title of Polygonal, or many-Sided. Finally when the number of the sides becomes so numerous, and the sides themselves so small, that the figure cannot be distinguished from a circle, he is included in the Circular or Priestly order; and this is the highest class of all.
- It is a Law of Nature with us that a male child shall have one more side than his father, so that each generation shall rise (as a rule) one step in the scale of development and nobility. Thus the son of a Square is a Pentagon; the son of a Pentagon, a Hexagon; and so on.

图 5-1

怎么看不见Back to top链接和锚呢？难道它们没有出现在页面上？简单地说，没有。虽然前面的两行代码创建了新的元素，但是还没有把它们添加页面中。为此，我们可以选择使用jQuery提供的众多插入方法中的一种。

5.2 插入新元素

jQuery提供了两种将元素插入到其他元素前面的方法：`.insertBefore()`和`.before()`。这两个方法作用相同，它们的区别取决于如何将它们与其他方法进行连缀。另外两个方法`.insertAfter()`和`.after()`之间也具有相同的关系，但正如它们的名字所暗示的，它们用于向其他元素后面插入元素。对于Back to top链接，我们使用`.insertAfter()`方法：

```

$(document).ready(function() {
  $('<a href="#top">back to top</a>').insertAfter('div.chapter p');
  $('<a id="top"></a>');
});

```

通过`.after()`方法也能完成与`.insertAfter()`相同的工作，只不过必须把选择符表达式放在这个方法的前面，而不是放在后面。在使用`.after()`方法时，`$(document).ready()`中的第一行代码可以改写成如下所示：

```
$('div.chapter p').after('<a href="#top">back to top</a>');
```

使用`.insertAfter()`，可以通过连缀更多方法连续地对所创建的`<a>`元素进行操作。而使用`.after()`，连缀的其他方法的操作对象就会变成`($('div.chapter p')`中选择符匹配的元素。

在将链接实际地插入到页面（也插入到DOM）中之后，`<div class="chapter">`中的每个段落后面，都应该出现`Back to top`链接，如图5-2所示。

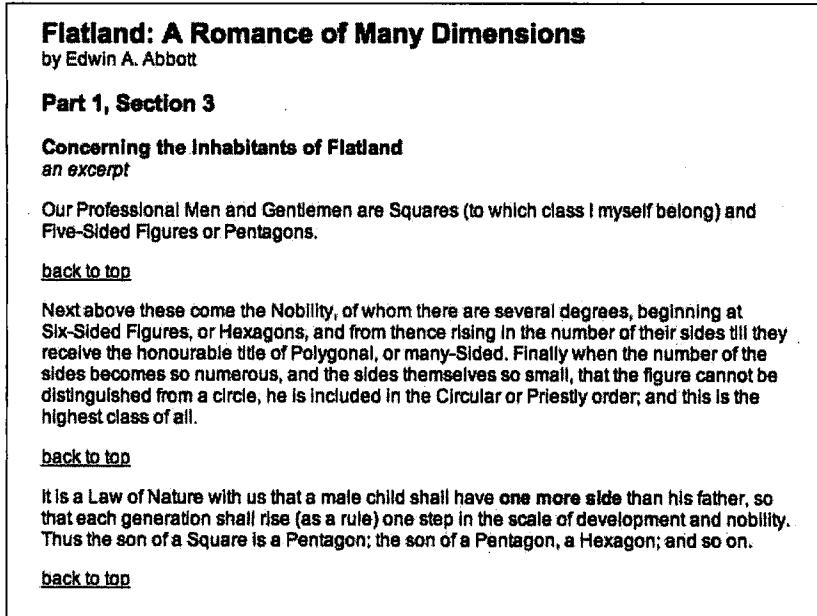


图 5-2

不过，现在的链接还不能用。因此，我们需要再插入`id="top"`的锚。要插入这个锚，可以选用一种在其他元素中插入元素的方法。

```

$(document).ready(function() {
  $('<a href="#top">back to top</a>').insertAfter('div.chapter p');
  $('<a id="top" name="top"></a>').prependTo('body');
});

```

新增加的代码会把锚准确地插入到<body>中的开始位置上；换句话说，就是插入到了页面的顶部。这样，通过.insertAfter()方法插入链接，并通过.prependTo()方法插入了作为目标的锚，我们就为页面添加了一组功能完备的Back to top链接。

但是，在页面顶部仍然可见的情况下，显示Back to top链接似乎意义不大。为此，可以对脚本进行一点简单的改进，即让这个链接到第4个段落后面再开始出现。这个改动很简单，只需将选择符表达式修改为：.insertAfter('div.chapter p:gt(2)').为什么在这里使用2？别忘了JavaScript中的索引都是从0开始的；因此，第1个段落的索引值是0，第2个是1，第3个是2，第4个是3。修改后的选择符表达式会在索引达到3时开始在每个段落后面插入链接（因为3是第一个大于2的数）。

在为HTML中添加了更多的段落后，修改选择符表达式的效果会变得更明显，如图5-3所示。

Concerning the Inhabitants of Flatland
an excerpt

Our Professional Men and Gentlemen are Squares (to which class I myself belong) and Five-Sided Figures or Pentagons.

Next above these come the Nobility, of whom there are several degrees, beginning at Six-Sided Figures, or Hexagons, and from thence rising in the number of their sides till they receive the honourable title of Polygonal, or many-Sided. Finally when the number of the sides becomes so numerous, and the sides themselves so small, that the figure cannot be distinguished from a circle, he is included in the Circular or Priestly order; and this is the highest class of all.

It is a Law of Nature with us that a male child shall have one more side than his father, so that each generation shall rise (as a rule) one step in the scale of development and nobility. Thus the son of a Square is a Pentagon; the son of a Pentagon, a Hexagon; and so on.

But this rule applies not always to the Tradesmen, and still less often to the Soldiers, and to the Workmen; who indeed can hardly be said to deserve the name of human Figures, since they have not all their sides equal. With them therefore the Law of Nature does not hold; and the son of an Isosceles (i.e. a Triangle with two sides equal) remains Isosceles still. Nevertheless, all hope is not such out, even from the Isosceles, that his posterity may ultimately rise above his degraded condition....

[back to top](#)

Rarely—in proportion to the vast numbers of Isosceles births—is a genuine and certifiable Equal-Sided Triangle produced from Isosceles parents.¹ Such a birth requires, as its antecedents, not only a series of carefully arranged intermarriages, but also a long-continued exercise of frugality and self-control on the part of the would-be ancestors of the coming Equilateral, and a patient, systematic, and continuous development of the Isosceles intellect through many generations.

[back to top](#)

图 5-3

5.3 移动元素

在Back to top链接的例子中，我们创建了新元素并把它们插入到了页面上。此外，也可以取得页面中某个位置上的元素，将它们插入到另一个位置上。动态地放置并格式化脚注，就是这种插入操作在实际中的一种应用。现在，Flatland的原始文本中已经包含了一个这样的脚注，但为了示范这种应用，下面我们还需要将文本其他几个部分指定为脚注：

<p>Rarely—and in proportion to the vast numbers of Isosceles births—and, is a genuine and certifiable Equal-Sided Triangle produced from Isosceles parents. "What need of a certificate?" a Spaceland critic may ask: "Is not the procreation of a Square Son a certificate from Nature herself, proving the Equal-sidedness of the Father?" I reply that no Lady of any position will marry an uncertified Triangle. Square offspring has sometimes resulted from a slightly Irregular Triangle; but in almost every such case the Irregularity of the first generation is visited on the third; which either fails to attain the Pentagonal rank, or relapses to the Triangular. Such a birth requires, as its antecedents, not only a series of carefully arranged intermarriages, but also a long-continued exercise of frugality and self-control on the part of the would-be ancestors of the coming Equilateral, and a patient, systematic, and continuous development of the Isosceles intellect through many generations.

</p>

<p>The birth of a True Equilateral Triangle from Isosceles parents is the subject of rejoicing in our country for many furlongs round. After a strict examination conducted by the Sanitary and Social Board, the infant, if certified as Regular, is with solemn ceremonial admitted into the class of Equilaterals. He is then immediately taken from his proud yet sorrowing parents and adopted by some childless Equilateral. The Equilateral is bound by oath never to permit the child henceforth to enter his former home or so much as to look upon his relations again, for fear lest the freshly developed organism may, by force of unconscious imitation, fall back again into his hereditary level.

</p>

<p>How admirable is the Law of Compensation! And how perfect a proof of the natural fitness and, I may almost say, the divine origin of the aristocratic constitution of the States of Flatland! By a judicious use of this Law of Nature, the Polygons and Circles are almost always able to stifle sedition in its very cradle, taking advantage of the irrepressible and boundless hopefulness of the human mind.…</p>

以上这几个段落中，分别包含一个位于标签中的脚注。通过以这种方式来标记HTML，能够保持脚注在上下文中的关系。在为这3个段落应用了样式表中的CSS规则后，它们的外观如图5-4所示。

接下来，可以提取出这些脚注，然后把它们插入到

>

和

>

之间。不过，这里我们要记住的一点是，即使是在隐式迭代的情况下，插入的顺序也是预定义的，即从DOM树的上方开始向下依次插入。由于维持脚注在页面上新位置中的顺序很重要，所以我们应该使用.insertBefore('#footer')。这样，footnote 1会被放在

>

和

>

之间，footnote 2会被放在footnote 1和

>

之间，然后依此类推。但是，如果在这里使用.insertAfter('div.chapter')，那么脚注的次序就会被颠倒。因此，当前的代码应该如下所示：

```
$(document).ready(function() {
    $('span.footnote').insertBefore('#footer');
});
```

Rarely—in proportion to the vast numbers of Isosceles births—is a genuine and certifiable Equal-Sided Triangle produced from Isosceles parents. "What need of a certificate?" a Spaceland critic may ask: "Is not the procreation of a Square Son a certificate from Nature herself, proving the Equal-sidedness of the Father?" I reply that no Lady of any position will marry an uncertified Triangle. Square offspring has sometimes resulted from a slightly Irregular Triangle; but in almost every such case the Irregularity of the first generation is visited on the third; which either fails to attain the Pentagonal rank, or relapses to the Triangular. Such a birth requires, as its antecedents, not only a series of carefully arranged intermarriages, but also a long-continued exercise of frugality and self-control on the part of the would-be ancestors of the coming Equilateral, and a patient, systematic, and continuous development of the Isosceles intellect through many generations.

[back to top](#)

The birth of a True Equilateral Triangle from Isosceles parents is the subject of rejoicing in our country for many furlongs round. After a strict examination conducted by the Sanitary and Social Board, the infant, if certified as Regular, is with solemn ceremonial admitted into the class of Equilaterals. He is then immediately taken from his proud yet sorrowing parents and adopted by some childless Equilateral. *The Equilateral is bound by oath never to permit the child henceforth to enter his former home or so much as to look upon his relations again, for fear lest the freshly developed organism may, by force of unconscious imitation, fall back again into his hereditary level.*

[back to top](#)

How admirable is the Law of Compensation! And how perfect a proof of the natural fitness and, I may almost say, the divine origin of the aristocratic constitution of the States of Flatland! By a judicious use of this Law of Nature, the Polygons and Circles are almost always able to stifle sedition in its very cradle, taking advantage of the irrepressible and boundless hopefulness of the human mind....

图 5-4

然而，在这行代码执行后，我们发现了一个大问题——由于脚注放在标签中，这就意味着它们在默认情况下应该显示为行内盒子，因此会导致这3个脚注前后相连，从视觉上无法将它们区分开来，如图5-5所示。

"What need of a certificate?" a Spaceland critic may ask: "Is not the procreation of a Square Son a certificate from Nature herself, proving the Equal-sidedness of the Father?" I reply that no Lady of any position will marry an uncertified Triangle. Square offspring has sometimes resulted from a slightly Irregular Triangle; but in almost every such case the Irregularity of the first generation is visited on the third; which either fails to attain the Pentagonal rank, or relapses to the Triangular. *The Equilateral is bound by oath never to permit the child henceforth to enter his former home or so much as to look upon his relations again, for fear lest the freshly developed organism may, by force of unconscious imitation, fall back again into his hereditary level.* And how perfect a proof of the natural fitness and, I may almost say, the divine origin of the aristocratic constitution of the States of Flatland!

图 5-5

解决这个问题的一种方案是修改CSS，使元素显示为块级盒子，但只针对位于

>外部的元素：

```
span.footnote {
    font-style: italic;
    font-family: "Times New Roman", Times, serif;
    display: block;
}
.chapter span.footnote {
    display: inline;
}
```

这样，我们的脚注就具备了雏形，如图5-6所示。

"What need of a certificate?" a Spaceland critic may ask: "Is not the procreation of a Square Son a certificate from Nature herself, proving the Equal-sidedness of the Father?" I reply that no Lady of any position will marry an uncertified Triangle. Square offspring has sometimes resulted from a slightly Irregular Triangle; but in almost every such case the Irregularity of the first generation is visited on the third; which either fails to attain the Pentagonal rank, or relapses to the Triangular.

The Equilateral is bound by oath never to permit the child henceforth to enter his former home or so much as to look upon his relations again, for fear lest the freshly developed organism may, by force of unconscious imitation, fall back again into his hereditary level.

And how perfect a proof of the natural fitness and, I may almost say, the divine origin of the aristocratic constitution of the States of Flatland!

图 5-6

至少，它们现在可以从视觉上明显地分开。然而，围绕这些脚注还有很多后续工作要做。更加健壮的一种脚注方案应该：

- (1) 在文本中标注出提取脚注的位置。
- (2) 为每个位置编号，并为相应的脚注添加对应的编号。
- (3) 在文本中的位置上创建一个指向对应脚注的链接，在脚注中创建返回文本位置的链接。

以上步骤可以在`.each()`方法中完成；不过，我们首先要为页面底部的脚注设置一个容器元素：

```
$(document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
});
```

由于要为脚注编号，所以在这里使用一个有序列表（`<ol id="notes">`）显然是非常合理的。为什么不使用一个能够替我们自动编号的元素呢？这里，我们为新创建的有序列表添加了值为`notes`的ID，然后把它插入到了`<div class="chapter">`后面。

5.3.1 标注、编号和链接到上下文

现在，可以对提取脚注的位置进行标注和编号了：

```
$(document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
  $('span.footnote').each(function(index) {
    $(this)
      .before('<a href="#foot-note-' + (index+1) +
        '" id="context-' + (index+1) + '" class="context"><sup>' +
        (index+1) + '</sup></a>');
  });
});
```

这里，一开始使用的是同简单的脚注例子中一样的选择符，但在它的后面连缀的则是`.each()`方法。

在`.each()`方法内部，第1行代码是`$(this)`，它表示迭代序列中的每一个脚注，然后在它的后面连缀了`.before()`方法。

出现在`.before()`方法的圆括号中的所有内容，都将被插入到相应的脚注（``）前面。这是一个相当长的连接字符串，但整个字符串的作用只是构建一个上标形式的链接。为清晰起见，下面我们还是仔细地分析一下这个字符串。

我们注意到，字符串中的前两部分构成了开始的`a`标签和一个`href`属性。这个`href`属性非常关键，因为它必须与脚注中的`id`属性（当然，不包括#）严格匹配：

```
.before('<a href="#foot-note-' + (index+1) + '" id="context-' +
(index+1) + '" class="context"><sup>' + (index+1) + '</sup></a>');
```

因为计数器从0开始，所以需要为`index`加1，以便让`href`属性从`#foot-note-1`开始编号。接下来的是`id`和`class`属性：

```
.before('<a href="#foot-note-' + (index+1) + '" id="context-' +
(index+1) + '" class="context"><sup>' +
(index+1) + '</sup></a>');
```

这一部分从以双引号关闭`href`属性开始。然后是`id`，`id`的值是为`context-`连接上`index+1`，以便与`href`属性中的编号保持一致。为了便于将来为这个标注链接添加样式，接着又为它指定了`context`类。

最后，是在`<sup>`元素中插入链接文本——同样，也是从数字1开始——并关闭链接：

```
.before('<a href="#foot-note-' + (index+1) + '" id="context-' +
(index+1) + '" class="context"><sup>' +
(index+1) + '</sup></a>');
```

于是，我们得到了3个如图5-7所示的指向脚注的标注链接。

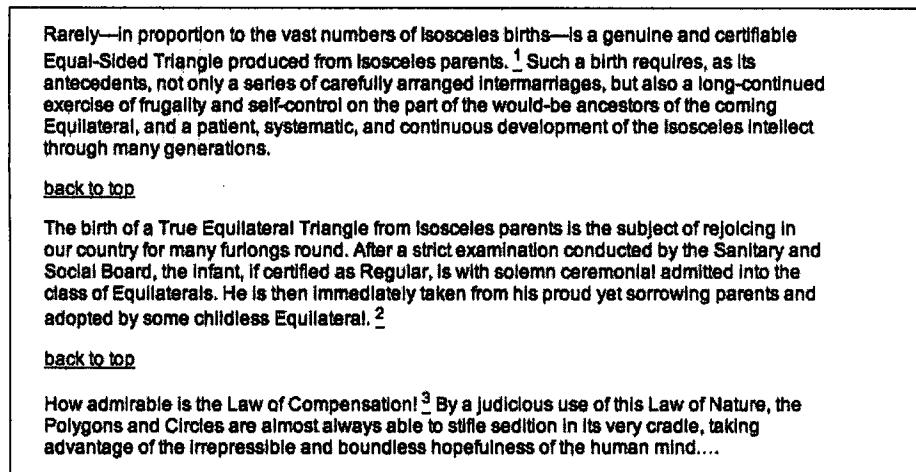


图 5-7

5.3.2 插入脚注

下一步是像我们在简单的例子中一样，移动``元素。不过，这一次要把它们插入到新创建的`<ol id="notes">`元素中。而且，为了保持正确的顺序，我们使用`.appendTo()`方法，这样会把连续的每个脚注依次插入到元素的末尾：

```
$(document).ready(function() {
    $('

</ol>').insertAfter('div.chapter');
    $('span.footnote').each(function(index) {
        $(this)
            .before('<a href="#foot-note-' + (index+1) +
                    '" id="context-' + (index+1) + '" class="context"><sup>' +
                    + (index+1) + '</sup></a>')
            .appendTo('#notes')
    });
});
```

这时候可别忘了`.appendTo()`仍然与`$(this)`保持着连缀关系，因此这些jQuery代码的含义是：把这些类为`footnote`的`span`，插入到ID为`notes`的元素中。

对于刚才移动的每个脚注，还需要为它们添加另一个链接，用于返回文本中的标注编号：

```
$(document).ready(function() {
    $('

</ol>').insertAfter('div.chapter');
    $('span.footnote').each(function(index) {
        $(this)
            .before('<a href="#foot-note-' + (index+1) + '"'
                    id="context-' + (index+1) + '" class="context"><sup>' +
                    + (index+1) + '</sup></a>')
            .appendTo('#notes')
            .append(' &ampnbsp(<a href="#context-' + (index+1) +
                    '">context </a>)')
    });
});
```

注意，最后插入的链接的`href`属性向后指向了对应标注的`id`。至此，应该能够看到添加链接之后的脚注，如图5-8所示。

"What need of a certificate?" a Spaceland critic may ask: "Is not the procreation of a Square Son a certificate from Nature herself, proving the Equal-sidedness of the Father?" I reply that no Lady of any position will marry an uncertified Triangle. Square offspring has sometimes resulted from a slightly Irregular Triangle; but in almost every such case the Irregularity of the first generation is visited on the third; which either fails to attain the Pentagonal rank, or relapses to the Triangular. ([context](#))

The Equilateral is bound by oath never to permit the child henceforth to enter his former home or so much as to look upon his relations again, for fear lest the freshly developed organism may, by force of unconscious imitation, fall back again into his hereditary level. ([context](#))

And how perfect a proof of the natural fitness and, I may almost say, the divine origin of the aristocratic constitution of the States of Flatland! ([context](#))

图 5-8

然而，每个脚注还缺少各自的编号。这是因为，虽然把它们放在了``元素内部，但还需要把每个脚注包装在一个``元素中。

5.4 包装元素

jQuery中用于将元素包装在其他元素中的方法，被贴切地命名为`.wrap()`。因为我们在这里希望将每个`$(this)`包装到``中，所以可以像下面这样来完成我们的脚注代码：

```
$ (document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
  $('span.footnote').each(function(index) {
    $(this)
      .before('<a href="#foot-note-' + (index+1) +
        '" id="context-' + (index+1) + '" class="context"><sup>' +
          (index+1) + '</sup></a>')
      .appendTo('#notes')
      .append(' &ampnbsp(<a href="#context-' + (index+1) + '">
        context </a>)')
    .wrap('<li id="foot-note-' + (index+1) + '"></li>');
  });
});
```

现在每个``元素都会带有与标注链接的`href`属性匹配的`id`。最终，我们得到了编号的并且带链接的脚注，如图5-9所示。

- 
1. "What need of a certificate?" a Spaceland critic may ask. "Is not the procreation of a Square Son a certificate from Nature herself, proving the Equal-sidedness of the Father?" I reply that no Lady of any position will marry an uncertified Triangle. Square offspring has sometimes resulted from a slightly Irregular Triangle; but in almost every such case the Irregularity of the first generation is visited on the third; which either fails to attain the Pentagonal rank, or relapses to the Triangular. ([context](#))
 2. The Equilateral is bound by oath never to permit the child henceforth to enter his former home or so much as to look upon his relations again, for fear last the freshly developed organism may, by force of unconscious imitation, fall back again into his hereditary level. ([context](#))
 3. And how perfect a proof of the natural fitness and, I may almost say, the divine origin of the aristocratic constitution of the States of Flatland! ([context](#))

图 5-9

当然，也可以像在段落中一样为每个脚注前面插入数字编号，但通过JavaScript动态生成具有语义的标记则是更令人满意的方案。

5.5 复制元素

本章到目前为止已经示范的操作包括：插入新创建的元素、将元素从文档中的一个位置移动到另一个位置，以及通过新元素来包装已有的元素。可是，有时候也会用到复制元素的操作。例如，可以复制出现在页面顶部的导航菜单，并把副本放到页脚上。实际上，无论何时，只要能通过复制元素增强页面的视觉效果，都是以重用代码来实现的好机会。毕竟，如果能够只编写一次代码并让jQuery替我们完成复制，何必要重写两遍同时又增加双倍的出错机会呢？

在复制元素时，需要使用jQuery的`.clone()`方法，这个方法能够创建任何匹配的元素集合的副本以便将来使用。与本章前面创建元素时一样，在为复制的元素应用一种插入方法之前，这些元素不会出现在文档中。例如，下面这行代码将创建`<div class="chapter">`中第1段落的副本：

```
$('div.chapter p:eq(0)').clone();
```

此时，通过图5-10可以看到，页面上没有变化。

Concerning the Inhabitants of Flatland
an excerpt

Our Professional Men and Gentlemen are Squares (to which class I myself belong) and Five-Sided Figures or Pentagons.

Next above these come the Nobility, of whom there are several degrees, beginning at Six-Sided Figures, or Hexagons, and from thence rising in the number of their sides till they receive the honourable title of Polygonal, or many-Sided. Finally when the number of the sides becomes so numerous, and the sides themselves so small, that the figure cannot be distinguished from a circle, he is included in the Circular or Priestly order; and this is the highest class of all.

It is a Law of Nature with us that a male child shall have one more side than his father, so that each generation shall rise (as a rule) one step in the scale of development and nobility. Thus the son of a Square is a Pentagon; the son of a Pentagon, a Hexagon; and so on.

图 5-10

为继续这个例子，我们可以让这个复制的段落出现在`<div class="chapter">`前面：

```
$('div.chapter p:eq(0)').clone().insertBefore('div.chapter');
```

现在，第1个段落在页面上出现了两次，如图5-11所示。而且，由于该段落的第一个实例没有位于`<div class="chapter">`内部，因此它不会带有与`div`相关的样式（最明显的就是宽度）：

Concerning the Inhabitants of Flatland
an excerpt

Our Professional Men and Gentlemen are Squares (to which class I myself belong) and Five-Sided Figures or Pentagons.

Our Professional Men and Gentlemen are Squares (to which class I myself belong) and Five-Sided Figures or Pentagons.

Next above these come the Nobility, of whom there are several degrees, beginning at Six-Sided Figures, or Hexagons, and from thence rising in the number of their sides till they receive the honourable title of Polygonal, or many-Sided. Finally when the number of the sides becomes so numerous, and the sides themselves so small, that the figure cannot be distinguished from a circle, he is included in the Circular or Priestly order; and this is the highest class of all.

It is a Law of Nature with us that a male child shall have one more side than his father, so

图 5-11

打一个多数人都熟悉的比方，`.clone()`之于插入方法，就相当于复制和粘贴操作。

5.5.1 复制的深度

在默认情况下，`.clone()`方法不仅会复制匹配的元素，也会复制其所有的后代元素。不过，

可以为这个方法传递一个参数，如果将这个参数设置为`false`，那么就只会复制匹配的元素。前面我们已经看到了，`$('.div.chapter p:eq(0)').clone()`复制的其实是下列HTML：

```
<p class="square">Our Professional Men and Gentlemen are Squares (to  
which class I myself belong) and Five-Sided Figures or Pentagons.  
</p>
```

下面我们把`false`放到它的圆括号中，像下面这样：

```
$('.div.chapter p:eq(0)').clone(false);
```

这次我们复制的只有段落元素：

```
<p class="square"></p>
```

内部的文本没有随同元素一起被复制，是因为文本自身也是一个DOM节点。

说明 `.clone()`方法不会复制元素中的事件。所以，不要忘记在所复制的元素上调用之前添加给它们的函数，以便重新应用处理程序。对这个问题有一个替代解决方案，即使用Brandon Aaron的插件方法`.cloneWithEvents()`直接复制事件。与复制事件相关的更多内容将在第10章中讨论。

5.5.2 通过复制创建突出引用

很多网站都和它们的印刷版一样，使用了突出引用（pull quote）来强调小块的文本并吸引读者的眼球。通过`.clone()`方法可以轻而易举地完成这种装饰效果。首先，我们来看一看例子文本的第3段：

```
<p>  
  <span class="pull-quote">It is a Law of Nature <span class="drop">  
  with us</span> that a male child shall have <strong>one more side  
  </strong> than his father</span>, so that each generation shall  
  rise (as a rule) one step in the scale of development and nobility.  
  Thus the son of a Square is a Pentagon; the son of a Pentagon, a  
  Hexagon; and so on.  
</p>
```

我们注意到这个段落以``元素开始，其中的类是为了复制而准备的。当把复制的``中的文本粘贴到其他位置上时，还需要修改它的样式属性，以便它与原来的文本区别开来。

1. 通过CSS使突出引用偏离正文

要实现这种样式，需要为复制的``添加一个`pulled`类，并在样式表中为这个类添加如下样式规则：

```
.pulled {  
  background: #e5e5e5;
```

```

position: absolute;
width: 145px;
top: -20px;
right: -180px;
padding: 12px 5px 12px 10px;
font: italic 1.4em "Times New Roman", Times, serif;
}

```

这样，就为pull-quote添加了浅灰色的背景、一些内边距和不同的字体。更重要的是将它绝对定位到了DOM中（绝对或相对）定位最近的祖先元素的上方20像素、右侧20像素。如果祖先元素中没有应用定位（除了static）的元素，那么pull-quote就会相对于文档中的<body>元素定位。为此，需要在jQuery代码中确保复制的pull-quote的父元素应用了position:relative属性。

虽然pull-quote盒子的上沿位置比较直观，但说到它的左边位于其定位的父元素右侧20像素时，恐怕就没有那么好理解了。要得到这个数字，需要先计算pull-quote盒子的总宽度，即width属性的值加上左右内边距，或者说145px + 5px + 10px，结果是160px。当为pull-quote设置right属性时，值为0会使pull-quote的右边与其父元素的右边对齐。因此，要使它的左边位于父元素右侧20像素，需要在相反的方向上将它移动比其总宽度多20像素的距离，即-180px。

2. 回到代码中

现在我们再回到jQuery代码中。首先，从匹配所有元素的选择符表达式开始，然后为选择的元素添加.each()方法，以便在循环遍历这些元素的过程中执行多项操作：

```

$(document).ready(function() {
  $('span.pull-quote').each(function(index) {
    ...
  });
});

```

接着，找到每个pull-quote的父元素并为它们应用CSS定位（position）属性：

```

$(document).ready(function() {
  $('span.pull-quote').each(function(index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
  });
});

```

这里，我们把作为父元素的段落保存在一个变量中，这是因为稍后还要用到这个对象。当需要多次引用一个jQuery对象时，最佳方式就是把它们保存到变量中。这样，通过减少对jQuery的\$()工厂函数的调用，能够提高遍历DOM的效率。

现在，我们肯定已经设置了CSS定位，因而可以复制pull-quote了。此时，我们先复制每个元素，然后为得到的副本添加pulled类，最后再把这个副本插入到段落的开始处：

```
$ (document).ready(function() {
  $('span.pull-quote').each(function(index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
    $(this).clone()
      .addClass('pulled')
      .prependTo($parentParagraph);
  });
});
```

因为前面对已经为这个复制的pull-quote元素设置了absolute的定位，因此它在段落中的位置是无所谓的。根据CSS规则中的设置，只要它处于这个段落的内部，它就会相对于段落的上边和右边进行定位。但是，假如我们想为这个突出引用应用float属性，那么它在段落中的位置就会影响到它的垂直位置。

目前，段落与其中插入的突出引用的外观如图5-12所示。

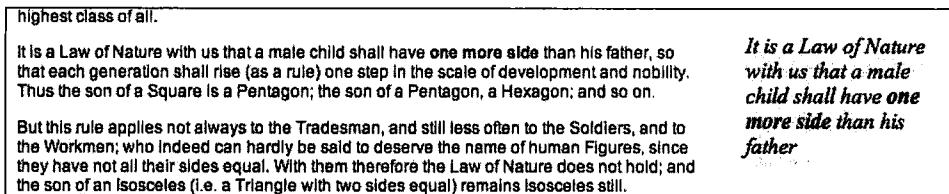


图 5-12

这个开头不错。但是，突出引用中通常不会保留与原文**one more side**一样的粗体字。也就是说，我们需要去掉文本中的*strong*、*em*、[及其他嵌入的标签](#)。而且，如果能够对这个pull-quote稍作修改，去掉一些文本并代之以省略号，那么效果会更好。为此，我们在例子文本中已经将某些文本包装在了元素中：

```
<p>
  <span class="pull-quote">It is a Law of Nature <span class="drop">
  with us</span> that a male child shall have <strong>one more side
  </strong> than his father</span>, so that each generation shall
  rise (as a rule) one step in the scale of development and nobility.
  Thus the son of a Square is a Pentagon; the son of a Pentagon, a
  Hexagon; and so on.
</p>
```

下面我们先来应用省略号，然后再去掉pull-quote中的所有HTML，只留下文本内容：

```
$ (document).ready(function() {
  $('span.pull-quote').each(function(index) {
```

```

var $parentParagraph = $(this).parent('p');
$parentParagraph.css('position', 'relative');
var $clonedCopy = $(this).clone();
$clonedCopy
    .addClass('pulled')
    .find('span.drop')
        .html('&hellip;')
    .end()
    .prependTo($parentParagraph);
var clonedText = $clonedCopy.text();
$clonedCopy.html(clonedText);
});
}
);

```

这次，我们一开始就把复制的对象保存在了变量中。由于我们并不是完全在同一个连缀方法序列中使用这个对象，所以将它保存到变量中是必要的。同时，也要注意，在找到>并将它的HTML内容替换为省略号(…)之后，我们使用了.end()方法结束上一次查询.find('span.drop')。这样，能够确保接下来插入到段落开始处的是整个副本，而不仅仅是省略号。

最后，我们又设置了另外一个变量clonedText，用它来保存副本中的纯文本内容。然后，又使用这些纯文本内容替换了副本中的HTML代码。现在，突出引用看起来如图5-13所示。

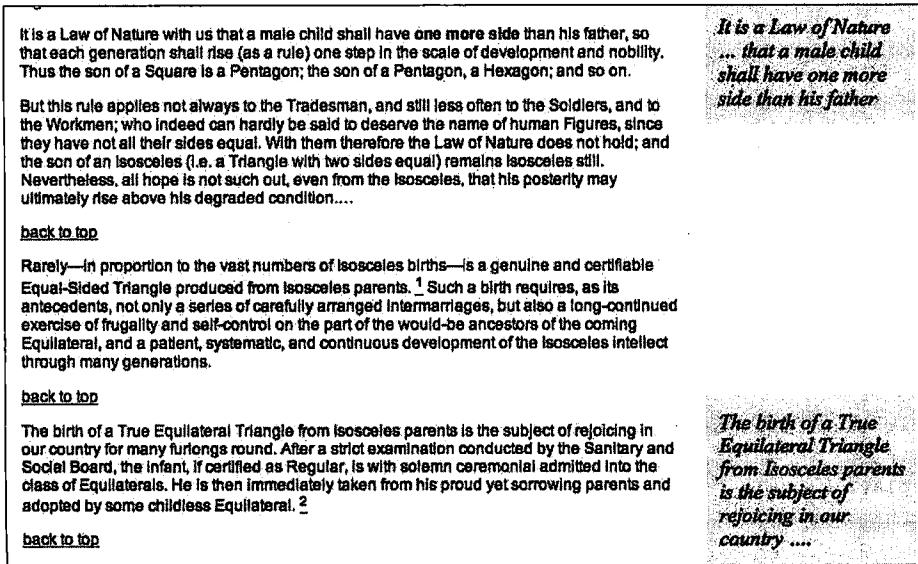


图 5-13

显然，后面的段落旁边也添加了>的另一个副本，说明我们的代码能够处理多个元素。

3. 修饰突出引用

复制的pull-quote元素都如期出现了，而且，不仅清除了其中的HTML元素，也通过省略号取代了被删除的文本。

毕竟，使用突出引用的目标之一就是增强视觉上的吸引力，因此下面我们要为pull-quote添加圆角和阴影效果。但是，pull-quote盒子的可变高度是个问题。因为至少在目前来看，除了最近的Safari版本，为一个元素应用两幅背景图像还是件不可能的事。

为了解决这个问题，我们可以在pull-quote外面再添加一个包装

：

```
$ (document).ready(function() {
    $('span.pull-quote').each(function(index) {
        var $parentParagraph = $(this).parent('p');
        $parentParagraph.css('position', 'relative');
        var $clonedCopy = $(this).clone();
        $clonedCopy
            .addClass('pulled')
            .find('span.drop')
            .html('&hellip;')
            .end()
            .prependTo($parentParagraph)
            .wrap('<div class="pulled-wrapper"></div>');
        var clonedText = $clonedCopy.text();
        $clonedCopy.html(clonedText);
    });
});
```

当然，还必须修改CSS来处理新添加的

，并应用两幅背景图像：

```
.pulled-wrapper {
    background: url(pq-top.jpg) no-repeat left top;
    position: absolute;
    width: 160px;
    right: -180px;
    padding-top: 18px;
}
.pulled {
    background: url(pq-bottom.jpg) no-repeat left bottom;
    position: relative;
    display: block;
    width: 140px;
    padding: 0 10px 24px 10px;
    font: italic 1.4em "Times New Roman", Times, serif;
}
```

这里，我们将以前应用给.pulled的一些规则转移到了.pulled-wrapper中。通过对宽度和内边距进行调整，使它们适应了背景图像的边框设计。同时，也修改了.pulled的position和

display属性，以便确保圆角背景在所有浏览器中都能正确显示。

图5-14展示了重新修饰之后的pull-quote的效果。

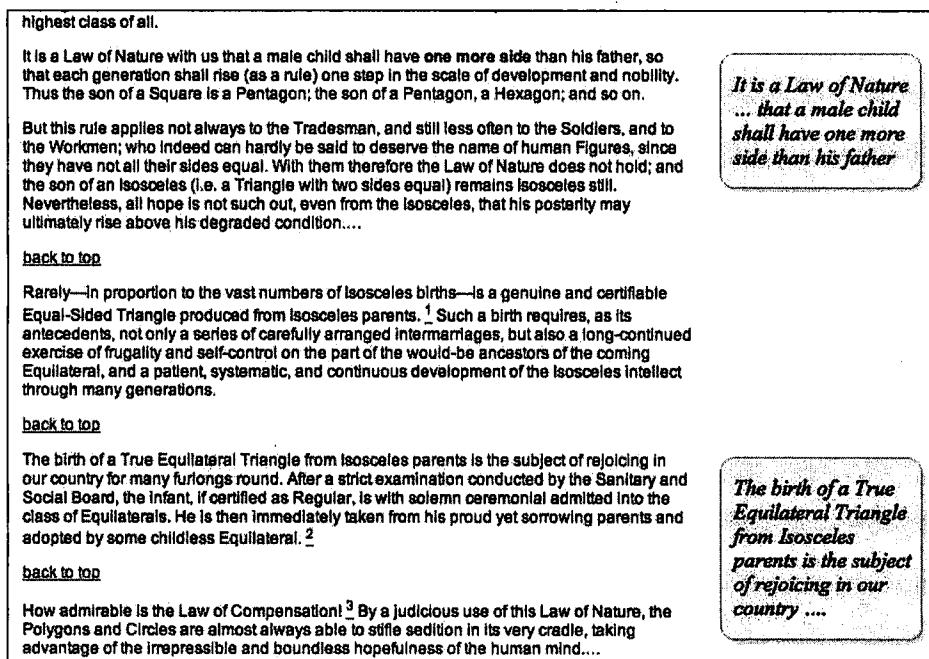


图 5-14

5.6 DOM 操作方法的简单归纳

对于jQuery提供的大量DOM操作方法，应该根据要完成的任务和元素的位置作出不同的选择。下面，我们简单地归纳出几乎能够包含任何情况下，完成任何任务所需要选用的相应方法。

(1) 要在每个匹配的元素中插入新元素，使用：

- ◆ .append()
- ◆ .appendTo()
- ◆ .prepend()
- ◆ .prependTo()

(2) 要在每个匹配的元素相邻的位置上插入新元素，使用：

- ◆ .after()
- ◆ .insertAfter()
- ◆ .before()
- ◆ .insertBefore()

(3) 要在每个匹配的元素外部插入新元素，使用：

◆ `.wrap()`

(4) 要用新元素或文本替换每个匹配的元素，使用：

◆ `.html()`

◆ `.text()`

(5) 要移除每个匹配的元素中的元素，使用：

◆ `.empty()`

(6) 要从文档中移除每个匹配的元素及其后代元素，但不实际删除它们，使用：

◆ `.remove()`

5.7 小结

在本章中，我们使用jQuery的DOM操作方法完成了元素的创建、复制、重组以及内容修饰等操作。通过在一个网页上应用这些方法，将一组普通的段落转换成了带脚注、突出引用、返回链接以及经过样式的文学摘录。

本书的教程部分到此已经接近尾声了，不过，在讨论更复杂的、扩展性的例子之前，下一章我们不妨先来通过jQuery的AJAX方法享受一次到服务器的往返旅行。

AJAX——让网站与时俱进

*Life's a bee without a buzz
It's going great till you get stung
—Devo,
“That's Good”*

AJAX是一位伟大的希腊勇士（实际上，是两位希腊勇士）的名字，他的传奇经历被收录在荷马史诗《伊利亚特》中。后来，AJAX成了一种家用清洁剂的名称^①。再后来，AJAX又被重新改造成一组Web技术的标签。

AJAX的最后一项含义最具现代意义，表示的是一组首字母缩写词——Asynchronous JavaScript and XML（异步JavaScript和XML）。一个AJAX解决方案中涉及的技术如下。

- JavaScript，通过用户或其他与浏览器相关事件捕获交互作用。
- XMLHttpRequest对象，通过这个对象可以在不中断其他浏览器任务的情况下向服务器发送请求。
- 服务器上的XML文件，或者其他类似的数据格式。
- 其他JavaScript，解释来自服务器的数据并将其呈现到页面上。

AJAX技术俨然已经被尊崇为Web前途的救世主，它能够将静态的网页转换成具有交互性的Web应用程序。由于浏览器对XMLHttpRequest对象实现的不一致性，许多框架纷纷拿出自己的方案来帮助开发者解决这个问题，jQuery也不例外。

AJAX真的能帮我们创造奇迹吗？

6.1 基于请求加载数据

在所有炒作和粉饰的背后，AJAX只不过是一种无需刷新页面即可从服务器上加载数据的手

^① 这里是指高露洁公司在1947年推出的多用清洁剂。这种清洁剂以强大的希腊英雄Ajax命名，因而也推出了Ajax产品的口号：“比污垢更强大！”。——译者注

段。这些数据的格式可以是很多种，而且，当数据到达时也有很多处理它们的方法可供选择。本章后面，当我们以多种方式完成同样的基本任务时，就能够清楚地看到这一点。

假设有一个显示字典中词条的页面，页面主体中的HTML如下所示：

```
<div id="dictionary">
</div>
```

对，没错！这个页面一开始没有内容。下面我们将使用jQuery的各种AJAX方法取得字典词条并用来填充这个

。

因为需要一种触发加载过程的方式，所以我们添加了几个调用事件处理程序的按钮：

```
<div class="letters">
  <div class="letter" id="letter-a">
    <h3>A</h3>
    <div class="button">Load</div>
  </div>
  <div class="letter" id="letter-b">
    <h3>B</h3>
    <div class="button">Load</div>
  </div>
  <div class="letter" id="letter-c">
    <h3>C</h3>
    <div class="button">Load</div>
  </div>
  <div class="letter" id="letter-d">
    <h3>D</h3>
    <div class="button">Load</div>
  </div>
</div>
```

再添加一些CSS规则，就得到了如图6-1所示的页面。

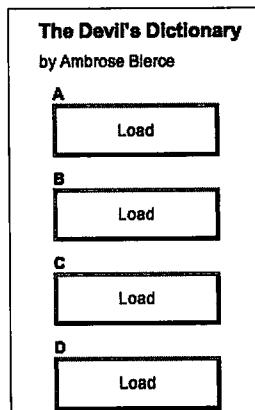


图 6-1

下面，我们关注的焦点就是如何向页面中填充内容。

6.1.1 追加 HTML

AJAX应用程序通常只不过是一个针对HTML代码块的请求。这种被称作AHAH(Asynchronous HTTP and HTML, 异步HTTP和HTML)的技术，通过jQuery来实现只是小菜一碟。首先，需要一些供插入用的HTML，我们把这些HTML放在与主文档位于同一目录下的a.html文件中。第2个HTML文件开始处的代码如下：

```

<div class="entry">
  <h3 class="term">ABDICTION</h3>
  <div class="part">n.</div>
  <div class="definition">
    An act whereby a sovereign attests his sense of the high
    temperature of the throne.
    <div class="quote">
      <div class="quote-line">Poor Isabella's Dead, whose
          abdication</div>
      <div class="quote-line">Set all tongues wagging in the Spanish
          nation.</div>
      <div class="quote-line">For that performance 'twere unfair to
          scold her:</div>
      <div class="quote-line">She wisely left a throne too hot to
          hold her.</div>
      <div class="quote-line">To History she'll be no royal riddle
          &mdash;</div>
      <div class="quote-line">Merely a plain parched pea that jumped
          the griddle.</div>
      <div class="quote-author">G.J.</div>
    </div>
  </div>
</div>

<div class="entry">
  <h3 class="term">ABSOLUTE</h3>
  <div class="part">adj.</div>
  <div class="definition">
    Independent, irresponsible. An absolute monarchy is one in which
    the sovereign does as he pleases so long as he pleases the
    assassins. Not many absolute monarchies are left, most of them
    having been replaced by limited monarchies, where the sovereign's
    power for evil (and for good) is greatly curtailed, and by
    republics, which are governed by chance.
  </div>
</div>
```

单独查看这个文档，结果显示它非常简单，如图6-2所示。

ABDICTION**n.**

An act whereby a sovereign attests his sense of the high temperature of the throne.
 Poor Isabella's Dead, whose abdication
 Set all tongues wagging in the Spanish nation.
 For that performance 'twere unfair to scold her:
 She wisely left a throne too hot to hold her.
 To History she'll be no royal riddle —
 Merely a plain parched pea that jumped the griddle.
 G.J.

ABSOLUTE**adj.**

Independent, irresponsible. An absolute monarchy is one in which the sovereign does as he pleases so long as he pleases the assassins. Not many absolute monarchies are left, most of them having been replaced by limited monarchies, where the sovereign's power for evil (and for good) is greatly curtailed, and by republics, which are governed by chance.

ACKNOWLEDGE**v.t.**

To confess. Acknowledgement of one another's faults is the highest duty imposed by our love of truth.

AFFIANCED**pp.**

Fitted with an ankle-ring for the ball-and-chain.

图 6-2

我们注意到，a.html并不是一个真正的HTML文档，它不包含<html>、<head>或者<body>，只包含最基本的代码。这个文件的唯一目的就是供插入到其他HTML文档中使用，插入的过程如下所示：

```
$ (document).ready(function() {
  $('#letter-a .button').click(function() {
    $('#dictionary').load('a.html');
  });
});
```

其中，.load()方法替我们完成了所有烦琐复杂的工作！这里，我们通过常规的jQuery选择符为HTML片段指定了目标位置，然后将要加载的文件的URL作为参数传递给.load()方法。现在，当单击第1个按钮时，这个文件就会被加载并插入到<div id="dictionary">内部。而且，当插入完成后，浏览器会立即呈现新的HTML，如图6-3所示。

从图6-3中可以看出，虽然这个HTML片段之前没有样式，但现在已经应用了样式。这些样式是主文档中的CSS规则所添加的，即当新HTML片段插入时，相应的CSS规则也会立即应用到它的标签上。

在这个例子中，当单击按钮时，字典中的解释会立即出现。这只是在本地运行应用程序的一种特殊情况。如果通过网络来传递相同的文档，那么需要多长的时间延迟是很难估计的。下面我们添加一个警告框，使其在加载完解释内容后立即显示：

```

$(document).ready(function() {
  $('#letter-a .button').click(function() {
    $('#dictionary').load('a.html');
    alert('Loaded!');
  });
});

```

The Devil's Dictionary
by Ambrose Bierce

A  B  C  D 	ABDICTION <i>n.</i> An act whereby a sovereign attests his sense of the high temperature of the throne. Poor Isabella's Dead, whose abdication Set all tongues wagging in the Spanish nation. For that performance 'twere unfair to scold her: She wisely left a throne too hot to hold her. To History she'll be no royal riddle — Merely a plain parched pea that jumped the griddle. G.J.
ABSOLUTE <i>adj.</i> Independent, irresponsible. An absolute monarchy is one in which the sovereign does as he pleases so long as he pleases the assassins. Not many absolute monarchies are left, most of them having been replaced by limited monarchies, where the sovereign's power for evil (and for good) is greatly curtailed, and by republics, which are governed by chance.	
ACKNOWLEDGE <i>v.t.</i> To confess. Acknowledgement of one another's faults is the highest duty imposed by our love of truth.	

图 6-3

根据代码中的结构，你可能会认为警告框只有在加载过程完成后才会显示。然而，由于网络延迟，警告框很可能先于加载完成就出现了。所有AJAX请求在默认情况下都是异步的(*asynchronous*)，否则，我们就要称它为SJAX了^①，而后者显然难以与AJAX相提并论^②！异步加载意味着在发出取得HTML片段的HTTP请求后，会立即恢复脚本执行，无需等待。在之后的某个时刻，当浏览器收到服务器的响应时，再对响应的数据进行处理。这通常都是人们期望的行为，但它不会导致在等待数据返回期间锁定整个Web浏览器。

对于必须要延迟到加载完成才能继续的操作，jQuery提供了一个回调函数。本章后面会展示使用回调函数的例子。

6.1.2 操作 JavaScript 对象

通过请求获取充分格式化的HTML虽然很方便，但有时候，在显示数据之前，需要脚本对数

① S是synchronous的首字母，即同步。——译者注

② 作者这里的意思是，如果不是AJAX，而是SJAX，即不是异步加载，而是同步加载，那么就不会有那么大的影响了。——译者注

据进行某些处理。在这种情况下，我们希望取得能够通过JavaScript进行遍历的数据结构。

1. 取得JavaScript对象

通过jQuery的选择符，虽然可以遍历取回的HTML并操作它，但HTML必须首先要插入到文档中。一种更本地化的JavaScript数据格式则意味着更少的编码工作量。

前面我们曾经看到过，JavaScript对象就是由一些“键-值”对组成的，而且可以方便地使用花括号（{}）来定义。另一方面，JavaScript的数组则可以使用方括号（[]）进行动态定义。将这两种语法组合起来，可以轻松地表达复杂而且庞大的数据结构。

Douglas Crockford为利用这种简单的语法起了一个名字，叫做JSON（JavaScript Object Notation，JavaScript对象表示法）。通过这种表示法能够方便地取代数据量庞大的XML格式：

```
{
  "key": "value",
  "key 2": [
    "array",
    "of",
    "items"
  ]
}
```

要了解有关JSON的各种用途，以及它在多种语言中的实现，请访问<http://json.org/>。

如果用这种格式对字典中的解释进行编码，那么可能会有很多种编码方式。这里，我们把一些字典的词条放在一个名叫b.json的JSON文件中：

```
[
  {
    "term": "BACCHUS",
    "part": "n.",
    "definition": "A convenient deity invented by the ancients as an
                  excuse for getting drunk.",
    "quote": [
      "Is public worship, then, a sin,",
      "That for devotions paid to Bacchus",
      "The lictors dare to run us in,",
      "And resolutely thump and whack us?"
    ],
    "author": "Jorace"
  },
  {
    "term": "BACKBITE",
    "part": "v.t.",
    "definition": "To speak of a man as you find him when he can't
                  find you."
  }
]
```

```

},
{
  "term": "BEARD",
  "part": "n.",
  "definition": "The hair that is commonly cut off by those who
justly execrate the absurd Chinese custom of shaving the head."
},

```

要取得这些数据，可以使用`$.getJSON()`方法，这个方法会在取得相应文件后对文件进行处理，并将处理得到的JavaScript对象提供给代码。

2. 全局jQuery函数

到目前为止，我们使用的所有jQuery方法都需要通过`$()`工厂函数构建的一个jQuery对象进行调用。通过选择符表达式，我们可以指定一组要操作的DOM节点，然后再用这些jQuery方法以某种方式对它们进行操作。然而，`$.getJSON()`却不一样。从逻辑上说，没有该方法适用的DOM元素；作为结果的对象只能提供给脚本，而不能插入到页面中。为此，`getJSON()`是作为全局jQuery对象的方法定义的，也就是说，它不是个别jQuery对象实例的方法。

如果JavaScript中有类似其他面向对象语言中的类，那我们可以把`$.getJSON()`称为类方法。而为了容易理解，我们在这里称其为全局函数；实际上，为了不与其他函数名称发生冲突，这些全局函数使用的是jQuery命名空间。

在使用这个函数时，我们还需要像以前一样为它传递文件名：

```

$(document).ready(function() {
  $('#letter-b .button').click(function() {
    $.getJSON('b.json');
  });
});

```

当单击按钮时，我们看不到以上代码的效果。因为虽然函数调用加载了文件，但是并没有告诉JavaScript对返回的数据如何处理。为此，我们需要使用一个回调函数。

`$.getJSON()`函数可以接受第2个参数，这个参数是当加载完成时调用的函数。如上所述，AJAX请求都是异步的，回调函数提供了一种等待数据返回的方式，而不是立即执行代码。回调函数也需要一个参数，该参数中保存着返回的数据。因此，我们的代码要写成：

```

$(document).ready(function() {
  $('#letter-b .button').click(function() {
    $.getJSON('b.json', function(data) {
      });
    });
});

```

这样，我们就可以在函数中通过`data`变量来遍历相应的数据结构了。具体来说，需要迭代顶级数组，为每个项构建相应的HTML代码。虽然可以在这里使用标准的`for`循环，但我们要借此

机会介绍jQuery的另一个实用全局函数`$.each()`，在第5章中，我们曾看到过它的对应方法`.each()`。`$.each()`函数不操作jQuery对象，它以一个数组或映射作为第1个参数，以一个回调函数作为第2个参数。此外，还需要将每次循环中数组或映射的当前索引和当前项作为回调函数的两个参数：

```
$(document).ready(function() {
    $('#letter-b .button').click(function() {
        $.getJSON('b.json', function(data) {
            $('#dictionary').empty();
            $.each(data, function(entryIndex, entry) {
                var html = '<div class="entry">';
                html += '<h3 class="term">' + entry['term'] + '</h3>';
                html += '<div class="part">' + entry['part'] + '</div>';
                html += '<div class="definition">';
                html += entry['definition'];
                html += '</div>';
                html += '</div>';
                $('#dictionary').append(html);
            });
        });
    });
});
```

在循环开始前，首先需要清空`<div id="dictionary">`，以便用重新构造的HTML填充它。然后，通过`$.each()`函数依次遍历每个项，并使用`entry`映射^①的内容构建起HTML代码结构。最后，把构建好的HTML添加到`<div>`，以便将其插入DOM树中。

说明 这种方法要求数据中包含可以直接用来构建HTML的安全内容，例如，数据中不能包含任何`<`字符。

现在所剩的就是处理词条中的引用语了，这需要使用另一个`$.each()`循环：

```
$(document).ready(function() {
    $('#letter-b .button').click(function() {
        $.getJSON('b.json', function(data) {
            $('#dictionary').empty();
            $.each(data, function(entryIndex, entry) {
                var html = '<div class="entry">';
                html += '<h3 class="term">' + entry['term'] + '</h3>';
                html += '<div class="part">' + entry['part'] + '</div>';
                html += '<div class="definition">';
                html += entry['definition'];
                
```

^① 在前面的JSON文件`b.json`中，每一对花括号`({})`表示一个`entry`（词条），即jQuery中的映射，或者说是JavaScript对象。——译者注

```

if (entry['quote']) {
    html += '<div class="quote">';
    $.each(entry['quote'], function(lineIndex, line) {
        html += '<div class="quote-line">' + line + '</div>';
    });
    if (entry['author']) {
        html += '<div class="quote-author">' + entry['author'] +
               '</div>';
    }
    html += '</div>';
}
html += '</div>';
html += '</div>';
$( '#dictionary' ).append( $( html ) );
);
});
});
);
});

```

编写完这些代码后，就可以单击下一个按钮来验证我们的成果了，如图6-4所示。

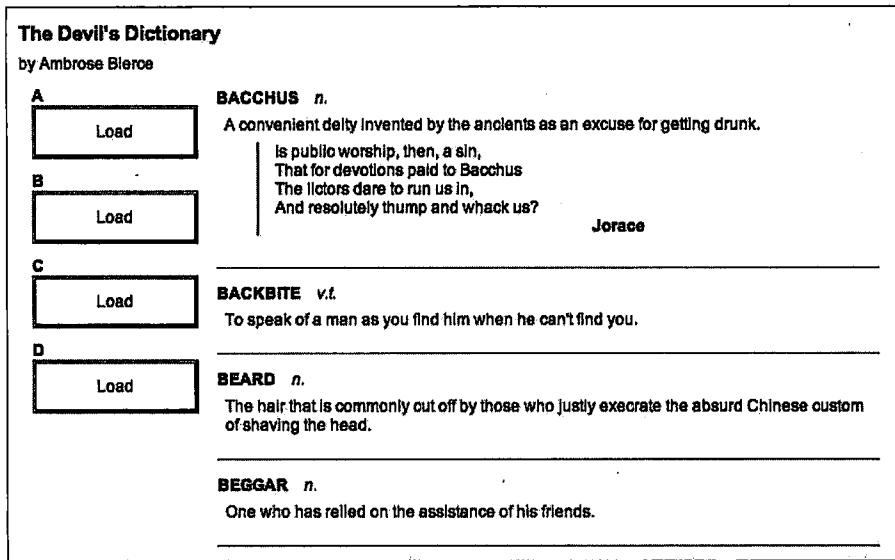


图 6-4

说明 尽管JSON格式很简洁，但它却不容许任何错误。所有方括号、花括号、引号和逗号都必须合理而且适当地存在，否则文件不会加载。而且，在多数浏览器中，当文件不会加载时我们都看不到错误信息；脚本只是默默地彻底中止运转。

3. 执行脚本

有时候，在页面初次加载时就取得所需的全部JavaScript也是没有必要的。具体需要取得哪个

脚本，要视用户的操作而定。虽然可以在需要时动态地引入<script>标签，但注入所需代码的更优雅的方式则是通过jQuery直接加载.js文件。

向页面中注入脚本与加载一个HTML片段一样简单。但在这种情况下，需要使用全局函数`$.getScript()`，这个全局函数与它的同辈函数们类似，接受一个URL参数以查找脚本文件：

```
$(document).ready(function() {
    $('#letter-c .button').click(function() {
        $.getScript('c.js');
    });
});
```

在前一个例子中，接下来要做的应该是处理结果数据，以便有效地利用加载的文件。然而，对于一个脚本文件来说，这个过程是自动化；换句话说，脚本会自动执行。

以这种方式取得的脚本会在当前页面的全局环境下执行。这意味着脚本有权访问在全局环境中定义的函数和变量，当然也包括jQuery自身。因而，我们可以模仿JSON的例子来准备脚本代码，以便在脚本执行时将HTML插入到页面中。现在，将以下脚本代码保存到c.js中：

```
var entries = [
    {
        "term": "CALAMITY",
        "part": "n.",
        "definition": "A more than commonly plain and unmistakable
                      reminder that the affairs of this life are not of
                      our own ordering. Calamities are of two kinds:
                      misfortune to ourselves, and good fortune to
                      others."
    },
    {
        "term": "CANNIBAL",
        "part": "n.",
        "definition": "A gastronome of the old school who preserves the
                      simple tastes and adheres to the natural diet of
                      the pre-pork period."
    },
    {
        "term": "CHILDHOOD",
        "part": "n.",
        "definition": "The period of human life intermediate between the
                      idiocy of infancy and the folly of youth —
                      two removes from the sin of manhood and three from
                      the remorse of age."
    }
];
var html = '';
$.each(entries, function() {
    html += '<div class="entry">';
    html += '<h3 class="term">' + this['term'] + '</h3>';
    html += '<div class="part">' + this['part'] + '</div>';
});
```

```

html += '<div class="definition">' + this['definition'] + '</div>';
html += '</div>';
});

$( '#dictionary' ).html( html );

```

最后，单击第3个按钮，应该会看到我们预期的结果，如图6-5所示。

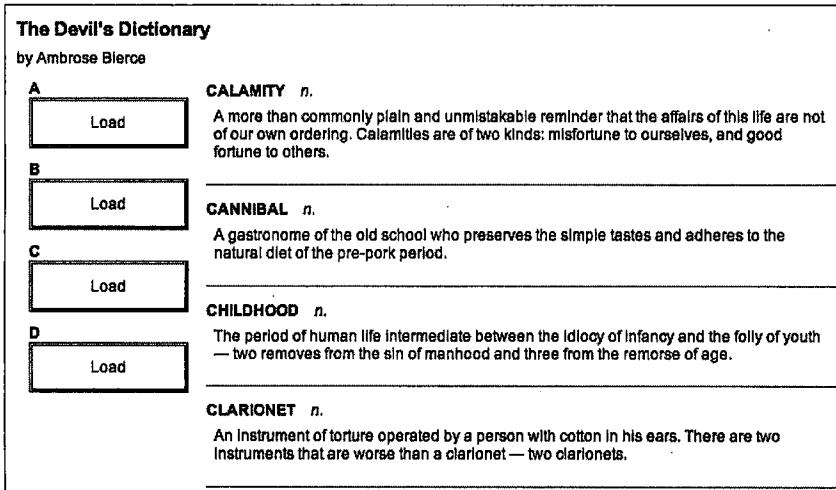


图 6-5

6.1.3 加载 XML 文档

XML是缩写词AJAX中的一部分，但我们至今还没有谈到加载XML文档。加载XML文档很简单，而且与JSON技术也相当接近。首先，需要将希望显示的数据包含在一个名为d.xml的XML文件中：

```

<?xml version="1.0" encoding="UTF-8"?>
<entries>
    <entry term="DANCE" part="v.i.">
        <definition>
            To leap about to the sound of tittering music, preferably with
            arms about your neighbor's wife or daughter. There are many
            kinds of dances, but all those requiring the participation of
            the two sexes have two characteristics in common: they are
            conspicuously innocent, and warmly loved by the vicious.
        </definition>
    </entry>
    <entry term="DAY" part="n.">
        <definition>
            A period of twenty-four hours, mostly misspent. This period is
            divided into two parts, the day proper and the night, or day
            improper <![CDATA[&mdash;]]> the former devoted to sins of
            business, the latter consecrated to the other sort. These two
            kinds of social activity overlap.
        </definition>
    </entry>

```

```

<entry term="DEBT" part="n.">
  <definition>
    An ingenious substitute for the chain and whip of the
    slave-driver.
  </definition>
  <quote author="Barlow S. Vode">
    <line>As, pent in an aquarium, the troutlet</line>
    <line>Swims round and round his tank to find an outlet,</line>
    <line>Pressing his nose against the glass that holds him,</line>
    <line>Nor ever sees the prison that enfolds him;</line>
    <line>So the poor debtor, seeing naught around him,</line>
    <line>Yet feels the narrow limits that impound him,</line>
    <line>Grieves at his debt and studies to evade it,</line>
    <line>And finds at last he might as well have paid it.</line>
  </quote>
</entry>
<entry term="DEFAME" part="v.t.">
  <definition>
    To lie about another. To tell the truth about another.
  </definition>
</entry>
</entries>

```

当然，通过XML来表示这些数据的形式有很多种，而其中一些能够非常近似地模仿我们已经确定的HTML结构或者前面使用的JSON。不过，这里我们示范了XML的一些更方便人类阅读的特性，例如使用属性term和part，而不是标签。

下面以我们熟悉的方式开始编写函数：

```

$(document).ready(function() {
  $('#letter-d .button').click(function() {
    $.get('d.xml', function(data) {
      });
    });
  });
});
```

这次，帮助我们完成任务的是`$.get()`函数。通常，这个函数只是简单地取得由URL指定的文件，然后将纯文本格式的数据提供给回调函数。但是，在根据服务器提供的MIME类型知道响应的是XML的情况下，提供给回调函数的将是XML DOM树。

好在，我们已经领略过了jQuery强大的DOM遍历能力。对XML文档就如同对HTML文档一样，也可以使用常规的`.find()`、`.filter()`及其他遍历方法：

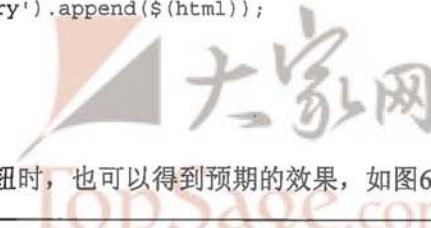
```

$(document).ready(function() {
  $('#letter-d .button').click(function() {
    $.get('d.xml', function(data) {
      $('#dictionary').empty();
      $(data).find('entry').each(function() {
        var $entry = $(this);
        var html = '<div class="entry">';
        $entry.html(html);
        $('#dictionary').append($entry);
      });
    });
  });
});
```

```

html += '<h3 class="term">' + $entry.attr('term') + '</h3>';
html += '<div class="part">' + $entry.attr('part') + '</div>';
html += '<div class="definition">';
html += $entry.find('definition').text();
var $quote = $entry.find('quote');
if ($quote.length) {
    html += '<div class="quote">';
    $quote.find('line').each(function() {
        html += '<div class="quote-line">' + $(this).text() +
               '</div>';
    });
    if ($quote.attr('author')) {
        html += '<div class="quote-author">' +
               $quote.attr('author') + '</div>';
    }
    html += '</div>';
}
html += '</div>';
html += '</div>';
$('#dictionary').append($('#html'));
});
});
});
});

```



这样，当单击第4个按钮时，也可以得到预期的效果，如图6-6所示。

The Devil's Dictionary

by Ambrose Bierce

A	<p>DANCE <i>v.i.</i></p> <p>To leap about to the sound of littering music, preferably with arms about your neighbor's wife or daughter. There are many kinds of dances, but all those requiring the participation of the two sexes have two characteristics in common: they are conspicuously innocent, and warmly loved by the vicious.</p> <hr/>
B	<p>Load</p>
C	<p>DAY <i>n.</i></p> <p>A period of twenty-four hours, mostly misspent. This period is divided into two parts, the day proper and the night, or day Improper — the former devoted to sins of business, the latter consecrated to the other sort. These two kinds of social activity overlap.</p> <hr/>
D	<p>DEBT <i>n.</i></p> <p>An ingenious substitute for the chain and whip of the slave-driver.</p> <p>As, pent in an aquarium, the troutlet Swims round and round his tank to find an outlet, Pressing his nose against the glass that holds him, Nor ever sees the prison that enfolds him; So the poor debtor, seeing naught around him, Yet feels the narrow limits that impound him, Grieves at his debt and studies to evade it, And finds at last he might as well have paid it.</p>

图 6-6

这是我们已知的DOM遍历方法的新用途，而且，jQuery对XPath支持的实用性由此也可见一斑。CSS的选择符语法典型地适合处理HTML页面，XPath则是为XML而构建的。这就意味着，虽然使用任何一种语法都可以找到期望的DOM元素，但现有的XPath表达式则可能会重用于使用相同的XML文件的其他系统中。

XML对任意标签和属性的习惯用法，以及不使用类作为标识，使得通过XPath来遍历它特别方便。例如，假设我们想把显示的内容限定为那些带有引用进而带有作者属性的词条。那么，通过将entry修改为entry[quote]就可以把词条限定为必须包含嵌套的引用元素。然后，还可以通过entry[quote[@author]]进一步限定词条中的引用元素必须包含author属性。这样，带有初始选择符的代码行应该写成：

```
$(data).find('entry[quote[@author]]').each(function() {
```

由图6-7可以看出，新的选择符表达式对返回的词条进行了适当的限制。

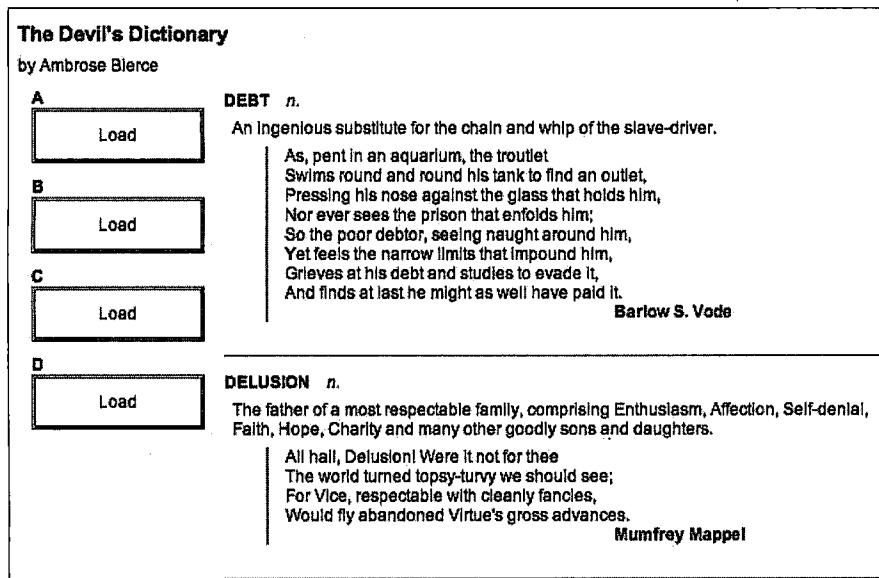


图 6-7

6.2 选择数据格式

我们已经看到了4种外部数据的格式，每种格式都可以通过jQuery本地的AJAX函数加以处理。而且，我们也亲自验证了这4种格式都能够用来方便地处理任务，在用户请求它时（而不是之前）将信息加载到现有的页面上。那么，当确定在应用程序中使用哪种格式时，应该考虑什么因素呢？

HTML片段实现起来只需要很小的工作量。这种格式的外部数据可以通过一种简单的方法加

载并插入到页面中，甚至连回调函数都不必使用。也就是说，对于将新HTML添加到现有页面中的简单任务来说，无需遍历数据。但另一方面，这种数据的结构方式却不一定能够在其他应用程序中得到重用，因为这种外部文件与它们的目标容器^①必须紧密结合。

JSON文件的结构使它可以方便地被重用。而且，它们非常简洁，也容易阅读。这种数据结构必须通过遍历来提取相关信息，然后再将信息呈现到页面上，不过通过标准的JavaScript技术就能做到这一点。由于调用一次JavaScript的eval()函数就能解析这种格式的文件，所以读取JSON文件的速度非常快。然而，使用eval()函数却会带来固有的风险。另外，JSON文件中的错误可能会导致页面上的脚本静默地中止运行，甚至还会带来其他的负面影响。因此，这种数据必须由信得过的人仔细进行构建。

JavaScript文件能够提供极大的灵活性，但它却不是一种真正的数据存储机制。因为这种文件针对特定的语言，所以不能通过它们将同样的信息提供给完全不同的系统。然而，能够加载JavaScript，则意味着可以将很少用到的行为提取到外部文件中，从而在加载该文件之前有效地减少页面中的代码量。

XML文档的可移植性是当之无愧的王者。由于XML已经成为了Web服务领域的“世界语”，因而以这种格式提供数据使它极有可能在其他地方被重用。比如，Flickr (<http://flickr.com/>)、del.icio.us (<http://del.icio.us/>) 和Upcoming (<http://upcoming.org/>) 都以XML格式输出它们的数据，从而催生了使用它们数据的很多有价值的Mashup应用^②。不过，XML格式的文件体积相对较大，所以同其他文件格式相比，解析和操作它们的速度要慢一些。

通过以上对各种数据格式优缺点的分析，我们知道在不需要与其他应用程序共享数据的情况下，以HTML片段提供外部数据一般来说是最简单的。如果数据需要重用，而且其他应用程序也可能因此受到影响，那么在性能和文件大小方面具有优势的JSON通常是不错的选择。而当远程应用程序未知时，XML则能够为良好的互操作性提供最可靠的保证。

最后一个要考虑的问题是，数据是否已经可以使用。如果是，那么这几种格式都可能成为首选，关键是作出最适合我们需求的决定。

6.3 向服务器传递数据

此前，我们的例子都是从Web服务器上取得静态的数据文件。然而，AJAX的价值只有当服务器能够基于浏览器的输入动态形成数据时才能得到体现。同样，在这种情况下jQuery也能为我们提供帮助；前面介绍的所有方法在经过修改之后，都可以实现双向的数据传送。

① 即上文提到的现有页面。——译者注

② Mashup应用，指利用几个相关或不相关的网站提供的API，将相应网站提供的内容直接或经过适当地加工之后整合显示在自己网站中的一种应用类型。——译者注

说明 由于示范这些技术需要同Web服务器进行交互，因此我们这里将首次用到服务器端代码。在给定的例子中，我们使用PHP脚本编程语言，该语言使用非常普遍而且能够免费取得。不过，我们不会在这里介绍如何设置支持PHP的Web服务器，相关的帮助可以在Apache (<http://apache.org/>) 或PHP (<http://php.net/>) 的网站上找到，或者也可以咨询为你的网站提供主机服务的公司。

6.3.1 执行GET请求

为了示范客户端与服务器之间的通信，我们要编写一个基于每次请求只向浏览器发送一个字典词条的脚本。词条的选择取决于从浏览器发送到服务器的参数。服务器端脚本将从如下内部数据结构中提取相应的数据：

```
<?php
$entries = array(
    'EAVESDROP' => array(
        'part' => 'v.i.',
        'definition' => 'Secretly to overhear a catalogue of the crimes
                           and vices of another or yourself.',
        'quote' => array(
            'A lady with one of her ears applied',
            'To an open keyhole heard, inside.',
            'Two female gossips in converse free &mdash;',
            'The subject engaging them was she.',
            '"I think," said one, "and my husband thinks",
            'That she\'s a prying, inquisitive minx!"',
            'As soon as no more of it she could hear',
            'The lady, indignant, removed her ear.',
            '"I will not stay," she said, with a pout.,
            '"To hear my character lied about!"',
        ),
        'author' => 'Gopete Sherany',
    ),
    'EDIBLE' => array(
        'part' => 'adj.',
        'definition' => 'Good to eat, and wholesome to digest, as a worm
                           to a toad, a toad to a snake, a snake to a pig,
                           a pig to a man, and a man to a worm.',
    ),
    'EDUCATION' => array(
        'part' => 'n.',
        'definition' => 'That which discloses to the wise and disquiases
                           from the foolish their lack of understanding.',
    ),
);
?>
```

在这个例子的产品版中，这些数据可能会保存在数据库中，并基于每次请求加载相应数据。

这里，由于我们把数据直接放在了脚本中，因此取得数据的代码非常直观。首先要对通过请求发送的数据进行检查，然后再构建返回给浏览器显示的HTML片段：

```
<?php
if (isset($entries[strtoupper($_REQUEST['term'])])) {
    $entry = $entries[strtoupper($_REQUEST['term'])];

    $html = '<div class="entry">';
    $html .= '<h3 class="term">';
    $html .= strtoupper($_REQUEST['term']);
    $html .= '</h3>';
    $html .= '<div class="part">';
    $html .= $entry['part'];
    $html .= '</div>';
    $html .= '<div class="definition">';
    $html .= $entry['definition'];
    if (isset($entry['quote'])) {
        $html .= '<div class="quote">';
        foreach ($entry['quote'] as $line) {
            $html .= '<div class="quote-line">' . $line . '</div>';
        }
        if (isset($entry['author'])) {
            $html .= '<div class="quote-author">' . $entry['author'] .
                    '</div>';
        }
        $html .= '</div>';
    }
    $html .= '</div>';

    $html .= '</div>';

    print($html);
}
?>
```

这样，当我们通过调用e.php来请求这个脚本时，它就会根据GET请求的参数返回相应的HTML片段。例如，在使用e.php?term=eavesdrop请求这个脚本时，会得到如图6-8所示的HTML片段。

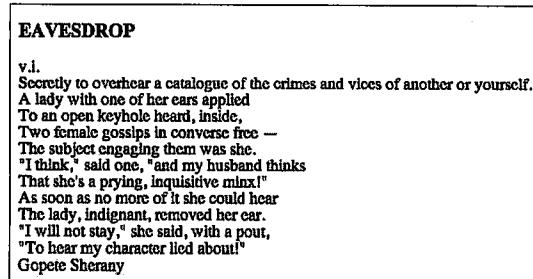


图 6-8

同样，这与我们在本章前面曾经看到过的HTML片段一样，由于还没有应用CSS规则，所以返回的HTML片段也缺少应有的样式。

由于我们要展示如何向服务器传送数据，所以这里不再借助一直沿用的独立按钮，而是使用另外一种方式请求词条——构建一个由要查询的词语组成的链接列表，通过单击其中任何一个链接，来加载相应的解释。下面是要添加到主页面中的HTML代码：

```
<div class="letter" id="letter-e">
<h3>E</h3>
<ul>
<li><a href="e.php?term=Eavesdrop">Eavesdrop</a></li>
<li><a href="e.php?term=Edible">Edible</a></li>
<li><a href="e.php?term=Education">Education</a></li>
<li><a href="e.php?term=Eloquence">Eloquence</a></li>
<li><a href="e.php?term=Elysium">Elysium</a></li>
<li><a href="e.php?term=Emancipation">Emancipation</a></li>
<li><a href="e.php?term=Emotion">Emotion</a></li>
<li><a href="e.php?term=Envelope">Envelope</a></li>
<li><a href="e.php?term=Envy">Envy</a></li>
<li><a href="e.php?term=Epitaph">Epitaph</a></li>
<li><a href="e.php?term=Evangelist">Evangelist</a></li>
</ul>
</div>
```

接下来，要通过JavaScript代码以正确的参数来调用前面的PHP脚本。虽然可以使用常规的`.load()`机制在URL后面添加查询字符串，即通过类似`e.php?term=eavesdrop`这样的地址直接取得数据。但是，在此我们想让jQuery基于我们提供给`$.get()`函数的映射来构建查询字符串：

```
$(document).ready(function() {
  $('#letter-e a').click(function() {
    $.get('e.php', {'term': $(this).text()}, function(data) {
      $('#dictionary').html(data);
    });
    return false;
  });
});
```

前面我们已经看到过jQuery提供的其他AJAX接口了，因此对这个函数的使用也应该熟悉。其中唯一的差别是第2个参数，该参数是一个用来构建查询字符串的键和值的映射。在这个例子中，键始终是`term`，而值则取自每个链接的文本。现在，单击列表中的第一个链接会导致相应词语的解释出现在页面中，如图6-9所示。

值得一提的是，列表中的链接无论有无代码使用它们都已经带有了给定的地址。这样，就为禁用了或者无法使用JavaScript的用户提供了查询相关信息的替代方法。但在正常情况下，为了防止单击这些链接时打开新URL，我们在事件处理程序中必须返回`false`。

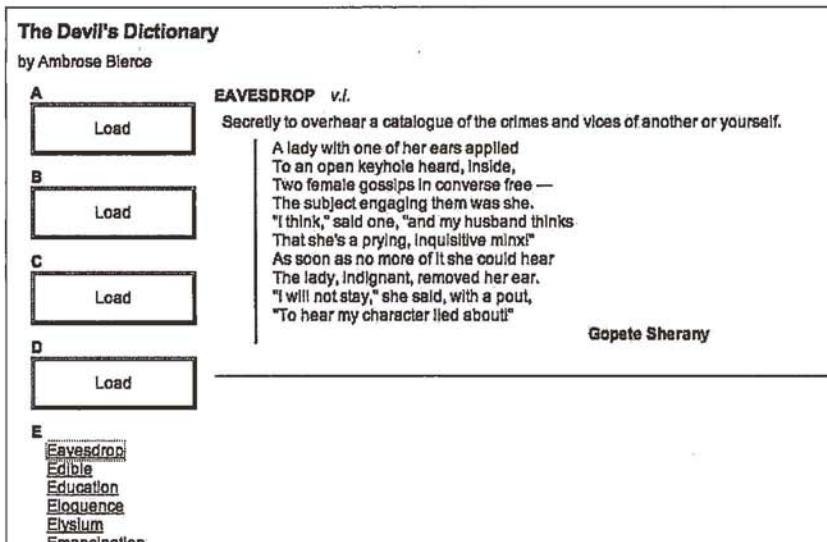


图 6-9

6.3.2 执行 POST 请求

使用POST方法与使用GET方法的HTTP请求几乎是一样的。从视觉上来看，它们之间一个最大的区别就是GET请求把参数放在作为URL一部分的查询字符串中，而POST请求则不是。但是，在AJAX请求中，即使是这种区别对一般用户而言也是不可见的。通常，决定使用哪种方法的唯一理由就是遵照服务器端代码的约定，或者要传输大量的数据——GET方法对传输的数据量有更严格的限制。由于我们编写的PHP代码能够妥善地处理任何一种方法发送的请求，因此只需改变调用的jQuery函数，就可以在GET和POST之间进行转换：

```
$(document).ready(function() {
    $('#letter-e a').click(function() {
        $.post('e.php', {'term': $(this).text()}, function(data) {
            $('#dictionary').html(data);
        });
        return false;
    });
});
```

虽然参数相同，但这里的请求是通过POST方法发送的。而通过使用.load()方法还可以进一步简化这些代码，因为.load()方法在接收到映射参数时，会默认使用POST方法发送请求：

```
$(document).ready(function() {
    $('#letter-e a').click(function() {
        $('#dictionary').load('e.php', {'term': $(this).text()});
        return false;
    });
});
```

当单击链接时，这个缩减版的函数仍然能起到相同的作用，如图6-10所示。

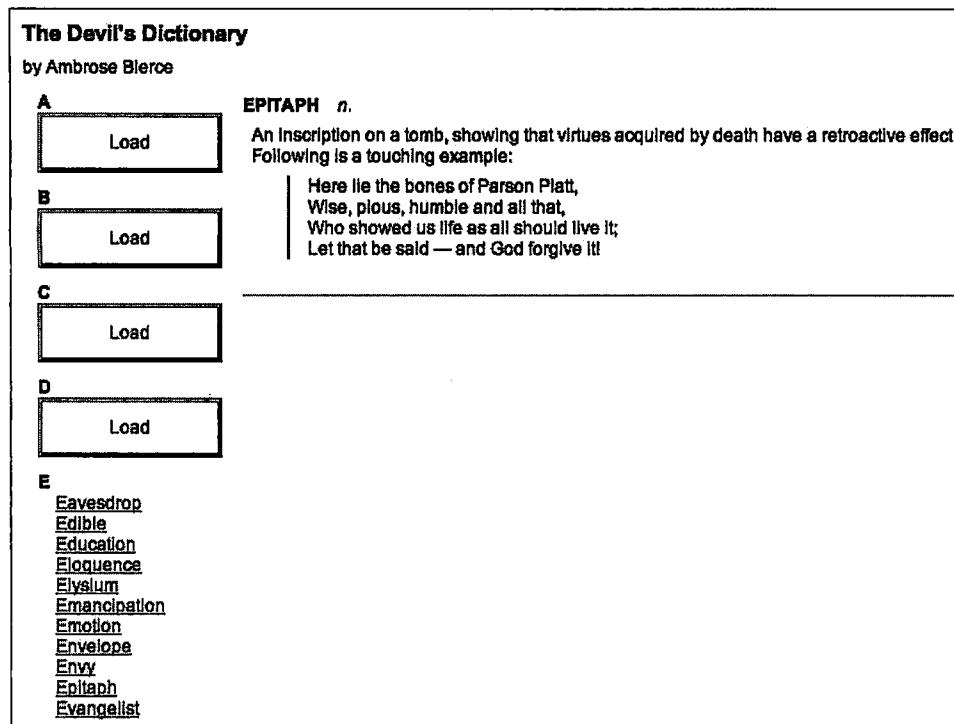


图 6-10

6.3.3 序列化表单

向服务器发送数据经常会涉及用户填写表单。常规的表单提交机制会在整个浏览器窗口中加载响应，而使用jQuery的AJAX工具箱则能够异步地提交表单，并将响应放到当前页面中。

为了试验，需要构建一个简单的表单：

```
<div class="letter" id="letter-f">
<form>
  <input type="text" name="term" value="" id="term">
  <input type="submit" name="search" value="search" id="search">
</form>
</div>
```

这一次，需要把提供的搜索词语作为字典中词条的子字符串来搜索，并从PHP脚本中返回一组词条。此时的数据结构与前面的格式相同，但逻辑稍有变化：

```
foreach ($entries as $term => $entry) {
  if (strpos($term, strtoupper($_REQUEST['term'])) !== FALSE) {
    $html = '<div class="entry">';
    $html .= '<h3 class="term">' .
```

```

$html .= $term;
$html .= '</h3>';
$html .= '<div class="part">';
$html .= $entry['part'];
$html .= '</div>';
$html .= '<div class="definition">';
$html .= $entry['definition'];
if (isset($entry['quote'])) {
    foreach ($entry['quote'] as $line) {
        $html .= '<div class="quote-line">' . $line . '</div>';
    }
    if (isset($entry['author'])) {
        $html .= '<div class="quote-author">' . $entry['author']
                . '</div>';
    }
}
$html .= '</div>';
$html .= '</div>';
print($html);
}
}

```

其中，调用的strpos()函数会扫描与提供的搜索字符串匹配的单词。接下来，我们可以通过遍历DOM树来响应表单提交并构造适当的查询字符串：

```

$(document).ready(function() {
    $('#letter-f form').submit(function() {
        $('#dictionary').load('f.php', {'term': $('input[@name="term"]').
val()});
        return false;
    });
});

```

虽然以上代码能够实现预期的效果，但通过名称属性逐个搜索输入字段并将字段的值添加到映射中总是有点麻烦。随着表单变得更复杂，这种方法也会明显变得缺乏弹性。好在，jQuery为这种常用的操作提供了一种简化方式.serialize()方法。这个方法作用于一个jQuery对象，将匹配的DOM元素转换成能够随AJAX请求传递的查询字符串。通过使用.serialize()方法，可以把前面的提交处理程序一般化为如下所示：

```

$(document).ready(function() {
    $('#letter-f form').submit(function() {
        $.get('f.php', $(this).find('input').serialize(), function(data)
        {
            $('#dictionary').html(data);
        });
        return false;
    });
});

```

这样，即使在增加表单中字段的情况下，同样的脚本仍然能够用于提交表单。如果现在进行一次搜索，页面中会显示出匹配的词条，如图6-11所示。

The screenshot shows a page from 'The Devil's Dictionary' by Ambrose Bierce. The search term 'force' has been entered in the search bar at the bottom left. The results are listed under section A:

- FORCE n.**

"Force is but might," the teacher said —
"That definition's just."
The boy said naught but thought instead,
Remembering his pounded head:
"Force is not might but must!"
- FORGETFULNESS n.**

A gift of God bestowed upon doctors in compensation for their destitution of conscience.
- D**

Load
- E**
 - Eavesdrop
 - Edible
 - Education
 - Eloquence
 - Elysium
 - Emanicipation
 - Emotion
 - Envelope
 - Envy
 - Epitaph
 - Evangelist
- F**

for

At the bottom right, there is a watermark for '大家网 TopSage.com'.

图 6-11

虽然`.serialize()`方法很方便，但它还无法完美地模拟浏览器的提交操作。特别是对于多选字段，使用这个方法只能保留一个选项。因此，在使用`.serialize()`时一定要多加注意。如果想真实地模拟浏览器常规的表单提交行为，那么可以选择jQuery的`form.js`表单插件。第10章将介绍与这个工具有关的更多信息。

6.4 关注请求

到现在为止，我们已经学习了如何调用AJAX方法，并且始终都在处理响应。然而，有时候多了解一些调用AJAX方法过程中的HTTP请求也会给我们带来方便。为满足这种需求，jQuery提供了一组函数，通过它们能够为各种与AJAX相关的事件注册回调函数。

其中，`.ajaxStart()`和`.ajaxStop()`方法就是这些“观察员”函数中的两个例子，可以把

它们添加给任何jQuery对象。当AJAX请求开始且尚未进行其他传输时，会触发`.ajaxStart()`的回调函数。相反，当最后一次活动请求终止时，则会执行通过`.ajaxStop()`注册的回调函数。所有这些观察员都是全局性的，因为无论创建它们的代码位于何处，当AJAX通信发生时都需要调用它们。

在网络连接的速度比较慢时，可以通过这些方法为用户提供一些反馈。页面中用作反馈的HTML可以包含适当的Loading...（加载中）信息，例如：

```
<div id="loading">
    Loading...
</div>
```

当然，其中也可以包含一幅动态的GIF图像作为动态指示器（throbber）。在为CSS文件中添加了样式之后，页面初始加载时的外观如图6-12所示。

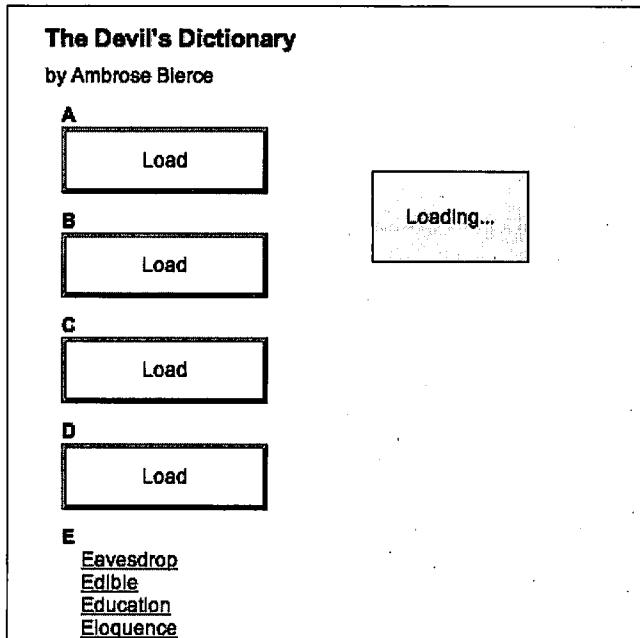


图 6-12

现在，我们添加一条`display:none`样式规则，以便在初始时隐藏这条信息。要在恰当的时机显示这条信息，只需通过`.ajaxStart()`将它注册为一个观察员即可：

```
$(document).ready(function() {
    $('#loading').ajaxStart(function() {
        $(this).show();
    });
});
```

而且，可以在此基础上继续连缀相应的隐藏行为：

```
$(document).ready(function() {
    $('#loading').ajaxStart(function() {
        $(this).show();
    }).ajaxStop(function() {
        $(this).hide();
    });
});
```

好的！我们的加载反馈系统已经就位了。

同样，我们注意到，这两个方法没有通过特别的方式与AJAX通信的开始建立联系。第1个按钮上的.load()和第2个按钮上的.getJSON()都可以导致反馈操作发生。在这种情况下，全局行为是有必要的。假如我们想要建立具体的联系，也有一些可控的选项。某些观察员方法，如.ajaxError()，会向它们的回调函数发送一个对XMLHttpRequest对象的引用。这样就可以做到区别不同的请求来提供不同的行为。其他更具体的处理可以通过使用低级的\$.ajax()函数来完成。前面我们讨论的所有AJAX函数都会在内部调用\$.ajax()。这个函数提供了大量的选项，其中有几个是与AJAX请求相关的具体事件的处理程序。

不过，与请求最常见的交互方式（我们已经介绍过了），是成功（success）回调函数。在前面的几个例子中，我们就是使用这个回调函数来解析从服务器返回的数据，然后将结果填充到页面上。当然，也可以使用其他回调函数。现在，仍然以.load()方法为例：

```
$(document).ready(function() {
    $('#letter-a .button').click(function() {
        $('#dictionary').load('a.html');
    });
});
```

此时，通过使加载的内容淡入视图而不是突然出现，可以从视觉上加入一些增强的效果。.load()方法可以接受一个加载完成时触发的回调函数：

```
$(document).ready(function() {
    $('#letter-a .button').click(function() {
        $('#dictionary').hide().load('a.html', function() {
            $(this).fadeIn();
        });
    });
});
```

以上代码首先隐藏了目标元素，然后开始加载。当加载完成时，又通过回调函数以淡入方式逐渐显示出新生成的元素。

6.5 AJAX 和事件

假设我们想在单击页面上任何一个

元素时，都能够突出地显示被单击的文本。而执行这一任务的代码对我们来说似乎已经可以信手拈来了：

```
$(document).ready(function() {
    $('h3').click(function() {
        $(this).toggleClass('highlighted');
    });
});
```

一切正常，当单击页面左侧的字母时，字母能够突出显示。但是，字典中的词条也是

元素，却不能通过单击突出显示。为什么？

由于字典中的词条在页面加载时还不是DOM的一部分，因此不会为它们绑定事件处理程序。这是使用事件处理程序与AJAX请求时的一个常见问题：加载的元素必须在适当的时机单独绑定自己的事件处理程序。

解决这个问题的第一步是把绑定过程提取到一个函数中，在文档就绪和AJAX请求之后都调用该函数：

```
$(document).ready(function() {
    var bindBehaviors = function() {
        $('h3').click(function() {
            $(this).toggleClass('highlighted');
        });
    };

    bindBehaviors();

    $('#letter-a .button').click(function() {
        $('#dictionary').hide().load('a.html', function() {
            bindBehaviors();
            $(this).fadeIn();
        });
    });
});
```

现在，当把所有事件处理程序都放在bindBehaviors()函数中之后，就可以在DOM变化时调用这个函数了。而且，现在单击一个字典词语也会看到突出显示的结果。然而，这样一来我们也促成了一种非常奇怪的行为，即当单击字母时，一开始字母能够正确地突出显示，但当单击了按钮（加载字典中的词条）之后，再单击字母，字母就不能突出显示了。

经过仔细地分析，我们发现在AJAX请求之后，之所以字母的突出显示会中断，是因为单击字母会触发两次相同的单击处理程序——执行两次.toggleClass()相当于什么也没有执行。导致这个问题的罪魁祸首是bindBehaviors()，该函数每次都会为所有

绑定单击事件。也就

是说，当单击按钮后，再单击

实际上会执行两个事件处理程序，而且这两个事件处理程序执行的是相同的操作。

6.5.1 限定事件绑定函数的作用域

解决这种“双触发”问题的一种有效方式，是每次调用bindBehaviors()时，都给它传递一个环境变量。\$()函数可以接受第2个参数，即用于限制搜索范围的DOM节点。通过在bindBehaviors()中使用这一特性，就可以避免多事件绑定的问题：

```
$document.ready(function() {
    var bindBehaviors = function(scope) {
        $('h3', scope).click(function() {
            $(this).toggleClass('highlighted');
        });
    };

    bindBehaviors(this);

    $('#letter-a .button').click(function() {
        $('#dictionary').hide().load('a.html', function() {
            bindBehaviors(this);
            $(this).fadeIn();
        });
    });
});
});
```

当第1次调用bindBehaviors()时，作用域是document，因此，文档中所有的

元素都会匹配并绑定单击事件。在AJAX请求之后，作用域换成了 元素，由于字母不再匹配，所以不会再重复绑定单击事件。

6.5.2 利用事件冒泡

向行为绑定函数中添加作用域，通常都是针对在AJAX请求之后绑定事件处理程序所带来问题的一种非常优雅的解决方案。然而，如果能够适当地利用事件冒泡，则能够完全避免此类问题。换句话说，我们不必将处理程序绑定到加载的元素上，而是绑定到一个它们公共的祖先元素上：

```
$document.ready(function() {
    $('body').click(function(event) {
        if ($(event.target).is('h3')) {
            $(event.target).toggleClass('highlighted');
        }
    });
});
```

这里，我们把单击事件处理程序绑定到了元素上。因为这个元素在生成AJAX请求时不会随着改变，因此就不会重复绑定事件处理程序。但是这样一来，事件的环境就不对了，因此

还需要通过检查事件的`target`属性来补偿这一点。也就是说，只有在事件目标的类型正确时，才会执行正常的操作；否则，什么也不做。

6.6 安全限制

尽管构建动态的Web应用程序非常实用，但`XMLHttpRequest`（jQuery的AJAX实现背后的底层浏览器技术）常常会受到严格限制。为了防止各种跨站点脚本攻击，一般情况下从提供原始页面的服务器之外的站点请求文档是不可能的。

这通常都是一种积极的情形。例如，有人指出通过`eval()`来解析JSON是不安全的。如果数据文件中存在恶意代码，那么通过`eval()`调用就会执行这些恶意代码。不过，因为数据文件肯定会与网页保存在相同的服务器上，如果能够向数据文件中注入代码，那么很大程度上就可以认为能够直接向网页中注入代码。也就是说，对于加载信得过的JSON文件而言，`eval()`并没有明显的安全性问题。

但是，从第三方来源中加载数据往往是很必要的。因而，也有许多方式可以绕过上述安全限制，即能够实现通过AJAX请求取得其他站点的数据。

其中一种方法是通过服务器加载远程数据，然后在客户请求时提供给浏览器。这是一种非常有效的手段，因为服务器能够对数据进行预处理。例如，可以从几个来源加载包含RSS新闻的XML文件，然后在服务器上将这些XML文件聚合到一个源文件中，当请求发生时再将这个新文件发布给客户。

如果想不通过服务器的参与加载远程地址中的数据，那我们就必须狡猾一些。例如，加载外来JavaScript文件的一种流行方法是根据请求注入`<script>`标签。由于jQuery能帮我们插入新的DOM元素，因此向文档中注入`<script>`标签非常简单：

```
$(document.createElement('script'))
    .attr('src', 'http://example.com/example.js')
    .appendTo('head');
```

此时，浏览器会执行加载的脚本，但却没有任何机制能够从脚本中取得结果。为此，使用这种技术要求同远程主机进行协作。加载的脚本必须执行某些操作，例如设置一个对本地环境有影响的全局变量。而远程主机上的服务除了发布能够通过这种方式执行的脚本外，还要提供一个API以便于同远程脚本进行交互^①。

另一种方法是使用`<iframe>`这个HTML标签来加载远程数据。可以为`<iframe>`元素指定任

① 由于在动态注入的`<script>`标签中，脚本可以来源于任何一个域（`src`属性可以指向任何第三方站点），也就意味着可以通过该脚本中的`XMLHttpRequest`对象取得任何其他域中的信息，因而就绕过了“同源策略”的安全限制。Google Map的Google Maps API (<http://google.com/apis/maps>) 就采用了这种动态生成`<script>`标签的技术。

——译者注

何URL作为其获取数据的来源，包括与提供页面的服务器不匹配的URL。因此，第三方服务器上的数据能够轻易地加载到`<iframe>`中，并在当前页面上显示出来。然而，要操作`<iframe>`中的数据，仍然存在同使用`<script>`标签时一样的协作需求；位于`<iframe>`中的脚本需要明确地向父文档中的对象提供数据。

6.7 小结

本章中，我们学习了使用jQuery提供的AJAX方法，在不刷新页面的情况下，从服务器上加载几种不同格式的数据。而且，我们也可以基于请求执行来自服务器的脚本，并且能够向服务器发送数据。

同时，我们还学习了如何处理常见的异步加载技术的挑战，例如在加载发生后绑定处理程序，以及从第三方服务器中加载数据。

这一章是本书教程部分的最后一章。到目前为止，我们已经学习了jQuery提供的主要工具：选择符、事件、效果、DOM操作和异步服务器请求。后面的参考资料部分将提供包含以上几类方法详细介绍的各种资源。在本书的第三部分中，我们将探讨一些综合运用以上技术的方式，并以新颖而有趣的形式来增强我们的网页。



表 格 操 作

*Let 'em wear gaudy colors
Or avoid display*

—Devo,
“Wiggly World”

在前6章中，我们通过一系列教程探索了jQuery库，借助众多的例子展示了jQuery中每个组件的实际应用。在第7章到第9章，我们要把这一过程颠倒过来，即以例子为出发点，来探讨如何使用jQuery提供的方法实现这些例子。

在这一部分中，我们将以一个在线书店的网站作为模型，但所涉及的技术同样广泛适用于各类网站，从博客到投资理财，从面向市场的商业站点到公司内部网等等。第7章和第8章主要介绍大多数网站中都会用到的两个元素——表格和表单，第9章主要讨论从视觉上增强信息集合的两种方式——滑移和翻转效果。

本章中，我们讨论使用jQuery来增强表格的可读性、可用性以及视觉冲击力，但是，我们不会介绍如何使用表格进行页面布局或设计。事实上，随着近年来Web标准运动越来越深入人心，基于表格的页面布局已经逐步被基于CSS的设计所取代。虽然表格在1990年代是用来创建多栏或其他复杂布局的常用的而且是必要的权宜之计，但设计表格的目的却完全不是为了进行页面布局。另一方面，CSS则是专门为表现而创建的一种技术。

不过，我们并不想在这里展开讨论表格的适当角色是什么。本章真正要讨论的主题是如何显示并与作为表格式数据（tabular data）语义容器的表格进行交互。要更深入地了解如何为表格应用具有语义的、可访问性的HTML标记，Roger Johansson的博客文章“Bring on the Tables”（http://www.456bereastreet.com/archive/200410/bring_on_the_tables/）是个不错的起点。

本章中为表格应用的一些技术也可以在Christian Bach和Table Sorter插件中找到。要了解与插件有关的更多信息，请访问“jQuery Plug-in Repository”（<http://plugins.jquery.com/>）。

7.1 排序

操作表格式数据的一种最常见任务就是排序。在一个大型的表格中，能够对要寻找的信息进行重新排列是非常重要的。然而，这个有用的操作实现起来也是最有技巧性的。用来完成排序的手段有两种，一种是服务器端排序，另一种是JavaScript排序。

7.1.1 服务器端排序

在服务器端进行数据排序是一种常见的解决方案。由于表格中的数据通常来自于数据库，因而从数据库中提取数据的代码可以按照给定的排序次序请求数据（例如，使用SQL语言中的ORDER BY子句）。如果我们能够控制服务器端代码，那么一开始就使用一种合理的默认排序次序非常简单。

排序的作用体现在用户能够决定排序次序。为此，一种常用的实现方式就是将可以排序的列标题转换为链接。这些链接仍然指向当前页面，但链接后面会添加一个表示排序依据的查询字符串：

```
<table id="my-data">
  <tr>
    <th class="name"><a href="index.php?sort=name">Name</a></th>
    <th class="date"><a href="index.php?sort=date">Date</a></th>
  </tr>
  ...
</table>
```

服务器可以根据查询字符串中的参数，以特定的顺序返回数据库中的内容。

避免刷新页面

这个方案虽然简单，但每次排序操作都必须刷新页面。如前所见，通过jQuery提供的AJAX方法，可以避免在这种情况下刷新页面。在前面将列标题设置为链接的基础上，我们可以通过添加jQuery代码将这些链接都转换为AJAX请求：

```
$(document).ready(function() {
  $('#my-data .name a').click(function() {
    $('#my-data').load('index.php?sort=name&type=ajax');
    return false;
  });
  $('#my-data .date a').click(function() {
    $('#my-data').load('index.php?sort=date&type=ajax');
    return false;
  });
});
```

现在，当单击这些锚时，jQuery会为同一个页面向服务器发送一次AJAX请求。而且，我们在查询字符串中也添加了另外一个参数，以便服务器能够确定该请求是AJAX请求。这样，就可

以据此来编写服务器端程序，当这个参数存在时只返回表格，而不是包含表格的整个页面。于是，在得到响应后，客户端代码也可以直接将响应插入到表格的位置上。

这是一个渐进增强的例子。也就是说，即使没有JavaScript，这个页面的功能也完好无缺，因为通过服务器端程序进行排序的链接仍然有效。不过，当JavaScript存在时，AJAX就会拦截页面请求并在不必加载整个页面的前提下完成排序。

7.1.2 JavaScript 排序

然而，有时候我们既不想在排序时等待服务器响应，又无权控制服务器端脚本。那么，这种情况下的一种可行的替代方案，就是在浏览器中完全使用JavaScript客户端脚本进行排序。

假设我们有一个表格，其中列出了很多书以及书的作者、出版日期和价格：

```
<table class="sortable">
    <thead>
        <tr>
            <th></th>
            <th>Title</th>
            <th>Author(s)</th>
            <th>Publish&nbsp;Date</th>
            <th>Price</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>
                
            </td>
            <td>Building Websites with Joomla! 1.5 Beta 1</td>
            <td>Hagen Graf</td>
            <td>Feb 2007</td>
            <td>$40.49</td>
        </tr>
        <tr>
            <td></td>
            <td>Learning Mambo: A Step-by-Step Tutorial to Building Your
                 Website</td>
            <td>Douglas Paterson</td>
            <td>Dec 2006</td>
            <td>$40.49</td>
        </tr>
        ...
    </tbody>
</table>
```

```
</tbody>
</table>
```

我们的目标是把这个表格的列标题转换成按钮，以便按照相应的列进行排序。下面我们就来尝试几种实现这一功能的方式。

1. 行组标签

在上面表格的标记中，我们使用了`<thead>`和`<tbody>`标签把数据分割为行组。很多HTML设计者都忽略了这两个标签的作用，其实，添加这两个标签有助于我们更方便地使用CSS选择符。例如，在想为这个表格添加典型的偶数/奇数行条纹效果时，如果想把效果范围限定在表格的主体内，可以编写如下代码：

```
$(document).ready(function() {
    $('table.sortable tbody tr:odd').addClass('odd');
    $('table.sortable tbody tr:even').addClass('even');
});
```

这样，就会为除了标题行之外的表格行应用交替的颜色，如图7-1所示。

	Title	Author(s)	Publish Date	Price
	Building Websites with Joomla! 1.5 Beta 1	Hagen Graf	Feb 2007	\$40.49
	Learning Mambo: A Step-by-Step Tutorial to Building Your Website	Douglas Paterson	Dec 2006	\$40.49
	Moodle E-Learning Course Development	William Rice	May 2006	\$35.99
	AJAX and PHP: Building Responsive Web Applications	Cristian Darie, Mihai Bucica, Filip Chereches-Tosa, Bogdan Brinzarea	Mar 2006	\$31.49

图 7-1

2. 基本的按字母排序

接下来，我们实现基于表格的Title列进行排序。首先，需要为标题行的单元格应用一个类，以便于选择：

```
<thead>
<tr>
    <th></th>
    <th class="sort-alpha">Title</th>
    <th>Author(s)</th>
    <th>Publish Date</th>
    <th>Price</th>
```

```
</tr>
</thead>
```

在实际进行排序时，可以使用JavaScript内置的`.sort()`方法。该方法能够对数组进行就地排序，而且可以接受一个排序函数作为参数。作为参数的排序函数要比较数组中的两个项，并且根据比较的结果返回一个正数或负数。因此，初始的排序代码如下所示：

```
$(document).ready(function() {
    $('table.sortable').each(function() {
        var $table = $(this);
        $('th', $table).each(function(column) {
            if ($(this).is('.sort-alpha')) {
                $(this).addClass('clickable').hover(function() {
                    $(this).addClass('hover');
                }, function() {
                    $(this).removeClass('hover');
                }).click(function() {
                    var rows = $table.find('tbody > tr').get();
                    rows.sort(function(a, b) {
                        var keyA = $(a).children('td').eq(column).text()
                            .toUpperCase();
                        var keyB = $(b).children('td').eq(column).text()
                            .toUpperCase();
                        if (keyA < keyB) return -1;
                        if (keyA > keyB) return 1;
                        return 0;
                    });
                    $.each(rows, function(index, row) {
                        $table.children('tbody').append(row);
                    });
                });
            }
        });
    });
});
```

这里要注意的第一件事，就是我们使用`.each()`方法进行了显式迭代，而没有直接使用`$('table.sortable th.sort-alpha').click()`选择并为每个带`sort-alpha`类的标题单元绑定单击事件处理程序。由于`.each()`方法会向它的回调函数中传递迭代索引，所以我们能够用它方便地捕获到一个至关重要的信息——单击标题的列索引，进而就能在后面使用这个列索引来找到每个数据行中的相关单元格。

在找到带`sort-alpha`类的标题单元之后，接下来我们取得了一个包含所有数据行的数组。这是一个通过`.get()`方法将jQuery对象转换为一个DOM节点数组的极好范例。之所以要进行这一转换，是因为虽然jQuery对象在很多方面都与数组类似，但它却不具有任何本地的数组方法，比如`.sort()`。

在能够调用`.sort()`方法后，剩下的代码就很简单了。总而言之，就是通过比较相关表格单元中的文本，实现对表格行的排序。这里，我们之所以知道要比较哪个单元格，是因为在`.each()`方法内部能够捕获到列的索引。而之所以要把单元格中的文本转换为大写，是因为JavaScript中的字符串比较是区分大小写的，而我们希望排序结果不分区分大小写。最后，通过循环遍历排序后的数组，将表格行重新插入到表格中。注意，因为`.append()`方法不会复制节点，因此该方法会移动表格行而不是复制表格行。于是，我们就能看到按字母排序的表格。

以上是渐进增强的另一面——平稳退化的一个例子。与我们前面讨论的AJAX解决方案不同，这个例子中的排序功能在没有JavaScript的情况下是无法使用的，因为我们假设服务器上没有处理这种情况的脚本程序。由于排序功能有赖于JavaScript，因此只通过JavaScript代码添加“clickable”类，可以确保在缺少脚本而无法使用排序功能时，不会显示可以排序的界面。页面中的表格也在没有JavaScript的情况下平稳退化，虽然无法进行排序，但仍然具有基本的功能。

由于排序过程中移动了表格行，因而导致原来交替显示的行颜色发生了混乱，如图7-2所示。

	Title	Author(s)	Publish Date	Price
	Advanced Microsoft Content Management Server Development	Angus Logan, Stefan Giebler, Lim Mei Ying, Andrew Connell	Nov 2005	\$53.99
	AJAX and PHP: Building Responsive Web Applications	Cristian Dane, Mihai Bucica, Filip Cherecheș-Toşa, Bogdan Brinzărescu	Mar 2006	\$31.49
	Alfresco Enterprise Content Management Implementation	Munwar Sharif	Jan 2007	\$53.99
	BPEL Cookbook: Best Practices for SOA-based Integration and composite applications development	Jerry Thomas, Doug Todd, Harish Gaur, Lawrence Pravin, Arun Podivil, The Hoa Nguyen, Yves Coene, Jeremy Bolle, Stany Blanvalet, Markus Zirm, Matjaz Juric, Sean Carey, Michael Cardella, Kevin Geminiani, Praveen Ramachandran	Jul 2006	\$40.49

图 7-2

这说明，当完成排序后还需要重新为表格行应用颜色。为此，我们可以把添加颜色的代码提取到一个函数中，以便需要时调用：

```
$ (document).ready(function() {
    var alternateRowColors = function($table) {
        $('tbody tr:odd', $table).removeClass('even').addClass('odd');
        $('tbody tr:even', $table).removeClass('odd').addClass('even');
    };

    $('table.sortable').each(function() {
        var $table = $(this);
        alternateRowColors($table);
        $('th', $table).each(function(column) {
            if ($(this).is('.sort-alpha')) {
```

```
$(this).addClass('clickable').hover(function() {
    $(this).addClass('hover');
}, function() {
    $(this).removeClass('hover');
}).click(function() {
    var rows = $table.find('tbody > tr').get();
    rows.sort(function(a, b) {
        var keyA = $(a).children('td').eq(column).text()
                    .toUpperCase();
        var keyB = $(b).children('td').eq(column).text()
                    .toUpperCase();
        if (keyA < keyB) return -1;
        if (keyA > keyB) return 1;
        return 0;
    });
    $.each(rows, function(index, row) {
        $table.children('tbody').append(row);
    });
    alternateRowColors($table);
});
});
});
});
```

这样，就在排序之后恢复了表格行的颜色，从而解决了我们的问题，如图7-3所示。

Title	Author(s)	Publish Date	Price
Advanced Microsoft Content Management Server Development	Angus Logan, Stefan Goßner, Lim Mei Ying, Andrew Connell	Nov 2005	\$53.99
AJAX and PHP: Building Responsive Web Applications	Cristian Darie, Mihai Budica, Filip Chereches-Tosca, Bogdan Brinzares	Mar 2006	\$31.49
Alfresco Enterprise Content Management Implementation	Munwar Sheriff	Jan 2007	\$53.99
SPEL Cookbook: Best Practices for SOA-based Integration and composite applications development	Jerry Thomas, Doug Todd, Harish Gaur, Lawrence Pravin, Arun Poduval, The Hoa Nguyen, Yves Coene, Jeremy Boile, Sany Bhamvalek, Markus Zim, Matjaz Juric, Sean Carey, Michael Cardella, Kevin Gemmeluk, Praveen Ramachandran	Jul 2006	\$40.49

图 7-3

3. 创建插件

前面我们编写的alternateRowColors()函数，完全有可能创建为一个jQuery插件。事实上，任何应用到一组DOM元素上的操作都可以简单地通过插件的形式来表示。在这个例子中，要创建一个插件只需对现有的函数稍作修改：

```
jQuery.fn.alternateRowColors = function() {
    $('tbody tr:odd', this).removeClass('even').addClass('odd');
    $('tbody tr:even', this).removeClass('odd').addClass('even');
    return this;
};
```

重要的修改有如下3处。

- 首先，此时的函数是作为jQuery.fn的一个新属性定义的，而非一个孤立的函数。这样就把该函数注册成了一个插件方法。
- 其次，使用关键字this取代了\$table参数。在一个插件方法内部，this引用的是调用该方法的jQuery对象。
- 最后，在函数的末尾返回了this。返回这个值可以使这个新方法能够再连缀其他方法。

有关编写jQuery插件的更多内容，请参见第10章。届时，我们将讨论如何编写、面向公众功能完备的插件，而不是像这个小例子中一样，编写只供我们自己使用的插件。

在定义了新的插件后，就可以这样来调用该插件：\$table.alternateRowColors()。显然，与alternateRowColors(\$table)相比，这是一种更自然的jQuery语法。

4. 性能问题

前面例子中的代码虽然能够正常运行，但速度却很慢。速度慢的原因在于比较器函数，该函数的工作量非常大。具体来说，在排序期间需要多次调用比较器函数，而这意味着该函数消耗的每一点额外的处理时间都将被放大。

JavaScript实际使用的排序算法在标准中没有定义。因此，有可能是一种简单的冒泡排序（在复杂计算情况下的最坏运行时间为 $\Theta(n^2)$ ），或者一种更完善的排序，如快速排序（平均运行时间为 $\Theta(n \log n)$ ）。但无论使用哪种排序算法，如果排序的项增加一倍，那么调用比较器函数的次数则不仅仅是增加一倍。

解决低效率比较器的方法是预先计算用于比较的关键字。下面，我们就从低效率的排序函数开始：

```
rows.sort(function(a, b) {
    keyA = $(a).children('td').eq(column).text().toUpperCase();
    keyB = $(b).children('td').eq(column).text().toUpperCase();
    if (keyA < keyB) return -1;
    if (keyA > keyB) return 1;
    return 0;
});
$.each(rows, function(index, row) {
    $table.children('tbody').append(row);
});
```

可以从中提取出对关键字的计算，并将这一过程交给另一个循环来完成：

```
$.each(rows, function(index, row) {
    row.sortKey = $(row).children('td').eq(column).text().toUpperCase();
});
rows.sort(function(a, b) {
    if (a.sortKey < b.sortKey) return -1;
    if (a.sortKey > b.sortKey) return 1;
    return 0;
});
$.each(rows, function(index, row) {
    $table.children('tbody').append(row);
    row.sortKey = null;
});
```

在新的循环中，我们完成了所有占用资源的工作，并把结果保存在一个新属性中。像这种添加给一个DOM元素，但又不是常规的DOM属性的属性，称为expando。由于每个表格行都需要这样一个用于比较的关键字，因此把它们保存到expando属性中非常合适。然后，比较器函数就可以直接比较这个属性，从而排序的速度也因此明显加快了。

提示 这里，我们在用完expando属性之后将其设置为null，以便手动释放内存。虽然这不是必需的，但却是一个良好的习惯。否则，把这些expando属性留在内存里可能会导致内存泄漏。要了解有关内存泄漏的更多信息，请参考附录C。

5. 巧妙处理排序关键字

现在，我们想把同样的排序行为应用到表格中的Author(s)列中。通过向相应的标题单元格中添加sort-alpha类，现有的代码就能实现按Author(s)列进行排序。但是，理想的情况下，应该是按照作者的姓（last name），而不是名字（first name）进行排序。由于某些书有多位作者，而且有的作者有中名（middle name），或者使用缩写的名字，因此，需要通过额外的标识来确定以文本中的哪一部分作为排序关键字。为此，可以通过将单元格中的部分文本包装在一个标签中来提供这种标识：

```
<tr>
<td>
    </td>
    <td>Building Websites with Joomla! 1.5 Beta 1</td>
    <td>Hagen <span class="sort-key">Graf</span></td>
    <td>Feb 2007</td>
    <td>$40.49</td>
</tr>
<tr>
<td>
    </td>
```

```

<td>
    Learning Mambo: A Step-by-Step Tutorial to Building Your Website
</td>
<td>Douglas <span class="sort-key">Paterson</span></td>
<td>Dec 2006</td>
<td>$40.49</td>
</tr>
<tr>
    <td>
        </td>
        Moodle E-Learning Course Development
    <td>William <span class="sort-key">Rice</span></td>
    <td>May 2006</td>
    <td>$35.99</td>
</tr>

```

这样，我们必须修改排序代码，将这个作为标识的标签考虑在内，同时还要确保不会干扰Title列的排序行为（该列的排序很正常）。通过把标签中的排序关键字放到以前计算的关键字前头，可以实现以按姓排序为主，以按单元格中的整个字符串排序为辅的操作：

```

$.each(rows, function(index, row) {
    var $cell = $(row).children('td').eq(column);
    row.sortKey = $cell.find('.sort-key').text().toUpperCase()
        + ' ' + $cell.text().toUpperCase();
});

```

现在按照Author(s)列排序使用的排序依据是姓，如图7-4所示。

Title	Author(s)	Publish Date	Price
 Programming Windows Workflow Foundation: Practical WF Techniques and Examples using XAML and C#	K. Scott Allen	Dec 2006	\$40.49
 Building Websites with XOOPS :A step-by-step tutorial	Steve Atwell	Oct 2006	\$26.99
 Learn OpenOffice.org Spreadsheet Macro Programming: OOoBasic and Calc automation	Dr. Mark Alexander Bain	Dec 2006	\$35.99
 UML 2.0 in Action: A project-based tutorial	Philippe Baumann, Henrike Baumann, Patrick Grassie	Sep 2005	\$31.49

图 7-4

假如有两个作者的姓相同，排序操作会再以整个字符串作为排序依据。

6. 其他类型的数据排序

我们的排序例程应该不仅能够处理Title列和Author列，而且也应该能够处理Publish Dates列

和Price列。由于已经改进了比较器函数，它能够处理各种数据，但是对于其他数据类型，则需要调整计算的排序关键字。例如，需要去掉价格中前导的\$字符，然后解析出剩余的数字，最后再进行比较：

```
var key = parseFloat($cell.text().replace(/^[\^d.]*/,''));
row.sortKey = isNaN(key) ? 0 : key;
```

要对parseFloat()返回的结果进行检查，是因为如果不能从文本中解析出数字（即返回NaN），那么将会对.sort()函数造成严重破坏。对于日期单元格而言，我们可以使用JavaScript的Date对象：

```
row.sortKey = Date.parse('1 ' + $cell.text());
```

表格中的日期只包含月和年，但Date.parse()方法需要一个完整的日期，因此我们前置了一个1，以便补足月和年前面的日。最后，组合的日期将被转换为时间戳，时间戳可以使用正常的比较器进行排序。

最后，我们将以上表达式分别放在不同的函数里，再基于应用到表格标题的类调用适当的函数：

```
$fn.alternateRowColors = function() {
    $('tbody tr:odd', this).removeClass('even').addClass('odd');
    $('tbody tr:even', this).removeClass('odd').addClass('even');
    return this;
};

$(document).ready(function() {
    var alternateRowColors = function($table) {
        $('tbody tr:odd', $table).removeClass('even').addClass('odd');
        $('tbody tr:even', $table).removeClass('odd').addClass('even');
    };

    $('table.sortable').each(function() {
        var $table = $(this);
        $table.alternateRowColors($table);
        $('th', $table).each(function(column) {
            var findSortKey;
            if ($(this).is('.sort-alpha')) {
                findSortKey = function($cell) {
                    return $cell.find('.sort-key').text().toUpperCase()
                        + ' ' + $cell.text().toUpperCase();
                };
            }
            else if ($(this).is('.sort-numeric')) {
                findSortKey = function($cell) {
                    var key = parseFloat($cell.text().replace(/^[\^d.]*/,''));
                    return isNaN(key) ? 0 : key;
                };
            }
            else if ($(this).is('.sort-date')) {
                findSortKey = function($cell) {
```

```
        return Date.parse('1 ' + $cell.text());
    };
}

if (findSortKey) {
    $(this).addClass('clickable').hover(function() {
        $(this).addClass('hover');
    }, function() {
        $(this).removeClass('hover');
    }).click(function() {
        var rows = $table.find('tbody > tr').get();
        $.each(rows, function(index, row) {
            row.sortKey =
                findSortKey($(row).children('td').eq(column));
        });
        rows.sort(function(a, b) {
            if (a.sortKey < b.sortKey) return -1;
            if (a.sortKey > b.sortKey) return 1;
            return 0;
        });
        $.each(rows, function(index, row) {
            $table.children('tbody').append(row);
            row.sortKey = null;
        });
        $table.alternateRowColors($table);
    });
}
});
```

其中，变量`findSortKey`既是一个计算排序关键字的函数，也是表示列标题是否已经通过某个类标记为可排序的标志。这样，我们就可以按照日期和价格进行排序了，如图7-5所示。

	Title	Author(s)	Publish Date	Price
	User Training for Busy Programmers	William Rice	Jun 2005	\$11.99
	Pluggable Authentication Modules: The Definitive Guide to PAM for Linux SysAdmins and C Developers	Kenneth Gesslert	Jan 2007	\$17.99
	Creating your MySQL Database: Practical Design Tips and Techniques	Marc Dellisie	Nov 2006	\$17.99
	The Microsoft Outlook Ideas Book	Barbara March	Mar 2006	\$22.49

图 7-5

7. 突出显示列

良好的用户界面增强应该从视觉上对过去发生了什么向用户给出提示。通过突出显示最近用于排序的列，可以把用户的注意力吸引到很可能包含相关信息的表格部分上。好在，我们已经知道了如何选择列中的表格单元，因为这些单元格应用类只是小事一桩：

```
$table.find('td').removeClass('sorted')
.filter(':nth-child(' + (column + 1) + ')').addClass('sorted');
```

我们注意到，由于`:nth-child()`选择符从1开始而不是从0开始，所以必须要给列索引加上1。添加了这些代码后，任何排序操作都可以使相应的列突出显示，如图7-6所示。

	Title	Author(s)	Publish Date	Price
	Building Websites with the ASP.NET Community Starter Kit	Cristian Darie, K. Scott Allen	May 2004	\$40.49
	Building Websites with Plone	Cameron Cooper	Nov 2004	\$44.99
	Windows Server 2003 Active Directory Design and Implementation: Creating, Migrating, and Merging Networks	John Savill	Jan 2005	\$53.99
	SSL VPN: Understanding, evaluating and planning secure, web-based remote access	Tim Speed, Joseph Steinberg	Mar 2005	\$44.99

图 7-6

8. 交替排序方向

与排序有关的最后一项增强，是实现既能够按升序排序也能够按降序排序。换句话说，当用户单击一个已经排序的表格列时，应该反转当前的排序次序。

要反转当前的排序，我们所要做的就是逆转由比较器返回的值。为此，只需要使用一个简单的变量即可：

```
if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
```

如果`newDirection`等于1，那么排序结果同以前一样。如果它等于-1，则排序方向会反转。我们可以使用类来标识某一列当前的排序次序：

```
$.fn.alternateRowColors = function() {
  $('tbody tr:odd', this).removeClass('even').addClass('odd');
  $('tbody tr:even', this).removeClass('odd').addClass('even');
  return this;
};
```

```

$(document).ready(function() {
    var alternateRowColors = function($table) {
        $('tbody tr:odd', $table).removeClass('even').addClass('odd');
        $('tbody tr:even', $table).removeClass('odd').addClass('even');
    };
    $('table.sortable').each(function() {
        var $table = $(this);
        $table.alternateRowColors($table);
        $('th', $table).each(function(column) {
            var findSortKey;
            if ($(this).is('.sort-alpha')) {
                findSortKey = function($cell) {
                    return $cell.find('.sort-key').text().toUpperCase() + ' ' +
                           $cell.text().toUpperCase();
                };
            }
            else if ($(this).is('.sort-numeric')) {
                findSortKey = function($cell) {
                    var key = parseFloat($cell.text().replace(/^\^[\^d.]*/, ''));
                    return isNaN(key) ? 0 : key;
                };
            }
            else if ($(this).is('.sort-date')) {
                findSortKey = function($cell) {
                    return Date.parse('1 ' + $cell.text());
                };
            }
            if (findSortKey) {
                $(this).addClass('clickable').hover(function() {
                    $(this).addClass('hover');
                }, function() {
                    $(this).removeClass('hover');
                }).click(function() {
                    var newDirection = 1;
                    if ($(this).is('.sorted-asc')) {
                        newDirection = -1;
                    }
                    var rows = $table.find('tbody > tr').get();

                    $.each(rows, function(index, row) {
                        row.sortKey =
                            findSortKey($(row).children('td').eq(column));
                    });
                    rows.sort(function(a, b) {
                        if (a.sortKey < b.sortKey) return -newDirection;
                        if (a.sortKey > b.sortKey) return newDirection;
                        return 0;
                    });
                });
            }
        });
    });
});

```

```
        });
        $.each(rows, function(index, row) {
            $table.children('tbody').append(row);
            row.sortKey = null;
        });
        $table.find('th').removeClass('sorted-asc')
                    .removeClass('sorted-desc');
        var $sortHead = $table.find('th').filter(
                        ':nth-child(' + (column + 1) + ')');
        if (newDirection == 1) {
            $sortHead.addClass('sorted-asc');
        } else {
            $sortHead.addClass('sorted-desc');
        }
        $table.find('td').removeClass('sorted')
                        .filter(':nth-child(' + (column + 1) + ')')
                            .addClass('sorted');
        $table.alternateRowColors($table);
    });
});
```



由于我们使用了类来保存排序方向，因而也可以利用这个类来为列标题添加样式，以便表明当前的排序状态，如图7-7所示。

Title	Author(s)	Publish Date	Price
 Enhancing Microsoft Content Management Server with ASP.NET 2.0	Lim Mei Ying, Spencer Harbar, Stefan Goßner	Aug 2006	\$34.19
 Openswan: Building and Integrating Virtual Private Networks	Paul Wouters, Ken Bantoft	Feb 2006	\$53.99
 Learning Jakarta Struts 1.2: a concise and practical tutorial	Stephan Wiesner	Aug 2005	\$31.49
 Implementing SugarCRM	Michael J.R. Whitehead	Feb 2006	\$44.99

图 7-7

7.2 分页

排序是从大量数据中查找信息的一种有效方式。然而，通过对数据进行分页也可以帮助用户

把注意力聚焦于大数据集中的某一部分上。分页可以通过两种方式实现——服务器端分页和JavaScript分页。

7.2.1 服务器端分页

与排序很相似，分页也常常是在服务器上实现的。如果要显示的数据保存在数据库中，那么很容易使用MySQL的LIMIT子句、Oracle的ROWNUM或其他数据库引擎中的等价方法一次提取出一个信息块。

同前面排序的例子一样，分页也可以通过在查询字符串中向服务器发送信息来触发，例如index.php?page=52。而且，也和以前一样，我们既可以通过加载完整的页面，也可以通过使用AJAX只提取一段表格来完成这个任务。这种策略是浏览器中立的，而且也能很好地处理大数据集。

如影随形的排序和分页

如果数据长到需要排序的程度，那么很可能也长到了需要分页的程度。希望在表现数据时组合使用这两种技术也是很正常的。不过，因为这两种技术都会影响到页面上存在的数据集，所以在实现这两种功能时一定要考虑到它们之间的相互影响。

无论排序还是分页都既可以在服务器上完成，也可以在Web浏览器中完成。但是，对这两种任务必须坚持同步的策略，否则，最终会导致混乱的行为。例如，在排序和分页都通过服务器来完成的情况下，如图7-8所示。

A	4
B	5
C	2
D	7
E	1
F	8
G	3
H	6

E	1
C	2
G	3
A	4
B	5
H	6
D	7
F	8

图 7-8

当表格按照数字重新排序之后，表格行的变化可以通过表中的Page 1表现出来。如果分页在服务器上完成，而排序通过浏览器完成，由于排序例程不能使用完整的数据集，因而就会导致错误的结果，如图7-9所示。

只有页面中已经存在的数据能够参与排序。为了避免这种问题，要么同时在服务器上执行这两个任务，要么同时在浏览器中执行这两个任务。

A	4
B	5
C	2
D	7

C	2
A	4
B	5
D	7

图 7-9

7.2.2 JavaScript 分页

下面，我们就来讨论一下如何通过JavaScript对浏览器中已经可排序的表格进行分页。首先，我们从显示一个特定的数据页开始，暂时不考虑用户交互：

```
$('document').ready(function() {
    $('table.paginated').each(function() {
        var currentPage = 0;
        var numPerPage = 10;
        var $table = $(this);
        $table.find('tbody tr').show()
            .lt(currentPage * numPerPage)
            .hide()
            .end()
            .gt((currentPage + 1) * numPerPage - 1)
            .hide()
            .end();
    });
});
```

以上代码用于显示第1页的10行数据。

这里，我们仍然依赖`<tbody>`元素的存在来从标题中分离数据——我们不希望在打开第2页时看不到标题或表注。为了选择包含数据的行，首先要显示所有行，然后选择当前页之前和之后的行，并隐藏它们。通过在“退出”(pop)当前的筛选程序和再次开始新的筛选程序之间使用`.end()`方法，jQuery支持的方法连缀能够在我们执行第二次筛选时，使匹配的行集再次出现。

在编写这种代码时最常出现的问题就是不会描述筛选程序中使用的表达式。为了使用`.lt()`和`.gt()`方法，必须明确地指定当前页第一行和最后一行的索引。对于第一行，只需用当前页数乘以每页的行数即可。而用当前页数加1乘以每页的行数，得到的是下一页的第一行；因此，要得到当前页的最后一行，必须从这个数中减掉1。

1. 显示分页指示器

为了让用户选择分页，还需要在表格旁边放置分页指示器。虽然可以在HTML标记中简单地插入指向各个分页的链接，但这样做违反了我们提倡的渐进增强的原则。所以，我们要使用JavaScript来添加这些链接，以便无法使用脚本的用户不会被无效的链接所误导。

要显示作为分页指示器的链接，需要计算页数并创建表示相应数字的DOM元素：

```
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);

var $pager = $('

</div>');
for (var page = 0; page < numPages; page++) {
    $('<span class="page-number">' + (page + 1) + '</span>')
        .appendTo($pager).addClass('clickable');
}
$pager.insertBefore($table);


```

用数据行数除以每页显示的行数可以得到页数。如果得到的结果不是整数，必须使用`Math.ceil()`向上舍入，以确保显示最后一页。然后，根据这个数字，就可以为每个分页创建按钮并把新的分页指示器放到表格的前面，如图7-10所示。

1 2 3 4 5 6 7				
	Title	Author(s)	Publish Date	Price
	Building Websites with Joomla! 1.5 Beta 1	Hagen Graf	Feb 2007	\$40.49
	Learning Mambo: A Step-by-Step Tutorial to Building Your Website	Douglas Paterson	Dec 2006	\$40.49
	Moodle E-Learning Course Development	William Rice	May 2006	\$35.99
	AJAX and PHP: Building Responsive Web Applications	Cristian Darie, Mihai Bucica, Filip Cherecheș-Toşa, Bogdan Brînzărea	Mar 2006	\$31.49

图 7-10

2. 启用分页按钮

要让这些新按钮发挥作用，需要更新`currentPage`变量，然后运行分页例程。乍一看，好像可以把`currentPage`设置为`page`，而`page`中保存着创建这些按钮的迭代器的当前值：

```
$(document).ready(function() {
    $('table.paginated').each(function() {
        var currentPage = 0;
        var numPerPage = 10;
        var $table = $(this);

        var repaginate = function() {
            $table.find('tbody tr').show()
                .lt(currentPage * numPerPage)
                .hide()
            .end()
```

```

.gt((currentPage + 1) * numPerPage - 1)
.hide()
.end();
},
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('

</div>');
for (var page = 0; page < numPages; page++) {
    $('<span class="page-number">' + (page + 1) + '</span>')
        .click(function() {
            currentPage = page;
            repaginate();
        })
        .appendTo($pager).addClass('clickable');
}
$pager.insertBefore($table);
repaginate();
});
});


```

这些代码多数是有效的。当页面加载时和单击按钮时都会调用新的repaginate()函数。可是，每个按钮打开的分页中都没有数据行，如图7-11所示。

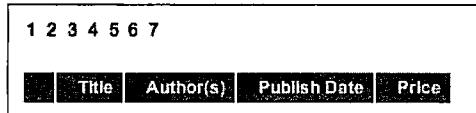


图 7-11

问题的原因是，我们在单击处理程序的定义中创建了一个闭包。单击处理程序引用了page变量，该变量在函数的外部定义。当在下次循环改变这个变量的值时，新的值也会影响到我们为早先的按钮设置的单击处理程序。最终的结果是，对于一个7页的分页指示器来说，每个按钮都会指向第8页（page的最终值）。要了解有关闭包工作原理的更多信息，请参考附录C。

为了修正这个问题，需要利用jQuery的事件绑定方法中的另一个高级特性。事实上，我们可以在绑定处理程序时为它添加一组数据，这组数据在最终调用相应的处理程序时仍然有效。有了这种可靠的技巧之后，可以将相应代码改写为如下所示：

```

$('<span class="page-number">' + (page + 1) + '</span>')
.bind('click', {'newPage': page}, function(event) {
    currentPage = event.data['newPage'];
    repaginate();
})
.appendTo($pager).addClass('clickable');

```

新页码作为事件对象的data属性传递到了处理程序中。这样一来，页码逃脱了闭包，并且在绑定处理程序时及时地冻结了它包含的值。现在，分页指示器按钮就可以正确地打开每个分页了，如图7-12所示。

1 2 3 4 5 6 7					
	Title	Author(s)	Publish Date	Price	
	PHPEclipse: A User Guide	Shu-Wai Chow	Feb 2008	\$31.49	
	Alfresco Enterprise Content Management Implementation	Munwar Sheriff	Jan 2007	\$69.99	
	Smarty PHP Template Programming and Applications	Jean Prado Mala, Hasin Hayder, Lucian Gheorghe	Apr 2006	\$35.99	
	JasperReports for Java Developers	David Heflininger	Aug 2008	\$40.49	

图 7-12

3. 标记当前分页

通过突出显示当前的页码可以为用户使用分页指示器提供更大的便利。为此，只需在每次单击一个按钮时更新相应的类即可：

```
var $pager = $('

</div>');
for (var page = 0; page < numPages; page++) {
    $('' + (page + 1) + '</span>')
        .bind('click', {'newPage': page}, function(event) {
            currentPage = event.data['newPage'];
            repaginate();
            $(this).addClass('active').siblings().removeClass('active');
        })
        .appendTo($pager).addClass('clickable');
}
$pager.find('span.page-number:first').addClass('active');
$pager.insertBefore($table);


```

这样，分页指示器上就会突出地显示出当前页码了，如图7-13所示。

1 2 3 4 5 6 7					
	Title	Author(s)	Publish Date	Price	
	Linux Email: Set up and Run a Small Office Email Server	Patrick Ben Koetter, Carl Taylor, Ralf Hildebrandt, David Rusenko, Alastair McDonald, Magnus Back	Jul 2005	\$36.99	
	Windows Small Business Server SBS 2003: A Clear and Concise Administrator's Reference and How-To	Stephanie Kreicht-Thumann	Aug 2005	\$33.29	
	Creating your MySQL Database: Practical Design Tips and Techniques	Marc Delisle	Nov 2006	\$17.99	
	Building Websites with VB.NET and DotNetNuke 4	Michael Washington, Steve Valenzuela, Daniel N. Egan	Oct 2006	\$35.99	

图 7-13

4. 带排序的分页

在前面开始讨论分页的问题时，我们提到过要注意避免排序和分页混淆结果。既然现在已经有了分页程序，那么下面我们就来讨论如何在不妨碍当前所选分页的前提下实现排序操作。

要实现在分页基础上的排序，其实只要每次执行排序时调用`repaginate()`函数即可。不过，函数的作用域会导致问题。由于排序例程包含在另外一个`$(document).ready()`处理程序中，所以在排序的代码中无法访问到`repaginate()`函数。虽然可以将这两段代码简单地合并起来，但我们要在这里采取一种更高级的方式——解耦（decouple）相应的行为，以便排序行为能够在重新分页行为存在时调用该行为，否则忽略该行为。为实现这一过程，需要将一个处理程序作为自定义事件。

我们在前面讨论到事件处理时，仅限于使用由Web浏览器触发的事件名称，例如`click`和`mouseup`。但是，`.bind()`和`.trigger()`方法并不局限于这些事件。换句话说，可以使用任何字符串作为事件名称。在当前的例子中，我们可以定义一个名为`repaginate`的事件，将它作为要调用函数的替身：

```
$table.bind('repaginate', function() {
    $table.find('tbody tr').show()
        .lt(currentPage * numPerPage)
            .hide()
        .end()
        .gt((currentPage + 1) * numPerPage - 1)
            .hide()
        .end();
});
```

于是，在需要调用`repaginate()`函数的地方，可以调用：

```
$table.trigger('repaginate');
```

同样，也可以把这条调用语句放到排序代码中。如果表格没有分页指示器^①，那么这条语句什么也不会做，因此我们可以按照预期来混合搭配这两种功能。

7.3 完成的代码

完成后的全部排序和分页代码如下所示：

```
$.fn.alternateRowColors = function() {
    $('tbody tr:odd', this).removeClass('even').addClass('odd');
    $('tbody tr:even', this).removeClass('odd').addClass('even');
```

^① 如果表格没有分页指示器，那么说明该表格仅具有排序功能（在这个例子中，也就是`<table>`中只有`.sortable`类），因而，就不会存在绑定的自定义函数`repaginate`。——译者注

```

        return this;
    };

$(document).ready(function() {
    var alternateRowColors = function($table) {
        $('tbody tr:odd', $table).removeClass('even').addClass('odd');
        $('tbody tr:even', $table).removeClass('odd').addClass('even');
    };

    $('table.sortable').each(function() {
        var $table = $(this);
        $table.alternateRowColors($table);
        $table.find('th').each(function(column) {
            var findSortKey;

            if ($(this).is('.sort-alpha')) {
                findSortKey = function($cell) {
                    return $cell.find('.sort-key').text().toUpperCase() +
                        ' ' + $cell.text().toUpperCase();
                };
            }
            else if ($(this).is('.sort-numeric')) {
                findSortKey = function($cell) {
                    var key = parseFloat($cell.text().replace(/[^\\d.]*/,' '));
                    return isNaN(key) ? 0 : key;
                };
            }
            else if ($(this).is('.sort-date')) {
                findSortKey = function($cell) {
                    return Date.parse('1 ' + $cell.text());
                };
            }
        });

        if (findSortKey) {
            $(this).addClass('clickable').hover(function() {
                $(this).addClass('hover');
            }, function() {
                $(this).removeClass('hover');
            }).click(function() {
                var newDirection = 1;
                if ($(this).is('.sorted-asc')) {
                    newDirection = -1;
                }

                rows = $table.find('tbody > tr').get();
                $.each(rows, function(index, row) {
                    row.sortKey =
                        findSortKey($(row).children('td').eq(column));
                });
                rows.sort(function(a, b) {

```

```

if (a.sortKey < b.sortKey) return -newDirection;
if (a.sortKey > b.sortKey) return newDirection;
return 0;
});
$.each(rows, function(index, row) {
  $table.children('tbody').append(row);
  row.sortKey = null;
});
$table.find('th').removeClass('sorted-asc')
  .removeClass('sorted-desc');
var $sortHead = $table.find('th').filter(':nth-child('
  + (column + 1) + ')');
if (newDirection == 1) {
  $sortHead.addClass('sorted-asc');
} else {
  $sortHead.addClass('sorted-desc');
}
$table.find('td').removeClass('sorted')
  .filter(':nth-child(' + (column + 1) + ')')
    .addClass('sorted');
  $table.alternateRowColors($table);
  $table.trigger('repaginate');
});
}
});
});
$(document).ready(function() {
  $('#table.paginated').each(function() {
    var currentPage = 0;
    var numPerPage = 10;

    var $table = $(this);

    $table.bind('repaginate', function() {
      $table.find('tbody tr').show()
        .lt(currentPage * numPerPage)
          .hide()
        .end()
        .gt((currentPage + 1) * numPerPage - 1)
          .hide()
        .end();
    });
    var numRows = $table.find('tbody tr').length;
    var numPages = Math.ceil(numRows / numPerPage);

    var $pager = $('

</div>');
    for (var page = 0; page < numPages; page++) {
      $('' + (page + 1) + '</span>')
    }
  });
})
});


```

```

        .bind('click', {'newPage': page}, function(event) {
            currentPage = event.data['newPage'];
            $table.trigger('repaginate');
            $(this).addClass('active').siblings().removeClass('active');
        })
        .appendTo($pager).addClass('clickable');
    }
    $pager.find('span.page-number:first').addClass('active');
    $pager.insertBefore($table);

    $table.trigger('repaginate');
});
});
}
);

```

7.4 高级行条纹效果

我们在本章前面曾看到过，通过两行简单的代码交替变换背景颜色就可以实现行条纹效果：

```

$(document).ready(function() {
    $('table.sortable tbody tr:odd').addClass('odd');
    $('table.sortable tbody tr:even').addClass('even');
});

```

如果像下面这样为odd和even类声明背景颜色，那么我们会看到行中交替显示的灰色条纹：

```

tr.even {
    background-color: #eee;
}
tr.odd {
    background-color: #ddd;
}

```

尽管这些代码对于简单的表格结构很有效，但如果我们向表格中引入非标准的行，例如副标题行（sub-heading），那么这种基本的“奇-偶”模式就不能满足需要了。假设有一个按年分组的新闻条目表格，表格中的列分别是日期（Date）、题目（Headline）、作者（Author）和主题（Topic）。表示这种信息的一种方式，就是将每一年的新闻条目包装在`<tbody>`元素中，并使用`<th colspan="4">`作为其副标题行。这样，相应表格的HTML（有删节）代码应该如下所示：

```


| Date | Headline | Author | Topic |
|------|----------|--------|-------|
|      |          |        |       |


```

```
<th colspan="4">2007</th>
</tr>
<tr>
  <td>Mar 11</td>
  <td>SXSWi jQuery Meetup</td>
  <td>John Resig</td>
  <td>conference</td>
</tr>
<tr>
  <td>Feb 28</td>
  <td>jQuery 1.1.2</td>
  <td>John Resig</td>
  <td>release</td>
</tr>
<tr>
  <td>Feb 21</td>
  <td>jQuery is OpenAjax Compliant</td>
  <td>John Resig</td>
  <td>standards</td>
</tr>
<tr>
  <td>Feb 20</td>
  <td>jQuery and Jack Slocum's Ext</td>
  <td>John Resig</td>
  <td>third-party</td>
</tr>
</tbody>
<tbody>
  <tr>
    <th colspan="4">2006</th>
  </tr>
  <tr>
    <td>Dec 27</td>
    <td>The Path to 1.1</td>
    <td>John Resig</td>
    <td>source</td>
  </tr>
  <tr>
    <td>Dec 18</td>
    <td>Meet The People Behind jQuery</td>
    <td>John Resig</td>
    <td>announcement</td>
  </tr>
  <tr>
    <td>Dec 13</td>
    <td>Helping you understand jQuery</td>
    <td>John Resig</td>
    <td>tutorial</td>
  </tr>
```

```

</tbody>
<tbody>
<tr>
  <th colspan="4">2005</th>
</tr>
<tr>
  <td>Dec 17</td>
  <td>JSON and RSS</td>
  <td>John Resig</td>
  <td>miscellaneous</td>
</tr>
</tbody>
</table>

```

在为`<thead>`和`<tbody>`中的`<th>`元素分别应用CSS样式后，这个表格的一部分如图7-14所示。

Date	Headline	Author	Topic
2007			
Mar 11	SXSWI jQuery Meetup	John Resig	conference
Feb 28	jQuery 1.1.2	John Resig	release
Feb 21	jQuery is OpenAjax Compliant	John Resig	standards
Feb 18	The jQuery IRC Channel	Yehuda Katz	announcement
Feb 14	jQuery Nightly Builds	Paul McLanahan	announcement
Feb 2	New jQuery Project Team Ms	ReyBango	announcement
Jan 15	Interface 1.1 Released	John Resig	Plug-in
Jan 14	jQuery Birthday: 1.1, New Site, New Docs	John Resig	announcement
Jan 13	jQuery wallpapers	Nate Cavanaugh	miscellaneous
Jan 11	Selector Speeds	John Resig	source
Jan 11	jQuery 1.1b	John Resig	release
2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Helping you understand jQuery	John Resig	tutorial
Dec 12	jQuery 1.0.4	John Resig	release

图 7-14

为了保证交替出现的灰色数据行不会覆盖副标题行的颜色，需要对选择符表达式进行相应调整：

```

$(document).ready(function() {
  $('table.stripped tbody tr:not([th]):odd').addClass('odd');
  $('table.stripped tbody tr:not([th]):even').addClass('even');
});

```

其中，添加的选择符`:not ([th])`用于从匹配的元素集合中移除包含`<th>`的表格行。现在，表格将如图7-15所示。

Date	Headline	Author	Topic
2007			
Mar 11	SXSWI jQuery Meetup	John Resig	conference
Feb 28	jQuery 1.1.2	John Resig	release
Feb 21	jQuery Is OpenAjax Compliant	John Resig	standards
Feb 18	The jQuery IRC Channel	Yehuda Katz	announcement
Feb 14	jQuery Nightly Builds	Paul McLanahan	announcement
Feb 2	New jQuery Project Team Ms	ReyBango	announcement
Jan 15	Interface 1.1 Released	John Resig	Plug-in
Jan 14	jQuery Birthday: 1.1, New Site, New Docs	John Resig	announcement
Jan 13	jQuery wallpapers	Nate Cavanaugh	miscellaneous
Jan 11	Selector Speeds	John Resig	source
Jan 11	jQuery 1.1b	John Resig	release
2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Helping you understand jQuery	John Resig	tutorial
Dec 12	jQuery 1.0.4	John Resig	release

图 7-15

7.4.1 三色交替模式

有时候，可能我们也需要为表格应用更复杂的条纹效果。例如，三行而不是两行颜色交替的模式。为实现这种模式，首先需要为第3行定义一条CSS规则。同时，还需要对第1行和第2行重用odd和even样式，即在原来的规则上再添加相应的类名：

```
tr.even,
tr.first {
    background-color: #eee;
}
tr.odd,
tr.second {
    background-color: #ddd;
}
tr.third {
    background-color: #ccc;
}
```

为应用这种模式，我们也像前一个例子那样开始——选择作为`<tbody>`元素后代的所有行，但从中排除包含`<th>`的行。不过，这一次需要使用`.each()`方法，以便使用该方法内置的`index`：

```
$(document).ready(function() {
    $('table.striped tbody tr').not(' [th] ').each(function(index) {
        // 应用到匹配的集合中每个元素上的代码
    });
});
```

为了利用索引(index)，可以将前面定义的3个类指定给数字键0、1和2。为此，需要创建一个对象，或者说一个映射：

```
$ (document).ready(function() {
    var classNames = {
        0: 'first',
        1: 'second',
        2: 'third'
    };
    $('table.striped tbody tr').not('[th]').each(function(index) {
        // 应用到匹配的集合中每个元素上的代码
    });
});
```

最终，按顺序添加与这3个数字对应的类，然后再重复这个序列。要完成这种计算，可以使用以%表示的求模运算符。所谓模，就是两个数相除得到的余数。而且，这个模或余数的值，始终介于0与除数减1之间。如果以3为例，那么可以看到如下模式：

```
3/3 = 1, 余0。
4/3 = 1, 余1。
5/3 = 1, 余2。
6/3 = 2, 余0。
7/3 = 2, 余1。
8/3 = 3, 余2。
```

依此类推。因为我们希望余数的范围在0~2之间，所以要使用3作为除数(第2个数)，将index的值作为被除数(第1个数)。下面，我们就把这个算式直接放在classNames之后的方括号中，以便在.each()方法逐个迭代匹配的行集合时从该对象中取得对应的类：

```
$ (document).ready(function() {
    var classNames = {
        0: 'first',
        1: 'second',
        2: 'third'
    };
    $('table.striped tbody tr').not('[th]').each(function(index) {
        $(this).addClass(classNames[index % 3]);
    });
});
```

编写完这些代码后，就可以看到带有3种交替的背景颜色的条纹式表格了，如图7-16所示。

当然也可以把这种模式扩展到4、5、6或更多种背景颜色，届时，只需向className对象中添加对应的“键-值”对，并增加classNames[index % n]中除数n的值即可。

Date	Headline	Author	Topic
2007			
Mar 11	SXSWI JQuery Meetup	John Resig	conference
Feb 28	jQuery 1.1.2	John Resig	release
Feb 21	jQuery is OpenAjax Compliant	John Resig	standards
Feb 18	The JQuery IRC Channel	Yehuda Katz	announcement
Feb 14	jQuery Nightly Builds	Paul McLanahan	announcement
Feb 2	New JQuery Project Team Ms	ReyBango	announcement
Jan 15	Interface 1.1 Released	John Resig	Plug-in
Jan 14	jQuery Birthday: 1.1, New Site, New Docs	John Resig	announcement
Jan 13	jQuery Wallpapers	Nate Cavanaugh	miscellaneous
Jan 11	Selector Speeds	John Resig	source
Jan 11	jQuery 1.1b	John Resig	release
Jan 8	jQuery 1.1a	John Resig	release
2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Helping you understand jQuery	John Resig	tutorial

图 7-16

7.4.2 三行一组交替

假设我们想使用两种颜色，但每种颜色一次要应用到三行。为此，仍然可以利用odd和even类，以及求模运算符。但是，我们还要在每次遇到包含`<th>`元素的行时重置相应的类。

如果不重置用于交替行的类，那么在条纹化的第一个行组之后有可能看到意外的颜色。到目前为止，由于我们例子表格的第一个行组包含12行，恰好能被2和3整除，所以可以避免这个问题。但对于现在三行一组交替的情况，我们要删除两行，即在第一个行组中剩下10行，以便示范重置类的重要性。

要实现这种条纹效果，我们从设置两个变量开始：`rowClass`和`rowIndex`。同样，这次还需要使用`.each()`方法，但我们不再依赖该方法内置的`index`，而是使用自定义的`rowIndex`变量，以便在遇到带`<th>`的行时能够重置该变量的值：

```
$(document).ready(function() {
  var rowClass = 'even';
  var rowIndex = 0;
  $('table.striped tbody tr').each(function(index) {
    $(this).addClass(rowClass);
  });
});
```

我们注意到，由于去掉了`:not([th])`选择符，那么在`.each()`方法内部就必须考虑相应的副标题行。不过，我们还是先来为整个表格应用三行一组的交替效果。目前，每个`<tr>`都会变成`<tr class="even">`。具体到每一行，我们可以检查`rowIndex % 3`是否等于0。如果是，就切换

rowClass的值。然后，再递增rowIndex的值：

```
$(document).ready(function() {
    var rowClass = 'even';
    var rowIndex = 0;
    $('table.striped tbody tr').each(function(index) {
        if (rowIndex % 3 == 0) {
            rowClass = (rowClass == 'even' ? 'odd' : 'even');
        };
        $(this).addClass(rowClass);
        rowIndex++;
    });
});
```

为了代码简洁，这里使用了三元或者条件运算符来设置rowClass值的转换。该行代码也可以重写为：

```
if (rowClass == 'even') {
    rowClass = 'odd';
} else {
    rowClass = 'even';
}
```

无论怎么写以上代码，都可以生成如图7-17所示的条纹化表格效果。

Date	Headline	Author	Topic
2007			
Mar 11	SXSW jQuery Meetup	John Resig	conference
Feb 28	jQuery 1.1.2	John Resig	release
Feb 21	jQuery Is OpenAjax Compliant	John Resig	standards
Feb 18	The jQuery IRC Channel	Yehuda Katz	announcement
Feb 14	jQuery Nightly Builds	Paul McLanahan	announcement
Feb 2	New jQuery Project Team Members	ReyBango	announcement
Jan 15	Interface 1.1 Released	John Resig	Plug-in
Jan 14	jQuery Birthday: 1.1, New Site, New Docs	John Resig	announcement
Jan 13	jQuery Wallpapers	Nate Cavanaugh	miscellaneous
Jan 11	Selector Speeds	John Resig	source
2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Helping You Understand jQuery	John Resig	tutorial
Dec 12	jQuery 1.0.4	John Resig	release
Dec 12	jQuery v1.0.3 API docs on gotAPI.com	ReyBango	documentation

图 7-17

或许你会感到吃惊，因为副标题行仍然保持着正确的格式。但是，千万不要被这种表象所迷惑。实际上，现在的HTML中，副标题行2007已经设置成了`<tr class="odd">`，而副标题行2006则已经设置成了`<tr class="even">`。只不过在样式表中，这两个元素的样式规则更具有针对性^①，因而权重超过了新添加的两个类：

```
#content tbody th {
    background-color: #6f93ce;
    padding-left: 6px;
}
tr.even {
    background-color: #eee;
}
tr.odd {
    background-color: #ddd;
}
```

由于`rowIndex`编号中没有排除这些副标题行，我们从一开始就为每一行都添加了错误的类——这一点从表格开始处的颜色变化发生在两行而不是三行之后可以明显地看出来。

因此，还需要在代码中添加其他条件，检查当前行是否包含`<th>`元素。如果是，则将`rowClass`的值设置为`subhead`，将`rowIndex`的值设置为`-1`。

```
$(document).ready(function() {
    var rowClass = 'even';
    var rowIndex = 0;
    $('table.striped tbody tr').each(function(index) {
        if ($(this).length) {
            rowClass = 'subhead';
            rowIndex = -1;
        } else if (rowIndex % 3 == 0) {
            rowClass = (rowClass == 'even' ? 'odd' : 'even');
        };
        $(this).addClass(rowClass);
        rowIndex++;
    });
});
```

当在副标题行中将`rowIndex`设置为`-1`后，这个变量会在下一行递增至`0`——而这恰好是我们想要的条纹行组的开始位置。现在，我们可以看到每一年的文章列表都以三行浅颜色的条纹行开始，并且每三行交替一次浅颜色和深颜色，如图7-18所示。

^① 根据CSS中选择符的权重计算法则，针对性更高的选择符对应的样式声明优先适用。比如：`#content tbody th`包含一个ID选择符和两个元素选择符，因而它的针对性就是`0、1、0、2`；而`tr.even`包含一个元素选择符和一个类选择符，所以针对性为`0、0、1、1`。由于前者的针对性更高，因此优先适用。——译者注

Date	Headline	Author	Topic
2007			
Mar 11	SXSWI jQuery Meetup	John Resig	conference
Feb 28	jQuery 1.1.2	John Resig	release
Feb 21	jQuery is OpenAjax Compliant	John Resig	standards
Feb 18	The jQuery IRC Channel	Yehuda Katz	announcement
Feb 14	jQuery Nightly Builds	Paul McLanahan	announcement
Feb 2	New jQuery Project Team Members	ReyBango	announcement
Jan 15	Interface 1.1 Released	John Resig	Plug-in
Jan 14	jQuery Birthday: 1.1, New Site, New Docs	John Resig	announcement
Jan 13	jQuery wallpapers	Nate Cavanaugh	miscellaneous
Jan 11	Selector Speeds	John Resig	source
2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Helping you understand jQuery	John Resig	tutorial
Dec 12	jQuery 1.0.4	John Resig	release
Dec 12	jQuery v1.0.3 API docs on gotAPI.com	ReyBango	documentation

图 7-18

关于实现表格条纹效果的代码，最后一点需要注意的是，虽然三元运算符确实简洁，但随着条件的复杂化，它也会变得更难以理解。在实现复杂的条纹变化效果时，使用基本的if-else条件语句反而可以使代码更容易管理：

```

$(document).ready(function() {
    var rowIndex = 0;
    $('tbody tr').each(function(index) {
        if ($(this).length) {
            $(this).addClass('subhead');
            rowIndex = -1;
        } else {
            if (rowIndex % 6 < 3) {
                $(this).addClass('even');
            }
            else {
                $(this).addClass('odd');
            }
        };
        rowIndex++;
    });
});

```

以上代码也能实现同前面一样的效果，但同时也使得包含更多的else if条件更容易了。

7.5 突出显示行

另一种可以应用到这个新闻条目表格的视觉增强效果，是基于用户的交互突出显示相关的行。这里，我们要根据用户单击的作者名字，突出显示作者单元格中包含相同名字的所有行。同添加行条纹效果一样，我们也是通过添加类来改变要突出显示的行的外观：

```
#content tr.highlight {
    background: #ff6;
}
```

为这个新的highlight类赋予足够的针对性^①非常重要，只有这样才能确保覆盖even或odd类定义的背景颜色。

接下来，需要选择适当的单元格并为其添加.click()方法：

```
$(document).ready(function() {
    var column = 3;
    $('table.striped td:nth-child(' + column + ')')
        .click(function() {
            // 根据单击执行操作
        });
});
```

我们注意到，选择符表达式中包含了:nth-child(n)伪类选择符，但并没有在该选择符中直接指定子元素的编号，而是为它传递了变量column。由于后面还要引用相同的nth-child，这样我们就可以在以后想要基于其他列突出显示行时，只在一个地方修改这个变量即可。

说明 与JavaScript中的索引不同，基于CSS的:nth-child(n)伪类选择符从1开始编号，而不是从0开始。

当用户单击第3列中的某个单元格时，我们想要将该单元格包含的文本与同一列中位于其他行的单元格包含的文本进行比较。如果匹配，则切换highlight类。换句话说，如果这个类不存在则添加该类，否则，如果存在就移除该类。这样，当单击某个作者单元格时，如果该单元格或包含相同作者的其他单元格已经被单击过了，就可以移除相应行的突出显示效果：

```
$(document).ready(function() {
    $('table.striped td:nth-child(' + column + ')')
        .click(function() {
            var thisClicked = $(this).text();
            $('table.striped td:nth-child(' + column + ')')
                .each(function(index) {
                    if (thisClicked == $(this).text()) {
                        $(this).parent().toggleClass('highlight');
```

^① 即在.highlight前面罗列足够多的高权重的选择符，从而提高整个组合选择符的针对性。——译者注

```

    };
  });
});
})
}

```

此时的代码，除非用户连续单击两个作者的名字，否则一切正常。当我们单击第2个作者的名字时，这些代码并没有像我们想象的那样把突出显示的行转换到第2个作者，而是将包含第2次单击的作者名字的行也添加到拥有class="highlight"的组中。为了避免这种行为，可以向代码中添加一条else语句，移除不包含单击的作者名字的所有行的highlight类：

```

$(document).ready(function() {
  $('table.striped td:nth-child(' + column + ')')
    .click(function() {
      var thisClicked = $(this).text();
      $('table.striped td:nth-child(' + column + ')')
        .each(function(index) {
          if (thisClicked == $(this).text()) {
            $(this).parent().toggleClass('highlight');
          } else {
            $(this).parent().removeClass('highlight');
          };
        });
    });
});
})

```

这样，当单击Rey Bango时，就会更容易找到与他相关的条目，如图7-19所示。

Feb 28	jQuery 1.1.2	John Resig	release
Feb 21	jQuery Is OpenAjax Compliant	John Resig	standards
Feb 18	The jQuery IRC Channel	Yehuda Katz	announcement
Feb 14	jQuery Nightly Builds	Paul McLanahan	announcement
Feb 2	New jQuery Project Team Members	Rey Bango	announcement
Jan 15	Interface 1.1 Released	John Resig	plug-in
Jan 14	jQuery Birthday: 1.1, New Site, New Docs	John Resig	announcement
Jan 13	jQuery wallpapers	Nate Cavanaugh	miscellaneous
Jan 11	Selector Speeds	John Resig	source
2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Helping you understand jQuery	John Resig	tutorial
Dec 12	jQuery 1.0.4	John Resig	release
Dec 12	jQuery v1.0.3 API Docs on GitHub	Rey Bango	documentation
Dec 12	jQuery Presentation In Phoenix on 11/26	Rey Bango	conference
Nov 28	jQuery at AZ PHP	Rey Bango	conference
Nov 14	Expandable Sidebar Menu Screencast	John Resig	tutorial

图 7-19

如果我们再单击任何单元格中John Resig的名字，突出显示效果就会从Rey Bango的行移除，然后添加到John Resig所在的行。

7.6 工具提示条

尽管行突出效果是一种很有用的特性，但就目前来讲，这些特性存在与否对用户而言并不明显。为了解决这个问题，我们可以先为所有包含作者名字的单元格添加clickable类，以便当用户的鼠标悬停在这些单元格上面时，光标会变成指示器的形状：

```
$(document).ready(function() {
    $('table.striped td:nth-child(' + column + ')')
        .addClass('clickable')
        .click(function() {
            var thisClicked = $(this).text();
            $('table.striped td:nth-child(' + column + ')')
                .each(function(index) {
                    if (thisClicked == $(this).text()) {
                        $(this).parent().toggleClass('highlight');
                    } else {
                        $(this).parent().removeClass('highlight');
                    }
                });
        })
    })
});
```

添加clickable类确实朝正确的方向迈进了一步，但用户仍然无从知晓单击这个单元格会发什么事情。不过，任何人都知道（当然，在没有看到代码的情况下）单击可能会把用户带到另一个页面上。为此，进一步添加与单击会发生什么情况有关的说明是必要的。

工具提示条是许多软件应用程序中常见的特性，Web浏览器也不例外。因此，可以通过自定义的文本来模拟工具提示条，比如：当用户的鼠标悬停在其中一个作者的单元格上时，显示“Click to highlight all rows authored by Rey Bango”。这样，就能提醒用户他们的操作会导致什么结果。

接下来，我们要在所有事件处理程序外部创建3个函数——showTooltip、hideTooltip和positionTooltip，然后在必要时调用或者引用它们。先从positionTooltip开始，我们会在鼠标移动到任何作者单元格上时引用这个函数：

```
var positionTooltip = function(event) {
    var tPosX = event.pageX - 5;
    var tPosY = event.pageY + 20;
    $('#div.tooltip').css({top: tPosY, left: tPosX});
};
```

这里，我们使用事件对象的pageX和pageY属性来设置工具提示条上边和左边的位置。当在.mousemove()方法中引用这个函数时，tPosX引用的是鼠标指针左侧5像素的位置，而tPosY

引用的是指针下方20像素的位置。我们可以把这个方法添加到包含.click()方法的同一个方法连缀序列中：

```
$ (document).ready(function() {
    var positionTooltip = function(event) {
        var tPosX = event.pageX - 5;
        var tPosY = event.pageY + 20;
        $('div.tooltip').css({top: tPosY, left: tPosX});
    };

    $('table.striped td:nth-child(' + column + ')')
        .addClass('clickable')
        .click(function() {
            // .....省略的代码.....
        })
        .mousemove(positionTooltip);
});
```

现在，虽然已经完成了工具提示条的定位，但是，我们还没有创建该提示条。下面的showTooltip函数将负责创建这个工具提示条。

在showTooltip函数中，首先要删除所有工具提示条。虽然这样做好像违反了直觉，但如果我想在鼠标指针每次悬停在一个作者单元格上时显示工具条，就必须这么做。否则，每当鼠标悬停到一个新单元格上时，都会新增一个工具提示条：

```
var showTooltip = function(event) {
    $('div.tooltip').remove();
};
```

下面可以创建工具提示条了。为此，可以把整个

及其内容包装到一个\$()函数中，然后将该函数再添加到文档的主体中：

```
var showTooltip = function(event) {
    $('div.tooltip').remove();
    var $thisAuthor = $(this).text();
    $('

Click to highlight all articles
written by ' + $thisAuthor + '</div>')
.appendTo('body');
};


```

当鼠标指针悬停在包含作者Rey Bango的单元格上时，工具提示条中将会显示“Click to highlight all articles written by Rey Bango”。但是，此时工具提示条却出现在了页面的底部。这正是需要使用positionTooltip函数的时候，我们可以简单地把该函数放到showTooltip函数的末尾：

```
var showTooltip = function(event) {
    $('div.tooltip').remove();
    var $thisAuthor = $(this).text();
```

```

$(`<div class="tooltip">Click to highlight all articles
written by ' + $thisAuthor + '</div>')
.appendTo('body');
positionTooltip(event);
};

```

可是，即使这样工具提示条的位置仍然不正确——除非我们把它从默认的position:static属性中解放出来。下面再在样式表中添加如下规则：

```

.tooltip {
  position: absolute;
  z-index: 2;
  background: #efd;
  border: 1px solid #ccc;
  padding: 3px;
}

```

这条样式规则在修改工具提示条定位的同时，也为它设置了一个比周围元素更高的z-index属性，以确保它出现在周围元素的上方。而且，通过应用背景颜色、边框及内边距，也使工具提示条的外观更加精美。

最后，我们来编写简单的hideTooltip函数：

```

var hideTooltip = function() {
  $('div.tooltip').remove();
};

```

既然用于显示、隐藏和定位工具提示条的函数都已经具备，就可以在代码中适当的位置来引用它们了：

```

$(document).ready(function() {
  var column = 3;
  // 定位工具提示条
  var positionTooltip = function(event) {
    var tPosX = event.pageX - 5;
    var tPosY = event.pageY + 20;
    $('div.tooltip').css({top: tPosY, left: tPosX});
  };
  // 显示（创建）工具提示条
  var showTooltip = function(event) {
    $('div.tooltip').remove();
    var $thisAuthor = $(this).text();
    $('<div class="tooltip">Click to highlight all articles written
by ' + $thisAuthor + '</div>').appendTo('body');
    positionTooltip(event);
  };
  // 隐藏（删除）工具提示条
  var hideTooltip = function() {
    $('div.tooltip').remove();
  };
}

```

```

$(`table.striped td:nth-child(' + column + ')`)
.addClass('clickable')
.click(function(event) {
    var thisClicked = $(this).text();
    $('table.striped td:nth-child(' + column + ')')
        .each(function(index) {
            if (thisClicked == $(this).text()) {
                $(this).parent().toggleClass('highlight');
            } else {
                $(this).parent().removeClass('highlight');
            };
        });
    });
.hover(showTooltip, hideTooltip)
.mousemove(positionTooltip);
})

```

我们注意到，`.hover()`和`.mousemove()`方法引用了在其他地方定义的函数。同样，这些函数都不带参数。而且，因为`showTooltip`函数内部也调用了`positionTooltip(event)`，所以当悬停事件发生时，工具提示条会立即获得正确的位置。然后，由于`.mousemove()`函数也以`positionTooltip`为参数，因而当鼠标指针在单元格上移动时，还会继续引用该函数为工具提示条进行定位。现在，带工具提示条的表格外观如图7-20所示。

一切都很正常。当鼠标悬停在作者单元格上时，工具提示条出现；当鼠标移出这些单元格时，工具提示条消失。而唯一的问题是，当相应作者的条目已经突出显示时，工具提示条还在建议单击单元格可以突出显示这些条目，如图7-21所示。

Paul McLanahan	announcement
Rey Bango	announcement
John Resig	announcement
Nate Cavanaugh	miscellaneous
John Resig	source
John Resig	source
John Resig	announcement

图 7-20

Paul McLanahan	announcement
Rey Bango	announcement
John Resig	Click to highlight all articles written by Rey Bango
John Resig	announcement
Nate Cavanaugh	miscellaneous
John Resig	source
John Resig	source
John Resig	announcement

图 7-21

这里，我们需要根据相应的行是否带有`highlight`类来修改工具提示条。好在，我们有一个单独的`showTooltip`函数，可以在这个函数中添加检查类的条件测试代码。具体来说，就是在创建工具提示条时，如果当前单元格的父`<tr>`元素包含`highlight`类，则在这个单词`highlight`前面

添加un-:

```

$(document).ready(function() {
    var highlighted = "";
    // .....省略的代码.....
    var showTooltip = function(event) {
        $('div.tooltip').remove();
        var $thisAuthor = $(this).text();
        if ($(this).parent().is('.highlight')) {
            highlighted = 'un-';
        } else {
            highlighted = '';
        };
        $('<div class="tooltip"> Click to '
            + highlighted + 'highlight all articles written by '
            + $thisAuthor + '</div>').appendTo('body');
        positionTooltip(event);
    };
});

```

假如不考虑单击单元格的同时也触发修改工具提示条的行为，那么工具提示条的开发任务就可以宣告结束了。然而，为了实现单击同时也改变工具提示条的内容，还需要在.click()事件处理程序中调用showTooltip函数：

```

$(document).ready(function() {
    // .....省略的代码.....
    .click(function(event) {
        var thisClicked = $(this).text();
        $('table.striped td:nth-child(' + column + ')')
        .each(function(index) {
            if (thisClicked == $(this).text()) {
                $(this).parent().toggleClass('highlight');
            } else {
                $(this).parent().removeClass('highlight')
            };
        });
        showTooltip.call(this, event);
    })
    // .....省略的代码.....
});

```

通过使用JavaScript的call()函数，可以调用showTooltip函数——就好像该函数是在.click()处理程序中定义的一样。因此，关键字this继承了.click()的作用域。而且，我们也传递了event对象，以便使用其pageX和pageY属性计算工具提示条的位置。

现在，当鼠标所在的行处于突出显示状态时，工具提示条会更智能地给出提示，如图7-22所示。

John Resig	standards
Yehuda Katz	announcement
Paul McLanahan	announcement
Ray Bango	announcement
<small>John Resig</small>	<small>Click to un-highlight all articles written by Ray Bango</small>
John Resig	announcement
Nate Cavanaugh	miscellaneous
John Resig	source
John Resig	source

图 7-22

7.7 折叠和扩展

当表格中以分组的方式包含大量数据时（例如新闻页面中按年分组的条目），如果能够折叠或者隐藏一部分表格的内容，就可以减少页面滚动，从而方便对表格中的所有数据进行更清晰地查看。

要折叠新闻条目表格中的各个部分，首先需要在表格中每个副标题行的第一个单元格前面添加一个减号图像。之所以通过JavaScript来插入这幅小图像，是因为在没有JavaScript而无法折叠表格行的情况下，该图像会给想要单击它来触发某种事件的人带来困扰：

```
$ (document).ready(function() {
    var toggleMinus = '../icons/bullet_toggle_minus.png';
    var togglePlus = '../icons/bullet_toggle_plus.png';
    var $subHead = $('tbody th:first-child');
    $subHead.prepend('');
});
```

这里，我们同时为减号和加号图像的位置设置了变量，以便单击图像折叠或扩展表格行时能够修改图像的src属性。

接下来，使用.addClass()方法让新创建的图像看起来可以单击：

```
$ (document).ready(function() {
    var toggleMinus = '../icons/bullet_toggle_minus.png';
    var togglePlus = '../icons/bullet_toggle_plus.png';
    var $subHead = $('tbody th:first-child');
    $subHead.prepend('');
    $('img', $subHead).addClass('clickable');
});
```

最后，就是在`.click()`方法中添加完成折叠和扩展操作的代码。为此，需要使用条件语句来检查这幅可单击图像的`src`属性。如果这个属性等于变量`toggleMinus`中保存的文件路径，那么就要隐藏同一个`<tbody>`中的其他所有`<tr>`元素，同时将这个`src`属性的值设置为`togglePlus`变量的值。否则，就要显示其他所有`<tr>`元素，并将`src`属性的值设置回`toggleMinus`变量的值：

```
$ (document).ready(function() {
    var toggleMinus = '../icons/bullet_toggle_minus.png';
    var togglePlus = '../icons/bullet_toggle_plus.png';
    var $subHead = $('tbody th:first-child');
    $subHead.prepend('');
    $('img', $subHead).addClass('clickable')
    .click(function() {
        var toggleSrc = $(this).attr('src');
        if ( toggleSrc == toggleMinus ) {
            $(this).attr('src', togglePlus)
            .parents('tr').siblings().fadeOut('fast');
        } else{
            $(this).attr('src', toggleMinus)
            .parents('tr').siblings().fadeIn('fast');
        };
    });
})
});
```

把这些代码放到适当的位置之后，单击2007旁边的减号图像会导致表格变成如图7-23所示。

Date	Headline	Author	Topic
+ 2007			
- 2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Helping you understand jQuery	John Resig	tutorial
Dec 12	jQuery 1.0.4	John Resig	release
Dec 12	jQuery v1.0.3 API docs on gotAPI.com	Rey Bango	documentation
Dec 12	jQuery Presentation in Phoenix on 11/28	Rey Bango	conference
Nov 28	jQuery at AZPhp	Rey Bango	conference

图 7-23

2007年的新闻并没有删除，只是暂时隐藏起来了。如果我们再单击出现在该行前头的加号图像，相关的新闻条目仍然会显示出来。

由于浏览器为表格行设置的可见`display`属性的值不同（`table-row`或`block`），所以对表格行应用动画存在特殊的障碍。因此，没有动画效果的`.hide()`和`.show()`方法在应用到表格行时始终是安全的。到了jQuery 1.1.3版，也可以使用`.fadeIn()`和`.fadeOut()`方法。

7.8 筛选

本章前面介绍了排序和分页等帮助用户聚焦于表格数据中相关部分的技术。而且，我们也看到这两种功能既可以通过服务器端技术，也可以通过JavaScript来实现。本节将要介绍的筛选是数据排列策略弹药库中的最后一种武器。通过只向用户显示匹配给定条件的表格行，可以省去用户不必要的分心。

我们已经看到过的一种筛选类型是突出显示一组表格行。接下来，我们就在突出显示表格行的基础上进一步扩展——实际地隐藏与筛选条件不匹配的行。

首先，需要创建放置筛选按钮的元素。按照惯例，我们要使用JavaScript来插入这些控件，以便不能使用JavaScript的用户看不到这些选项：

```
$ (document).ready(function() {
    $('table.filterable').each(function() {
        var $table = $(this);
        $table.find('th').each(function (column) {
            if ($(this).is('.filter-column')) {
                var $filters = $('

<h3>Filter by ' +
                    + $(this).text() + ':</h3></div>');
                $filters.insertBefore($table);
            }
        });
    });
});


```

这里，我们从列标题中取得了筛选框的标签，以便将同样的代码轻松地重用到其他表格中。现在，筛选框有了标题但还缺少按钮，如图7-24所示。

Date	Headline	Author	Topic	
2007				
Mar 11	SXSWi jQuery Meetup	John Resig	conference	
Feb 28	jQuery 1.1.2	John Resig	release	
Feb 21	jQuery is OpenAjax Compliant	John Resig	standards	
Feb 18	The jQuery IRC Channel	Yehuda Katz	announcement	
Feb 14	jQuery Nightly Builds	Paul McElanahan	announcement	
Feb 2	New jQuery Project Team Members	Rey Bangó	announcement	
Jan 15	Interface 1.1 Released	John Resig	plug-in	
Jan 14	jQuery Birthday: 1.1, New Site, New Docs	John Resig	announcement	
Jan 13	jQuery wallpapers	Nate Cavanaugh	miscellaneous	
Jan 11	Selector Speeds	John Resig	source	
2006				
Dec 27	The Path to 1.1	John Resig	source	

图 7-24

7.8.1 筛选选项

下面，我们就可以真正地来实现筛选功能了。作为开始，我们要为两个已知的主题添加筛选

按钮。相应的代码与前面突出显示作者行例子中的代码非常类似：

```
var keywords = ['conference', 'release'];
$.each(keywords, function (index, keyword) {
    $('<div class="filter"></div>').text(keyword).bind('click',
        {'keyword': keyword}, function(event) {
        $table.find('tbody tr').each(function() {
            if ($('td', this).filter(':nth-child(' + (column + 1) +
                ')').text() == event.data['keyword']) {
                $(this).show();
            }
            else if ($('th', this).length == 0) {
                $(this).hide();
            }
        });
        $(this).addClass('active').siblings().removeClass('active');
    }).addClass('clickable').appendTo($filters);
});
});
```

首先，定义了一个包含筛选关键字的静态数组，然后，通过循环遍历为每个关键字创建一个按钮。同分页的例子一样，这里也需要使用.bind()方法为事件对象添加data参数，以避免意外的闭包问题。随后，在单击处理程序中，通过将每个单元格与关键字进行比较，隐藏那些不匹配的行。同时，还必须检查不匹配的行是不是一个副标题行，以免在此过程中隐藏它们。

现在这两个按钮都实现了相应的筛选功能，如图7-25所示。

The screenshot shows a table with columns: Date, Headline, Author, and Topic. The table has three rows of data. To the right of the table is a sidebar titled 'Filter by Topic' with two buttons: 'conference' and 'release'. The 'conference' button is highlighted.

Date	Headline	Author	Topic
Feb 28	jQuery 1.1.2	John Resig	release
Dec 12	jQuery 1.0.4	John Resig	release
Oct 27	jQuery 1.0.3	John Resig	release
Oct 13	Minor API Changes in 1.0.2	John Resig	release
2006			

图 7-25

1. 从内容中收集筛选选项

接下来，我们要扩展筛选选项以包含表格中所有可用的主题。这里，我们不是要硬编码所有的主题，而是要从表格中已经存在的文本中收集这些主题。下面，我们把keywords的定义修改为：

```
var keywords = {};
$table.find('tbody tr td').filter(':nth-child(' + (column + 1) +
    ')').each(function() {
    keywords[$(this).text()] = $(this).text();
});
```

以上代码依赖于如下两个技巧。

- 通过使用映射而不是数组来保存找到的关键字，可以自动排除副本。
- jQuery的\$.each()函数既可以操作数组也可以操作映射，因此不必修改后面的代码。现在，我们已经补足了所有的筛选选项，如图7-26所示。

Date	Headline	Author	Topic	Filter by Topic:
Feb 28, 2007	jQuery 1.1.2	John Resig	release	conference release standards
Dec 12, 2006	jQuery 1.0.4	John Resig	release	announcement plug-in
Oct 27, 2006	jQuery 1.0.3	John Resig	release	miscellaneous source
Oct 13, 2006	Minor API Change in 1.0.2	John Resig	release	tutorial documentation third-party
Oct 12, 2005				

图 7-26

2. 反向筛选

出于完整性的考虑，我们还需要添加筛选后恢复完整列表的方式。而添加针对所有主题的筛选选项很简单：

```
$('<div class="filter">all</div>').click(function() {
    $table.find('tbody tr').show();
    $(this).addClass('active').siblings().removeClass('active');
}).addClass('clickable active').appendTo($filters);
```

这样，就添加了一个all按钮，通过它可以再次显示所有的表格行。此外，我们还将这个按钮标记为初始时的活动按钮。

7.8.2 同其他代码整合

通过编写排序和分页的代码，我们知道前面实现的这些功能并不是孤立存在的。事实上，我们构建的这些行为能够以某种特别的方式进行交互，为此，有必要回顾一下前面的工作，以便找出整合它们与刚添加的新筛选功能的方法。

1. 行条纹效果

由于新添加了筛选功能，前面运行正常的高级行条纹效果出现了问题。主要表现在执行完一次筛选后，表格行都保持着原来的颜色，就好像筛选掉的行仍然存在一样。

为处理筛选掉的行，实现条纹效果的代码必须能够找到它们。为此，我们可以为筛选掉的行添加一个类：

```

$(document).ready(function() {
    $('table.filterable').each(function() {
        var $table = $(this);

        $table.find('th').each(function (column) {
            if ($(this).is('.filter-column')) {
                var $filters = $('

<h3>Filter by ' +
                    $(this).text() + ':</h3></div>');
                var keywords = {};
                $table.find('tbody tr td').filter(':nth-child(' + (column +
                    1) + ')').each(function() {
                    keywords[$(this).text()] = $(this).text();
                });
                $('<div class="filter">all</div>').click(function() {
                    $table.find('tbody tr').show().removeClass('filtered');
                    $(this).addClass('active').siblings().removeClass('active');
                    $table.trigger('stripe');
                }).addClass('clickable active').appendTo($filters);
                $.each(keywords, function (index, keyword) {
                    $('<div class="filter"></div>').text(keyword).bind('click',
                        {'keyword': keyword}, function(event) {
                        $table.find('tbody tr').each(function() {
                            if ($('td', this).filter(':nth-child(' + (column + 1) +
                                ')').text() == event.data['keyword']) {
                                $(this).show().removeClass('filtered');
                            }
                            else if ($('th', this).length == 0) {
                                $(this).hide().addClass('filtered');
                            }
                        });
                        $table.removeClass('active').siblings().removeClass('active');
                        $table.trigger('stripe');
                    }).addClass('clickable').appendTo($filters);
                });
                $filters.insertBefore($table);
            }
        });
    });
});


```

只要当前的筛选器改变，就会触发stripe事件。这里，我们使用了分页指示器与排序例程协作时曾经使用过的一个技巧——添加了一个新的自定义事件。为此，必须重写实现条纹效果的代码以便定义这个事件：

```

$(document).ready(function() {
    $('table.striped').each(function() {
        $(this).bind('stripe', function() {

```

```

var rowIndex = 0;
$('tbody tr:not(.filtered)', this).each(function(index) {
    if($('th', this).length) {
        $(this).addClass('subhead');
        rowIndex = -1;
    } else {
        if(rowIndex % 6 < 3) {
            $(this).removeClass('odd').addClass('even');
        }
        else {
            $(this).removeClass('even').addClass('odd');
        }
    };
    rowIndex++;
});
});
$(this).trigger('stripe');
});
});
});

```

现在，查找表格行的选择符跳过了被筛选掉的行。而且，因为这些代码可能会被多次执行，所以还必须移除行中废弃不用的类。在定义了新的事件处理程序及相应的触发器之后，筛选操作与条纹效果紧密结合起来，如图7-27所示。

Date	Headline	Author	Topic	Filter by Topic:
2007				conference
Feb 28	jQuery 1.1.2	John Resig	release	release
2006				standards
Dec 12	jQuery 1.0.4	John Resig	release	announcement
Oct 27	jQuery 1.0.3	John Resig	release	plug-in
Oct 13	Minor API Change in 1.0.2	John Resig	release	miscellaneous
2005				source
				tutorial
				documentation
				third-party

图 7-27

2. 扩展和折叠

前面添加的扩展和折叠行为同样和筛选操作存在冲突。也就是说，当表格中的某一部分处于折叠状态时，如果单击了另一个筛选按钮，那么匹配的条目都会显示出来——即使该条目位于折叠的部分中。反之，如果在表格处于筛选后的状态时扩展某个部分，那么扩展后的部分中将显示出所有条目——不论这些条目是否与筛选器匹配。

由于通过筛选按钮移除表格行时，已经为这些行都添加了`filtered`类，因此可以在折叠脚本的单击处理程序内部检查这个类：

```

var toggleSrc = $(this).attr('src');
if ( toggleSrc == toggleMinus ) {
    $(this).attr('src', togglePlus)
    .parents('tr').siblings().addClass('collapsed').fadeOut('fast');
} else{
    $(this).attr('src', toggleMinus)
    .parents('tr').siblings().removeClass('collapsed')
        .not('.filtered').fadeIn('fast');
};

}

```

而且，当折叠或扩展行时，也向相应的行中添加或移除了另一个新类。下面，就需要通过这个类来解决另一半问题。换句话说，筛选代码可以使用这个类来确定当筛选条件改变时应该显示哪一行：

```

$tbody.find('tbody tr').each(function() {
    if ($('td', this).filter(':nth-child(' + (column + 1) + ')').text()
        == e.data['keyword']) {
        $(this).removeClass('filtered').not('.collapsed').show();
    }
    else if ($('th', this).length == 0) {
        $(this).addClass('filtered').hide();
    }
});

```

现在，所有功能都运转正常，每一个组件都可以独立地隐藏和显示表格行。

7.9 完成的代码

到目前为止，第2个例子页面中已经实现的功能包括表格行的条纹效果、突出显示效果、工具提示条、折叠/扩展和筛选。把这些代码放到一起，整个页面中所有的JavaScript代码如下所示：

```

$(document).ready(function() {
    var highlighted = "";
    var column = 3;

    var positionTooltip = function(event) {
        var tPosX = event.pageX;
        var tPosY = event.pageY + 20;
        $('div.tooltip').css({top: tPosY, left: tPosX});
    };
    var showTooltip = function(event) {
        $('div.tooltip').remove();
        var $thisAuthor = $(this).text();
        if ($(this).parent().is('.highlight')) {
            highlighted = 'un-';
        } else {
            highlighted = '';
        };
    };
}

```

```

    $('<div class="tooltip">Click to ' + highlighted +
      'highlight all articles written by ' +
      $thisAuthor + '</div>').appendTo('body');
    positionTooltip(event);
  });
  var hideTooltip = function() {
    $('div.tooltip').remove();
  };

  $('table.striped td:nth-child(' + column + ')')
    .addClass('clickable')
    .click(function(event) {
      var thisClicked = $(this).text();
      $('table.striped td:nth-child(' + column + ')')
        .each(function(index) {
          if (thisClicked == $(this).text()) {
            $(this).parent().toggleClass('highlight');
          } else {
            $(this).parent().removeClass('highlight');
          }
        })
      showTooltip.call(this, event);
    })
    .hover(showTooltip, hideTooltip)
    .mousemove(positionTooltip);
  });

$(document).ready(function() {
  $('table.striped').each(function() {
    $(this).bind('stripe', function() {
      var rowIndex = 0;
      $('tbody tr:not(.filtered)', this).each(function(index) {
        if ($('th', this).length) {
          $(this).addClass('subhead');
          rowIndex = -1;
        } else {
          if (rowIndex % 6 < 3) {
            $(this).removeClass('odd').addClass('even');
          } else {
            $(this).removeClass('even').addClass('odd');
          }
        }
        rowIndex++;
      });
    });
    $(this).trigger('stripe');
  });
});

```

```

$(document).ready(function() {
    $('table.filterable').each(function() {
        var $table = $(this);

        $table.find('th').each(function (column) {
            if ($(this).is('.filter-column')) {
                var $filters = $('

<h3>Filter by ' +
                               $(this).text() + ':</h3></div>');
                var keywords = {};

                $table.find('tbody tr td').filter(':nth-child(' + (column +
                    1) + ')').each(function() {
                    keywords[$(this).text()] = $(this).text();
                })

                $('<div class="filter">all</div>').click(function() {
                    $table.find('tbody tr').removeClass('filtered')
                        .not('.collapsed').show();
                    $(this).addClass('active').siblings().removeClass('active');
                    $table.trigger('stripe');
                }).addClass('clickable active').appendTo($filters);

                $.each(keywords, function (index, keyword) {
                    $('<div class="filter"></div>').text(keyword).bind('click',
                        {'keyword': keyword}, function(event) {
                        $table.find('tbody tr').each(function() {
                            if ($('td', this).filter(':nth-child(' + (column + 1)
                                + ')').text() == event.data['keyword']) {
                                $(this).removeClass('filtered').not('.collapsed')
                                    .show();
                            }
                            else if ($('th', this).length == 0) {
                                $(this).addClass('filtered').hide();
                            }
                        });
                        $(this).addClass('active').siblings().removeClass(
                            'active');
                        $table.trigger('stripe');
                    }).addClass('clickable').appendTo($filters);
                });
                $filters.insertBefore($table);
            }
        });
    });
});

$(document).ready(function() {


```

```
var toggleMinus = '../icons/bullet_toggle_minus.png';
var togglePlus = '../icons/bullet_toggle_plus.png';
var $subHead = $('tbody th:first-child');
$subHead.prepend('');

$('img', $subHead).addClass('clickable')
.click(function() {
    var toggleSrc = $(this).attr('src');
    if ( toggleSrc == toggleMinus ) {
        $(this).attr('src', togglePlus)
        .parents('tr').siblings().addClass('collapsed').fadeOut('fast');
    } else {
        $(this).attr('src', toggleMinus)
        .parents('tr').siblings().removeClass('collapsed')
        .not('.filtered').show().fadeIn('fast');
    }
});
})
```

7.10 小结

本章中，我们探索了对网站上的表格进行交叉操作的一些方式，将它们改造成了兼具漂亮界面和实用功能的数据容器。介绍了如何对表格中的数据进行排序，包括使用不同的数据类型（文字、数值、日期）作为排序关键字；以及对表格应用分页技术，将大型表格分割成便于查看的数据块。学习了复杂的行条纹技术和JavaScript驱动的工具提示条。也讨论了扩展和折叠表格，以及根据给定的标准筛选和突出显示表格行的内容。

此外，本章我们也接触到了一些比较高级的主题，例如通过服务器端代码和AJAX技术实现排序和分页，动态计算元素在页面上的坐标。而且，还编写了一个jQuery插件。

通过本章的学习，我们看到在具有适当语义的HTML表格中，隐藏着大量的细节和复杂性。好在，jQuery能够帮我们驯服这些小生灵，使得表格式数据的能量得以充分地释放出来。

构建功能型表单

*I'm shoutin'
We're waiting for a reply
—Devo,
"Shout"*

几乎每一个要求用户反馈的网站都会以某种形式使用表单。自因特网诞生至今，表单就在扮演驮骡的角色，它负责把信息从终端用户运送给网站的发布者——踏实肯干、任劳任怨，但外表或风度却不尽如人意。这种天资的不足可能是因为往返服务器的旅途单调而辛苦，也可能同表单必须与之合作的不妥协的因素有关，或者只是它们保守而不赶时髦的本性所致。不管是什么原因，这一切在前不久已经得到了改观，随着客户端脚本编程的复兴，表单被注入了新的活力、用途和形象。本章中，我们将探索为表单赋予生机的各种方式，包括增强它们的样式、为它们创建验证例程、使用它们进行计算，以及在无人值守的情况下把它们的结果发送到服务器。

8.1 渐进增强的表单设计

当在网站中使用jQuery时，我们必须时常提醒自己如果用户禁用了JavaScript，那么页面看起来会怎样、功能是否还健全（当然，除非我们知道用户是谁，而且知道他们会怎样配置浏览器）。但是，这并不意味着我们不能为启用JavaScript的用户创建更美观或者功能更强大的网站。渐进增强的原则在JavaScript开发者中间如此流行，就是因为它在为多数人提供额外功能的同时，还能照顾到全体用户的需求。

下面我们就来创建一个表单，即一个联系表单，通过它来示范渐进增强在外观和行为两方面的应用。在JavaScript无效的情况下，表单的第一个控件组（fieldset）如图8-1所示。

尽管表单中有足够的信息可以引导用户填写每个字段，换句话说，它能够正常使用；但是，显然这个表单也有很多需要改进之处。我们要从以下3个方面来渐进增强这组表单控件。

- (1) 修改DOM以便灵活地为<legend>元素应用样式。
- (2) 把必填字段的提示信息修改为星号(*)，把特殊字段（只在相应的复选框被选中时需要

填写)的提示信息修改为双星号 (**). 将这两种必填字段的标签修改为粗体字, 再在表单上方放一条解释星号和双星号含义的说明。

(3) 在页面加载时隐藏每个复选框对应的文本输入框, 当用户选择或取消选择复选框时切换这些文本输入框——让它们显示或者隐藏。

图 8-1

我们首先来看一看`<fieldset>`元素中的HTML代码:

```

<fieldset>
    <legend>Personal Info</legend>
    <ol>
        <li><label for="first-name">First Name</label><input
            class="required" type="text" name="first-name"
            id="first-name" /> <span>(required)</span></li>
        <li><label for="last-name">Last Name</label><input
            class="required" type="text" name="last-name"
            id="last-name" /> <span>(required)</span></li>
        <li>How would you like to be contacted? (choose at least one
            method)
            <ul>
                <li><label for="by-email"><input type="checkbox"
                    name="by-contact-type" value="E-mail" id="by-email" />
                    by E-Mail</label>
                    <input class="conditional" type="text" name="email"
                        id="email" /> <span>(required when corresponding
                        checkbox checked)</span></li>
                <li><label for="by-phone"><input type="checkbox" name="by-
                    contact-type" value="Phone" id="by-phone" />
                    by Phone</label>
                    <input class="conditional" type="text" name="phone"
                        id="phone" /> <span>(required when corresponding
                        checkbox checked)</span></li>
                <li><label for="by-fax"><input type="checkbox" name="by-
                    contact-type" value="Fax" id="by-fax" /> by Fax</label>
                    <input class="conditional" type="text" name="fax" id="fax"
                        /> <span>(required when corresponding checkbox
                        checked)</span></li>
            </ul>
        </li>
    </ol>

```

```
</ol>
</fieldset>
```

这里要注意的是，每个表单元素或者元素对都包含在一个列表项（``）中。所有元素被包含在一个有序列表（``）中，而复选框（以及相应的文本字段）被包含在一个嵌套的无序列表（``）中。而且，我们使用`<label>`元素标出了每个字段的名称。对于文本字段，`<label>`放在`<input>`前面；对于复选框，`<label>`包含`<input>`。

在了解了HTML的结构之后，现在就可以通过jQuery来实现渐进增强了。

8.1.1 图标符号

表单的图标符号（`<legend>`）难以通过CSS添加样式是出了名的。浏览器实现的不一致性和定位方法的限制，使得处理图标符号常常成为一件令人沮丧的事。然而，如果我们注意使用有意义的、结构良好的页面元素，那么图标符号也是一个用于在表单的`<fieldset>`中显示标题的有吸引力的元素（如果不是要求美观的满足）。

如果只有HTML和CSS，我们只能寄希望于语义化的标记或者灵活的设计。但是现在，我们可以在页面加载时修改HTML，对于查看页面的人而言，把每个`<legend>`变成一个`<h3>`，而对于读取该页面的计算机（以及禁用JavaScript的用户）来说，看到的仍然是`<legend>`。

对每个`<fieldset>`，我们要取得`<legend>`元素中包含的文本并保存到一个变量中，然后移除`<legend>`：

```
$(document).ready(function() {
    $('fieldset').each(function() {
        var heading = $('legend', this).remove().text();
    });
});
```

这里使用`.each()`方法，是因为完整的联系表单中包含3个`<fieldset>`元素。如果我们操作的是只包含一个控件组的表单，可以简单地依靠jQuery的隐式迭代机制。同时也要注意，针对`heading`变量使用的选择符表达式是`($('legend', this))`。由于每次迭代一个`<fieldset>`都会设置一次`heading`，所以需要使用`this`作为`<legend>`的环境选择符，以确保每次只取得一个`<legend>`中的文本。否则，第一次迭代就会取得全部3个`<legend>`中的文本，而第2次和第3次则什么也得不到。

接着，创建`<h3>`元素，将它插入到每个`<fieldset>`的开始处，同时将保存在`heading`变量中的文本放入其中：

```
$(document).ready(function() {
    $('fieldset').each(function() {
        var heading = $('legend', this).remove().text();
        $('

### ' + heading + '

')
            .text(heading)
});
```

```

    .prependTo( this );
  });
});

```

这里我们采取了一种费事的创建和插入新元素的方式——首先创建这个元素，然后插入文本，最后再向文档中添加该元素，使用了3行代码。而完成同样的任务，也可以只用一行代码，比如：

```
$(this).prepend('<h3>' + heading + '</h3>');
```

但是，编写3行码却降低了出错的概率：通过使用`.text()`方法可以保证对任何特殊的HTML字符都进行适当的转义。

现在，当在样式表中为这个`<h3>`元素应用蓝色的背景和白色的文本颜色后，表单的第一个控件组将如图8-2所示。

图 8-2

表单的`legend`元素按照我们的意图添加了足够的样式。下面应该着手清理必填字段的提示信息了。

8.1.2 必填字段的提示信息

在这个联系表单中，必填字段都带有`class="required"`属性以便应用样式和响应用户的输入；而每种联系方式的输入字段都带有`class="conditional"`属性。我们就是要通过这些类来修改每个输入框右侧圆括号中的使用说明。

首先，我们来设置变量`requiredFlag`和`conditionalFlag`，然后再向每个必填和条件字段后面的``元素中填入这两个变量中保存的文本：

```

$(document).ready(function() {
  var requiredFlag = ' * ';
  var conditionalFlag = ' ** ';

  $('form :input').filter('.required')
    .next('span').text(requiredFlag);
}

```

```

  $('form :input').filter('.conditional')
    .next('span').text(conditionalFlag);
});

```

由于一个星号(*)可能不会立即吸引用户的注意力，所以还需要为每个必填字段的<label>添加class="req-label"，并为这个类应用font-weight:bold样式：

```

$(document).ready(function() {
  var requiredFlag = ' * ';
  var conditionalFlag = ' ** ';

  $('form :input').filter('.required')
    .next('span').text(requiredFlag).end()
    .prev('label').addClass('req-label');

  $('form :input').filter('.conditional')
    .next('span').text(conditionalFlag);
});

```

为了适当地选择label，必须在前一行中添加.end()方法。整个方法链首先选择全部表单输入字段，再筛选出只包含class="required"的字段，然后选择直接位于筛选出的输入字段后面的元素。而为这个方法链添加.end()方法，会使选择符表达式向后倒退一步；换句话说，退回到所有带class="required"属性的表单输入字段。因此，尾随的.prev('label')才会按照预期找到元素。现在，修改了文本并添加了类之后的字段组的外观如图8-3所示。

图 8-3

还不错。不过，required字段和conditional字段后面的提示信息其实还是很有用的——只不过重复的次数太多罢了。下面我们就来取得每条提示信息的第一个实例，并将它们显示到表单上方、表示它们的形象化“标记”的旁边。

在生成保存提示信息及相应标记的元素之前，需要把初始的信息保存到两个变量中。然后，可以使用正则表达式来去掉文本中的圆括号：

```

$(document).ready(function() {
    var requiredFlag = ' * ';
    var requiredKey = $('input.required:first').next('span').text();
    requiredKey = requiredFlag + requiredKey
        .replace(/^((.+)\)$/, "$1");
    var conditionalFlag = ' ** ';
    var conditionalKey = $('input.conditional:first').next(
        'span').text();
    conditionalKey = conditionalFlag + conditionalKey
        .replace(/^((.+)\)$/, "$1");

    $('form :input').filter('.required')
        .next('span').text(requiredFlag).end()
        .prev('label').addClass('req-label');

    $('form :input').filter('.conditional')
        .next('span').text(conditionalFlag);
});

```

新增代码的第1行用于将信息中的文本保存到变量中。第2行用于将每个标记与相应的信息（去掉了圆括号）连接起来。这里用到的正则表达式以及字符串的.replace()方法，可能需要进一步解释一下。

1. 题外话：正则表达式

上面代码中的正则表达式，包含在`/^((.+)\)$/`的两个斜杠之间。第1个字符`^`，表示后面跟着的应该是字符串的开始位置。`^`后面的两个字符`\(`好像是开始圆括号。但其中用于转义的反斜杠，会告诉正则表达式解释程序按照字面含义解释它后面的字符。由于圆括号在正则表达式中是一个具有特殊含义的字符（后面会介绍到），因此对它转义是必要的。再后面的4个字符是`(.+)`，用于查找一个或多个`(+)`同一行中的任意字符`(.)`。最后3个字符`\)$`，用于查找字符串结尾处的结束圆括号。因此，整个正则表达式用于选择一个开始圆括号，后跟一组字符，并且以一个结束圆括号结尾。

另一方面，.replace()方法会在特定的环境中查找由一个正则表达式描述的字符串，并用另一个字符串替换找到的字符串。相应的语法如下所示：

```
context-string.replace(/regular-expression/, "replacement-string")
```

前面代码中调用.replace()方法的两个环境字符串分别是变量`requiredKey`和`conditionalKey`。随后是包含在两个斜杠中的正则表达式，我们刚刚分析过了。然后，是分隔正则表达式和替换字符串的逗号。最后是替换字符串——在前面的代码中两个替换字符串都是`$1`。作为占位符的`$1`，表示的是正则表达式中的第一个分组。也就是说，这里使用的正则表达式中有一个包含一个或多个字符的分组，这个分组两边有一对圆括号。`$1`表示的用作替换字符串的这个分组，代表的是位于圆括号内部但不包含圆括号的所有字符。

2. 插入字段提示信息

既然已经取得了不带圆括号的字段提示信息，下面就可以将这些信息连同相应的标记一块插入到表单上方了：

```
$ (document).ready(function() {
    var requiredFlag = ' * ';
    var requiredKey = $('input.required:first').next('span').text();
    requiredKey = requiredFlag + requiredKey.replace(/^\((.+)\)$/, "$1");

    var conditionalFlag = ' ** ';
    var conditionalKey =
        $('input.conditional:first').next('span').text();
    conditionalKey = conditionalFlag +
        conditionalKey.replace(/\((.+)\)/, "$1");

    $('form :input').filter('.required')
        .next('span').text(requiredFlag).end()
        .prev('label').addClass('req-label');

    $('form :input').filter('.conditional')
        .next('span').text(conditionalFlag);

    $('<p></p>')
        .addClass('field-keys')
        .append(requiredKey + '<br />')
        .append(conditionalKey)
        .insertBefore('#contact');
});
```

新添加的5行代码现在看起来应该不陌生。它们完成的任务如下：

- (1) 创建一个新的段落元素。
- (2) 为这个段落添加field-keys类。
- (3) 将requiredKey和一个换行标记添加到这个段落中。
- (4) 将conditionalKey添加到这个段落中。
- (5) 将这个段落及刚才添加到段落中的一切插入到联系表单的前面。

当像我们这里一样，在使用.append()方法添加HTML字符串标记时，要注意对其中特殊的HTML字符进行转义处理。不过，在上面的代码中，.text()方法已经替我们完成了这一操作。

在通过样式表为.field-keys应用了一些样式之后，结果如图8-4所示。

为第一个字段组编写的jQuery代码差不多就要完成了。

* required
** required when corresponding checkbox checked

Personal Info	
First Name	*
Last Name	*
How would you like to be contacted? (choose at least one method)	
<input type="checkbox"/> by E-Mail	**
<input type="checkbox"/> by Phone	**
<input type="checkbox"/> by Fax	**

图 8-4

8.1.3 根据条件显示的字段

下面，我们再围绕询问用户愿意使用哪种联系方式来进一步增强这组字段。因为只有当用户选择了相应的复选框之后，才需要填写后面的文本输入字段，所以可以在文档加载完成时首先隐藏它们：

```
$ (document) . ready (function () {
    $ ('input.conditional') . hide () . next ('span') . hide ();
});
```

图8-5就是对这个字段组精简后的界面。

Personal Info	
First Name	*
Last Name	*
How would you like to be contacted? (choose at least one method)	
<input type="checkbox"/> by E-Mail	
<input type="checkbox"/> by Phone	
<input type="checkbox"/> by Fax	

图 8-5

为显示文本输入字段和相应的提示标记，需要给每个复选框添加.click()方法。为了便于设置两个可重用的变量，我们在每个条件文本输入字段的环境中添加这个行为：

```

$(document).ready(function() {
    $('input.conditional').hide().next('span').hide();
    $('input.conditional').each(function() {
        var $thisInput = $(this);
        var $thisFlag = $thisInput.next('span').hide();
        $thisInput.prev('label').find(':checkbox').click(function() {
            // .....省略的代码.....
        });
    });
});

```

这样，我们就设置了保存着当前文本输入字段和当前标记的变量。当用户单击复选框时，需要检查复选框是否被选中；如果是，则显示文本输入字段，显示提示标记，然后再为父元素<label>添加req-label类：

```

$(document).ready(function() {
    $('input.conditional').hide().next('span').hide();
    $('input.conditional').each(function() {
        var $thisInput = $(this);
        var $thisFlag = $thisInput.next('span').hide();
        $thisInput.prev('label').find(':checkbox').click(function() {
            if (this.checked) {
                $thisInput.show();
                $thisFlag.show();
                $(this).parent('label').addClass('req-label');
            };
        });
    });
});

```

在测试复选框是否被选中时，首选的方案应该是检查this.checked，因为通过this关键字可以直接访问这个DOM节点。当不能直接访问DOM节点时，我们可以使用\$('selector').is(':checked')来代替，因为.is()返回一个布尔值(true或false)。

现在所剩的操作就是添加一个else语句，以便在复选框没有被选中时，隐藏条件元素并移除req-label类：

```

$(document).ready(function() {
    $('input.conditional').hide().next('span').hide();
    $('input.conditional').each(function() {
        var $thisInput = $(this);
        var $thisFlag = $thisInput.next('span').hide();
        $thisInput.prev('label').find(':checkbox').click(function() {
            if (this.checked) {
                $thisInput.show();
                $thisFlag.show();
            } else {
                $thisInput.hide();
                $thisFlag.hide();
            }
        });
    });
});

```

```
    $(this).parent('label').addClass('req-label');
} else {
    $thisInput.hide();
    $thisFlag.hide();
    $(this).parent('label').removeClass('req-label');
},
});
});
});
});
```

至此，增强表单过程中添加样式的一部分就结束了。下面，我们要为表单添加一些客户端验证功能。

8.2 表单验证

在通过jQuery向表单添加验证功能之前，必须记住一条重要的规则：客户端验证不能取代服务器端验证。同样，也不能依赖用户启用JavaScript。如果我们真想要求必须填写或者以特殊的格式填写某些字段，只靠JavaScript不能保证得到想要的结果。有的用户喜欢禁用JavaScript，有的设备可能不支持JavaScript，而且，少数用户还会绕过JavaScript的限制故意提交恶意数据。

8.2.1 即时反馈

既然如此，为什么还要通过jQuery实现验证功能呢？使用jQuery的客户端表单验证具有服务器端验证无法比拟的一个优势——即时反馈。服务器端代码，无论是ASP、PHP，还是其他什么富有想象力的缩写词，都需要重载页面才能生效（除非是异步访问，当然，异步访问仍然需要JavaScript）。通过jQuery，可以在必填字段失去焦点(blur)或者用户按下某个键盘按键(keyup)时，利用灵活的客户端代码实现验证功能。

1. 必填字段

对于我们例子中的联系表单来说，可以在用户按下Tab键或在输入字段外部单击时，检查每个输入字段的required类。不过，在开始编写检查代码之前，我们应该简单地回顾一下条件文本字段。为了简化验证例程，我们要在<input>显示时为它添加required类，当<input>隐藏时再移除这个类。这一部分的代码如下所示：

```
$thisInput.prev('label').find(':checkbox').click(function() {
    if (this.checked) {
        $thisInput.show().addClass('required');
        $thisFlag.show();
        $(this).parent('label').addClass('req-label');
    } else {
        $thisInput.hide().removeClass('required');
        $thisFlag.hide();
        $(this).parent('label').removeClass('req-label');
```

```
    };
});
```

在添加了所有required类之后，就可以准备在用户留空这些字段时给出的反馈了。此时，应该在必填标记后面放入一条信息，而且要通过class="warning"使字段所在的取得样式，以便对用户给出警示：

```
$(document).ready(function() {
    $('form :input').blur(function() {
        if ($(this).is('.required')) {
            var $listItem = $(this).parents('li:first');
            if (this.value == '') {
                var errorMessage = 'This is a required field';
                $('</span>')
                    .addClass('error-message')
                    .text(errorMessage)
                    .appendTo($listItem);
                $listItem.addClass('warning');
            }
        }
    });
});
```

以上代码会在每个表单输入字段发生blur事件时检查两个if语句：首先检查required类，然后检查空字符串。如果两个条件都满足，则构建一条错误信息放在中，然后再将它们添加到父元素中。

如果我们想对条件文本字段给出一条不同的信息——只在对应的复选框被选中时必填，可以把限定信息连接到标准的错误信息上。这样，还需要在代码中再添加一条if语句，只在前两个if条件满足的情况下检查conditional类：

```
$(document).ready(function() {
    $('form :input').blur(function() {
        if ($(this).is('.required')) {
            var $listItem = $(this).parents('li:first');
            if (this.value == '') {
                var errorMessage = 'This is a required field';
                if ($(this).is('.conditional')) {
                    errorMessage += ', when its related checkbox is checked';
                }
                $('</span>')
                    .addClass('error-message')
                    .text(errorMessage)
                    .appendTo($listItem);
                $listItem.addClass('warning');
            }
        }
    });
});
```

当用户留空一个字段时，这些代码运行得很好。然而，当用户随后填写了这个字段并离开该字段时，就会出现两个明显的问题，如图8-6所示。

The screenshot shows a 'Personal Info' form with several fields. The 'First Name' field contains 'Karl'. The 'Last Name' field is empty and has a red asterisk next to it, with the text 'This is a required field' below it. Below these, there is a question 'How would you like to be contacted? (choose at least one method)'. Underneath, there are three checkboxes: 'by E-Mail' (checked), 'by Phone' (unchecked), and 'by Fax' (unchecked). A tooltip message 'This is a required field, when its related checkbox is checked' is displayed above the 'by E-Mail' checkbox.

图 8-6

如果字段仍然是空的，错误信息会重复显示与用户离开字段一样多的次数。如果字段中已经填写了文本，不会移除class="warning"。显然，我们只想让一个字段显示一条信息，而且，当用户修改了错误时还应该移除相应的信息。为解决这两个问题，可以在运行验证检查的代码之前，首先响应字段失去焦点的事件，移除当前字段的父元素``中的class="warning"和位于同一个``元素中的所有``标签：

```
$ (document).ready(function() {
    $('form :input').blur(function() {
        $(this).parents('li:first').removeClass('warning')
            .find('span.error-message').remove();
        if ($(this).is('.required')) {
            var $listItem = $(this).parents('li:first');
            if (this.value == '') {
                var errorMessage = 'This is a required field';
                if ($(this).is('.conditional')) {
                    errorMessage += ', when its related checkbox is checked';
                }
                $('')
                    .addClass('error-message')
                    .text(errorMessage)
                    .appendTo($listItem);
                $listItem.addClass('warning');
            };
        };
    });
});
```

最后，我们得到了针对必填字段和条件必填字段进行验证的有效脚本。即使重复地填写和离开必填字段，错误信息也能够正确地显示，如图8-7所示。

First Name

Last Name

How would you like to be contacted? (choose at least one method)

by E-Mail ** This is a required field, when its related checkbox is checked

by Phone ** This is a required field, when its related checkbox is checked

by Fax

图 8-7

不过等等!我们还要在用户取消对一个复选框的选定时,移除``的`warning`类和它的``元素。为此,我们还要修改前面编写的复选框代码,以便当复选框取消选定时,在相应的文本字段上触发`blur`事件:

```
if (this.checked) {
    $thisInput.show().addClass('required');
    $thisFlag.show();
    $(this).parent('label').addClass('req-label');
} else {
    $thisInput.hide().removeClass('required').blur();
    $thisFlag.hide();
    $(this).parent('label').removeClass('req-label');
};
```

这样,当取消对一个复选框的选定时,相关的警告样式和错误信息将会从我们的视野中也从我们的脑海中消失。

2. 必要的格式

对于我们的联系表单来说,还需要实现另一种类型的验证——纠正输入的格式。有时候,如果填写到某个字段中的文本不正确(不是简单的留空),也需要提供警告信息。适用于这种类型验证的表单字段主要有电子邮件、电话和信用卡。作为示范,我们介绍针对电子邮件字段的一个相对简单的正则表达式测试。在深入探讨正则表达式之前,我们先来看一看用于电子邮件验证的完整代码:

```
$(document).ready(function() {
// .....省略的代码.....
    if ($(this).is('#email')) {
        var $listItem = $(this).parents('li:first');
        if (this.value != '' && !/.+@.+\.+[a-zA-Z]{2,4}\$/ .test(this.value)) {
            var errorMessage = 'Please use proper e-mail format'
                + '(e.g. joe@example.com)';
            $('<span></span>')
```

```

    .addClass('error-message')
    .text(errorMessage)
    .appendTo($listItem);
    $listItem.addClass('warning');
}
};

// .....省略的代码.....
});

```

这些代码执行的任务如下。

- 测试电子邮件字段的id，如果测试成功，
- 将父列表项保存到一个变量中。
- 再用两个条件测试电子邮件字段——值不是空字符串，而且不匹配正则表达式；如果这两个测试成功：
- ◆ 创建一条错误信息。
- ◆ 将这条信息插入到>中。
- ◆ 将>元素及其内容添加到父列表项中。
- ◆ 为父列表项添加warning类。

现在，我们单独来看一下正则表达式：

```
!/.+@.+\.[a-zA-Z]{2,4}\$.test(this.value)
```

虽然这个正则表达式与我们在本章前面创建的那个相似，但这里使用了`.test()`方法而不是`.replace()`方法，因为我们只需要它返回`true`或`false`。同以前一样，这里的正则表达式也位于两个正斜杠之间。这个正则表达式用于测试位于`.test()`方法圆括号中的一个字符串，即这里的电子邮件字段的值。

在这个正则表达式中，我们看到了一组一个或多个非换行字符`(.+)`，后跟一个`@`符号，再后跟另一组一个或多个非换行字符。到目前为止，像`lucia@example`这样的字符串，以及其他类似的数百万种排列组合都能够通过测试。不过，这还不是一个有效的电子邮件地址。

为了使测试更精确，还需要在这个字符串的末尾查找一个`.`字符，后跟2~4个位于`a`和`z`之间的字符。这个正则表达式剩下的部分恰好可以完成这个任务。它首先查找一个位于`a`和`z`或`A`和`Z``([a-zA-Z])`之间的字符，然后要求位于该范围内的字母只能出现2~4次`{2,4}`。最后，它强调这2~4个字母要出现在字符串的末尾`(\$)`。现在，字符串`lucia@example.com`将返回`true`，而`lucia@example.2fn`、`lucia@example.example`或`lucia-example.com`则不会。

但是，我们希望只在用户填写的电子邮件地址格式不正确时返回`true`（并执行创建错误信息等操作）。这就是要在正则表达式前面加上一个感叹号（求反运算符）的原因：

```
!/.+@.+\.[a-zA-Z]{2,4}\$.test(this.value)
```

8.2.2 最终检查

现在，对联系表单进行验证的代码差不多快完成了。事实上，我们有必要在用户提交表单时再检查一遍表单中的字段，这次是整体性的检查。通过表单上的`.submit()`事件处理程序（而不是`Send`按钮），可以在所有必填字段上触发`blur`事件：

```
$(document).ready(function() {
    $('form').submit(function() {
        $('#submit-message').remove();
        $(":input.required").trigger('blur');
    });
});
```

你可能注意到了，我们悄悄地添加了一行代码，用于移除一个还不存在的元素。我们会在下一步中添加这个元素。之所以要在这里提前移除它，是因为根据本章前面生成多条错误信息的经验，我们知道需要这样做。

在触发了`blur`事件之后，我们取得了当前表单中包含的`warning`类的总数。如果存在`warning`类，就创建一个新的`id`为`submit-message`的`<div>`元素，并把它插入到`Send`按钮前面，因为这是用户最容易看到的地方。最后，还要阻止表单提交：

```
$(document).ready(function() {
    $('form').submit(function() {
        $('#submit-message').remove();
        $(":input.required").trigger('blur');
        var numWarnings = $('.warning', this).length;
        if (numWarnings) {
            $('').attr({'id': 'submit-message',
                'class': 'warning'})
                .append('Please correct errors with ' + numWarnings
                    + ' fields')
                .insertBefore('#send');
            return false;
        }
    });
});
```

除了提出修复错误的常规要求之外，这条信息也指出了需要修复的字段数目，如图8-8所示。

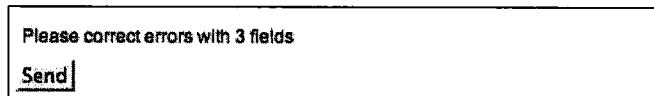


图 8-8

不过，我们还应该提供比这更好的反馈。除了显示错误的数目之外，还应该为用户列出包含错误的字段名称：

```

$(document).ready(function() {
    $('form').submit(function() {
        $('#submit-message').remove();
        $(":input.required").trigger('blur');
        var numWarnings = $('.warning', this).length;
        if (numWarnings) {
            var fieldList = [];
            $('.warning label').each(function() {
                fieldList.push($(this).text());
            });
            $('<div></div>')
                .attr({'id': 'submit-message', 'class': 'warning'})
                .append('Please correct errors with the following ' +
                        numWarnings + ' fields:<br />')
                .append('&bull; ' + fieldList.join('<br />&bull; '))
                .insertBefore('#send');
            return false;
        };
    });
});

```

首先，需要在代码中添加把fieldList变量声明为空数组的语句。然后，取得每个带warning类的元素的后代<label>元素，将该标签元素中的文本“推”到fieldList数组中（使用JavaScript本地的push函数）。这样，每个标签中的文本就构成了fieldList数组中一个独立的元素。

接着，我们又修改了前面创建的submit-message元素，将fieldList数组中的内容添加到这个<div>元素中。这里，我们使用JavaScript的本地函数join把数组转换成字符串，将每个数组元素与一个换行符和一个圆点符号连接了起来，如图8-9所示。

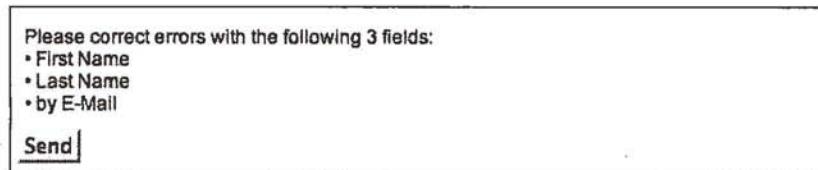


图 8-9

不错，这里字段列表的HTML标记只是为了显示而不具有语义。然而，对于一个临时性的列表——由JavaScript在提交表单的最后一步生成，而且随时可能废弃不用——我们出于简单明了的考虑，可以容许这点粗糙的代码。

8.3 复选框操作

为了更完整地增强联系表单，我们要帮助用户管理Miscellaneous部分的复选框列表。一个组中包含10个复选框会使人望而却步，尤其是当用户希望单击其中大多数复选框时。比如下列要求用户选择“知道我们的方式”的一组复选框，如图8-10所示。

The screenshot shows a web page with a title bar 'Miscellaneous'. Below it is a question: 'How did you discover us? (Check all that apply)'. A vertical list of checkboxes follows, each preceded by a small square checkbox icon. The options listed are: Magazine, Website, Television, Movie, School, Mom, Billboard, Graffiti, Detritus, and Hate Mail.

图 8-10

在这种情况下，提供一个选定或不选定所有复选框的选项肯定有用。因此，下面我们就来创建这样一个选项。

首先，要创建一个新元素，向其中填入一个<label>，在<label>中再放入<input type="checkbox" id="discover-all">和相应的文本，最后再把这个元素及其后代元素添加到<li class="discover">中的元素前面：

```
$(document).ready(function() {
  $('<li></li>')
    .html('<label><input type="checkbox" id="discover-all" />' +
          '<em>check all</em></label>')
    .prependTo('li.discover > ul');
});
```

这样，文档中就多出了一个新的带标签的check all复选框。不过，这个新复选框现在什么都还做不了。所以，接下来还要为它添加.click()方法：

```
$(document).ready(function() {
  $('<li></li>')
    .html('<label><input type="checkbox" id="discover-all" />' +
          '<em>check all</em></label>')
    .prependTo('li.discover > ul');

  $('#discover-all').click(function() {
    var $checkboxes = $(this).parents('ul:first').find(':checkbox');
    if (this.checked) {
      $checkboxes.attr('checked', 'true');
    } else {
      $checkboxes.attr('checked', '');
    };
  });
});
```

在这个事件处理程序中，我们首先设置了\$checkboxes变量，该变量中保存着包含当前列表中所有复选框的一个jQuery对象。设置完这个变量后，如果发现check all复选框被选中，则选中所有复选框，如果check all复选框未被选中，则取消对所有复选框的选定。

最后，可以为check all复选框的标签添加一些CSS样式，然后，在用户选中该复选框时将其文本修改为un-check all：

```
$(document).ready(function() {
    $('<li></li>')
        .html('<label><input type="checkbox" id="discover-all" /> <em>check
all</em></label>')
        .prependTo('li.discover > ul');
    $('#discover-all').click(function() {
        var $checkboxes = $(this).parents('ul:first').find(':checkbox');
        if (this.checked) {
            $(this).next().text(' un-check all');
            $checkboxes.attr('checked', 'true');
        } else {
            $(this).next().text(' check all');
            $checkboxes.attr('checked', '');
        }
    })
    .parent('label')
    .css({
        borderBottom: '1px solid #ccc',
        color: '#777',
        lineHeight: 2
    });
});
```

现在，这组复选框和check all复选框在页面上的效果如图8-11所示。

The screenshot shows a web page with a title bar "Miscellaneous". Below it is a heading "How did you discover us? (Check all that apply)". A list of checkboxes follows:

- check all
- Magazine
- Website
- Television
- Movie
- School
- Mom
- Billboard
- Graffiti
- Detritus
- Hate Mail

图 8-11

当选中check all复选框时，相应的界面外观如图8-12所示。

Miscellaneous
How did you discover us? (Check all that apply)

un-check all
 Magazine
 Website

图 8-12

8.4 完成的代码

下面，是完成之后的增强联系表单的代码：

```

$(document).ready(function() {

    // 增强表单元素的样式

    $('fieldset').each(function(index) {
        var heading = $('legend', this).remove().text();
        $('

### ' + heading + '

')
            .text(heading)
            .prependTo(this);
    });

    var requiredFlag = ' * ';
    var requiredKey = $('input.required:first').next('span').text();
    requiredKey = requiredFlag + requiredKey.replace(/^\((.+)\)\$/,"$1");
    var conditionalFlag = ' ** ';
    var conditionalKey =
        $('input.conditional:first').next('span').text();
    conditionalKey = conditionalFlag +
        conditionalKey.replace(/\((.+)\)/,"$1");

    $('form :input').filter('.required')
        .next('span').text(requiredFlag).end()
        .prev('label').addClass('req-label');

    $('form :input').filter('.conditional')
        .next('span').text(conditionalFlag);

    $('<p></p>')
        .addClass('field-keys')
        .append(requiredKey + '<br />')
        .append(conditionalKey)
        .insertBefore('#contact');

    // 条件文本输入字段，复选框切换
}
)

```

```

    $('input.conditional').hide().each(function() {
        var $thisInput = $(this);
        var $thisFlag = $thisInput.next('span').hide();
        $thisInput.prev('label').find(':checkbox').click(function() {
            if (this.checked) {
                $thisInput.show().addClass('required');
                $thisFlag.show();
                $(this).parent('label').addClass('req-label');
            } else {
                $thisInput.hide().removeClass('required').blur();
                $thisFlag.hide();
                $(this).parent('label').removeClass('req-label');
            };
        });
    });
}

// 在发生blur事件时验证字段

$('form :input').blur(function() {
    $(this).parents('li:first').removeClass('warning')
        .find('span.error-message').remove();

    if ($(this).is('.required')) {
        var $listItem = $(this).parents('li:first');
        if (this.value == '') {
            var errorMessage = 'This is a required field';
            if ($(this).is('.conditional')) {
                errorMessage += ', when its related checkbox is checked';
            };
            $('</span>')
                .addClass('error-message')
                .text(errorMessage)
                .appendTo($listItem);
            $listItem.addClass('warning');
        };
    };

    if ($(this).is('#email')) {
        var $listItem = $(this).parents('li:first');
        if (this.value != '' && !/.+@.+\.+[a-zA-Z]{2,4}$/,
            .test(this.value)) {
            var errorMessage = 'Please use proper e-mail format'
                + '(e.g. joe@example.com)';
            $('</span>')
                .addClass('error-message')
                .text(errorMessage)
                .appendTo($listItem);
            $listItem.addClass('warning');
        };
    };
}
);

```

```

    };
});

// 在提交时验证表单

$('form').submit(function() {
    $('#submit-message').remove();
    $(':input.required').trigger('blur');
    var numWarnings = $('.warning', this).length;
    if (numWarnings) {
        var fieldList = [];
        $('.warning label').each(function() {
            fieldList.push($(this).text());
        });
        $('')
            .attr({
                'id': 'submit-message',
                'class': 'warning'
            })
            .append('Please correct errors with the following ' +
                    numWarnings + ' fields:<br />')
            .append('&bull; ' + fieldList.join('<br />&bull; '))
            .insertBefore('#send');
        return false;
    };
});

// 复选框

$('form :checkbox').removeAttr('checked');

// 带(un)check all的复选框

$(<li></li>).html('<label><input type="checkbox" ' +
    + ' id="discover-all" /> <em>check all</em>' +
    + '</label>').prependTo('li.discover > ul');

$('#discover-all')
.click(function() {
    var $checkboxes = $(this).parents('ul:first').find(':checkbox');
    if (this.checked) {
        $(this).next().text(' un-check all');
        $checkboxes.attr('checked', 'true');
    } else {
        $(this).next().text(' check all');
        $checkboxes.attr('checked', '');
    };
});
.parent('label')
.css({
    borderBottom: '1px solid #ccc',
    color: '#777',
})

```

```

        lineHeight: 2
    });
});

```

尽管我们对本例中的联系表单进行了相当多的增强，但要做的仍然还有很多。例如，仅验证操作就可以分为很多类型。要找到一个更具灵活性的验证插件，请访问jQuery的插件仓库 <http://plugins.jquery.com/>。

8.5 字段的占位符文本

有些表单比联系表单要简单得多。事实上，许多网站在每个页面中都包含带一个字段的表单——网站的搜索表单。对页面中这个目的单纯的部分而言，通常的表单组件——字段标签、提交按钮和文本等，都显得多余。通过jQuery，我们可以为这样的表单“减肥”，只保留其应有的功能。

表单字段的label元素是维持网站可访问性的一个基本组件。每个字段都应该有一个对应的标签元素，因此屏幕阅读器和其他辅助设备能够据此以识别出每个字段都有什么用途。即使是在HTML源代码中，标签元素也有助于描述字段：

```

<form id="search" action="search/index.php" method="get">
    <label for="search-text">search the site</label>
    <input type="text" name="search-text" id="search-text" />
</form>

```

在不添加样式的情况下，标签位于字段的正上方，如图8-13所示。

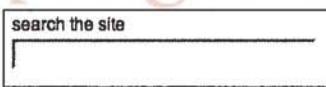


图 8-13

虽然标签在这里占据的空间不大，但在某些网站的布局中，即使是一行文本也太多了。可以通过CSS来隐藏这行文本，但这样会导致用户无法知悉字段的用途。为此，我们可以使用jQuery将这个标签转换成位于字段内部的占位符文本。

要实现这个转换，需要在DOM加载后移除标签并使用它的文本填充字段：

```

$(document).ready(function() {
    var searchLabel = $('#search-label').remove().text();
    $('#search-text').addClass('placeholder').val(searchLabel);
});

```

因为.remove()方法能够把一个元素从DOM树中拽出来而不删除它，所以可以在取得标签的文本之前先移除标签。取得的文本随即被设置为字段的值。而添加的类则使该文本变灰，以表明它是一个占位符，如图8-14所示。

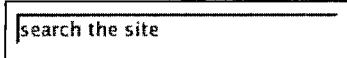


图 8-14

效果还不错，但是这样一来会给搜索字段造成一个不利的影响。由于字段的值已经改变，在字段中单击可以继续追加这个值的内容，而不是替换现有的值。这可能会导致搜索到意外的结果，如图8-15所示。

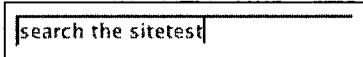


图 8-15

为避免这个问题，需要当字段获得焦点时移除其中的文本，而当字段失去焦点时再把文本放回原处。实现这个操作的代码很简单：

```
$(document).ready(function() {
    var searchLabel = $('#search label').remove().text();
    $('#search-text').addClass('placeholder').val(searchLabel)
        .focus(function() {
            if (this.value == searchLabel) {
                $(this).removeClass('placeholder').val('');
            };
        });
    }).blur(function() {
        if (this.value == '') {
            $(this).addClass('placeholder').val(searchLabel);
        };
    });
});
```

当字段获得焦点时，我们测试字段的值是否等于原先插入的占位符文本。如果是，移除该文本。这个检查很重要，因为我们不想丢掉用户原先填写的任何文本。

当字段失去焦点时，则执行相反的过程。如果用户什么也没有输入，字段的值应该为空。在这种情况下，需要恢复以前存在的占位符文本。

与此同时，我们还移除或添加了placeholder CSS类，因而，当用户在字段中输入文本时，会为输入的文本应用适当的样式，如图8-16所示。

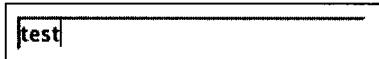


图 8-16

最后的增强还会导致一个小问题，即如果用户在没有输入内容的情况下提交表单，字段中仍然会包含占位符文本。为避免这个问题，可以在表单提交时移除占位符文本：

```
$('#search').submit(function() {
  if ($('#search-text').val() == searchLabel) {
    $('#search-text').val('');
  }
});
```

不管提供初始搜索界面的服务器怎样，现在我们已经为启用了JavaScript的用户提供了视觉上的增强效果。

8.6 AJAX 自动完成

通过为搜索表单提供自动完成内容的功能，还可以进一步增强表单。这个功能可以让用户在输入搜索关键字的开头字符串时，看到以该字符串开头的全部可能的关键字。因为关键字列表可能是从驱动网站的数据库中取得的，所以用户会知道如果使用相应的关键字，那么搜索结果将是现成的。而且，如果数据库提供的关键字按照流行程度或结果数目进行了排序，那么也会对用户取得更有用的结果起到指导作用。

自动完成是一个非常复杂的话题，而且用户的交互方式不同也会带来很多微妙的问题。接下来我们要构建一个实用的例子，但并没有涉及所有高级的概念，比如限制请求速度或多关键字完成等。我们推荐使用基于jQuery的自动完成插件在真实的应用中创建简单的自动完成功能，而且也可以将相应的插件作为添加更复杂功能的起点。有关插件的更多内容将在第10章介绍。

自动完成例程背后的基本思想就是响应敲击键盘的操作，向服务器发送一个包含字段内容的AJAX请求。返回的结果将会包含针对该字段的可能的完成项列表。然后，脚本把返回的列表作为一个下拉菜单呈现在字段下方。

8.6.1 服务器端代码

服务器端需要包含处理请求的代码。虽然真实的应用通常要依赖数据库生成一个可能的完成项列表，但为了方便我们可以把结果内置于简单的PHP脚本中：

```
<?php
if (strlen($_REQUEST['search-text']) < 1) {
  print '[]';
  exit;
}
$terms = array(
  'access',
  'action',
  // .....省略的列表内容.....
  'xaml',
  'xoops',
);
$possibilities = array();
foreach ($terms as $term) {
```

```

if (strpos($term, strtolower($_REQUEST['search-text'])) === 0) {
    $possibilities[] = "'". str_replace("'", "\'", $term) . "'";
}
print ('['. implode(', ', $possibilities) .']');

```

这个PHP页面会将提供的字符串与每个可能的关键字进行比较，然后生成一个匹配的JSON数组。

8.6.2 浏览器端脚本

接下来，我们就可以在JavaScript代码中向这个PHP脚本发送请求了：

```

$(document).ready(function() {
    var $autocomplete = $('

</ul>').hide();
    insertAfter('#search-text');

    $('#search-text').keyup(function() {
        $.ajax({
            'url': '/bookstore/search/autocomplete.php',
            'data': {'search-text': $('#search-text').val()},
            'dataType': 'json',
            'type': 'POST',
            'success': function(data) {
                if (data.length) {
                    $autocomplete.empty();
                    $.each(data, function(index, term) {
                        $('- </li>').text(term).appendTo($autocomplete);
                    });
                    $autocomplete.show();
                }
            }
        });
    });
});

```

这里，应该使用`keyup`而不是`keydown`或`keypress`来触发AJAX请求。后两个事件会在按键过程中，以及字符实际地输入到字段中之前发生。如果响应这两个事件来发送请求，那么建议项(suggestion)列表将会滞后于搜索文本。举例来说，在输入第3个字符时，只会使用前两个字符生成AJAX请求。通过响应`keyup`事件，就可以避免这个问题。

在样式表中，我们为建议项列表设置了绝对定位，以便它覆盖下方的文本。现在，当我们在搜索字段中输入字符时，就会看到可能的关键字列表出现在搜索字段下方，如图8-17所示。

为了适当地显示建议项列表，在此必须考虑到某些Web浏览器内置的自动完成机制。浏览器通常会记住用户在表单字段中输入过的字符串，并在用户下次使用表单时对相应的条目给出建议。这些机制如果与我们创建的自动完成功能同时发生作用，就会导致混乱的结果，如图8-18所示。

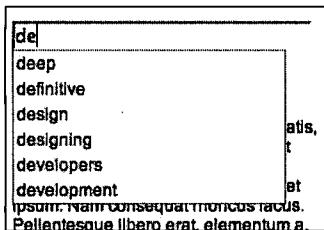


图 8-17

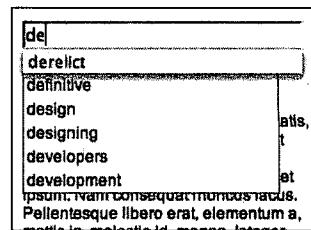


图 8-18

好在，这个问题可以通过禁用浏览器内置的自动完成机制来解决，即将表单字段的 `autocomplete` 属性设置为 `off`。虽然也可以在HTML中设置这个属性，但这样一来就会违反渐进增强的原则。为此，我们还是要通过脚本来设置这个属性：

```
$('#search-text').attr('autocomplete', 'off')
```

8.6.3 填充搜索字段

如果不能把建议项列表中的关键字放到搜索框中，那么这个功能的效果就会大打折扣。所以，我们要允许通过鼠标选定一个建议项：

```
'success': function(data) {
    if (data.length) {
        $autocomplete.empty();
        $.each(data, function(index, term) {
            $('- </li>').text(term)
                .appendTo($autocomplete).click(function() {
                    $('#search-text').val(term);
                    $autocomplete.hide();
                });
        });
        $autocomplete.show();
    }
}

```

经过这次修改后，可以通过单击将列表项目设置为搜索字段中的文本。在此之后，由于无需再使用这个列表，所以我们隐藏了该列表。

8.6.4 键盘导航

因为用户此时正在通过键盘输入搜索关键字，如果也能够让用户通过键盘从建议项列表中选择关键字则会更加方便。要实现这个功能，需要知道当前选择的建议项。为此，必须添加一个辅助函数，用于保存建议项的索引，然后，再通过必要的视觉效果显示出当前选择的建议项：

```
var selectedItem = null;
var setSelectedItem = function(item) {
    selectedItem = item;
    if (selectedItem === null) {
```

```

$autocomplete.hide();
return;
}
if (selectedItem < 0) {
    selectedItem = 0;
}
if (selectedItem >= $autocomplete.find('li').length) {
    selectedItem = $autocomplete.find('li').length - 1;
}
$autocomplete.find('li').removeClass('selected')
.eq(selectedItem).addClass('selected');
$autocomplete.show();
};

```

在没有选择任何建议项时，`selectedItem`变量会被设置为`null`。通过持续地调用`setSelectedItem()`来改变这个变量的值，能够保证只在选择了一个建议项时才会显示建议项列表。

函数中对`selectedItem`数字值的测试用于把结果限制在适当的范围内。如果不添加这些测试代码，`selectedItem`可能会是任意值，甚至是负值。这个函数确保了`selectedItem`的当前值始终是建议项列表中有效的索引值。

下面，我们就来修改现有的代码以使用这个新函数：

```

$('#search-text').attr('autocomplete', 'off').keyup(function() {
    $.ajax({
        'url': '/bookstore/search/autocomplete.php',
        'data': { 'search-text': $('#search-text').val() },
        'dataType': 'json',
        'type': 'POST',
        'success': function(data) {
            if (data.length) {
                $autocomplete.empty();
                $.each(data, function(index, term) {
                    $('- </li>').text(term)
                        .appendTo($autocomplete).mouseover(function() {
                            setSelectedItem(index);
                        }).click(function() {
                            $('#search-text').val(term);
                            $autocomplete.hide();
                        });
                });
                setSelectedItem(0);
            }
        else {
            setSelectedItem(null);
        }
    });
}

```

```

        }
    });
});
}

```

这次修改立即产生了一些效果。首先，如果没有为输入的搜索关键字返回建议项，那么建议项列表会被隐藏。其次，能够添加mouseover处理程序以突出显示鼠标指针下方的建议项。第三，当建议项列表显示时第一个建议项会立即突出显示，如图8-19所示。

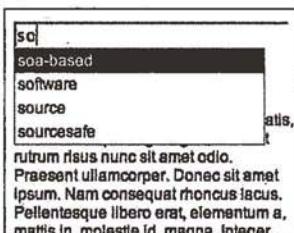


图 8-19

现在，需要允许用户通过键盘上的键来改变列表中的当前活动项。

1. 处理方向键

通过事件对象的keyCode属性可以确定用户按下的是哪个键。因而，通过检查与上、下方向键对应的代码38和40，可以分别作出不同的响应：

```

$('#search-text').attr('autocomplete', 'off').keyup(function(event) {
    if (event.keyCode > 40 || event.keyCode == 8) {
        // 代码为40和以下的键是特殊键
        // (回车键、方向键、退出键等)
        // 代码为8的键是回格键

        $.ajax({
            'url': '/bookstore/search/autocomplete.php',
            'data': {'search-text': $('#search-text').val()},
            'dataType': 'json',
            'type': 'POST',
            'success': function(data) {
                if (data.length) {
                    $autocomplete.empty();
                    $.each(data, function(index, term) {
                        $('- </li>').text(term)
                            .appendTo($autocomplete).mouseover(function() {
                                setSelectedItem(index);
                            }).click(function() {
                                $('#search-text').val(term);
                                $autocomplete.hide();
                            });
                }
            }
        });
    }
});

```

```

        });
        setSelectedItem(0);
    }
    else {
        setSelectedItem(null);
    }
}
});
else if (event.keyCode == 38 && selectedItem != null) {
    // 用户按了上方向键
    setSelectedItem(selectedItem - 1);
    event.preventDefault();
}
else if (event.keyCode == 40 && selectedItem != null) {
    // 用户按了下方向键
    setSelectedItem(selectedItem + 1);
    event.preventDefault();
}
});

```

现在，keyup处理程序通过检查接收到的keyCode，可以执行相应的操作。而且，如果用户按下的是特殊键（例如方向键或退出键），AJAX请求还可以跳过这些事件。如果检测到用户按下的是方向键，并且建议项列表当前处于显示状态，该处理程序会按照适当的方式逐个改变选择的建议项。因为setSelectedItem()函数中已经限定了列表索引值的范围，所以无需担心用户会离开列表的任何一端。

2. 将建议项插入到字段中

接下来需要处理回车键。当建议项列表处于显示状态时，按下回车键应该把当前选择的建议项填充到搜索字段中。由于我们要在两个地方执行这一操作，所以应该把前面为鼠标按键编写的字段填充代码提取到一个独立的函数中：

```

var populateTextField = function() {
    $('#search-text').val($autocomplete
        .find('li').eq(selectedItem).text());
    setSelectedItem(null);
};

```

这样，click处理程序就可以简单地调用这个函数。而且，在处理回车键时也可以调用这个函数：

```

$('#search-text').keypress(function(event) {
    if (event.keyCode == 13 && selectedItem != null) {
        // 用户按了回车键
        populateTextField();
        event.preventDefault();
    }
});

```

我们注意到，这个处理程序响应的是`keypress`事件，而不是前面的`keyup`事件。为了避免这个按键操作提交表单，必须要作此修改。如果我们等到`keyup`事件被触发，那么提交表单的操作就已经开始执行了。

3. 移除建议项列表

现在，还需要对自动完成的行为增强代码进行最后一次调整。当用户决定在当前页面上做其他事时，应该隐藏建议项列表。首先，可以在`keyup`处理程序中响应退出键，让用户能够通过按该键取消列表：

```
else if (event.keyCode == 27 && selectedItem !== null) {
    // 用户按了退出键
    setSelectedItem(null);
}
```

更重要的是，我们应该在搜索字段失去焦点时隐藏列表。为此，可以简单地编写下列代码：

```
$('#search-text').blur(function(event) {
    setSelectedItem(null);
});
```

但是，这样会在无意间导致副作用。由于在列表上面单击鼠标会将焦点从字段上移开，所以这个处理程序会执行并隐藏列表。这意味着我们在前面定义的`click`处理程序将永远得不到调用，因此，通过鼠标与列表交互也变得不可能了。

这个问题没有简单的解决方案。`blur`处理程序始终会先于`click`处理程序被调用。一种方案是在字段失去焦点时，延迟极短的时间再隐藏列表：

```
$('#search-text').blur(function(event) {
    setTimeout(function() {
        setSelectedItem(null);
    }, 250);
});
```

这样，就在建议项列表隐藏之前为在列表上面触发`click`事件提供了机会。

8.6.5 自动完成与实时搜索

前面的例子聚焦于文本字段的自动完成，因为这是一种适用于许多表单的技术。但是，还有一种更新颖的搜索方案，叫做**实时搜索**（live search）。实时搜索会在用户输入搜索关键字时，实际地搜索相关的内容。

从功能上看，自动完成和实时搜索非常相似。在这两种情况下，按下键盘都会向服务器提交AJAX请求，同时在请求中传递当前字段中的内容。而返回的结果也会显示在字段下方的下拉列表框中。此时，对于自动完成来说（我们已经看到了），结果是可能的搜索关键字。而对于实时

搜索而言，结果则是包含已经输入的搜索关键字的实际页面。

在JavaScript脚本中，用于构建这两种功能的代码几乎相同，所以我们就不在这里详细介绍了。在决定使用哪种功能时，需要权衡利弊。实时搜索虽然可以让用户不必费太多力气就能得到更多的信息，但是，这种功能占用的资源也会更多。

8.7 完成的代码

用于增强搜索字段外观和提供自动完成行为的完成后的代码如下所示：

```
$document.ready(function() {
    var searchLabel = $('#search label').remove().text();
    $('#search-text').addClass('placeholder').val(searchLabel)
        .focus(function() {
            if (this.value == searchLabel) {
                $(this).removeClass('placeholder').val('');
            };
        }).blur(function() {
            if (this.value == '') {
                $(this).addClass('placeholder').val(searchLabel);
            };
        });
    $('#search').submit(function() {
        if ($('#search-text').val() == searchLabel) {
            $('#search-text').val('');
        }
    });

    var $autocomplete = $('

</ul>').hide().
        insertAfter('#search-text');
    var selectedItem = null;

    var setSelectedItem = function(item) {
        selectedItem = item;
        if (selectedItem === null) {
            $autocomplete.hide();
            return;
        }
        if (selectedItem < 0) {
            selectedItem = 0;
        }
        if (selectedItem >= $autocomplete.find('li').length) {
            selectedItem = $autocomplete.find('li').length - 1;
        }
        $autocomplete.find('li').removeClass('selected').eq(selectedItem)
            .addClass('selected');
        $autocomplete.show();
    };
});
```

```

};

var populateSearchField = function() {
    $('#search-text').val($('#autocomplete').find('li').eq(selectedItem)
        .text());
    setSelectedItem(null);
};
$('#search-text').attr('autocomplete', 'off').keyup(function(event)
{
    if (event.keyCode > 40 || event.keyCode == 8) {
        // 代码为40和以下的键是特殊键（回车键、方向键、退出键等）
        // 代码为8的键是回格键

        $.ajax({
            'url': '/bookstore/search/autocomplete.php',
            'data': {'search-text': $('#search-text').val()},
            'dataType': 'json',
            'type': 'POST',
            'success': function(data) {
                if (data.length) {
                    $autocomplete.empty();
                    $.each(data, function(index, term) {
                        $('- </li>').text(term).appendTo($autocomplete)
                            .mouseover(function() {
                                setSelectedItem(index);
                                }).click(populateSearchField);
                    });
                    setSelectedItem(0);
                }
                else {
                    setSelectedItem(null);
                }
            }
        });
    }
    else if (event.keyCode == 38 && selectedItem !== null) {
        // 用户按了上方向键
        setSelectedItem(selectedItem - 1);
        event.preventDefault();
    }
    else if (event.keyCode == 40 && selectedItem !== null) {
        // 用户按了下方向键
        setSelectedItem(selectedItem + 1);
        event.preventDefault();
    }
    else if (event.keyCode == 27 && selectedItem !== null) {
        // 用户按了退出键
        setSelectedItem(null);
    }
}

```

```

}).keypress(function(event) {
  if (event.keyCode == 13 && selectedItem !== null) {
    // 用户按了回车键
    populateSearchField();
    event.preventDefault();
  }
}).blur(function(event) {
  setTimeout(function() {
    setSelectedItem(null);
  }, 250);
});
});

```

8.8 输入掩码

我们已经介绍了一些适用于用户进行文本性输入的表单功能。然而，有时候表单中的内容也会以数字为主。当处理作为表单内容的数字值时，也可以实现另外几种表单增强功能。

在网上书店的例子中，数字表单的典型代表就是购物车。在购物车表单中，应该允许用户更新购买的商品数量，同时，也要为用户提供价格和总金额的数值反馈。

8.8.1 购物车表格结构

购物车的HTML代码将涉及一个迄今为止较难处理的表格结构：

```

<form action="checkout.php" method="post">
  <table id="cart">
    <thead>
      <tr>
        <th class="item">Item</th>
        <th class="quantity">Quantity</th>
        <th class="price">Price</th>
        <th class="cost">Total</th>
      </tr>
    </thead>
    <tfoot>
      <tr class="subtotal">
        <td class="item">Subtotal</td>
        <td class="quantity"></td>
        <td class="price"></td>
        <td class="cost">$152.95</td>
      </tr>
      <tr class="tax">
        <td class="item">Tax</td>
        <td class="quantity"></td>
        <td class="price">6%</td>
        <td class="cost">$9.18</td>
      </tr>
      <tr class="shipping">

```

```

<td class="item">Shipping</td>
<td class="quantity">5</td>
<td class="price">$2 per item</td>
<td class="cost">$10.00</td>
</tr>
<tr class="total">
    <td class="item">Total</td>
    <td class="quantity"></td>
    <td class="price"></td>
    <td class="cost">$172.13</td>
</tr>
<tr class="actions">
    <td></td>
    <td><input type="button" name="recalculate"
               value="Recalculate" id="recalculate" /></td>
    <td></td>
    <td><input type="submit" name="submit"
               value="Place Order" id="submit" /></td>
</tr>
</tfoot>
<tbody>
    <tr>
        <td class="item">Building Telephony Systems With Asterisk</td>
        <td class="quantity"><input type="text" name="quantity-2"
           value="1" id="quantity-2" maxlength="3" /></td>
        <td class="price">$26.99</td>
        <td class="cost">$26.99</td>
    </tr>
    <tr>
        <td class="item">Smarty PHP Template Programming and
                           Applications</td>
        <td class="quantity"><input type="text" name="quantity-1"
           value="2" id="quantity-1" maxlength="3" /></td>
        <td class="price">$35.99</td>
        <td class="cost">$71.98</td>
    </tr>
    <tr>
        <td class="item">Creating your MySQL Database: Practical
                           Design Tips and Techniques</td>
        <td class="quantity"><input type="text" name="quantity-3"
           value="1" id="quantity-3" maxlength="3" /></td>
        <td class="price">$17.99</td>
        <td class="cost">$17.99</td>
    </tr>
    <tr>
        <td class="item">Drupal: Creating Blogs, Forums, Portals, and
                           Community Websites</td>
        <td class="quantity"><input type="text" name="quantity-4"
           value="1" id="quantity-4" maxlength="3" /></td>
        <td class="price">$35.99</td>
        <td class="cost">$35.99</td>
    </tr>
</tbody>
</table>
</form>

```

这个表格中引入了另一个在现实中很少见的元素`<tfoot>`。与`<thead>`一样，这个元素也用于分组表格行。而且，虽然`<tfoot>`在源代码中位于表体（`<tbody>`）之前，但在通过页面呈现它时，浏览器会把它放到表体下方，如图8-20所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	1	\$26.99	\$26.99
Smarty PHP Template Programming and Applications	2	\$36.99	\$71.98
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.95
Tax		6%	\$9.18
Shipping	5	\$2 per item	\$10.00
Total			\$172.13
	Recalculate		Place Order

图 8-20

源代码的这种排列顺序，虽然对设计者从视觉上想象表格的呈现效果不太直观，但对于有视力障碍的人却很有帮助。当视力有障碍的用户通过辅助设备读取这个表格时，会在读取可能较长的内容之前先读取脚注（`<tfoot>`），这样他们就能对后面是什么内容有一个概括性的了解。

而且，我们还在表格的每个单元格中都添加了一个类，用于识别包含该单元格的表格列。在前一章中，我们示范了通过单元格在表格行中的索引来查找列中的单元格。这里，我们采取一个折衷的方案，通过提高一点HTML源代码的复杂性，来简化JavaScript代码。在有了标识每个单元格所在列的类之后，选择符就会变得很简单。

在操作表单字段之前，我们先为表格应用标准的行条纹效果，美化一下表格的外观：

```
$(document).ready(function() {
    $('#cart tbody tr:even').addClass('even');
    $('#cart tbody tr:odd').addClass('odd');
});
```

同样，我们只选择了表体中的行作为着色的目标，如图8-21所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	1	\$26.99	\$26.99
Smarty PHP Template Programming and Applications	2	\$36.99	\$71.98
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.95
Tax		6%	\$9.18
Shipping	5	\$2 per item	\$10.00
Total			\$172.13
	Recalculate		Place Order

图 8-21

8.8.2 拒绝非数字输入

在改进联系表单时，我们曾讨论过一些验证输入的技术。通过JavaScript，可以检查用户输入的内容是否符合我们的要求，以便在将表单发送到服务器之前对用户给出反馈。下面，我们来介绍输入验证的另一种技术，叫做输入掩码。

输入验证是根据某些有效输入的标准来检查用户输入的过程。输入掩码会在用户填写内容的同时应用标准，并简单地禁止无效的按键操作。比如，在这个购物车表单的例子中，必须在输入字段中填写数字。通过使用输入掩码验证，可以在这些字段获得焦点时，让非数字按键操作不起作用：

```
$('.quantity input').keypress(function(event) {
  if (event.charCode && (event.charCode < 48 || event.charCode > 57))
  {
    event.preventDefault();
  }
});
```

在为了实现字段的自动完成功能而捕获按键操作时，我们监听的是`keyup`事件。`keyup`事件会在事件对象中提供`.keyCode`属性，通过检查该属性能够确定用户按下了哪个按键。在这里，我们要观察的是`keypress`事件，这个事件不提供`.keyCode`属性，但提供了`.charCode`属性。`.charCode`属性中保存的是代表刚才按键的ASCII字符。

如果通过按键输入了一个字符（即没有按方向键、删除键或其他编辑功能键）并且该字符不在表示数字的ASCII编码的范围内，我们就可以在相应的事件对象上调用`.preventDefault()`方法。前面我们曾经看到过，调用`.preventDefault()`方法会阻止浏览器响应相应的事件，而在里面，就意味着不能把字符插入到字段中。也就是说，所有数量字段只能接受数字。

8.9 数字计算

接下来，我们继续讨论一下如何操作用户在购物车表单中输入的数字。表单中有一个`Recalculate`（重新计算）按钮，单击这个按钮会把表单提交到服务器，通过重新计算总金额再把表单展示给用户。但是，这个往返过程是不必要的；所有工作都可以在浏览器中通过jQuery来完成。

这个表单中最简单的计算，就是在`Shipping`行中显示订购商品的数量。当用户修改了其中一行的数量时，我们要合计所有输入值以得到新的数量，然后将总数显示在相应的单元格中：

```
$('.quantity input').change(function() {
  var totalQuantity = 0;
  $('.quantity input').each(function() {
    var quantity = parseInt(this.value);
    totalQuantity += quantity;
  });
  $('.shipping .quantity').text(String(totalQuantity));
});
```

对于这个重新计算的操作而言，有几种监听的事件可供选择。比如，可以观察`keyup`事件，通过每次按键操作来触发重新计算。另外，也可以观察`blur`事件，该事件会在用户每次离开字段时触发。不过，为了更加稳健地使用CPU，我们只在`change`事件发生时执行计算。这样，只有当用户离开字段并填写了同以前不一样的值时，才会导致重新计算总数量。

计算总数量使用了一个简单的`.each()`循环。由于字段的`.value`属性中保存的是字段值的字符串形式，所以我们使用内置的`parseInt`函数把字符串转换成了可以用来计算的整数。这种习惯可以避免把数字相加搞成字符串连接所导致的奇怪结果（因为这两种操作使用的是同一个运算符）。恰恰相反，当显示计算结果时，我们需要为jQuery的`.text()`方法传递一个字符串值。因此，要使用`String`函数以计算的总数量作为参数构建一个新字符串。

于是，修改数量就能自动地更新总数了，如图8-22所示。



Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	11	\$26.99	\$26.99
Smarty PHP Template Programming and Applications	2	\$35.99	\$71.98
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$36.99	\$36.99
Subtotal			\$152.95
Tax		6%	\$9.18
Shipping	15	\$2 per item	\$10.00
Total			\$172.13
Recalculate		Place Order	

图 8-22

8.9.1 解析和格式化货币值

现在，我们继续讨论右侧表格列中的总费用（Total）。每一行的总费用应该用输入的数量乘以商品的价格得到。由于需要对每一行执行多项操作，所以我们把前面计算数量的代码重新解构，变基于字段的计算为基于行的计算：

```
$('#cart tbody tr').each(function() {
    var quantity = parseInt($('.quantity input', this).val());
    totalQuantity += quantity;
});
```

这些代码可以得到同以前一样的结果，但是，它为插入计算每一行总费用的代码提供了方便：

```
$('.quantity input').change(function() {
    var totalQuantity = 0;
    $('#cart tbody tr').each(function() {
        var price = parseFloat($('.price', this).text()
            .replace(/[^.\d.]*/, ''));
```

```

price = isNaN(price) ? 0 : price;
var quantity = parseInt($('.quantity input', this).val());
var cost = quantity * price;
$('.cost', this).text('$' + cost);
totalQuantity += quantity;
});
$('.shipping .quantity').text(String(totalQuantity));
});

```

通过使用前面按价格排序表格时使用的相同技术，我们取得了表格中每件商品的价格。正则表达式首先去掉了价格前面的货币符号，然后把得到的字符串传递给`parseFloat()`，该函数把价格转换成了浮点数。由于要根据结果进行计算，所以必须确保找到了数字值，如果没有则将其设置为0。最后，用价格乘以数量，再把\$与计算结果连接起来放到总费用列中。这样，就可以在修改数量时实时地计算总费用了，如图8-23所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	11	\$26.99	\$296.89
Smarty PHP Template Programming and Applications	3	\$35.99	\$107.97
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.95
Tax		6%	\$9.18
Shipping	16	\$2 per item	\$10.00
Total			\$172.13
Recalculate		Place Order	

图 8-23

8.9.2 处理小数位

虽然我们为总费用前面添加了美元符号，但JavaScript并不知道这是在处理货币值。从计算机的角度看，这些值就是数字，而且应该如实地显示它们。这意味着，如果总费用的小数点后面以0结尾，那么这个0会被切掉，如图8-24所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	11	\$26.99	\$296.89
Smarty PHP Template Programming and Applications	30	\$35.99	\$1079.7
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Edums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.95
Tax		6%	\$9.18
Shipping	43	\$2 per item	\$10.00
Total			\$172.13
Recalculate		Place Order	

图 8-24

更糟糕的是，JavaScript精度的限制有时候会造成舍入错误。而这些问题会导致计算的结果难以令人信服，如图8-25所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	11	\$26.99	\$296.89
Smarty PHP Template Programming and Applications	30	\$35.99	\$1079.7
Creating your MySQL Database: Practical Design Tips and Techniques	10	\$17.99	\$179.89999999999998
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.95
Tax		6%	\$9.18
Shipping	52	\$2 per item	\$10.00
Total			\$172.13
		Recalculate	Place Order

图 8-25

好在，修复这两个问题也很简单。JavaScript的Number类中提供了一些处理这种问题的方法，其中.toFixed()恰好符合我们的要求。这个方法接受一个表示小数点位数的参数，返回一个表现舍入到相应小数位后的浮点数的字符串：

```
$('#cart tbody tr').each(function() {
    var price = parseFloat($('.price', this).text()
        .replace(/^\[^.\]*/, ''));

    price = isNaN(price) ? 0 : price;
    var quantity = parseInt($('.quantity input', this).val());
    var cost = quantity * price;
    $('.cost', this).text('$' + cost.toFixed(2));
    totalQuantity += quantity;
});
```

这样，总费用看起来与正常的货币值就没有什么区别了，如图8-26所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	11	\$26.99	\$296.89
Smarty PHP Template Programming and Applications	30	\$35.99	\$1079.7
Creating your MySQL Database: Practical Design Tips and Techniques	10	\$17.99	\$179.90
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.95
Tax		6%	\$9.18
Shipping	52	\$2 per item	\$10.00
Total			\$172.13
		Recalculate	Place Order

图 8-26

8.9.3 其他计算

这个页面中的其余计算也都遵循类似的模式。以小计金额（Subtotal）为例，我们可以在计算完每一行的费用之后合计总费用，然后使用同前面一样的货币格式将结果显示到相应的单元格中（如图8-27所示）：

```
$('.quantity input').change(function() {
    var totalQuantity = 0;
    var totalCost = 0;
    $('#cart tbody tr').each(function() {
        var price = parseFloat($('.price', this).text())
            .replace(/[^d.*/, ''));
        price = isNaN(price) ? 0 : price;
        var quantity = parseInt($('.quantity input', this).val());
        var cost = quantity * price;
        $('.cost', this).text('$' + cost.toFixed(2));
        totalQuantity += quantity;
        totalCost += cost;
    });
    $('.shipping .quantity').text(String(totalQuantity));
    $('.subtotal .cost').text('$' + totalCost.toFixed(2));
});
```

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	11	\$26.99	\$296.89
Smarty PHP Template Programming and Applications	1	\$35.99	\$35.99
Creating your MySQL Database: Practical Design Tips and Techniques	10	\$17.99	\$179.90
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$548.77
Tax		6%	\$9.18
Shipping	23	\$2 per item	\$10.00
Total			\$172.13
	Recalculate		Place Order

图 8-27

1. 舍入数值

为了计算税金（Tax），需要用相应的数字除以100得到税率，再用小计金额乘以税率。不过，计算税金时总是要向上舍入的，因此必须保证在显示和计算时使用正确的值。JavaScript的Math.ceil函数可以将一个数值向上舍入到最接近的整数。但是，由于我们要处理元和分，所以还需要使用一点技巧：

```
var taxRate = parseFloat($('.tax .price').text()) / 100;
var tax = Math.ceil(totalCost * taxRate * 100) / 100;
```

```
$('.tax .cost').text('$' + tax.toFixed(2));
totalCost += tax;
```

用税金乘以100会使它变成一个以分而不是元表示的值，通过Math.ceil()舍入这个值是安全的。舍入之后，再除以100就可以将这个值转换回以元表示的值。最后，同以前一样调用.toFixed()就可以得到正确的结果，如图8-28所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$26.99	\$80.97
Smarty PHP Template Programming and Applications	1	\$35.99	\$35.99
Creating your MySQL Database: Practical Design Tips and Techniques	2	\$17.99	\$35.98
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$188.93
Tax		6%	\$11.34
Shipping	7	\$2 per item	\$10.00
Total			\$172.13
Recalculate	Place Order		

图 8-28

2. 最后的调整

计算运费（Shipping）比计算税金更简单，因为它不涉及舍入问题。用商品数量乘以单件商品的运输费率就可以得到总运费（如图8-29所示）：

```
$('.shipping .quantity').text(String(totalQuantity));
var shippingRate = parseFloat($('.shipping .price')
    .text().replace(/[^^\d.]*/,''));
var shipping = totalQuantity * shippingRate;
$('.shipping .cost').text('$' + shipping.toFixed(2));
totalCost += shipping;
```

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$26.99	\$80.97
Smarty PHP Template Programming and Applications	1	\$35.99	\$35.99
Creating your MySQL Database: Practical Design Tips and Techniques	2	\$17.99	\$35.98
Drupal: Creating Blogs, Forums, Portals, and Community Websites	2	\$35.99	\$71.98
Subtotal			\$224.92
Tax		6%	\$13.50
Shipping	8	\$2 per item	\$16.00
Total			\$172.13
Recalculate	Place Order		

图 8-29

最后，当然还要计算总金额。不过，现在我们需要做的就是在最后一个单元格中以适当的格式显示totalCost（如图8-30所示）：

```
$('.total .cost').text('$' + totalCost.toFixed(2));
```

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	1	\$26.99	\$26.99
Smarty PHP Template Programming and Applications	2	\$35.99	\$71.98
Creating your MySQL Database: Practical Design Tips and Techniques	2	\$17.99	\$35.98
Drupal: Creating Blogs, Forums, Portals, and Community Websites	2	\$35.99	\$71.98
Subtotal			\$208.93
Tax		6%	\$12.42
Shipping	7	\$2 per item	\$14.00
Total			\$233.35
Recalculate		Place Order	

图 8-30

现在，我们已经完全重写了可能会发生的任何服务器端计算。因此，下面可以安全地隐藏Recalculate按钮了（如图8-31所示）：

```
$('#recalculate').hide();
```

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	1	\$26.99	\$26.99
Smarty PHP Template Programming and Applications	2	\$35.99	\$71.98
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	2	\$35.99	\$71.98
Subtotal			\$188.94
Tax		6%	\$11.34
Shipping	6	\$2 per item	\$12.00
Total			\$212.28
Place Order			

图 8-31

这一变化再次应用了渐进增强的原则：首先确保页面在没有JavaScript的情况下正常运行，然后使用jQuery在可能的情况下更优雅地完成相同的任务。

8.10 删除商品

如果光临我们网站的购物者在把商品放到购物车中之后改变了主意，他们可以将相应商品的Quantity字段修改为0。然而，我们还可以提供一种更可靠的行为，即为每件商品都添加一个明确的Delete按钮。虽然单击这个按钮的实际效果与修改Quantity字段相同，但这种视觉上的反馈可以

强化不会购买相应商品的事实。

首先，需要添加新按钮。由于这些按钮没有JavaScript无法使用，因此不能把它们放到HTML中。下面，我们使用jQuery把它们添加到每一行：

```
$('<th>&nbsp;</th>').insertAfter('#cart thead th:nth-child(2)');
$('#cart tbody tr').each(function() {
  $deleteButton = $('

```

不过，除此之外还需要在表头和表注中创建作为占位符的空单元格，以便表格列依旧能够正确地对齐。而且，创建的按钮也只添加到了表体包含的行中，如图8-32所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$28.99	\$86.97
Smarty PHP Template Programming and Applications	2	\$35.99	\$71.98
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$206.93
Tax		6%	\$12.42
Shipping	7	\$2 per item	\$14.00
Total			\$233.35
Place Order			

图 8-32

接下来，需要让这些按钮做点什么。为此，我们修改按钮的定义，添加一个单击处理程序：

```
$deleteButton = $('

```

添加的处理程序会找到与按钮同在一行的数量字段，然后将它的值设置为0。现在，数量字段是更新了，但相应的计算却没有同步执行，如图8-33所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$26.99	\$80.97
Smarty PHP Template Programming and Applications	0	\$35.99	\$71.98
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$206.93
Tax		6%	\$12.42
Shipping	7	\$2 per item	\$14.00
Total			\$233.35
Place Order			

图 8-33

因此，还需要像用户手工修改数量字段的值一样，触发相应的计算：

```
$deleteButton = $('<img />').attr({
  'width': '16',
  'height': '16',
  'src': '../icons/cross.png',
  'alt': 'remove from cart',
  'title': 'remove from cart',
  'class': 'clickable'
}).click(function() {
  $(this).parents('tr').find('.quantity input')
    .val(0).trigger('change');
});
```

现在总金额会随着单击删除按钮而更新了，如图8-34所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$26.99	\$80.97
Smarty PHP Template Programming and Applications	0	\$35.99	\$0.00
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$134.95
Tax		6%	\$8.10
Shipping	5	\$2 per item	\$10.00
Total			\$153.05
Place Order			

图 8-34

下面，我们为用户提供一个视觉反馈——隐藏刚才单击的行，以便将商品明显地从购物车中

移除（如图8-35所示）：

```
$deleteButton = $('').attr({
  'width': '16',
  'height': '16',
  'src': '../icons/cross.png',
  'alt': 'remove from cart',
  'title': 'remove from cart',
  'class': 'clickable'
}).click(function() {
  $(this).parents('tr').find('.quantity input')
    .val(0).trigger('change')
  .end().hide();
});

});
```

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	[3]	\$26.99	\$80.97
Creating your MySQL Database: Practical Design Tips and Techniques	[1]	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	[1]	\$35.99	\$35.99
Subtotal			\$134.95
Tax			6% \$8.10
Shipping			5 \$2 per item \$10.00
Total			\$153.05
Place Order			

图 8-35

虽然隐藏了相应的商品行，但数量字段仍然存在于表单中。也就是说，它也会与表单的其他字段一起被提交，届时，服务器上的代码将会删除该商品。

不过，随着行的移除，行条纹效果也受到了影响。为解决这个问题，可以先把已有的实现条纹效果的代码放到一个函数中，然后在需要时再调用这些代码：

```
var stripe = function() {
  $('#cart tbody tr:visible:even').removeClass('odd')
    .addClass('even');
  $('#cart tbody tr:visible:odd').removeClass('even')
    .addClass('odd');
};

stripe();
```

与此同时，我们还修改了选择符表达式，以便计算奇偶行的数目时排除不可见的行。而且，还要确保在应用even类时先删除odd类，反之亦然。现在，就可以在移除行之后调用该函数了：

```
$deleteButton = $('').attr({
  'width': '16',
  'height': '16',
  'src': '../icons/cross.png',
```

```

'alt': 'remove from cart',
'title': 'remove from cart',
'class': 'clickable'
}).click(function() {
  $(this).parents('tr').find('.quantity input')
    .val(0).trigger('change')
  .end().hide();
  stripe();
});
}

```

这样，移除的行会没有痕迹地从表格中消失了，如图8-36所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$26.99	\$80.97
Creating your MySQL Database: Practical Design Tips and Techniques	1	\$17.99	\$17.99
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$134.95
Tax		6%	\$8.10
Shipping	5	\$2 per item	\$10.00
Total			\$153.05
Place Order			

图 8-36

至此，我们就通过jQuery完成了另一项对服务器上的代码毫无干扰的增强功能。从服务器端代码的角度上看，用户是在数量输入字段中填写了0；而从用户的角度上看，这是一个与修改商品数量不同的移除操作。

8.11 修改送货信息

购物车页面中还有一个与送货信息有关的表单。实际上，在页面加载时，并不存在这个表单；而且，如果不通过JavaScript来增强，它只是放在内容区右侧的一个小方框，其中包含一个链接，打开该链接之后，用户可以在相应的页面中修改送货信息，如图8-37所示。

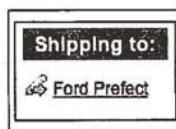


图 8-37

但是，在启用JavaScript的情况下，加上强大的jQuery的帮助，我们可以把这个小链接转换成一个功能完善的表单。为此，需要从一个PHP页面中请求该表单。通常，填充表单的数据会被保存在某种数据库中。但为了方便起见，我们使用PHP数组中保存的静态数据来进行示范。

为了取得表单并使它出现在Shipping to方框内，需要在.click事件处理程序中使用\$.get方法：

```
$(document).ready(function() {
  $('#shipping-name').click(function() {
    $.get('shipping.php', function(data) {
      $('#shipping-name').remove();
      $(data).hide().appendTo('#shipping').slideDown();
    });
    return false;
  });
});
```

在\$.get方法的回调函数中，我们移除了刚才单击的用户名，然后在它的位置上添加了表单及来自shipping.php中的数据。最后，通过添加return false语句取消单击链接引发的默认操作（加载链接的href属性指向的页面）。现在，Shipping to方框内出现了可以编辑的表单，如图8-38所示。

The screenshot shows a modal dialog box titled "Shipping to:". Inside, there are input fields for shipping information:

- First name: Ford
- Last name: Prefect
- Address: 52 Rodeo Drive
- City: Beverly Hills
- State: CA
- ZIP: 90210

Below these fields is a "Gift message:" label followed by a text area containing the text: "Enjoy these great books! You'll love the cliff-hangers!"

At the bottom of the dialog is a "Save" button.

图 8-38

这样，用户就可以在不离开当前页面的情况下，修改自己的送货信息。

下一步是hijack^①（拦截）表单提交事件，通过jQuery把修改后的数据发送回服务器。首先，要对表单中的数据进行序列化处理，并把处理后的结果保存到变量postData中。然后，再使用shipping.php将数据发送回服务器：

```
$(document).ready(function() {
  $('#shipping-form').submit(function() {
    var postData = $('#shipping :input').serialize();
    $.post('shipping.php', postData);
```

① Jeremy Keith利用这个词的意思，提出了一语双关的Hijax的概念。读者可以参考译者网站中的一篇文章：<http://www.cn-cuckoo.com/2007/11/14/the-origin-of-the-concept-of-unobtrusive-144.html>。——译者注

```

        return false;
    );
});

```

此时，有必要移除表单并恢复Shipping to方框的原始状态。我们可以在刚才使用的`$.post`方法的回调函数中完成相应的恢复操作：

```

$(document).ready(function() {
    $('#shipping form').submit(function() {
        var postData = $('#shipping :input').serialize();
        $.post('shipping.php', postData, function(data) {
            $('#shipping form').remove();
            $(data).appendTo('#shipping');
        });
        return false;
    });
});

```

现在，唯一的问题就是这些代码还不起作用。按照现在的方式，应该在DOM加载后立即把`.submit`事件处理程序绑定到Shipping to表单。但是，在用户单击Shipping to链接之前，这个表单在DOM中是不存在的。不能把事件绑定到不存在的DOM对象上。

为解决这个问题，可以把创建表单的代码放到一个名为`editShipping`的函数中，把提交表单或移除表单的代码放到一个名为`saveShipping`的函数中。然后，在`$.get()`方法的回调函数中、在创建了表单之后，绑定`saveShipping`函数。同样地，还需要在DOM就绪时和在`$.post()`方法的回调函数中重新创建了用户名的链接之后，绑定`editShipping`函数：

```

$(document).ready(function() {
    var editShipping = function() {
        $.get('shipping.php', function(data) {
            $('#shipping-name').remove();
            $(data).hide().appendTo('#shipping').slideDown();
            $('#shipping form').submit(saveShipping);
        });
        return false;
    };
    var saveShipping = function() {
        var postData = $('#shipping :input').serialize();
        $.post('shipping.php', postData, function(data) {
            $('#shipping form').remove();
            $(data).appendTo('#shipping');
            $('#shipping-name').click(editShipping);
        });
        return false;
    };
    $('#shipping-name').click(editShipping);
});

```

以上代码构建了某种循环模式，即一个函数在重新绑定它们各自的事件处理程序时，需要使用另一个函数。

8.12 完成的代码

综合到一起，我们为购物车页面编写的代码只有79行——相对于它们完成的功能而言，这已经相当少了；而且，如果考虑到通过简洁的代码实现的最佳可读性，这一点就显得更为突出。由于jQuery具有方法连缀能力，所以我们也会特别关注合并了许多行之后的行数。不管怎样，以下就是我们为购物车页面编写的全部代码，而且，专门讨论表单的这一章到此也结束了：

```

$(document).ready(function() {
    // 购物车
    var stripe = function() {
        $('#cart tbody tr:visible:even').removeClass('odd')
            .addClass('even');
        $('#cart tbody tr:visible:odd').removeClass('even')
            .addClass('odd');
    };
    stripe();
    $('#recalculate').hide();
    $('.quantity input').keypress(function(event) {
        if (event.charCode && (event.charCode < 48 ||
            event.charCode > 57)) {
            event.preventDefault();
        }
    }).change(function() {
        var totalQuantity = 0;
        var totalCost = 0;
        $('#cart tbody tr').each(function() {
            var price = parseFloat($('.price', this).text()
                .replace(/[^\\d.]*/, ''));
            price = isNaN(price) ? 0 : price;
            var quantity = parseInt($('.quantity input', this).val());
            var cost = quantity * price;
            $('.cost', this).text('$' + cost.toFixed(2));
            totalQuantity += quantity;
            totalCost += cost;
        });
        $('.subtotal .cost').text('$' + totalCost.toFixed(2));
        var taxRate = parseFloat($('.tax .price').text()) / 100;
        var tax = Math.ceil(totalCost * taxRate * 100) / 100;
        $('.tax .cost').text('$' + tax.toFixed(2));
        totalCost += tax;
        $('.shipping .quantity').text(String(totalQuantity));
        var shippingRate = parseFloat($('.shipping .price').text()
            .replace(/[^\\d.]*/, ''));
        var shipping = totalQuantity * shippingRate;
    });
});

```

```

$('.shipping .cost').text('$' + shipping.toFixed(2));
totalCost += shipping;
$('.total .cost').text('$' + totalCost.toFixed(2));
});

$(<th>&nbsp;</th>).insertAfter('#cart thead th:nth-child(2)');
$('#cart tbody tr').each(function() {
  $deleteButton = $('<img />').attr({
    'width': '16',
    'height': '16',
    'src': '../icons/cross.png',
    'alt': 'remove from cart',
    'title': 'remove from cart',
    'class': 'clickable'
  }).click(function() {
    $(this).parents('tr').find('.quantity input')
      .val(0).trigger('change')
      .end().hide();
    stripe();
  });
  $(<td></td>).insertAfter($('td:nth-child(2)', this))
    .append($deleteButton);
});
$(<td>&nbsp;</td>).insertAfter('#cart tfoot td:nth-child(2)');
});

// 修改送货信息

$(document).ready(function() {
  var editShipping = function() {
    $.get('shipping.php', function(data) {
      $('#shipping-name').remove();
      $(data).hide().appendTo('#shipping').slideDown();
      $('#shipping form').submit(saveShipping);
    });
    return false;
  };
  var saveShipping = function() {
    var postData = $('#shipping :input').serialize();
    $.post('shipping.php', postData, function(data) {
      $('#shipping form').remove();
      $(data).appendTo('#shipping');
      $('#shipping-name').click(editShipping);
    });
    return false;
  };
  $('#shipping-name').click(editShipping);
});
}

```

8.13 小结

本章中，我们研究了改进常见的HTML表单元素的外观和行为的不同方式。学习了在保留原始标记语义的同时，增强表单的样式、基于其他字段的值有条件地隐藏和显示字段，以及在提交之前和在数据填写期间验证字段的内容。而且，介绍了如何通过AJAX自动完成文本字段、如何限制在字段中输入特殊字符，以及如何基于字段中的数值执行计算。此外，也讨论了使用AJAX而不是页面刷新提交表单的内容。

表单元素往往是体现网站交互性的关键所在。通过jQuery，可以轻松地改进用户填写表单的体验，而且，还能保持原有的可用性和灵活性。

第9章

滑移和翻转

*Spin that wheel
Go along for the ride
—Devo,
“Spin the Wheel”*

只在Web上发表文学名著已经没有那么吸引人了。人们强烈要求得到更多。他们想看到文字移动。他们想看到漂亮的画面。他们想看到滑移和滚动效果！在实例部分的最后一章，也就是本章中，我们将探讨如何使用超前的动画、超High的AJAX，以及超爽的视觉体验来满足人们的需要，让Web真真切切地旋转起来。

9.1 标题翻转效果

在第一个翻转效果的例子中，我们要取得一个新闻源并滚动显示新闻条目的标题及内容摘要，每次一条。与通常的新闻Ticker水平移动新闻条目不同，我们这个例子中的新闻条目是向上滚动。一条新闻滚进视图之后，会暂停几秒钟，然后继续向上滚动，淡出视图；与此同时，下一条新闻接着滚入视图。

9.1.1 设置页面

在最基本的层面上，这个功能不是很难实现。不过，稍后我们会看到，把基本的页面标记转换为成品确实需要一定的技巧。

同往常一样，我们仍然从一个HTML标记块开始。本例中的新闻源要显示在页面的侧边栏中：

```
<div id="sidebar">
  <!-- 省略的代码 -->
  <h3>Recent News</h3>
  <div id="news-feed">
    <a href="news/index.html">News Releases</a>
  </div>
</div>
```

现在，新闻源（news-feed）`<div>`中只包含一个指向主新闻页面的链接。这是我们的后备力

量，万一用户没有启用JavaScript，这个链接就可以披挂上阵。而我们要操作的内容来自一个实际的RSS源。

应用到这个

的CSS样式非常重要，因为这些样式不仅可以决定一次显示几个新闻条目，而且还能控制新闻条目在页面上出现的位置。将应用到单个新闻条目的样式规则放到一起，会得到如下CSS：

```
#news-feed {
    position: relative;
    height: 200px;
    width: 17em;
    overflow: hidden;
}

.headline {
    position: absolute;
    height: 200px;
    top: 210px;
    overflow: hidden;
}
```

我们注意到，单个新闻条目（由.headline类表示）和它们容器元素的height都是200px。而且，因为.headline是相对于#news-feed绝对定位的，所以可以将新闻条目的顶边设置为恰好位于它们容器的底边下方。这样，当为#news-feed设置overflow:hidden属性后，就可以有效地使新闻条目在它们的初始位置上隐藏起来。

为新闻条目设置position:absolute还有另外一个原因。任何元素如果想在页面上实现基于位置的动画效果，那么该元素必须采取absolute或relative定位，而不能是默认的static定位。

既然HTML和CSS都已经准备好了，接下来应该从RSS源中提取新闻条目。首先，我们要把代码包装在一个.each()方法中，这就像是某种if语句在一个私有的命名空间中包含着相应的代码：

```
$(document).ready(function() {
    $('#news-feed').each(function() {
        $(this).empty();
    });
});
```

当使用#news-feed选择符创建jQuery对象时，有两种可能的结果。第1种，\$()工厂函数会生成一个与带有news-feed ID的唯一元素匹配的jQuery对象；第2种，\$()函数没有在页面中找到带有该ID的元素，并生成一个空的jQuery对象。只有当返回的jQuery对象非空时，调用的.each()方法才会执行包含的代码。

在.each()方法之后，我们立即清空了新闻源

，以便盛放新内容。

9.1.2 取得新闻源

为取得新闻源，可以使用`$.get()`方法，这是jQuery提供的许多与服务器通信的方法中的一种。要了解与`$.get()`方法及其他AJAX方法有关的更多内容，请参见第6章。

新闻源的内容会以XML结构的形式传递给第1个参数（`data`），随后，这个参数将会用作选择符的环境：

```
$ (document).ready(function() {
    $('#news-feed').each(function() {
        $(this).empty();

        $.get('news/feed.xml', function(data) {
            $('rss//item', data).each(function() {

                // 省略的代码

            });
        });
    });
});
```

可以针对新闻源中的条目再使用一个`.each()`方法，以便把每个新闻条目的各个部分组合成一个可用的HTML标记块。首先，我们来构建新闻条目的链接：

```
$ (document).ready(function() {
    $('#news-feed').each(function() {
        $(this).empty();

        $.get('news/feed.xml', function(data) {
            $('rss//item', data).each(function() {
                var title = $('title', this).text();
                var linkText = $('link', this).text();
                var $link = $('</a>')
                    .attr('href', linkText)
                    .text(title);
                $link = $('

### </h3>').html($link); }); }); }); });


```

这里，我们分别取得每个条目中`<title>`和`<link>`元素的文本，然后构造一个`<a>`元素，将`linkText`变量设置为其`href`属性，将`title`变量设置为出现在`<a>`和``标签之间的文本。最后，将每个`<a>`元素再包装到一个`<h3>`元素中。

除了链接之外，我们还要重新格式化并插入每个条目的发布日期，添加每条摘要的HTML，将它们分别包装在各自的`<div>`中：

```

$(document).ready(function() {
    $('#news-feed').each(function() {
        $(this).empty();

        $.get('news/feed.xml', function(data) {
            $('rss//item', data).each(function() {
                var title = $('title', this).text();
                var linkText = $('link', this).text();
                var $link = $('')
                    .attr('href', linkText)
                    .text(title);
                $link = $('

### 

').html($link);

                var pubDate = new Date($('pubDate', this).text());
                var pubMonth = pubDate.getMonth() + 1;
                var pubDay = pubDate.getDate();
                var pubYear = pubDate.getFullYear();
                var $pubDiv = $('')
                    .addClass('publication-date')
                    .text(pubMonth + '/' + pubDay + '/' + pubYear);

                var summaryText = $('description', this).text();
                var $summary = $('')
                    .addClass('summary')
                    .html(summaryText);
            });
        });
    });
});

```



将新闻条目放到页面中的最后一步，涉及到再创建一个

并为其添加headline类，然后把\$link、\$pubDiv和\$summary都添加到这个

中。最后，再把前面创建的所有元素都添加到

中，而

已经存在于HTML中了：

```

$(document).ready(function() {
    $('#news-feed').each(function() {
        $(this).empty();

        $.get('news/feed.xml', function(data) {
            $('rss//item', data).each(function() {
                var title = $('title', this).text();
                var linkText = $('link', this).text();
                var $link = $('')
                    .attr('href', linkText)
                    .text(title);
                $link = $('

### 

').html($link);

                var pubDate = new Date($('pubDate', this).text());
                var pubMonth = pubDate.getMonth() + 1;
                var pubDay = pubDate.getDate();
                var pubYear = pubDate.getFullYear();
                var $pubDiv = $('')

```

```

.addClass('publication-date')
.text(pubMonth + '/' + pubDay + '/' + pubYear);
var summaryText =($('description', this).text();
var $summary = $('

</div>');
.addClass('summary')
.html(summaryText);

$('<div></div>')
.addClass('headline')
.append($link)
.append($pubDiv)
.append($summary)
.appendTo('#news-feed');

});
});
});
});
});


```

这样，页面中就包含了多个`<div class="headline">`元素，每个元素都带有一个标题、日期、链接和摘要——都已经准备就绪，随时可以显示。

9.1.3 设置翻转效果

在深入到标题翻转代码之前，还需要进行一番设置。首先，要设置两个变量，一个用于当前可见的标题^①，另一个用于刚刚滚动出视图的标题。在初始状态下，这两个变量的值都是0。

```
var currentHeadline = 0, oldHeadline = 0;
```

然后，要设置所有标题的初始位置。还记得吗，我们在样式表中把标题的`top`属性设置为比它们容器的`height`大10像素的值，从而隐藏了这些标题。为了后面方便，我们要把这个属性值保存在一个变量中，以便当把某个标题滚动出可见区域后，再把它重设回原来隐藏它的位置。我们也希望在页面加载后，立即显示第1个标题，因此可以吧它的`top`属性设置为0：

```
var hiddenPosition = $('#news-feed').height() + 10;
$('#div.headline: eq(' + currentHeadline + ')').css('top', '0');
```

这样，页面上的翻转区域已经具备了雏形，如图9-1所示。

最后，我们还要保存标题的总数并定义一个计时变量，这个计时变量将用于控制每次翻转之间的暂停：

```
var headlineCount = $('div.headline').length;
var headlineTimeout;
```

^① 这里的“标题”指的是与`.headline`匹配的元素，即包含标题、日期、链接和摘要的“标题新闻”。下同。

——译者注

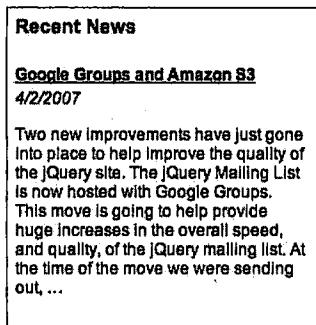


图 9-1

现在还不必为headlineTimeout设置值，该变量的值将在每次翻转发生时进行设置。但是，为了避免与同名的全局变量发生冲突，必须要使用var来声明变量。

9.1.4 标题翻转函数

现在，我们已经准备好翻转标题以及它们对应的日期和摘要了。为了重用代码，下面要为这个任务定义一个函数。这个函数的第一行会为currentHeadline的值加上1，然后再对headlineCount执行求模运算。这样，currentHeadline将会等于oldHeadline + 1，直到后者的值相当于headlineCount的值时，又会把currentHeadline的值重设为0。要了解有关模运算符的更详细的讨论，请参见7.4.1节。

这个函数的最后一行会在翻转发生之后将oldHeadline的值设置为currentHeadline的值。有了函数中这两行书夹式的代码，我们就能通过这两个变量来索引当前标题和上一个可见的标题了：

```
var headlineRotate = function() {
    currentHeadline = (oldHeadline + 1) % headlineCount;
    // 这里将填充翻转标题行的代码……
    oldHeadline = currentHeadline;
};
```

在这两行代码之间，我们要编写实际移动标题的代码。首先，要取得索引为oldHeadline的

元素，为该元素的top属性设置动画效果，即将该元素向上移动到看不见为止。然后，上述动画效果一旦完成，就将该元素的top属性设置回其原始的hiddenPosition:

```
var headlineRotate = function() {
    currentHeadline = (oldHeadline + 1) % headlineCount;
    $('div.headline:eq(' + oldHeadline + ')')
        .animate({top: -hiddenPosition}, 'slow', function() {
            $(this).css('top', hiddenPosition);
        });
    oldHeadline = currentHeadline;
};
```

由于hiddenPosition比

的高度大一些，所以将标题的顶部通过动画移动到-headlinePosition，会将该标题一直向上移动到隐藏在它的包含元素上方。然后，通过使用.animate方法的回调函数，确保了当动画完成后再把标题重新定位到其原始位置。

当前的标题也要同时向上滑入视图。然后，随着它的动画效果完成，我们使用setTimeout函数在5秒（5000毫秒）的暂停后，再次调用headlineRotate函数：

```
var headlineRotate = function() {
    currentHeadline = (oldHeadline + 1) % headlineCount;
    $('div.headline: eq(' + oldHeadline + ')')
        .animate({top: -hiddenPosition}, 'slow', function() {
            $(this).css('top', hiddenPosition);
        });
    $('div.headline: eq(' + currentHeadline + ')')
        .animate({top: 0}, 'slow', function() {
            headlineTimeout = setTimeout(headlineRotate, 5000);
        });
    oldHeadline = currentHeadline;
};
```

在编写完成headlineRotate函数后，还必须调用它。虽然它能够在动画运行后通过内部代码调用自己，但仍然需要经过初始调用才能在文档就绪时启动效果。为此，我们要做的就是在函数的定义之后重复一次headlineTimeout所在的那行代码。在添加了这行代码后，迄今为止的完整代码应该如下所示：

```
$(document).ready(function() {
    $('#news-feed').each(function() {
        $(this).empty();

        // 取得新闻源
        $.get('news/feed.xml', function(data) {
            $('rss//item', data).each(function() {
                var title = $('title', this).text();
                var linkText = $('link', this).text();
                var $link = $('</a>')
                    .attr('href', linkText)
                    .text(title);
                $link = $('<h3></h3>').html($link);

                var pubDate = new Date($('pubDate', this).text());
                var pubMonth = pubDate.getMonth() + 1;
                var pubDay = pubDate.getDate();
                var pubYear = pubDate.getFullYear();
                var $pubDiv = $('<div></div>')
                    .addClass('publication-date')
                    .text(pubMonth + '/' + pubDay + '/' + pubYear);
                var summaryText = $('description', this).text();
                var $summary = $('<div></div>')
```

```

.addClass('summary')
.html(summaryText);
$('<div></div>')
.append($link)
.append($pubDiv)
.append($summary)
.appendTo('#news-feed');
});

// 设置翻转变量
var currentHeadline = 0, oldHeadline = 0;
var hiddenPosition = ($('#news-feed').height() + 10);
$('div.headline: eq(' + currentHeadline + ')').css('top', '0');

var headlineCount = $('div.headline').length;
var headlineTimeout;

// 执行翻转
var headlineRotate = function() {
    currentHeadline = (oldHeadline + 1) % headlineCount;
    $('div.headline: eq(' + oldHeadline + ')')
        .animate({top: -hiddenPosition}, 'slow', function() {
            $(this).css('top', hiddenPosition);
        });
    $('div.headline: eq(' + currentHeadline + ')')
        .animate({top: 0}, 'slow', function() {
            headlineTimeout = setTimeout(headlineRotate, 5000);
        });
    oldHeadline = currentHeadline;
};

headlineTimeout = setTimeout(headlineRotate, 5000);
}); // $.get()方法结束
}); // #news-feed的.each()方法结束
});

```

9.1.5 悬停时暂停

尽管标题翻转的功能完全实现了，但我们还必须提到一个可用性问题——当用户想单击某个标题中的链接时，如果该标题滚动出了可见区域，那么用户就必须等到整整一组标题全部循环完毕。如果当用户把鼠标放到标题中的任意位置上，滚动效果会暂停下来，那么发生上述问题的可能性就会降低。

```

$('#news-feed').hover(function() {
    clearTimeout(headlineTimeout);
}, function() {
    headlineTimeout = setTimeout(headlineRotate, 250);
});

```

.hover方法中的代码会在

元素上发生mouseover事件时，调用JavaScript的clearTimeout函数，从而有效地阻止再次调用headlineRotate函数。当mouseout

事件发生时，会在250毫秒的短暂延迟后再次调用headlineRotate函数。

这几行简单的代码基本上能够维持正常运行。但是，如果用户的鼠标反复在这个

元素上面移入移出，那么就会导致一个非常令人讨厌的问题：多个标题会相互层叠在可见区域中，如图9-2所示。

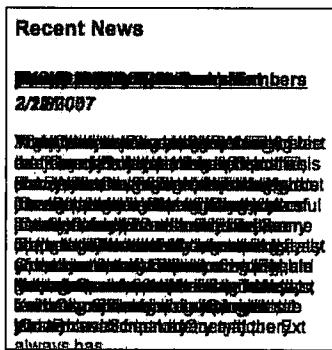


图 9-2

遗憾的是，我们必须通过一次“大手术”来除掉这个毒瘤。

首先，要在headlineRotate函数之前再引入一个变量：

```
var rotateInProgress = false;
```

然后，在这个函数中的第1行检查翻转效果当前是否处于执行过程中。只有当rotateInProgress的值是false时，才允许再次运行后面的代码。为此，我们要把函数中的所有代码包装在一个if语句当中。在if语句之后，立即将rotateInProgress变量设置为true。然后，在第2个.animate方法的回调函数中，再将这个变量的值设置回false：

```
var headlineRotate = function() {
    if (!rotateInProgress) {
        rotateInProgress = true;
        currentHeadline = (oldHeadline + 1) % headlineCount;
        $('div.headline: eq(' + oldHeadline + ')')
            .animate({top: -hiddenPosition}, 'slow', function() {
                $(this).css('top', hiddenPosition);
            });
        $('div.headline: eq(' + currentHeadline + ')')
            .animate({top: 0}, 'slow', function() {
                rotateInProgress = false;
                headlineTimeout = setTimeout(headlineRotate, 5000);
            });
        oldHeadline = currentHeadline;
    }
};
```

新添加的这些代码有效地增强了标题翻转效果。重复的mouseover-mouseout行为不会再导致标题相互堆积的结果。不过，这个重复行为仍然给我们留下了一个小问题：后续的标题会按照不同的步调出现，两三个标题会很快地前后跟随着出现，而不是按照5秒钟的间隔有序地滑入。

问题的原因是，如果用户的鼠标离开这个

时，现有的计时器还没有完成，就会导致并发存在多个计时器。因而，我们还需要再添加一个安全装置，即在这个函数的顶部和.hover()方法内部的clearTimeout()语句之后，将headlineTimeout变量设置为false。然后，在使用headlineTimeout调用headlineRotate函数的两个位置上，通过检查确保该变量的值是false。这样，就可以保证在现有的计时器没有完成之前，不会设置新计时器：

```
var headlineRotate = function() {
    if (!rotateInProgress) {
        rotateInProgress = true;
        headlineTimeout = false;

        currentHeadline = (oldHeadline + 1) % headlineCount;
        $('div.headline:eq(' + oldHeadline + ')')
            .animate({top: -hiddenPosition}, 'slow', function() {
                $(this).css('top', hiddenPosition);
            });
        $('div.headline:eq(' + currentHeadline + ')')
            .animate({top: 0}, 'slow', function() {
                rotateInProgress = false;
            });
        if (!headlineTimeout) {
            headlineTimeout = setTimeout(headlineRotate, 5000);
        }
    });
    oldHeadline = currentHeadline;
};

headlineTimeout = setTimeout(headlineRotate, 5000);

$('#news-feed').hover(function() {
    clearTimeout(headlineTimeout);
    headlineTimeout = false;
}, function() {
    if (!headlineTimeout) {
        headlineTimeout = setTimeout(headlineRotate, 250);
    }
});
```

最终，我们构建的标题翻转效果能够经受住鼠标的各种不正常行为。

9.1.6 从不同的域中取得新闻源

我们在前面的例子中使用的新闻源是一个本地文件，但是，我们想要的是从另外一个网站取得新闻源。虽然针对跨站点的数据检索有许多种解决方案，但是我们这里只介绍使用PHP的方案。为此，需要创建一个名为feed.php的新文件（不是feed.xml了），然后在\$.get方法中引用它：

```
$.get('news/feed.php', function(data) {
    // 省略的代码
})
```

在feed.php文件内部，需要取得跨站点新闻源的内容，相应的代码如下所示：

```
<?php
header('Content-Type: text/xml');
print file_get_contents('http://jquery.com/blog/feed');
?>
```

这里需要注意的是，必须明确地将响应的Content-Type设置为text/xml，以便jQuery取得并正确地解析新闻源。而且，出于安全方面的考虑，有些网站主机提供商可能不允许使用PHP的file_get_contents函数。

取决于很多因素，加载这样的远程文件可能会花一定的时间。因此，我们可以在\$.get()请求启动时添加一幅图像，告诉用户正在加载标题。当请求完成时，再移除这幅图像：

```
$(document).ready(function() {
    $('#news-feed').each(function() {
        $(this).empty();

        var $newsLoading = $('')
            .attr({
                'src': '/cookbook/images/loading.gif',
                'alt': 'loading. please wait'
            })
            .addClass('news-wait');
        $(this).ajaxStart(function() {
            $(this).append($newsLoading);
        }).ajaxStop(function() {
            $newsLoading.remove();
        });
    });
    // 省略的代码
});
});
```

现在，当页面初次加载时，如果取得标题内容遇到了网络延迟，我们就能看到一幅加载中图像，而不是一个空白区域，如图9-3所示。

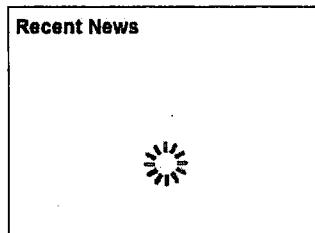


图 9-3

这幅图像是一个GIF动画，因此在浏览器中明显要比印刷在纸上更有动感。

9.1.7 附加的内部渐变效果

在完成标题翻转效果之前，我们再修饰最后一笔——让标题中的文本呈现出逐渐淡入到背景中的效果。要实现这种视觉效果，可以创建一系列

元素，并为每个元素递增地设置一定的`opacity`和`top`值。所有这些div“切片”都具有一些相同的样式属性，我们可以在样式表中声明这些属性：

```
.fade-slice {
    position: absolute;
    width: 20em;
    height: 2px;
    background: #efd;
    z-index: 3;
}
```

它们具有同包含元素

一样的width和background-color。接下来，我们通过设置所有这些

加在一起的高度，来确定要创建的

元素的数量。在这里，我们希望所有这些

达到

高度的25%。然后，运行for循环，从0开始递增到组合的渐变区域的高度，每次增量为2：

```
$(document).ready(function() {
    $('#news-feed').each(function() {
        var $this = $(this);
        $this.empty();

        var totalheight = $this.height();
        var fadeHeight = $('#news-feed').height() / 4;
        for (var i = 0; i < fadeHeight; i+=2) {
            $('')
                .addClass('fade-slice')
                .appendTo(this);
        }
        // 省略的代码
    });
});
```

由于下面需要非常频繁地使用\$(this)这个jQuery对象，所以我们为它声明了一个\$this变量，以便在不过多占用系统资源的情况下重用这个对象。

在for循环中，我们没有使用标准的`i++`增量，而使用了`i+=2`，即增量为2。这是因为每个切片的高度为2像素。在

的高度被设置为200像素的情况下，`fadeHeight`的值就是50，这样就会依次生成25个

元素，按照样式表中的定义，它们每个都是2像素高。

现在，只需通过一些数学计算来确定每个元素的`opacity`和`top`属性：

```

$(document).ready(function() {
  $('#news-feed').each(function() {
    var $this = $(this);
    $this.empty();

    var totalheight = $this.height();
    var fadeHeight = $totalheight() / 4;
    for (var i = 0; i < fadeHeight; i+=2) {
      $('<div></div>')
        .css({
          opacity: i / fadeHeight,
          top: $totalHeight - fadeHeight + i
        })
        .addClass('fade-slice')
        .appendTo(this);
    }
    // 省略的代码
  });
});

```

如图9-4所示，`opacity`的值从0开始，步长为.04，并且会持续递增到.96——几乎完全不透明。其间，`top`值从150开始，每次增量为2，直至达到198为止。

i	/	fadeHeight	=	opacity		totalHeight	/	fadeHeight	+	i	=	top
1	0 /	50	=	0.00		200	-	50	+	0 =	150	
2	2 /	50	=	0.04		200	-	50	+	2 =	152	
3	4 /	50	=	0.08		200	-	50	+	4 =	154	
4	6 /	50	=	0.12		200	-	50	+	6 =	156	
5	8 /	50	=	0.16		200	-	50	+	8 =	158	
6	10 /	50	=	0.20		200	-	50	+	10 =	160	
7	12 /	50	=	0.24		200	-	50	+	12 =	162	
8	14 /	50	=	0.28		200	-	50	+	14 =	164	
9	16 /	50	=	0.32		200	-	50	+	16 =	166	
10	18 /	50	=	0.36		200	-	50	+	18 =	168	
11	20 /	50	=	0.40		200	-	50	+	20 =	170	
12	22 /	50	=	0.44		200	-	50	+	22 =	172	
13	24 /	50	=	0.48		200	-	50	+	24 =	174	
14	26 /	50	=	0.52		200	-	50	+	26 =	176	
15	28 /	50	=	0.56		200	-	50	+	28 =	178	
16	30 /	50	=	0.60		200	-	50	+	30 =	180	
17	32 /	50	=	0.64		200	-	50	+	32 =	182	
18	34 /	50	=	0.68		200	-	50	+	34 =	184	
19	36 /	50	=	0.72		200	-	50	+	36 =	186	
20	38 /	50	=	0.76		200	-	50	+	38 =	188	
21	40 /	50	=	0.80		200	-	50	+	40 =	190	
22	42 /	50	=	0.84		200	-	50	+	42 =	192	
23	44 /	50	=	0.88		200	-	50	+	44 =	194	
24	46 /	50	=	0.92		200	-	50	+	46 =	196	
25	48 /	50	=	0.96		200	-	50	+	48 =	198	

图 9-4

因为最后一个`<div class="fade-slice">`的上边位置是198，而它的高度是2像素，因此它

会完美地覆盖包含它的200像素高的

底部。

编写完这些代码之后，页面标题区域中的文本在从

底部向上滚动时，就会显示出从不透明到透明的优美的渐变效果，如图9-5所示。

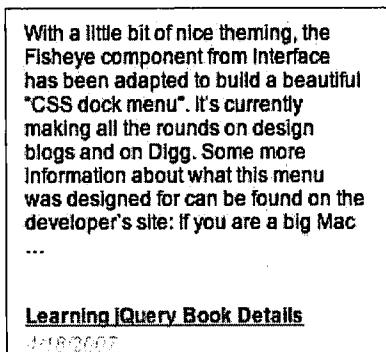


图 9-5

9.2 图像传送带

作为滑移页面内容的另一个例子，我们要在网上书店的首页中实现一个图像画廊。这个画廊用于展示一些热销的特色图书，每本图书都带有一个指向大幅封面图像的链接。与前面新闻Ticker例子中的标题基于既定的时间表移动不同，这里，我们要在用户单击一个封面时，通过jQuery实现图像在屏幕上横向滑动的效果。

基于jQuery的插件jCarousel实现了另一种循环滚动一组图像的机制。虽然与我们下面要实现的机制不同，但通过该插件能够以非常少的代码实现高水平的滑移效果。有关使用插件的更多信息将在第10章中讨论。

9.2.1 设置页面

同以前一样，我们仍然是从构建基本的HTML和CSS开始，以便没有JavaScript的用户也能通过美观和实用的界面获得这些信息：

```
<div id="featured-books">
<div class="covers">
<a href="covers/large/1847190871.jpg"
    title="Community Server Quickly">
    
    <span class="price">$35.99</span>
</a>
<a href="covers/large/1847190901.jpg"
    title="Deep Inside osCommerce: The Cookbook">
```

```


<span class="price">$44.99</span>
</a>
<a href="covers/large/1847190979.jpg" title="Learn OpenOffice.org
    Spreadsheet Macro Programming: OOoBasic and Calc automation">
    
    <span class="price">$35.99</span>
</a>
<a href="covers/large/1847190987.jpg" title="Microsoft AJAX C#
    Essentials: Building Responsive ASP.NET 2.0 Applications">
    
    <span class="price">$31.99</span>
</a>
<a href="covers/large/1847191002.jpg"
      title="Google Web Toolkit GWT Java AJAX Programming">
    
    <span class="price">$40.49</span>
</a>
<a href="covers/large/1847192386.jpg"
      title="Building Websites with Joomla! 1.5 Beta 1">
    
    <span class="price">$40.49</span>
</a>
</div>
</div>

```

其中，每幅图像都包含在一个指向放大封面的锚标签中。每个封面都带有一个价格标签，但现在它们是被隐藏的，稍后我们会通过JavaScript在适当的时机显示它们。

为节省首面的空间，我们希望每次只显示其中3个封面。在没有JavaScript的情况下，可以通过将容器元素的overflow属性设置为scroll，并适当地调整容器的宽度来实现这一点：

```

.featured-books {
    position: relative;
    background: #ddd;
    width: 440px;
    height: 186px;
    overflow: scroll;
    margin: 1em auto;
    padding: 0;
    text-align: center;
    z-index: 2;
}

```

```
#featured-books .covers {
    position: relative;
    width: 840px;
    z-index: 1;
}
#featured-books a {
    float: left;
    margin: 10px;
    height: 146px;
}
#featured-books .price {
    display: none;
}
```

这些样式有必要讨论一下。首先，最外层的元素需要具有比它内部的元素更大的z-index属性，以保证在IE中能够隐藏内部元素延伸到容器外面的部分。其次，我们把外部元素的宽度设置为440px，恰好可以容纳3幅图像（每幅图像宽度为120px）、每幅图像10像素的外边距以及20像素的垂直滚动条。

在设置了这些样式之后，通过标准的系统滚动条也可以浏览所有图像，如图9-6所示。



图 9-6

9.2.2 通过 JavaScript 修改样式

目前，我们已经完成了在没有JavaScript的情况下可以使用的图像画廊。接下来，需要撤销一些现有的细节。一个是在实现滚动机制时多余的滚动条，另一个是应用了float属性的自动封面布局，这个浮动布局会妨碍我们在为封面添加动画效果时使用定位。因此，第1件事就是重写一些样式：

```
$(document).ready(function() {
    var spacing = 140;
    $('#featured-books').css({
        'width': spacing * 3,
```

```

    'height': '166px',
    'overflow': 'hidden'
}).find('.covers a').css({
    'float': 'none',
    'position': 'absolute',
    'left': 1000
});

var $covers = $('#featured-books .covers a');

$covers.eq(0).css('left', 0);
$covers.eq(1).css('left', spacing);
$covers.eq(2).css('left', spacing * 2);
});

```

此外，后面会在很多需要计算的地方用到变量spacing，它表示封面图像的宽度加上左右两边的外边距。由于不再需要为滚动条留出空间，所以我们可以把包含元素的宽度设置为恰好容纳3幅图像。没错，通过把overflow属性修改为hidden，我们跟水平滚动条也说再见了。

接下来，我们对所有封面图像都采取了绝对定位，并让它们的左坐标从1000开始，这样就把它都放到了可见区域之外。然后，我们移动前3个封面的位置，每次移动一个。而保存所有锚元素的变量\$covers也将在后面派上用场。

这样，在撤销了滚动条机制的同时，我们让前3个封面显示到了可见区域中，如图9-7所示。

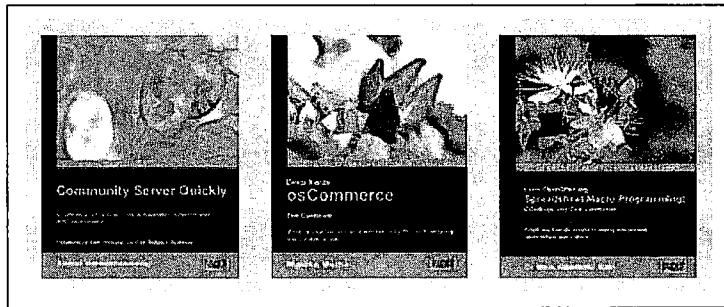


图 9-7

9.2.3 通过单击滑移图像

下面，要为单击两端图像的事件添加响应代码，并按照需要对封面图像重新排序。当用户单击左侧的封面时，说明用户希望看到更多左侧的图像，也就是说，需要把封面向右移动。类似地，当用户单击右侧的封面时，则必须向左移动封面。总而言之，我们需要把这条“传送带”首尾相连，也就是要把左侧减少的图像续接到右侧，反之亦然。首先，只改变图像的位置，不添加动画效果：

```

$(document).ready(function() {
    var spacing = 140;

```

```

$( '#featured-books' ).css({
  'width': spacing * 3,
  'height': '166px',
  'overflow': 'hidden'
}).find('.covers a').css({
  'float': 'none',
  'position': 'absolute',
  'left': 1000
});
var setUpCovers = function() {
  var $covers = $('#featured-books .covers a');
  $covers.unbind('click');

  // 左侧的图像; 当单击时向右滚动 (以便查看左侧的图像)
  $covers.eq(0).css('left', 0).click(function(event) {
    $covers.eq(2).css('left', 1000);
    $covers.eq($covers.length - 1).prependTo(
      '#featured-books .covers');
    setUpCovers();
    event.preventDefault();
  });

  // 右侧的图像; 当单击时向左滚动 (以便查看右侧的图像)
  $covers.eq(2).css('left', spacing * 2).click(function(event) {
    $covers.eq(0).css('left', 1000);
    $covers.eq(0).appendTo('#featured-books .covers');
    setUpCovers();

    event.preventDefault();
  });

  // 中间的图像
  $covers.eq(1).css('left', spacing);
};

setUpCovers();
});

```

这个新setUpCovers函数整合了我们前面编写的图像定位代码。通过把这些代码封装到函数中，可以在对元素进行重新排序后，方便地重新定位图像。

在这个例子中，总共有6幅图像（JavaScript将通过数字0~5引用它们），而编号为0、1和2的图像可见。当用户单击0号图像时，需要把所有图像都向右移动一个位置。因为2号图像在这次移动后将不可见，所以先把2号图像移出可见区域。然后，再把位于队尾的图像（5号）转移到队列的前头，这样会导致所有图像重新排序。因此，当再次调用setUpCovers()时，原来的5号就变成了现在的0号，而原来的0号变成了1号，1号则变成了2号。于是，现有的这些定位代码就足以把封面移动到各自的新位置上了，如图9-8所示。

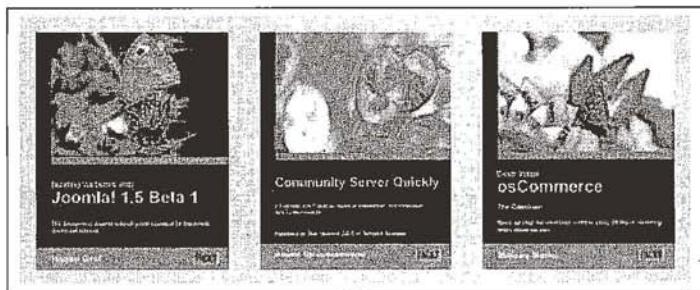


图 9-8

单击2号图像会触发相反的过程。此时会隐藏0号图像，并把它移动到队尾。因而，会使1号变成0号，2号变成1号，3号变成2号。

此外，为了避免用户操作导致不正常的结果，还要注意两个细节：

(1) 由于封面都链接到相应的大幅图像，所以要在单击处理程序中调用`.preventDefault()`。如果不调用这个方法，那么单击封面会打开链接，从而导致用户看不到滑移效果。

(2) 需要在`setUpCovers()`函数一开始取消对单击处理程序的绑定。否则，在“传送带”转动期间，一幅图像会被绑定多个处理程序。

9.2.4 添加滑移效果

现在，当单击图像时，由于封面都同时移动，所以很难说清图像是怎么变化的——虽然看起来确实变了，但没有看到它们移动。为了揭示封面移动的过程，可以通过添加动画效果让封面滑移到新位置，而不是突然出现在新位置。为此，我们需要修改`setUpCovers`函数：

```
var setUpCovers = function() {
    var $covers = $('#featured-books .covers a');

    $covers.unbind('click');

    // 左侧的图像：当单击时向右滚动（以便查看左侧的图像）
    $covers.eq(0).css('left', 0).click(function(event) {
        $covers.eq(0).animate({'left': spacing}, 'fast');
        $covers.eq(1).animate({'left': spacing * 2}, 'fast');
        $covers.eq(2).animate({'left': spacing * 3}, 'fast');
        $covers.eq($covers.length - 1).css('left', -spacing).animate({
            'left': 0}, 'fast', function() {
            $(this).prependTo('#featured-books .covers');
            setUpCovers();
        });
        event.preventDefault();
    });

    // 右侧的图像：当单击时向左滚动（以便查看右侧的图像）
}
```

```

$covers.eq(2).css('left', spacing * 2).click(function(event) {
  $covers.eq(0).animate({'left': -spacing}, 'fast', function() {
    $(this).appendTo('#featured-books .covers');
    setUpCovers();
  });
  $covers.eq(1).animate({'left': 0}, 'fast');
  $covers.eq(2).animate({'left': spacing}, 'fast');
  $covers.eq(3).css('left', spacing * 3).animate({
    'left': spacing * 2}, 'fast');

  event.preventDefault();
});

// 中间的图像
$covers.eq(1).css('left', spacing);
};

```

此时，当单击左侧图像时，我们先将所有3幅可见的图像都向右移动一幅图像的宽度（重用了前面定义的spacing变量）。这一部分的代码很直观，但我们还必须考虑如何让新图像滑进视图。为此，需要先从队尾取得该图像，并把它在屏幕上的位置移动到恰好位于画面左侧之外。然后，再把它连同其他图像一起滑进视图，如图9-9所示。

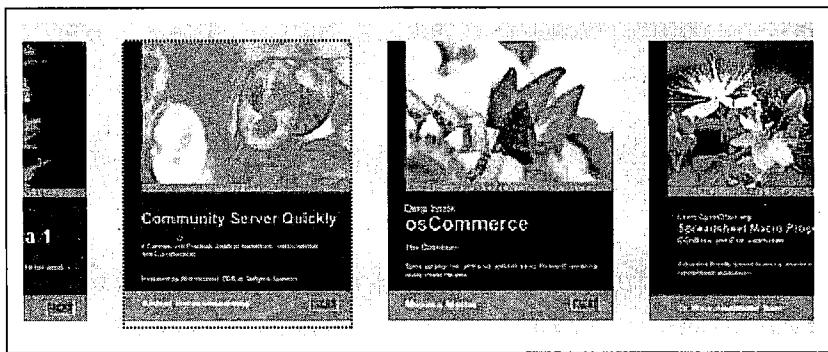


图 9-9

尽管动画效果负责完成了初始的移动，我们仍然需要通过再次调用setUpCovers()改变封面的顺序。如果不调用setUpCovers()，那么下一次单击将无法正常发生作用。由于setUpCovers()用于改变封面的位置，必须要延迟到动画完成后再调用它，所以我们把调用语句放在了动画的回调函数中。

9.2.5 显示操作图标

现在，图像“传送带”实现了平滑的移动效果。但是，我们尚未给用户提供任何单击封面可以移动图像的提示。为此，可以在用户鼠标悬停到图像上时，显示一个适当的提示性图标。

在这个例子中，我们将把图标放到现有图像的上层。通过设置`opacity`属性，可以使用户在图标显示时仍然能够看到底层的封面。为保证封面不会太模糊，我们使用简单的单色图标，如图9-10所示。

在这个例子中，总共需要3个图标，分别用于表示向左滑动、向右滑动和显示中间封面的放大版。下面，我们创建这3个图标，将它们保存到变量中以便后面使用：

```
var $leftRollover = $('')
    .attr('src', 'images/left.gif')
    .addClass('control')
    .css('opacity', 0.6)
    .hide();
var $rightRollover = $('')
    .attr('src', 'images/right.gif')
    .addClass('control')
    .css('opacity', 0.6)
    .hide();
var $enlargeRollover = $('')
    .attr('src', 'images/enlarge.gif')
    .addClass('control')
    .css('opacity', 0.6)
    .hide();
```

但是，以上代码存在过多的冗余。可以把这些代码提取到一个函数中，然后在需要创建图标时调用它：

```
function createControl(src) {
    return $('')
        .attr('src', src)
        .addClass('control')
        .css('opacity', 0.6)
        .hide();
}
var $leftRollover = createControl('images/left.gif');
var $rightRollover = createControl('images/right.gif');
var $enlargeRollover = createControl('images/enlarge.gif');
```

在页面的CSS中，我们为这些控件设置了比图像更高的`z-index`属性，然后对它们进行绝对定位，以便它们与封面重叠起来：

```
#featured-books .control {
    position: absolute;
    z-index: 3;
    left: 0;
    top: 0;
}
```

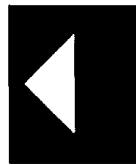


图 9-10

这些翻转图标都共享了相同的control类，因此您可能想在这个CSS样式表中设置它们的opacity属性。然而，不同的浏览器处理元素不透明度的方式并不一致，比如在IE中，表示60%的不透明度的语法就是filter: alpha(opacity=60)。为了避免这些差异，应该使用jQuery的.css方法来设置opacity样式，该方法隐藏了浏览器间的不一致行为。

现在，我们要做的就是在hover处理程序中把这些图标放到正确的DOM位置上：

```
var setUpCovers = function() {
    var $covers = $('#featured-books .covers a');

    $covers.unbind('click').unbind('mouseover').unbind('mouseout');

    // 左侧的图像：当单击时向右滚动（以便查看左侧的图像）
    $covers.eq(0).css('left', 0).click(function(event) {
        $covers.eq(0).animate({'left': spacing}, 'fast');
        $covers.eq(1).animate({'left': spacing * 2}, 'fast');
        $covers.eq(2).animate({'left': spacing * 3}, 'fast');
        $covers.eq($covers.length - 1).css('left', -spacing).
        animate({'left': 0}, 'fast', function() {
            $(this).prependTo('#featured-books .covers');
            setUpCovers();
        });
        event.preventDefault();
    }).hover(function() {
        $leftRollover.appendTo(this).show();
    }, function() {
        $leftRollover.hide();
    });
});

// 右侧的图像：当单击时向左滚动（以便查看右侧的图像）
$covers.eq(2).css('left', spacing * 2).click(function(event) {
    $covers.eq(0).animate({'left': -spacing}, 'fast', function() {
        $(this).appendTo('#featured-books .covers');
        setUpCovers();
    });
    $covers.eq(1).animate({'left': 0}, 'fast');
    $covers.eq(2).animate({'left': spacing}, 'fast');
    $covers.eq(3).css('left', spacing * 3).animate(
        {'left': spacing * 2}, 'fast');

    event.preventDefault();
}).hover(function() {
    $rightRollover.appendTo(this).show();
}, function() {
    $rightRollover.hide();
});

// 中间的图像
$covers.eq(1).css('left', spacing).hover(function() {
    $enlargeRollover.appendTo(this).show();
```

```

}, function() {
    $enlargeRollover.hide();
});
}
;

```

同处理click事件时一样，为了不导致悬停行为的累积，我们在setUpCovers()函数的开始也取消了对mouseover和mouseout处理程序的绑定。

现在，当鼠标指针放到一个封面上时，相应的翻转图像就会覆盖到该封面的上方，如图9-11所示。

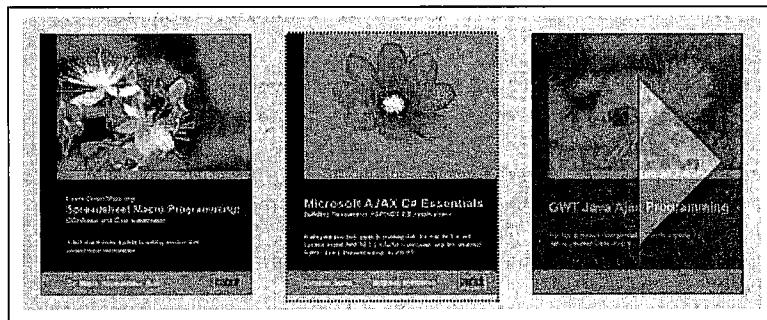


图 9-11

9.3 放大图像

我们的图像画廊已经具有了足够的功能，用户通过“传送带”可以找到想要的图像，而单击中间的图像会看到相应封面的放大视图。不过，这里的图像放大功能还有待于进一步改进。

与其在用户单击中间的图像时打开另一个URL，不如将大幅图像封面叠放在当前页面上。基于jQuery的Thickbox插件提供了另一种在页面上显示叠放信息的方式，但我们这里将在不使用插件的情况下实现这一功能。有关使用插件的更多信息可以参见第10章。

为显示大幅封面图像，还需要创建一个新的图像元素。可以在实例化悬停图标的同时创建这个图像元素：

```

var $enlargedCover = $('')
    .addClass('enlarged')
    .hide()
    .appendTo('body');

```

而且，还要像以前一样为这个新类添加一组类似的样式规则：

```

img.enlarged {
    position: absolute;
    z-index: 5;
    cursor: pointer;
}

```

通过绝对定位可以使这个大幅封面悬浮在其他已经定位的图像上方，因为它的z-index值到目前为止是最高的。接下来，需要在用户单击“传送带”中间的图像时实际地定位这幅放大的图像：

```
// 中间的图像：当单击时放大封面
$covers.eq(1).css('left', spacing).click(function(event) {
  $enlargedCover.attr('src', $(this).attr('href')).css({
    'left': ($('body').width() - 360) / 2,
    'top' : 100,
    'width': 360,
    'height': 444
  }).show();
  event.preventDefault();
}).hover(function() {
  $enlargeRollover.appendTo(this).show();
}, function() {
  $enlargeRollover.hide();
});
```

通过HTML源代码中的链接，可以找到大幅封面图像在服务器上的存放位置。因此，我们从链接的href属性中找到相应的地址，并将该URL设置为大幅封面图像的src属性。

下面必须要定位图像。目前，图像的top、width和height属性采用的都是硬编码方式，只有left属性进行了一些计算。我们想让大幅图像居中显示在页面上，但事先却不知道实现相应定位的适当坐标值。为此，可以通过取得body元素的宽度并除以2得到页面水平中点位置的坐标。而大幅图像的一半位置可以放在页面中点坐标的任何一边。因此这幅图像的左坐标应该是`($('body').width()-360)/2`，其中360是大幅封面图像的宽度。这样，我们通过适当地定位，就把大幅封面图像水平居中地放到了页面上，如图9-12所示。



图 9-12

9.3.1 隐藏大幅封面

在显示大幅封面后，还需要通过某种机制来隐藏它。最简单的方式就是通过这个封面的单击事件将它淡出视图：

```
// 中间的图像：当单击时放大封面
$covers.eq(1).css('left', spacing).click(function(event) {
  $enlargedCover.attr('src', $(this).attr('href')).css({
    'left': ($('#body').width() - 360) / 2,
    'top' : 100,
    'width': 360,
    'height': 444
  }).show()
  .one('click', function() {
    $enlargedCover.fadeOut();
  });

  event.preventDefault();
}).hover(function() {
  $enlargeRollover.appendTo(this).show();
}, function() {
  $enlargeRollover.hide();
});
```

这里，我们使用`.one`方法绑定了单击处理程序，从而避开了两个可能的问题。如果使用常规的`.bind()`方法，用户在图像淡出后仍然可以单击该图像。而且，由于每次显示大幅封面时都重用了相同的图像元素，所以每次放大操作都会绑定处理程序。如果不解除这些绑定的处理程序，就会导致它们随时间推移而累积。使用`.one()`方法则可以确保这些处理程序一经触发就会被移除。

显示关闭按钮

虽然这个行为能够移除大幅封面图像，但我们还没有就单击可以隐藏封面向用户给出提示。为此，需要在大幅封面图像上添加一个关闭按钮作为标记。同创建前面使用的单个元素类似，可以调用前面定义的辅助函数来创建这个按钮：

```
var $closeButton = createControl('images/close.gif')
.addClass('enlarged-control')
.appendTo('body');
```

当用户单击了中间的封面并显示出大幅封面后，需要定位并显示这个按钮：

```
$closeButton.css({
  'left': ($('#body').width() - 360) / 2,
  'top' : 100
}).show();
```

关闭按钮的坐标与大幅封面的坐标相同，因此它们的左上角是对齐的，如图9-13所示。

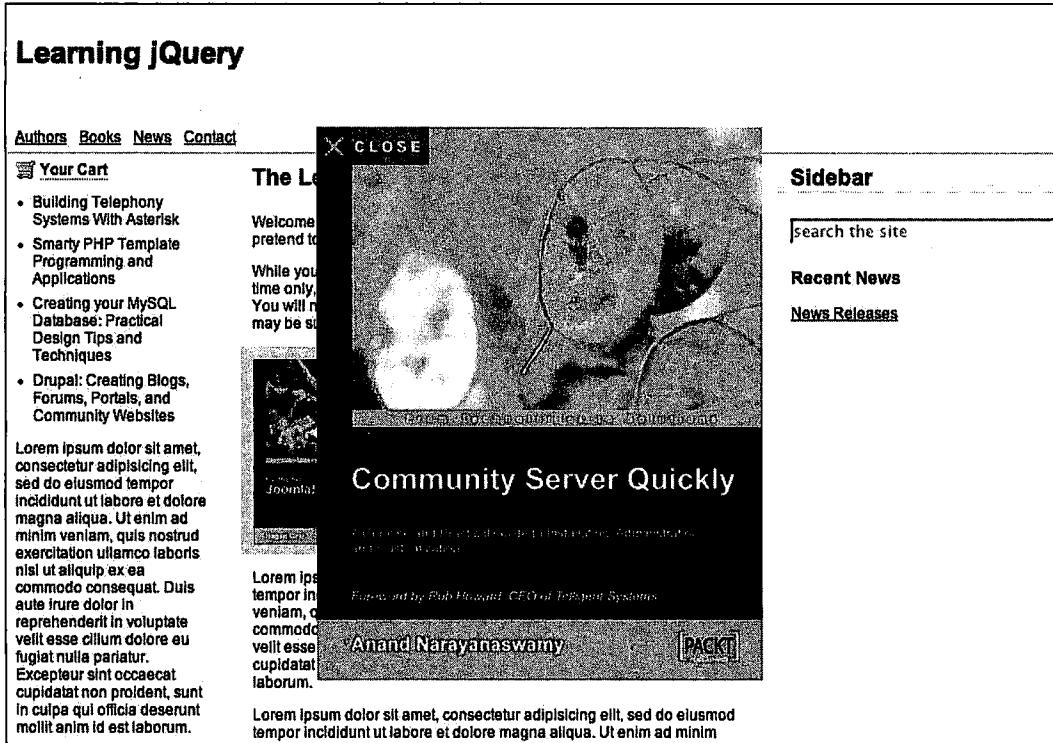


图 9-13

由于我们在大幅封面上已经绑定了单击隐藏它的行为，而在这种情况下通常可以依靠事件冒泡让单击关闭按钮也具有同样的行为。但是，在这个例子中，无论从外观上看怎么样，关闭按钮并不是大幅封面图像的后代元素。虽然关闭按钮被绝对地定位在了封面上，但在这个按钮上发生的单击事件不会传递到大幅图像。为此，必须在关闭按钮自身上处理单击事件：

```
// 中间的图像：当单击时放大封面
$covers.eq(1).css('left', spacing).click(function(event) {
    $enlargedCover.attr('src', $(this).attr('href')).css({
        'left': ($('body').width() - 360) / 2,
        'top': 100,
        'width': 360,
        'height': 444
    }).show()
    .one('click', function() {
        $closeButton.unbind('click').hide();
        $enlargedCover.fadeOut();
    });
});

$ccloseButton.css({
    'left': ($('body').width() - 360) / 2,
    'top': 100
});
```

```

}).click(function() {
    $enlargedCover.click();
}).show();

event.preventDefault();
}).hover(function() {
    $enlargeRollover.appendTo(this).show();
}, function() {
    $enlargeRollover.hide();
});

```

这样，当显示关闭按钮时，我们就为它绑定了一个单击事件处理程序。而所有这些处理程序要做的，就是触发已经绑定在大幅封面上的单击处理程序。不过，此时还需要修改大幅封面上的单击处理程序，通过它来隐藏关闭按钮。最后，还必须解除对这个单击处理程序的绑定，以防止随着时间推移造成处理程序的累积。

9.3.2 更有价值的标记

我们记得，在HTML源代码中还包含着每本书的价格信息，而当显示大幅图书封面时，也可以将这个附加信息显示在封面上。这次，我们要把应用于关闭按钮的技术再应用到一个文本化的内容元素上。

同样，需要在JavaScript代码的开始处创建一个新元素：

```

var $priceBadge = $('')
    .addClass('enlarged-price')
    .css('opacity', 0.6)
    .css('display', 'none')
    .appendTo('body');

```

由于价格标记也是部分透明的，所以字体颜色和背景之间保持高对比度能够取得最佳效果：

```

.enlarged-price {
    background-color: #373c40;
    color: #fff;
    width: 80px;
    padding: 5px;
    font-size: 18px;
    font-weight: bold;
    text-align: right;
    position: absolute;
    z-index: 6;
}

```

在显示价格标记之前，需要用HTML中的实际价格信息填充这个元素。还记得吗，在中间封面的单击处理程序中，this引用的是相应的链接元素，而价格信息就包含在这个链接中的元素里。因此，获得这个文本非常简单：

```
var price = $(this).find('.price').text();
```

下面，就可以在显示大幅封面时同时显示价格标记了：

```
$priceBadge.css({
  'right': ($('body').width() - 360) / 2,
  'top' : 100
}).text(price).show();
```

如图9-14所示，价格标记被定位在了放大图像的右上角。

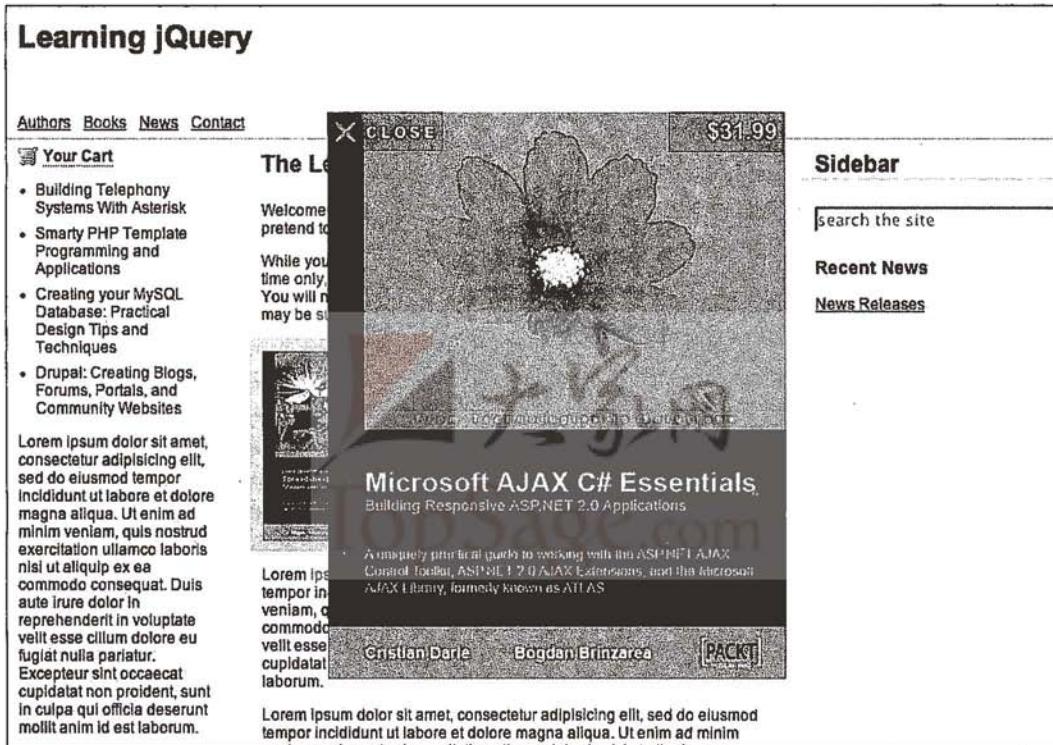


图 9-14

随着我们把清除价格标记的代码`$priceBadge.hide();`添加到大幅封面的单击处理程序中，这一阶段的代码就编写完成了。

9.3.3 为封面放大添加动画效果

此时，当用户单击中间的封面时，放大版的封面会毫无修饰地出现在页面中央。不过，通过使用jQuery内置的动画能力，可以实现从封面的缩略图到放大版之间平滑地变换。

为此，需要知道动画的起始坐标，即中间封面在页面上的位置。通过累加这个图像及它在DOM树中祖先元素的`offsetTop`和`offsetLeft`属性可以得到它的位置：

```

var element = $(this).find('img').get(0);
var coverLeft = 0;
var coverTop = 0;
var coverWidth = element.width;
var coverHeight = element.height;
while (element.offsetParent) {
    coverLeft += element.offsetLeft;
    coverTop += element.offsetTop;
    element = element.offsetParent;
}

```

基于jQuery的Dimensions插件为类似这样的计算提供了便捷的手段。要了解与插件有关的更多信息请参见第10章。

要实现动画效果，首先需要把大幅封面设置为同封面缩略图的大小、位置一样。然后，在调用.animate()方法时以完整的大小作为目标：

```

$enlargedCover.attr('src', $(this).attr('href')).css({
    'left': coverLeft,
    'top' : coverTop,
    'width': coverWidth,
    'height': coverHeight
}).animate({
    'left': ($('body').width() - coverWidth * 3) / 2,
    'top' : 100,
    'width': coverWidth * 3,
    'height': coverHeight * 3
}, 'normal', function() {
    $enlargedCover.one('click', function() {
        $closeButton.unbind('click').hide();
        $priceBadge.hide();
        $enlargedCover.fadeOut();
    });
    $closeButton.css({
        'left': ($('body').width() - coverWidth * 3) / 2,
        'top' : 100
    }).click(function() {
        $enlargedCover.click();
    }).show();
    $priceBadge.css({
        'right': ($('body').width() - coverWidth * 3) / 2,
        'top' : 100
    }).text(price).show();
});

```

既然已经捕获到了缩略图的宽度和高度，那么就可以使用这些值来计算放大后的图像尺寸，而不必硬编码这些数字了。这里，我们假设放大后的图像始终等于缩略图的3倍。另外，对关闭按钮和价格标记的定位操作也需要延迟到动画完成之后，因此，我们把相应的代码放在了.animate()方法的回调函数中。这样，就实现了从小封面到大封面的平滑变换效果（如图9-15至图9-18所示）。

Learning jQuery

[Authors](#) [Books](#) [News](#) [Contact](#)

Your Cart

- Building Telephony Systems With Asterisk
- Smarty PHP Template Programming and Applications
- Creating your MySQL Database: Practical Design Tips and Techniques
- Drupal: Creating Blogs, Forums, Portals, and Community Websites

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

The Learning jQuery Fake Bookstore

Welcome to our fake bookstore. Feel free to browse through the site and pretend to buy books. We'll pretend to sell them to you.

While you're here we'd like to present to you our special offer: For a limited time only, you can click on anything you like in this site for free. That's right! You will not be charged one peso. So stop reading, and start clicking. You may be surprised by what you find.

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sidebar

[Search the site](#)

Recent News

[News Releases](#)

图 9-15

Learning jQuery

[Authors](#) [Books](#) [News](#) [Contact](#)

Your Cart

- Building Telephony Systems With Asterisk
- Smarty PHP Template Programming and Applications
- Creating your MySQL Database: Practical Design Tips and Techniques
- Drupal: Creating Blogs, Forums, Portals, and Community Websites

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

The Learning jQuery Fake Bookstore

Welcome to our fake bookstore. Feel free to browse through the site and pretend to buy books. We'll pretend to sell them to you.

While you're here we'd like to present to you our special offer: For a limited time only, you can click on anything you like in this site for free. That's right! You will not be charged one peso. So stop reading, and start clicking. You may be surprised by what you find.

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lore ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sidebar

[Search the site](#)

Recent News

[News Releases](#)

图 9-16

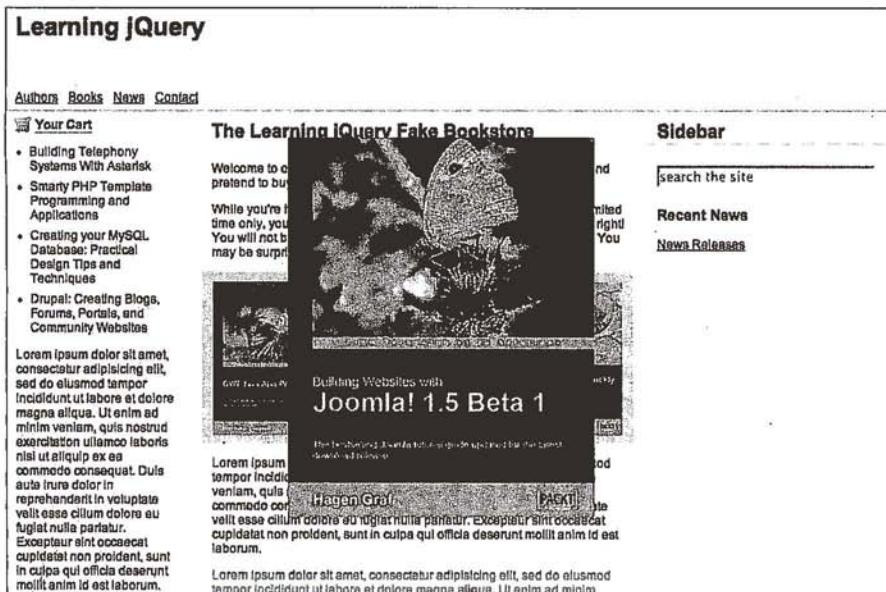


图 9-17

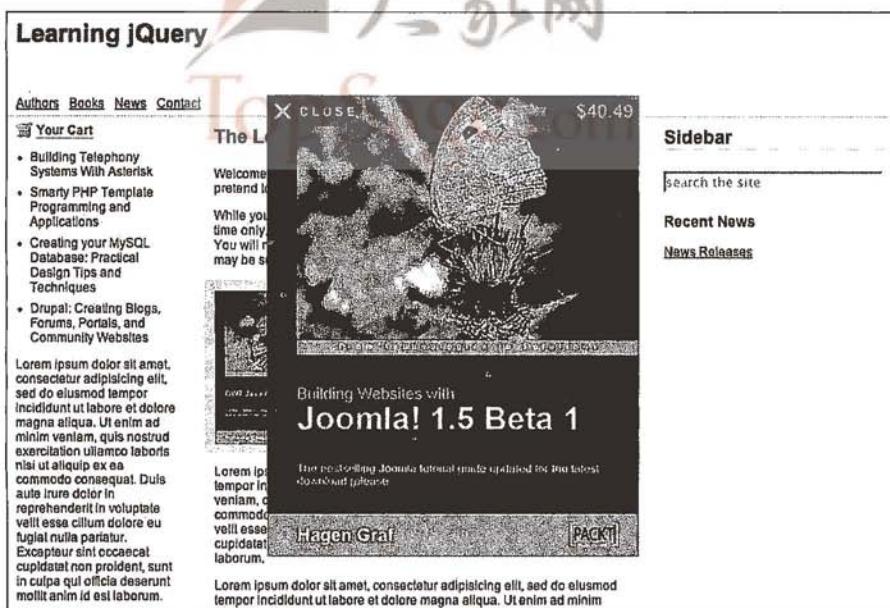


图 9-18

1. 将动画延迟到图像加载之后

事实上，前面动画效果的平滑流畅有赖于到这个网站的快速连接。如果下载大幅封面需要一

定的时间，那么在动画开始的一瞬间可能会显示一个表示图像无效的红叉。而通过等到图像加载完成后再开始动画，则可以使变换效果显得更加优雅：

```
$enlargedCover.attr('src', $(this).attr('href')).css({
  'left': coverLeft,
  'top' : coverTop,
  'width': coverWidth,
  'height': coverHeight
});
var animateEnlarge = function() {
  $enlargedCover.animate({
    'left': ($('body').width() - coverWidth * 3) / 2,
    'top' : 100,
    'width': coverWidth * 3,
    'height': coverHeight * 3
  }, 'normal', function() {
    $enlargedCover.one('click', function() {
      $closeButton.unbind('click').hide();
      $priceBadge.hide();
      $enlargedCover.fadeOut();
    });
    $closeButton.css({
      'left': ($('body').width() - coverWidth * 3) / 2,
      'top' : 100
    }).click(function() {
      $enlargedCover.click();
    }).show();
    $priceBadge.css({
      'right': ($('body').width() - coverWidth * 3) / 2,
      'top' : 100
    }).text(price).show();
  });
};

if ($enlargedCover[0].complete) {
  animateEnlarge();
}
else {
  $enlargedCover.bind('load', animateEnlarge);
}
```

对我们来说，这是load事件比jQuery的自定义ready事件更有用处的一种少见的情况。因为load事件在文档、图像或帧中的内容完全加载后都有可能被触发，所以需要观察这个事件以确保所有图像都已经加载到了内存中——只有此时才会执行事件处理程序，动画才会启动。

说明 由于.load()方法也是一个AJAX方法，为了清晰起见，我们在这里使用了.bind('load')语法，而不是简写的.load()方法；这两种语法是可以互换的。

当图像已经加载到了浏览器的缓存中时，IE和Firefox会对这个事实给出不同的解释。此时，Firefox会立即向JavaScript发送load事件，但IE认为并没有“加载”行为发生，因而不会发送这一事件。为补偿这一差异，我们使用了图像元素的complete属性。这个属性只有在图像加载完成后才会被设置为true。因此，我们首先来测试这个值并在图像就绪后执行动画；如果图像还没有加载完成，那么就需要等到load事件被触发。

2. 添加一个加载指示器

但是，当网速较慢而加载图像需要花较长时间时，就会出现一种尴尬的情况。也就是说，当处于图像下载期间时，页面会对用户的单击毫无反应。事实上，同加载新闻标题时一样，我们也需要在此时为用户提供一个指示器，告诉用户某些处理正在进行中。

所谓的指示器，其实就是另外一个会在适当的时候显示的图像：

```
var $waitThrobber = $('')
    .attr('src', 'images/wait.gif')
    .addClass('control')
    .css('z-index', 4)
    .hide();
```

这里，我们要使用一幅动态的GIF图，以便让用户明显地感觉到活动已经发生了，如图9-19所示。

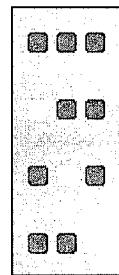


图 9-19

在创建这个图像元素后，把等待指示器放到适当的位置上只需两行代码。在中间图像的单击处理程序中，在开始做其他工作之前，也就是在该处理程序的第一行，就需要显示这个指示器：

```
$waitThrobber.appendTo(this).show();
```

而在animateEnlarge函数的开始处，当知道图像已经加载完成时，则需要从视图中把它移除：

```
$waitThrobber.hide();
```

这些就是通过等待指示器来标记即将放大的封面的全部代码。在等待封面放大期间，这个动态图像会叠放在封面缩略图的左上角（如图9-20至图9-23所示）。

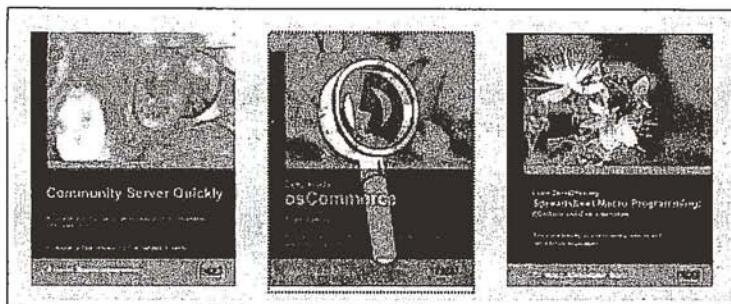


图 9-20

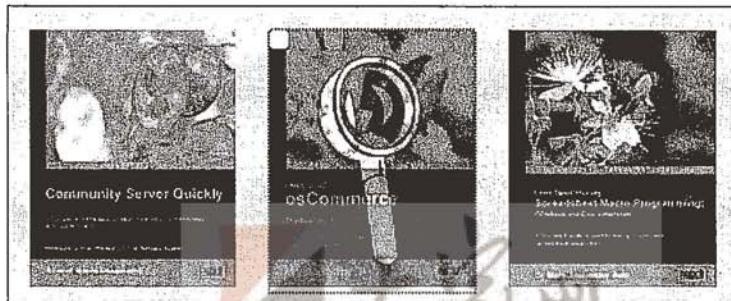


图 9-21

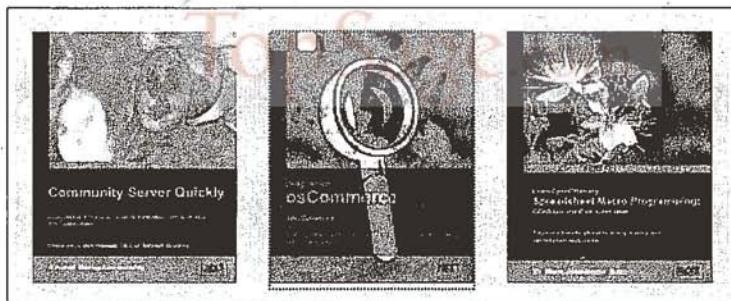


图 9-22

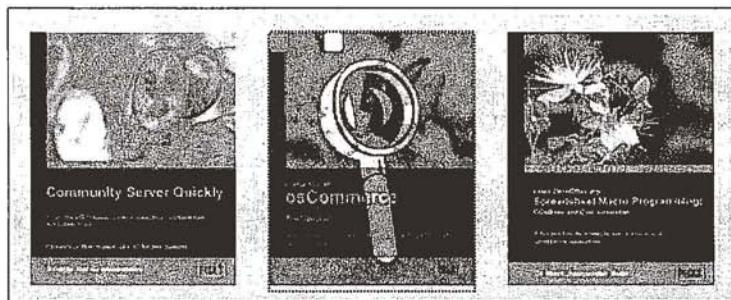


图 9-23

9.4 完成的代码

本章所介绍的为图像和文本添加动画及翻转效果，只是能够在Web上实现的一小部分功能。综合起来，标题翻转效果和图像“传送带”的代码如下所示：

```
$ (document).ready(function() {
    // 把.each()方法作为if语句来使用
    // 将代码包含在私有的命名空间中
    $('#news-feed').each(function() {
        var $this = $(this);
        $this.empty();

        var totalHeight = $this.height();
        var fadeHeight = totalHeight / 4;

        for (var i = 0; i < fadeHeight; i+=2) {
            $('<div></div>').css({
                opacity: i / fadeHeight,
                top: totalHeight - fadeHeight + i
            }).addClass('fade-slice').appendTo(this);
        }

        var $newsLoading = $('<img/>')
            .attr({
                'src': '/cookbook/images/loading.gif',
                'alt': 'loading. please wait'
            })
            .addClass('news-wait');

        $this.ajaxStart(function() {
            $this.append($newsLoading);
        }).ajaxStop(function() {
            $newsLoading.remove();
        });
    });

    // 取得新闻源
    $.get('news/feed.php', function(data) {
        $('</rss//item', data).each(function() {
            var title = $('title', this).text();
            var linkText = $('link', this).text();
            var $link = $('<a></a>')
                .attr('href', linkText)
                .text(title);
            $link = $('<h3></h3>').html($link);

            var pubDate = new Date($('pubDate', this).text());
            var pubMonth = pubDate.getMonth() + 1;
            var pubDay = pubDate.getDate();
            var pubYear = pubDate.getFullYear();
            var $pubDiv = $('<div></div>')
                .addClass('publication-date')
                .text(pubMonth + '/' + pubDay + '/' + pubYear);
        });
    });
});
```

```

var summaryText = $('description', this).text();
var $summary = $('

</div>')
    .addClass('summary')
    .html(summaryText);

$('<div></div>')
    .addClass('headline')
    .append($link)
    .append($pubDiv)
    .append($summary)
    .appendTo('#news-feed');
});

// 设置翻转变量
var currentHeadline = 0, oldHeadline = 0;
var hiddenPosition = totalHeight + 10;
$('div.headline:eq(' + currentHeadline + ')').css('top', '0');
var headlineCount = $('div.headline').length;
var headlineTimeout;
var rotateInProgress = false;

// 翻转函数
var headlineRotate = function() {
    if (!rotateInProgress) {
        rotateInProgress = true;
        headlineTimeout = false;
        currentHeadline = (oldHeadline + 1) % headlineCount;
        $('div.headline:eq(' + oldHeadline + ')')
            .animate({top: -hiddenPosition}, 'slow', function() {
                $(this).css('top', hiddenPosition);
            });
        $('div.headline:eq(' + currentHeadline + ')')
            .animate({top: 0}, 'slow', function() {
                rotateInProgress = false;
                if (!headlineTimeout) {
                    headlineTimeout = setTimeout(headlineRotate, 5000);
                }
            });
        oldHeadline = currentHeadline;
    }
};
headlineTimeout = setTimeout(headlineRotate, 5000);

// 当悬停时，清除计时器并把headlineTimeout重设为false
$('#news-feed').hover(function() {
    clearTimeout(headlineTimeout);
    headlineTimeout = false;
}, function() {
    // 当鼠标离开后，很快开始翻转
    if (!headlineTimeout) {
        headlineTimeout = setTimeout(headlineRotate, 250);
    }
});


```

```

        }
    });
}); // .hover()方法结束
}); // $.get()方法结束
}); // #news-feed的.each()方法结束
});

*****=图像“传送带”*****
$(document).ready(function() {
    var spacing = 140;

    function createControl(src) {
        return $('<img/>')
            .attr('src', src)
            .addClass('control')
            .css('opacity', 0.6)
            .css('display', 'none');
    }

    var $leftRollover = createControl('images/left.gif');
    var $rightRollover = createControl('images/right.gif');
    var $enlargeRollover = createControl('images/enlarge.gif');
    var $enlargedCover = $('<img/>')
        .addClass('enlarged')
        .hide()
        .appendTo('body');
    var $closeButton = createControl('images/close.gif')
        .addClass('enlarged-control')
        .appendTo('body');
    var $priceBadge = $('<div/>')
        .addClass('enlarged-price')
        .css('opacity', 0.6)
        .css('display', 'none')
        .appendTo('body');
    var $waitThrobber = $('<img/>')
        .attr('src', 'images/wait.gif')
        .addClass('control')
        .css('z-index', 4)
        .hide();

    $('#featured-books').css({
        'width': spacing * 3,
        'height': '166px',
        'overflow': 'hidden'
    }).find('.covers a').css({
        'float': 'none',
        'position': 'absolute',
        'left': 1000
    });
});

```

```

var setUpCovers = function() {
    var $covers = $('#featured-books .covers a');
    $covers.unbind('click').unbind('mouseover').unbind('mouseout');

    // 左侧的图像; 当单击时向右滚动 (以便查看左侧的图像)
    $covers.eq(0).css('left', 0).click(function(event) {
        $covers.eq(0).animate({'left': spacing}, 'fast');
        $covers.eq(1).animate({'left': spacing * 2}, 'fast');
        $covers.eq(2).animate({'left': spacing * 3}, 'fast');
        $covers.eq($covers.length - 1).css('left', -spacing)
            .animate({'left': 0}, 'fast', function() {
                $(this).prependTo('#featured-books .covers');
                setUpCovers();
            });
        event.preventDefault();
    }).hover(function() {
        $leftRollover.appendTo(this).show();
    }, function() {
        $leftRollover.hide();
    });
    // 右侧的图像; 当单击时向左滚动 (以便查看右侧的图像)
    $covers.eq(2).css('left', spacing * 2).click(function(event) {
        $covers.eq(0).animate({'left': -spacing}, 'fast', function() {
            $(this).appendTo('#featured-books .covers');
            setUpCovers();
        });
        $covers.eq(1).animate({'left': 0}, 'fast');
        $covers.eq(2).animate({'left': spacing}, 'fast');
        $covers.eq(3).css('left', spacing * 3).animate({
            'left': spacing * 2}, 'fast');

        event.preventDefault();
    }).hover(function() {
        $rightRollover.appendTo(this).show();
    }, function() {
        $rightRollover.hide();
    });
    // 中间的图像; 当单击时放大封面
    $covers.eq(1).css('left', spacing).click(function(event) {
        $waitThrobber.appendTo(this).show();

        var price = $(this).find('.price').text();
        var element = $(this).find('img').get(0);
        var coverLeft = 0;
        var coverTop = 0;
        var coverWidth = element.width;
        var coverHeight = element.height;
        while (element.offsetParent) {
            coverLeft += element.offsetLeft;
        }
    });
}

```

```

        coverTop += element.offsetTop;
        element = element.offsetParent;
    }

$enlargedCover.attr('src', $(this).attr('href')).css({
    'left': coverLeft,
    'top' : coverTop,
    'width': coverWidth,
    'height': coverHeight
});
var animateEnlarge = function() {
    $waitThrobber.hide();
    $enlargedCover.animate({
        'left': $('body').width() - coverWidth * 3) / 2,
        'top' : 100,
        'width': coverWidth * 3,
        'height': coverHeight * 3
    }, 'normal', function() {
        $enlargedCover.one('click', function() {
            $closeButton.unbind('click').hide();
            $priceBadge.hide();
            $enlargedCover.fadeOut();
        });
        $closeButton.css({
            'left': $('body').width() - coverWidth * 3) / 2,
            'top' : 100
        }).click(function() {
            $enlargedCover.click();
        }).show();
        $priceBadge.css({
            'right': $('body').width() - coverWidth * 3) / 2,
            'top' : 100
        }).text(price).show();
    });
};

if ($enlargedCover[0].complete) {
    animateEnlarge();
}
else {
    $enlargedCover.bind('load', animateEnlarge);
}

event.preventDefault();
}).hover(function() {
    $enlargeRollover.appendTo(this).show();
}, function() {
    $enlargeRollover.hide();
});
};

```

```
setUpCovers();  
});
```

9.5 小结

本章中，我们研究了几种能够随着时间变化的页面元素，有的变化是自动地变化，有的则是响应用户操作变化。这些滑移和翻转效果是真正现代的Web设计与传统网站设计的分水岭。概括地讲，我们介绍了如何在页面中展示XML信息源，同时也展示了如何基于时间延迟将新闻条目滚入或滚出视图。针对用“传送带”式的画廊可导航地显示一组图像的情况，我们也讨论了以平滑的动画效果放大图像，从而得到近距离的视图，同时还以不唐突的方式提供了一组用户界面控件。

通过以各种方式综合运用这些技术，能够为平淡的页面注入生机和活力。而且，借助jQuery的强大功能可以使平常实现起来冗长乏味的动画效果，变得轻松而简单。



插件

*Like a plug without a socket
I'm just waitin' 'round for you
—Devo,
“Don't You Know”*

贯穿本书，我们介绍了使用jQuery库完成各种任务的许多方式。但是，唯独还没有深入讨论的一个方面就是同它的核心一样强大的扩展能力。通过使用jQuery简洁的插件架构，开发者们能够把jQuery的功能扩展得更加丰富。

尽管jQuery的发布还不到两年时间^①，但它支持的插件已经超过了100个——小到选择符助手，大到全屏的用户界面部件。在本章中，我们将简单地介绍3个流行的jQuery插件，然后再讨论创建我们自己的插件。

在第7章中，我们曾经提到过插件的强大之处，而且也创建了一个简单的插件。本章中，我们就来体验一下如何在网页中使用已有的插件，并深入地探讨如何构建我们自己的插件。

10.1 使用插件

使用jQuery插件很简单。第1步是把插件包含在文档的<head>标签内，并确保它位于主jQuery源文件之后：

```
<head>
  <meta http-equiv="Content-Type" content="text/html;
                                         charset=utf-8"/>
  <script src="jquery.js" type="text/javascript"></script>
  <script src="jquery.plug-in.js" type="text/javascript"></script>
  <script src="custom.js" type="text/javascript"></script>
  <title>Example</title>
</head>
```

然后，剩下的就是包含一个自定义的JavaScript文件，并在其中使用插件创建或扩展的方法。

① jQuery是2006年1月14日发布的，读者可以参考<http://jquery.com/blog/2007/01/14/jquery-birthday-11-new-site-new-docs/>。——译者注

比如使用Form插件时，在这个自定义文件的`$(document).ready()`方法中添加一行代码就可以通过AJAX来提交表单：

```
$(document).ready(function() {
  $('#myForm').ajaxForm();
});
```

许多插件也具有一些内置的灵活性，比如提供一些供我们设置的可选参数，用来改变插件的行为。这样，我们既可以按照需要自定义插件的操作，也可以简单地保持其默认行为。

10.2 流行的插件

在jQuery的网站上（<http://plugins.jquery.com/>），提供了一个相当长的可用插件的列表，同时也包含让用户能够对插件进行评级和评论的功能。

本章中，我们要探索3个官方插件——选择它们的原因是它们有着成熟的基本代码、很实用，而且也很好地遵循了jQuery项目设置的一系列编程和文档标准。

10.2.1 Dimensions

由Paul Bakaus和Brandon Aaron合作开发的Dimensions插件，用于消除CSS盒模型和开发者对准确测量文档中元素的高度和宽度的需求之间的差距。这个插件还能够提供页面上任何元素的精确到像素级别的上偏移量和左偏移量。

1. height和width

Dimensions提供了3组方法来获得高度和宽度：

- (1) `.height()`和`.width()`。
- (2) `.innerHeight()`和`.innerWidth()`。
- (3) `.outerHeight()`和`.outerWidth()`。

当把`.height`和`.width`方法应用到元素时，这两个方法只是使用同名的jQuery核心方法。但是，Dimensions还扩展了jQuery的这两个方法，可以用它们来取得浏览器窗口和文档的大小。比如，使用`$(window).width()`可以返回浏览器的像素宽度值，而`$(document).width()`则单独返回文档的宽度。如果存在垂直滚动条，`$(window).width()`会包含滚动条，而`$(document).width()`则不会。

inner和outer方法在度量包含内边距（inner）和边框（outer）的元素的宽度和高度时非常有用。下面我们来看一个应用了下列CSS规则的示例元素`<div class="dim-outer">`：

```
.dim-outer {
  height: 200px;
  width: 200px;
  margin: 10px;
  padding: 1em;
  border: 5px solid #e3e3e3;
```

```

    overflow: auto;
    font-size: 12px;
}

```

此时，纯粹的`$('div.dim-outer').width()`方法返回200，因为这确实就是CSS中定义的宽度。然而，这个宽度值并没有精确地反映从左边框内边缘到右边框内边缘之间的宽度。为此，我们可以使用`$('div.dim-outer').innerWidth()`，它的返回值是224。多出的24像素来自于左内边距和右内边距的和。具体来说，因为内边距的值是1em，而em等于font-size，即12px，所以左内边距和右内边距加在一起就得到了额外的24像素。最后，`$('div.dim-outer').outerWidth()`方法会把左边框和右边框(5+5)的宽度加到元素宽度(+200)和内边距宽度(+24)上，得到元素左外边缘到右外边缘的总宽度234，如图10-1所示。

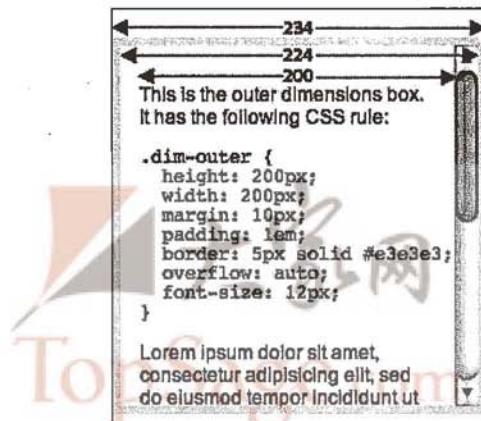


图 10-1

2. scrollTop和scrollLeft

当用户滚动浏览器或文档内部的可滚动元素时，`.scrollTop`和`.scrollLeft`方法分别返回向下和向右滚动的像素值。如果给这两个方法传递一个数字参数，也可以把页面滚动到指定的位置。

3. offset

Dimensions插件中最强大的特性可能就是它的`.offset()`方法了，通过该方法我们可以把任何元素的top和left位置定位到页面上的任何位置，无论这个元素的position是static、relative还是absolute，也不管在将overflow设置为auto时是否包含窗口滚动条或者元素滚动条。通过使用分解为外边距、边框、内边距和滚动参数的选项，`.offset()`方法提供了极大的灵活性和精确度。通过Dimensions的测试页面，可以看出这个方法强大的适应能力，如图10-2所示。

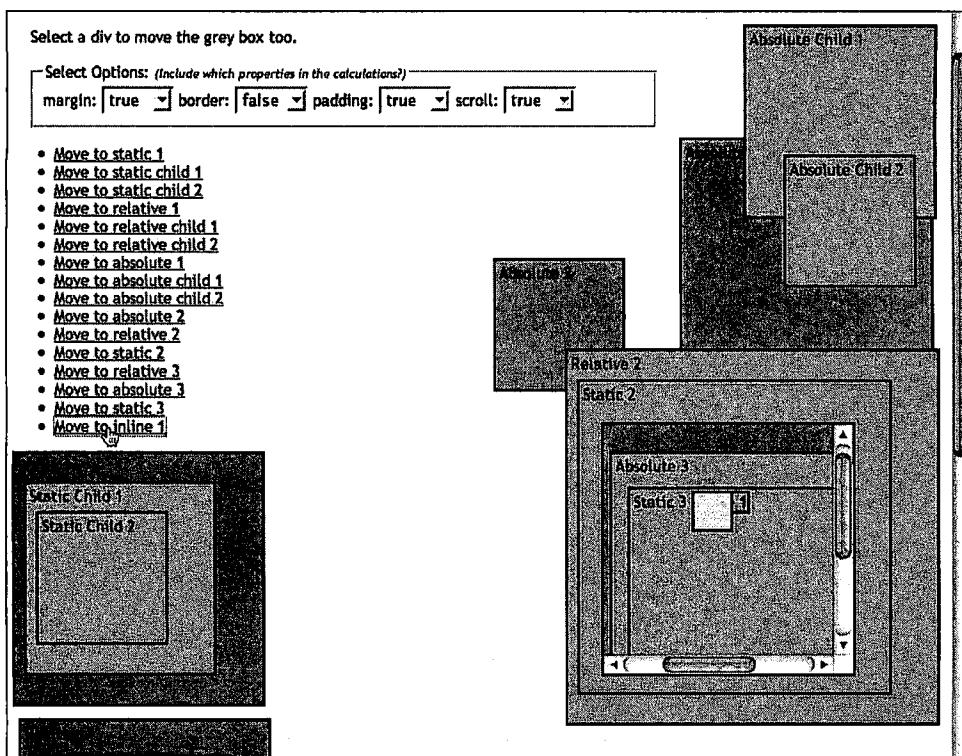


图 10-2

在图10-2中，通过单击Move to inline 1链接把灰盒子移动到了与inline 1元素相同的位置上。而且，由于border选项被设置为false，灰盒子与inline 1元素的上和左边框发生重叠。要亲自体验一下其他的偏移变换，请访问Dimensions的测试页面<http://brandong.jquery.com/plugins/dimensions/test/offset.html>。

10.2.2 Form

Form插件是把困难、复杂的任务变得绝对简单的一个令人叫绝的例子。

这个插件的核心是.ajaxForm方法。正如我们在10.1节中所看到的，基于常规的表单生成一次AJAX表单请求只需一行代码：

```
$(document).ready(function() {
  $('#myForm').ajaxForm();
});
```

以上代码可以用于提交带有id="myForm"属性的表单，并且不会刷新当前页面。虽然这已经很不错了，但.ajaxForm()方法的真正强大之处则体现在我们为该方法传递的选项映射上面。例如，下列代码在调用.ajaxForm()方法时传递了target、beforeSubmit和success选项：

```

$(document).ready(function() {
    function validateForm() {
        // 这里包含验证表单的代码
        // 通过返回false来阻止提交
    };

    $('#test-form').ajaxForm({
        target: '.log',
        beforeSubmit: validateForm,
        success: function() {
            alert('Thanks for your comment!');
        }
    });
});

```

其中，`target`选项表示通过服务器响应的内容来更新的一个或多个目标元素，这里指的是带有`class="log"`的任何元素。

而`beforeSubmit`选项表示在提交表单之前执行的任务。这里，我们调用了`validateForm`函数。如果该函数返回`false`，则不会提交表单。

最后，`success`选项表示在成功提交表单之后执行的任务。在上面的例子中，我们只是简单地为用户提供一个警告信息，以告知用户表单已经成功提交。

可以在`.ajaxForm()`及类似的`.ajaxSubmit()`方法中使用的选项，还有如下几个。

- `url`: 接收表单数据的URL。在不同于表单`action`属性的值时指定。
- `type`: 用于提交表单的方法GET或POST。默认值取自表单的`method`属性，如果该属性为空，则默认使用GET。
- `dataType`: 期望的服务器响应的数据类型。可能的值有`null`、`xml`、`script`或`json`。默认值为`null`。
- `resetForm`: 布尔值，默認為`false`。如果设置为`true`，则会在提交成功后将所有表单字段重置为各自的默认值。
- `clearForm`: 布尔值，默認為`false`。如果设置为`true`，则会在提交成功后清除所有表单字段的值。

此外，Form插件还提供了辅助处理表单及其数据的许多其他方法。要了解这些方法并体验更多示例，请访问<http://www.malsup.com/jquery/form/>。

提示和技巧

在默认情况下，`.ajaxForm()`和`.ajaxSubmit()`方法都使用表单的`action`和`method`属性的值。只要能够适当地标记表单，这个插件就可以如期地完成任务，无需任何调整。

一般来说，在提交表单时，如果用于提交表单的元素具有`name`属性，那么该元素的“名/值”

也会随同其他表单数据一起提交。在这一点上，`.ajaxForm()`方法经过了预先设置，因此，通过为所有提交元素添加单击处理程序，它知道是哪个元素提交的表单。另一方面，`.ajaxSubmit()`方法是反应性的，它无法确定这些信息。换句话说，它不会捕获提交元素。同样的差别也适用于图像输入元素——`.ajaxForm()`会处理它们，而`.ajaxSubmit()`会忽略它们。

无论是`.ajaxForm()`还是`.ajaxSubmit()`，都把它们的`options`参数传递给jQuery核心的`$.ajax()`方法。因此，可以通过Form插件传递任何对`$.ajax()`方法有效的选项。了解了这些特性之后，我们就可以把AJAX表单响应代码构造得更加健壮，比如：

```
$("#myForm").ajaxForm({
  timeout: 2000,
  error: function (xml, status, e) {
    alert(e.message);
  }
});
```

此外，也可以为`.ajaxForm`和`.ajaxSubmit`方法传递一个函数而不是一个`options`作为参数。因为这个函数将被视为成功处理程序，所以，可以直接从中取得服务器返回的响应文本，比如：

```
$("#myForm").ajaxForm(function(responseText) {
  alert(responseText);
});
```

10.2.3 Interface

Dimensions和Form插件只做一件事，并且做得很好；而Interface插件却能够做很多事（同样也做得很好）。事实上，Interface并不只是一个插件，而是一整套插件。

Interface最早由Stefan Petre创建，Paul Bakaus也为该插件的开发做出了主要贡献。通过Interface，开发者们可以轻松地构建拖动、放置、排序、高级动画以及丰富的视觉反馈等部件，能够使用户通过网页获得类似桌面应用程序一般的体验。

下面，我们简单地介绍Animate和Sortables插件。

1. Animate

同Dimensions插件的`.height`和`.width`方法一样，Interface中的`.animate`方法也扩展自jQuery的核心方法。虽然jQuery核心的`.animate()`方法对于可以作为参数的选项给出了一定的限制，但这个方法的Interface版则开放了这些选项，从而囊括了几乎任何CSS属性，甚至可以接受类名。例如，Interface的`.animate()`方法可以从一个类定义的一组属性，通过动画方式过渡到另一个类定义的另一组属性。假设有一个元素`<div class="boxbefore">`，它应用了下列CSS规则：

```
.boxbefore {
  width: 300px;
```

```

margin: 1em 0;
padding: 5px;
overflow: auto;
background-color: #fff;
color: #000;
border: 10px solid #333;
}

```

这些样式定义了300像素宽的盒子，5像素的四边内边距，10像素的深灰色边框，以及常见的白色背景上的黑色文本。而且，由于将overflow属性设置为auto，那么当这个元素盒子不足以显示所有内容时就会出现滚动条。但是，由于没有预先指定高度，这个盒子会为了适应其中的内容而增大到必需的尺寸。在应用了以上属性后，这个盒子的外观如图10-3所示。

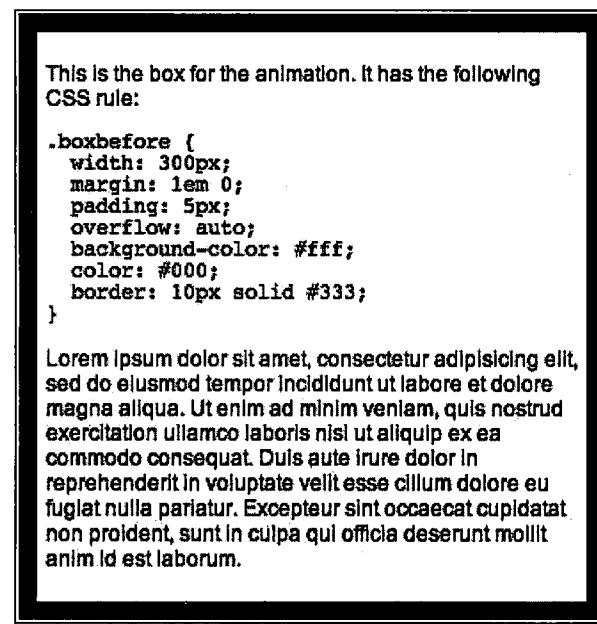


图 10-3

现在，我们就以动画方式来实现从boxbefore类到定义了下列属性的新的boxafter类的转换：

```

.boxafter {
  height: 180px;
  width: 500px;
  padding: 15px;
  background-color: #000;
  color: #fff;
  border: 5px solid #ccc;
}

```

在这条CSS规则中，我们将盒子的高度设置为180像素，将它的宽度增大为500像素，减少了边框的宽度但加亮了边框的颜色，另外，还增加了内边距并反转了文本和背景的颜色。因为没有定义新的overflow和margin属性，所以它们保持不变。

要通过动画呈现这个显著的变化，可以简单地编写下面几行代码：

```
$(document).ready(function() {
  $('div.boxbefore').animate({className:'boxafter'}, 1000);
});
```

在动画进行到一半多一点的时候，这个盒子的外观如图10-4所示。

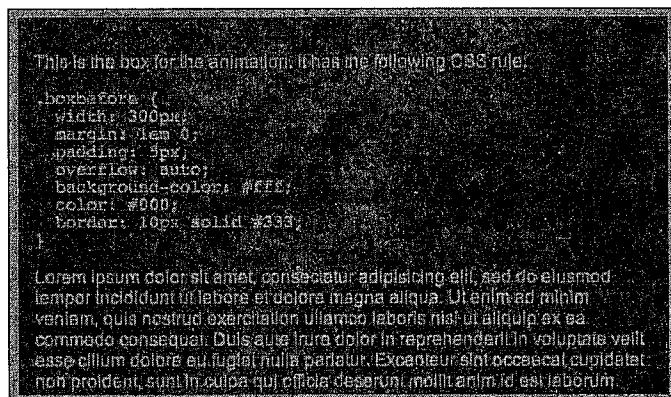


图 10-4

随着动画效果的结束，盒子将会完全应用boxafter类中定义的样式，而且overflow: auto; 和减少的高度会导致元素盒子中出现垂直滚动条，如图10-5所示。

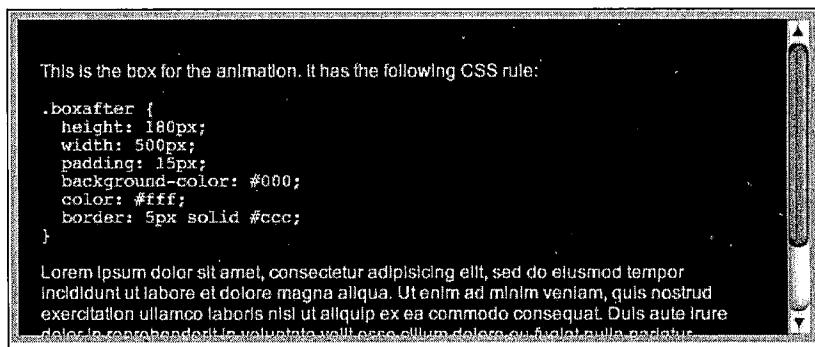


图 10-5

2. Sortables

Interface中的Sortables插件模块几乎能够把任何元素组合转换为可拖放式的列表。这里，我

们以每个项都应用了一些CSS样式的无序列表为例，如图10-6所示。

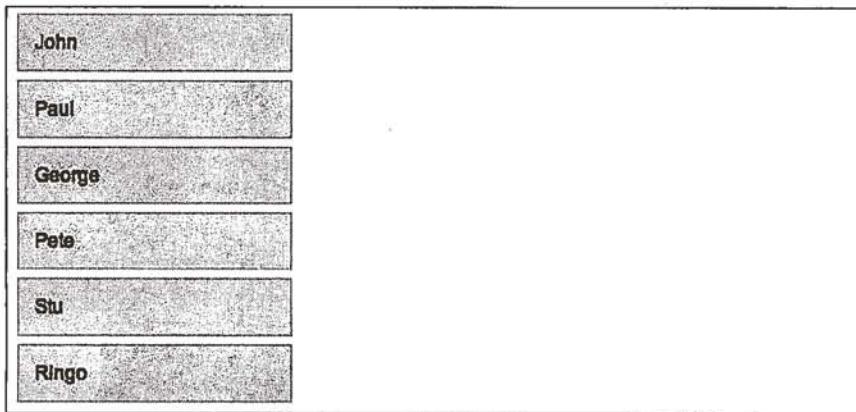


图 10-6

这个无序列表的HTML非常简单：

```
<ul id="sort-container" class="content">
    <li id="item1" class="sort-item">John</li>
    <li id="item2" class="sort-item">Paul</li>
    <li id="item3" class="sort-item">George</li>
    <li id="item4" class="sort-item">Pete</li>
    <li id="item5" class="sort-item">Stu</li>
    <li id="item6" class="sort-item">Ringo</li>
</ul>
```

其中，每个列表项都有唯一的id和一个公共的class。现在，为了能对这个列表进行排序，我们只需编写下列简单的代码：

```
$(document).ready(function() {
    $('#sort-container').Sortable({
        accept : 'sort-item',
        hoverclass : 'hover',
        helperclass : 'helper',
        opacity: 0.5
    });
});
```

以上代码中包含一个Sortable方法和一个参数映射。其中，第1个参数accept是强制性的，其他参数都是可选的。事实上，我们在这个脚本中还省略了很多选项。

结果，这个方法把带有class="sort-item"的所有列表项都变得可以排序了。而且，该方法还在鼠标指针悬停时为下方的项添加了一个类(hoverclass: 'hover')并标记了用作helper项的类(helperclass: 'helper')。在这个例子中，helper类只是一个红色的点划线边框，如图10-7所示。

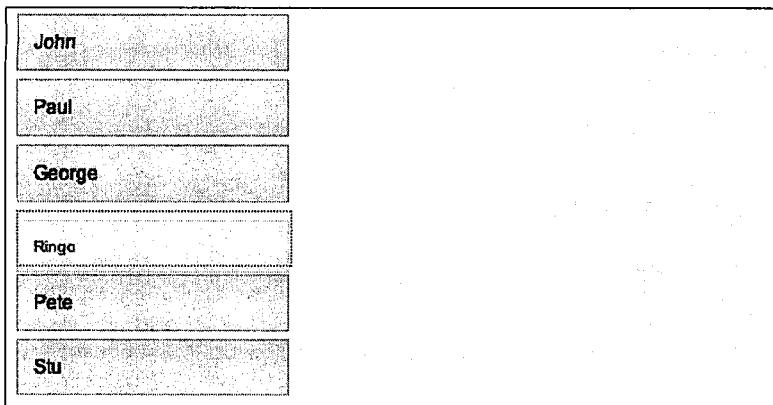


图 10-7

类似Sortables这样的Interface插件有助于在Web应用程序中添加类似桌面应用程序的功能。要了解与全部Interface插件有关的更多内容，请访问<http://interface.eyecon.ro/>。

10.3 查找插件文档

在查找插件的文档时，“jquery.com的插件仓库”(<http://plugins.jquery.com/>)是一个不错的起点。仓库中列出的每个插件都有相应的链接，指向可以下载该插件的页面。此外，许多链接的页面中还包含演示、例子代码和教程。

官方的jQuery插件在自身的源代码中还提供了丰富的注释。其他很多插件中的注释语法，都与jquery.js文件中的注释语法保持一致，这些注释为每个方法提供了说明和至少一个例子。一致的注释语法意味着能够查看jQuery文档的工具，同样可以用来查看兼容的插件。

例如，在Dimensions插件的.offset方法中，包含如下注释^①：

```
/**
 * Returns the location of the element in pixels from the top left
 * corner of the viewport.
 *
 * For accurate readings make sure to use pixel values for margins,
 * borders and padding.
 *
 * @example $("#testdiv").offset()
 * @result { top: 100, left: 100, scrollTop: 10, scrollLeft: 10 }
 *
 * @example $("#testdiv").offset({ scroll: false })
 * @result { top: 90, left: 90 }
```

^① 此段代码中的注释不是原书作者编写的示例，而是引用其他库中方法的注释，只是为了说明注释的格式。如果要翻译成中文，读者在这个库中将找不真正的“原始信息”。——译者注

```

*
* @example var offset = {}
* $("#testdiv").offset({ scroll: false }, offset)
* @result offset = { top: 90, left: 90 }
*
* @name offset
* @param Object options A hash [map] of options describing what
* should be included in the final calculations of the offset.
* The options include:
*   margin: Should the margin of the element be included in the
*           calculations? True by default.
*           If set to false the margin of the element is subtracted
*           from the total offset.
*   border: Should the border of the element be included in the
*           calculations? True by default.
*           If set to false the border of the element is subtracted
*           from the total offset.
*   padding: Should the padding of the element be included in the
*            calculations? False by default.
*            If set to true the padding of the element is added to the
*            total offset.
*   scroll: Should the scroll offsets of the parent elements be
*           included in the calculations? True by default. When true,
*           it adds the total scroll offsets of all parents to the
*           total offset and also adds two properties to the returned
*           object, scrollTop and scrollLeft. If set to false the
*           scroll offsets of parent elements are ignored.
*           If scroll offsets are not needed, set to false to get a
*           performance boost.
* @param Object returnObject An object to store the return value in,
* so as not to break the chain. If passed in, the chain will not be
* broken and the result will be assigned to this object.
*
* @type Object
* @cat Plugins/Dimensions
* @author Brandon Aaron (brandon.aaron@gmail.com ||
*                      http://brandonaaron.net)
*/

```

这里，注释的开头先对方法进行了一般性的介绍，然后提供了一些有关使用像素值的简单建议。在这些介绍性的文字后面，是一组更详细的信息列表，每个列表项都以一个@符号开始。而且，方法名（`@name offset`）放在了所有例子的后面。这里，按照复杂程度先后提供了3个例子。

方法名的后面是方法可以接受的参数。注释中对这些参数（特别是对`object`选项）给出了非常详尽的介绍。我们应该重点关注这些参数的默认值和应用这些参数之后的结果。

最后3个项提供了与这个方法有关的更多信息，包括方法返回的数据类型、所属的类别以及作者。

如果在插件仓库、作者的网站和插件的注释中仍然找不到问题的答案，还可以求助于jQuery的讨论列表。很多插件的作者都会频繁地在这个列表中发表自己的文章，而且总是乐于帮助解决新用户碰到的任何问题。可以在<http://docs.jquery.com/Discussion>中找到如何订阅这个讨论列表的说明。

10.4 开发新插件

众多的第三方插件虽然能够增强我们的编程体验，但有时候我们还需要走得更远一些。当我们编写的代码可以供其他人甚至我们自己重用的时候，我们会希望把这些代码打包成一个新插件。好在，这个过程并不比编写代码更麻烦。

10.4.1 添加新的全局函数

jQuery内置的某些功能是通过全局函数提供的。所谓全局函数，实际上就是jQuery对象的方法，但从实践的角度上看，它们是位于jQuery命名空间内部的函数。

使用这种技术的一个典型的例子就是`$.ajax`函数。`$.ajax()`所做的一切都可以通过简单地调用一个名为`ajax()`的常规全局函数来实现，但是，这种方式会给我们带来函数名冲突的问题。通过把这个函数放在jQuery的命名空间内，我们只需避免它与其他的jQuery方法冲突即可。

要向jQuery的命名空间中添加一个函数，只需将这个新函数指定为jQuery对象的一个属性：

```
jQuery.foo = function() {
    alert('This is a test. This is only a test.');
};
```

于是，我们就可以在使用这个插件的任何代码中，编写如下代码：

```
jQuery.foo();
```

而且，也可以使用\$别名并写成：

```
$.foo();
```

这样，就如同调用jQuery中的其他函数一样，会显示一个警告信息。

1. 添加多个函数

如果我们想在插件中提供多个全局函数，可以独立地声明这些函数：

```
jQuery.foo = function() {
    alert('This is a test. This is only a test.');
};

jQuery.bar = function(param) {
    alert('This function takes a parameter, which is "' + param + '".');
};
```

这样，就定义了两个方法；而且，也可以通过正常的方式来调用它们：

```
$ .foo();
$.bar('baz');
```

还可以使用`$.extend()`函数来使函数的定义更加清晰：

```
jQuery.extend({
  foo: function() {
    alert('This is a test. This is only a test.');
  },
  bar: function(param) {
    alert('This function takes a parameter, which is "' + param +
'."');
  }
});
```

以上代码也会得到同样的结果。不过，这样也会使我们面临不同命名空间冲突的风险。即使我们通过使用jQuery命名空间屏蔽了多数JavaScript函数和变量名，但仍然有可能同其他jQuery插件定义的函数名冲突。为避免这个问题，最好是把属于一个插件的所有全局函数都封装到一个对象中：

```
jQuery.myPlugin = {
  foo: function() {
    alert('This is a test. This is only a test.');
  },
  bar: function(param) {
    alert('This function takes a parameter, which is "' + param +
'."');
  }
};
```

尽管我们仍然可以把这些函数当成全局函数来看待，但从技术上说，它们现在都是全局jQuery函数的方法，因此调用这些函数的方式也会稍微有些变化：

```
$.myPlugin.foo();
$.myPlugin.bar('baz');
```

通过使用这一技术（及一个足够特别的插件名称），完全可以避免在全局函数中发生命名空间的冲突。

2. 关键所在

现在，我们已经掌握了开发插件的基础知识。在把定义的函数保存到一个名为`jquery.mypluginname.js`的文件中之后，就可以在页面中包含这个脚本并在其他脚本中调用这些函数。那么，这个文件同我们创建并包含的其他JavaScript文件有什么区别呢？

前面我们已经讨论过把代码都封装在jQuery对象中能够获得的命名空间保护的好处了。此

外，把我们的函数库编写为jQuery扩展的另一个关键的好处在于：这些函数可以使用jQuery对象本身。通过为代码打上插件的标签，就等于明确地要求页面中始终要包含jQuery文件。

提示 即使页面中包含了jQuery文件，也不应该假设简写方式\$始终是有效的。我们编写的插件应该始终使用jQuery来调用jQuery方法，或者像后面介绍的那样，在内部定义自己的\$。

但是，这些都只是组织结构上面的好处。要真正领略到jQuery插件的强大之外，还需要学习如何在个别jQuery对象的实例上创建新方法。

10.4.2 添加jQuery对象方法

jQuery中大多数内置的功能都是通过它的方法提供的，而且这些方法也是插件之所以诱人的关键。当函数需要操作DOM元素时，就是将函数创建为新方法的好机会。

前面我们已经看到，添加全局函数需要以新方法来扩展jQuery对象。添加实例方法也与此类似，但扩展的却是jQuery.fn对象：

```
jQuery.fn.xyzzy = function() {
    alert('Nothing happens.');
}
```

提示 jQuery.fn对象是jQuery.prototype的别名，使用别名是出于简洁的考虑。

然后，就可以在使用任何选择符表达式之后调用这个新方法了：

```
$('.div').xyzzy();
```

当调用这个方法时会弹出一个警告框。由于这里并没有在任何地方用到匹配的DOM节点，所以为此编写一个全局函数也是一样的。由此可见，一个合理的实例方法应该包含对它的环境的操作。

1. 对象方法的环境

在任何插件方法内部，关键字this引用的都是当前的jQuery对象。因而，可以在this上面调用任何内置的jQuery方法，或者提取它包含的DOM节点并操作该节点：

```
jQuery.fn.showAlert = function() {
    alert('You called this method on "' + this[0] + '".');
}
```

但是，不要忘记jQuery的选择符可能会匹配零、一或多个元素。因此，在设计插件时必须考虑到所有这些可能的情况。要做到这一点，最简单的方式就是始终在方法的环境上调用.each()方法；这样就会执行隐式迭代，而执行隐式迭代对于维护插件与内置方法的一致性是至关重要的。在调用的.each()方法内部，this依次引用每个DOM元素：

```
jQuery.fn.showAlert = function() {
    this.each(function() {
        alert('You called this method on "' + this + '".');
    });
}
```

这样，根据前面选择符表达式匹配的元素不同，我们定义的方法也会给出不同的警告信息^①。

2. 方法连缀

除了隐式迭代之外，jQuery用户也应该能够正常使用连缀行为。因而，我们必须在所有插件方法中返回一个jQuery对象，除非相应的方法明显用于取得不同的信息。返回的jQuery对象通常就是this所引用的对象。如果我们使用.each()迭代遍历this，那么可以只返回迭代的结果^②：

```
jQuery.fn.showAlert = function() {
    return this.each(function() {
        alert('You called this method on "' + this + '".');
    });
}
```

在添加return之后，就可以在我们的插件方法上面连缀内置的方法了：

```
$('div').showAlert().hide('slow');
```

10.4.3 DOM 遍历方法

在某些情况下，我们的方法可能会改变jQuery对象引用的DOM元素。比如，假设我们想要添加一个查找匹配元素的祖父级元素的DOM遍历方法：

```
jQuery.fn.grandparent = function() {
    var grandparents = [];
    jQuery.each(this, function(index, value) {
        grandparents.push(value.parentNode.parentNode);
    });
    grandparents = $.unique(grandparents);
    return this.setArray(grandparents);
};
```

这个方法创建了一个新的grandparents数组，并通过迭代由当前jQuery对象引用的全部元素来填充这个数组。具体来说，就是使用内置的.parentNode属性查找祖父级元素，并把找到的结果推到数组中。然后，调用\$.unique()方法去掉数组中重复的元素。最后，jQuery的.setArray

^① 事实上，这里仅从弹出的警告框中还是看不出来什么不同。因为每个警告框中都会显示“You called this method on “[object]””，但如果读者把this修改为this.tagName或this.className，那么根据情况就能够看到针对不同元素的信息了。——译者注

^② 这里说返回结果也不确切，实际上仍然是返回引用jQuery对象的this——只不过，此时的this中包含了扩展的.showAlert()方法。读者可以在试试\$('div').showAlert().showAlert()；，这条语句会重复显示两遍警告框。这样就能理解.showAlert()方法返回了带.showAlert()方法的jQuery对象了。——译者注

方法把这组匹配的元素转换成新数组。现在，就可以使用这个方法来查找并操作一个元素的祖父级元素了：

```
$('.foo').grandparent().addClass('bar');
```

然而，这个方法具有破坏作用，因为它修改了实际的jQuery对象。如果我们把jQuery对象保存到一个变量中，就能明显地看到这一改变：

```
var $frood = $('.hoopy');
$frood.grandparent().hide();
$frood.show();
```

以上代码先隐藏祖父级元素，然后再显示它们。最终，保存在\$frood变量中的jQuery对象变成了引用祖父级元素。如果我们以非破坏性的方式来编写这个方法，就不会发生这种混乱的情况：

```
jQuery.fn.grandparent = function() {
  var grandparents = [];
  jQuery.each(this, function(index, value) {
    grandparents.push(value.parentNode.parentNode);
  });
  grandparents = $.unique(grandparents);
  return this.pushStack(grandparents);
};
```

.pushStack方法会创建一个新的jQuery对象，而不是修改旧对象。这样我们就修复了刚才遇到的问题。现在，\$frood.show()中变量\$frood仍然会引用原始的\$('.hoopy')对象。使用.pushStack()的另一个好处是，它会允许.end方法操作我们的新方法。因此，新方法就可以适当地与其他方法连缀使用了：

```
$('.fred').grandparent().addClass('grandma').end()
           .addClass('grandson');
```

说明 在jQuery 1.0中，像.children()之类的DOM遍历方法都是破坏性的。但在jQuery 1.1之后，都变成了非破坏性的。

方法的参数

传递给任何方法的最重要的参数就是关键字this。不过，我们当然可以随便定义更多的参数。要想让我们插件的API尽可能地好用，就需要在参数列表的开始处列出必要的参数。虽然也可以在参数列表中提供可选的参数，但通常用映射来表示可选参数会更简单也更方便。

举例来说，假设我们的方法可以接受一个字符串和一个数字。那么，可以把方法定义为接受两个参数：

```
jQuery.fn.myMethod = function(aString, aNumber) {
  alert('The string is "' + aString + '"');
  alert('The number is ' + aNumber + '.');
}
```

但是，如果这些参数是可选的，那么就必须考虑到4种可能性：

```
$('div').myMethod('hello', 52);
$('div').myMethod('hello');
$('div').myMethod(52);
$('div').myMethod();
```

然后，需要检查这些参数是否有定义，并在没有定义时提供默认值：

```
jQuery.fn.myMethod= function(aString, aNumber) {
    if (aString == undefined) {
        aString = 'goodbye';
    }
    if (aNumber == undefined) {
        aNumber = 97;
    }
    alert('The string is "' + aString + '" .');
    alert('The number is ' + aNumber + ' .');
}
```

这样，无论是在两个参数都存在，还是只提供了字符串，或者两个参数都没有提供的情况下，我们的方法都能正常使用。但是，当提供了数字而没有提供字符串时，数字就会作为aString参数传入。为此，还需要检测参数的数据类型：

```
jQuery.fn.myMethod= function(aString, aNumber) {
    if (aString == undefined) {
        aString = 'goodbye';
    }
    if (aNumber == undefined) {
        if (aString.constructor == Number) {
            aNumber = aString;
            aString = 'goodbye';
        }
        else {
            aNumber = 97;
        }
    }
    alert('The string is "' + aString + '" .');
    alert('The number is ' + aNumber + ' .');
}
```

在只有两个参数的情况下，进行这样的检测还比较容易。但是，随着参数的增加，很快就会导致令人头痛的问题。为了避免这些麻烦，可以使用映射作为参数：

```
jQuery.fn.myMethod= function(parameters) {
    defaults = {
        aString: 'goodbye',
        aNumber: 97
    };
    jQuery.extend(defaults, parameters);
```

```

    alert('The string is "' + defaults.aString + '"');
    alert('The number is ' + defaults.aNumber + '.');
}

```

通过使用jQuery.extend()方法，可以方便提供随时可能被传入的参数覆盖的默认值。此时，对方法的调用会大致保持相同，只不过要传入一个映射而不是一个纯参数列表：

```

$('div').myMethod({aString: 'hello', aNumber: 52});
$('div').myMethod({aString: 'hello'});
$('div').myMethod({aNumber: 52});
$('div').myMethod();

```

这种策略与检测数据类型相比不仅灵活而且更简洁。此外，使用命名参数也意味着再添加新选项也不会影响过去编写的代码，并且使用这个插件的脚本也会更加直观明了。

10.4.4 添加新的简写方法

jQuery库必须在方便和复杂之间维持一个微妙的平衡。添加到这个库中的每个方法都有助于开发者简化某些代码的编写，但也会增加基础代码的整体大小并可能影响性能。考虑到这个原因，内置功能的许多简写方法都移交到了插件中实现，以便开发者可以挑选出对某个开发项目有用的方法，并省略那些无关的方法。

当我们发现自己在代码中需要多次重复使用某个方法时，可能会想到为该方法创建一种简写的形式。核心jQuery库中就包含一些这种简写方法，例如.click()就是bind('click')的简写方式。这些插件创建起来很简单，只需为核心函数传递相应的参数并加上我们自己的代码即可。

假设我们要使用内置的“滑动”和“淡化”技术频繁地为元素添加动画效果。把这两个效果放到一起意味着要同时变换元素的高度和不透明度。而使用animate()方法可以简化这个操作：

```
.animate({height: 'hide', opacity: 'hide'});
```

为此，我们可以创建两个简写的方法，以便在需要显示和隐藏元素时执行相应的动画效果：

```

jQuery.fn.slideFadeOut = function() {
    return this.animate({height: 'hide', opacity: 'hide'});
}

jQuery.fn.slideFadeIn = function() {
    return this.animate({height: 'show', opacity: 'show'});
}

```

现在，我们就可以在需要时调用\$('.myClass').slideFadeOut()并触发相应的动画效果。因为在插件方法的定义中，this引用当前的jQuery对象，所以这个动画会立即在所有匹配的元素上面执行。

出于完整性的考虑，我们的新方法也应该支持与内置的简写方法相同的参数。具体来说，应该像.fadeIn()方法一样能够自定义速度和回调函数。由于.animate()方法也接受这些参数，

所以这个过程就很简单了——只需接受这些参数并转交给`.animate()`即可：

```
jQuery.fn.slideFadeOut = function(speed, callback) {
    return this.animate({height: 'hide', opacity: 'hide'},
                        speed, callback);
}

jQuery.fn.slideFadeIn = function(speed, callback) {
    return this.animate({height: 'show', opacity: 'show'},
                        speed, callback);
}
```

这样，我们就有了与内置的简写方法具有类似功能的自定义的简写方法。

10.4.5 维护多事件日志

作为一名JavaScript开发者，我们有时候也需要在各种事件发生时显示日志事件。JavaScript的`alert()`函数通常用于示范，并不适合显示频繁而且实时的信息。对此，一个比较好的替代方案是Firefox和Safari中的`console.log()`函数，这个函数可以将信息输出到一个独立的日志中，因而不会中断页面的交互过程。但是，由于IE没有提供这个函数，我们需要使用一个自定义的函数来实现这种风格的消息日志。

说明 **Firebug Lite脚本**（将在附录B中介绍）提供了一种非常稳定的跨平台的日志工具。我们这里开发的方法只适合于一般性的用途，Firebug Lite通常是更好的方案。

记录日志信息的一种简单的方式，就是创建一个把信息添加到页面中某个特殊元素中的全局函数：

```
jQuery.log = function(message) {
    $('

甚至还可以搞点花样，让新日志信息以动画形式出现：



```
jQuery.log = function(message) {
 $('

这样，就可以调用$.log('foo')在页面上的日志元素中显示“foo”了。

但是，有的时候一个页面中可能会包含多个例子，如果能够分别记录每个例子的信息会非常方便。为此，我们可以定义一个对象方法，而不是使用全局函数：

www.TopSage.com


```


```

```
jQuery.fn.log = function(message) {
    return this.each(function() {
        $('<div class="log-message" />')
            .text(message)
            .hide()
            .appendTo(this)
            .fadeIn();
    });
};
```

现在，调用`$('.log').log('foo')`会得到同前面调用全局函数一样的效果。但是，通过修改选择符表达式则可以把信息添加到不同的日志元素中。

然而，理想的情况下`.log`方法应该足够聪明，即不必使用明确的选择符就能找到与显示日志信息最相关的元素。通过利用传递到这个方法中的环境，我们可以遍历DOM并找到距离选择的元素最近的日志元素：

```
jQuery.fn.log = function(message) {
    return this.each(function() {
        $context = $(this);
        while ($context.length) {
            $log = $context.find('.log');
            if ($log.length) {
                $('<div class="log-message" />').text(message).hide()
                    .appendTo($log).fadeIn();
                break;
            }
            $context = $context.parent();
        }
    });
};
```

以上代码首先会在匹配的元素中查找用于显示日志信息的元素，如果没有找到，则会通过遍历DOM继续寻找。

最后，我们还有可能需要显示一个对象的内容。由于打印输出对象只会得到类似`[object Object]`这样的信息，因此可以通过检测参数的类型并在参数为对象时执行我们自己的“优质打印”操作：

```
jQuery.fn.log = function(message) {
    if (typeof(message) == 'object') {
        string = '{';
        $.each(message, function(key, value) {
            string += key + ': ' + value + ', ';
        });
        string += '}';
        message = string;
    }
};
```

```

return this.each(function() {
  $context = $(this);
  while ($context.length) {
    $log = $context.find('.log');
    if ($log.length) {
      $('<div class="log-message" />').text(message).hide()
        .appendTo($log).fadeIn();
      break;
    }
    $context = $context.parent();
  }
});
};

```

这样，我们定义的方法就能同时处理对象和字符串，并把它们的信息写入到页面上相关的元素中。

10.4.6 添加选择符表达式

jQuery内置的组件也可以扩展。与添加新方法不同，我们可以自定义现有的组件。比如，一个常见的需求就是扩展jQuery提供的选择符表达式，以便得到更高级的选择符。

通过jQuery实现的伪类选择符: nth-child()^①，可以在父元素中查找位于特定位置上的元素。假设我们构造了下面这个包含10个项的有序列表：

```

<ol class="nthchild">
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
</ol>

```

选择符表达式\$('li:nth-child(4)')^②会查找这个列表中的第4个项。前面我们也看到过这个选择符的能力。但是，这个选择符在CSS规范中更加强大。在CSS 3中，:nth-child()伪类选择符不仅能够接受整数参数，而且能够接受an+b形式的任何表达式。如果某个项的位置等于这个表达式（或者等于n在任何整数值下算出的值），这个项就会匹配。例如，:nth-child(4n+1)会匹配第1、5、9……个项。我们可以通过一个插件为jQuery的选择符引擎添加这种能力。

jQuery的选择符解析器首先会使用一组正则表达式来分解选择符表达式。然后，针对分解出的每个选择符执行一个函数，以找到可能匹配的节点。这个函数可以在jQuery.expr^③映射中找到。

^① 读者可以在jquery.js源文件中搜索expr来找到这个映射。——译者注

下面，我们使用`$.extend()`来覆盖`:nth-child()`伪类选择符的内置行为：

```
jQuery.extend(jQuery.expr[':'], {
  'nth-child': 'jQuery.nthchild(a, m)',
});
```

这个映射的值是一个字符串，其中包含用来筛选元素的JavaScript表达式。在这些表达式中，`a`引用要测试的DOM元素，而`m`则是保存选择符组件的一个数组。

`m`的实际内容会根据我们实现的选择符的格式而变化，因此第1步是检查位于`jquery.js`内部的`jQuery.parse`中的正则表达式。通过观察完成的匹配，我们看到对于`:x(y(z))`格式的伪类选择符，`m`中的组件分别是：

```
m[0] == ':x(y(z))'
m[1] == ':'
m[2] == 'x'
m[3] == 'y(z)'
m[4] == '(z)'
```

我们代码中针对`:nth-child()`伪类选择符的代码会调用jQuery命名空间中名为`nthchild()`的函数，该函数就是我们工作的重点（通过这个机会把`a`和`m`分别重命名为更容易理解的`element`和`components`）：

```
jQuery.nthchild = function(element, components) {
  var index = $(element).parent().children().index(element) + 1;

  var numbers = components[3].match(/((\d+)n)?\+?(\d+)?/);
  if (numbers[2] == undefined) {
    return index == numbers[3];
  }
  if (numbers[3] == undefined) {
    numbers[3] = 0;
  }

  return (index - numbers[3]) % numbers[2] == 0;
}
```

这个函数首先从同辈元素中找到当前节点的索引。虽然使用纯粹的DOM遍历函数会使这个操作的速度更快一些，但使用jQuery方法则会使代码更容易理解。由于CSS规定`:nth-child()`伪类选择符是从1开始的而不是从0开始的，所以还要在结果上加1。

在找到索引之后，我们把数字表达式分解成相应的组件。以表达式`4n+1`为例，这个表达式在被分割后，`numbers[2]`中是4，而`numbers[3]`中是1。然后，我们针对像`4n`和`1`这样的特殊情况进行了相应的处理。

最后，通过一点数学运算我们发现，如果 $an+b = i$ ，那么 $(i-b)/a = n$ 。这样，就得到了一个算式，而执行该算式能够确定当前的索引是否能够通过测试。如果这个元素应该是结果集中的一个，

那么返回true；否则返回false。

在安装了这个新插件后，我们就可以使用像`$('li:nth-child(3n+2)')`这样的jQuery选择符，在列表中轻松地找到从第2个项开始算的所有第3个项了。

10.4.7 创建缓动样式

在调用动画方法时，我们可以指定要变换的每个属性的起点和终点。并且，还能告诉该方法从A点到B点以多快的速度移动。然而，我们却不能指定从A到B之间以什么方式移动。动画效果不一定总是匀速运动，而且事实上在默认情况下的确不是。

假设在一个动画效果中，元素从左向右移动，同时减少其不透明度：

```
$('.sprite').animate({'left': 791, 'opacity': 0.1}, 5000);
```

如果我们在观看动画的过程中，以均等的时间间隔捕获这个元素的位置，那么就会对它在变换过程中的速度变化有一个直观的印象，如图10-8所示。

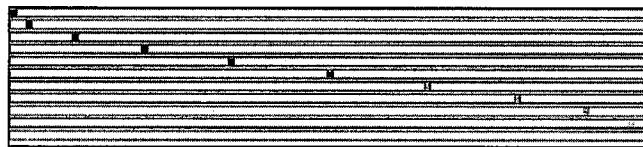


图 10-8

通过这幅插图可以看到，动画元素开始速度较慢，而在大部分动画期间是加速运动，快到终点时速度又慢了下来。在实践中，这种非匀速地执行动画效果的样式叫做缓动（easing）。而默认的缓动样式叫做悬摆（swing），这种缓动效果与匀速运动相比会令人感觉更自然，而且不生硬。

通过为`.animate()`方法提供一个额外的参数，我们可以改变jQuery动画使用的缓动样式。这个参数用来标识应该使用哪个缓动函数。内置于jQuery中的唯一函数就是实现我们刚才看到的缓动样式的默认函数^①；要使用其他函数，则必须求诸于插件或者自己来编写。

添加新的缓动函数与添加新的选择符表达式类似。可以通过扩展全局jQuery对象为它的`easing`对象添加属性。每个属性都对应着一个缓动函数。

假设我们想要实现一个纯粹的线性缓动样式，即让动画从头到尾匀速执行。那么，可以通过一个单行缓动函数来实现这个样式：

```
jQuery.extend({
  'easing': {
    'linear': function(fraction, elapsed, attrStart, attrDelta,
               duration) {
```

^① 自在jQuery 1.1.3开始到1.2.1，内置的缓动函数有两个，一个是linear另一个是swing。——译者注

```

        return fraction * attrDelta + attrStart;
    }
}
});

```

1. 缓动函数的参数

所有缓动函数都接受如下5个参数。

- fraction: 动画的当前位置, 按照时间衡量, 取值范围是0(动画的开始)到1(动画的结束)。
- elapsed: 自动画开始所经过的毫秒数(很少使用)。
- attrStart: 即将变换的CSS属性的开始值。
- attrDelta: 即将变换的CSS属性开始值与结束值的差。
- duration: 动画总共要经历的毫秒数(很少使用)。

缓动函数需要使用这个5个参数生成一个数字, 用于表示在任何给定的时间要变换的参数应该是什么值。假设使用我们的线性缓动函数将一个元素的高度从20像素变换到30像素, 如图10-9所示。

在这种简单的情况下, 需要用attrDelta的值乘以fraction得到参数到目前为止变换的增量距离。其中, elapsed的值会从0变到duration, fraction始终等于elapsed/duration, 而函数的值从attrStart变换到attrStart + attrDelta。

现在, 可以使用我们新的缓动样式来重复前面的动画效果:

```
$('.sprite').animate({'left': 791, 'opacity': 0.1}, 5000, 'linear');
```

在使用这个缓动函数后, 对动画的定时捕获会得到一个不同的画面, 如图10-10所示。

fraction	elapsed	attrStart	attrDelta	duration	function value
0	0	20	10	100	20
.25	25	20	10	100	22.5
.5	50	20	10	100	25
.75	75	20	10	100	27.5
1	100	20	10	100	30

图 10-9

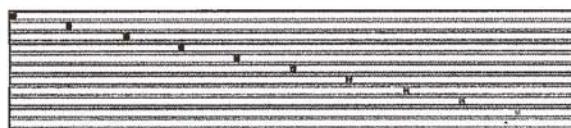


图 10-10

现在，动画以匀速前进。

2. 复合缓动样式

要得到某种更有趣的动画效果，可以编写一个在动画区间内沿着不同的特性曲线变换的缓动函数：

```
jQuery.extend({
  'easing': {
    'back-n-forth': function(fraction, elapsed, attrStart, attrDelta,
                      duration) {
      if (fraction < 0.33)
        return fraction * (1.0 / 0.33) * attrDelta + attrStart;
      if (fraction < 0.66)
        return (-fraction + 0.66) * (1.0 / 0.33) * attrDelta +
                           attrStart;
      return (fraction - 0.66) * (1.0 / 0.34) * attrDelta + attrStart;
    }
  }
});
```

这个函数把动画区间分成了3等块，每一块都沿着直线运动。我们可以像前面一样测试这个缓动样式：

```
$('.sprite').animate({'left': 791, 'opacity': 0.1}, 5000,
                     'back-n-forth');
```

这个动画的效果看起来是向前、再向后，然后再向前，如图10-11所示。

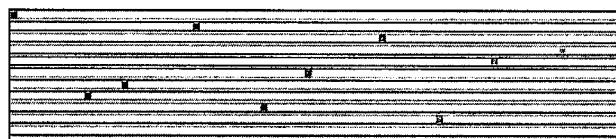


图 10-11

构建更复杂的缓动样式的主要问题，就是发现相应的数学表达式（或者表达式），以便生成动画需要遵行的曲线，然后把这个表达式通过JavaScript程序编写出来。

在像Interface之类的插件中，已经提供了很多现成的缓动函数。

10.4.8 做个好公民

为了保证我们编写的插件能够与其他代码和平共处，需要遵循一些规则。其中一些规则前面已经介绍过了，但为了便于参考，我们在这里再总结一下。

1. 命名约定

所有插件文件必须命名为jQuery.myPlugin.js，其中myPlugin是插件的名称。在这个文件

中，所有全局函数都应该组合到一个名为`jQuery.myPlugin`的对象中。除非插件中只有一个函数，在这种情况下，这个函数可能是`jQuery.myPlugin()`。

对象方法的命名可以更灵活一些，但应该尽可能保持唯一性。如果只定义了一个方法，那么该方法应该叫做`jQuery.fn.myPlugin()`。如果定义了多个方法，可以在方法名前面添加插件名作为前缀，以保持清晰。不要使用太短的、含糊的方法名，例如`.load()`或`.get()`，这样可能会导致与其他插件中定义的方法混淆。

2. 别名\$的使用

`jQuery`插件不能假设`$`有效。相反，每次都应该使用完整的`jQuery`名称。

在较长的插件中，许多开发者都觉得不使用`$`简写方式会使代码不易阅读。为解决这个问题，可以通过定义并执行函数的方式，在插件的作用域内定义局部的简写方式。定义并立即执行函数的语法如下所示：

```
(function ($) {
    // 函数的代码
})(jQuery);
```

这个包装函数接受一个参数，在此我们为这个参数传递的是全局`jQuery`对象。由于参数被命名为`$`，因此在这个函数的内部可以使用`$`别名而不会导致冲突。

3. 方法接口

所有`jQuery`方法都是在一个`jQuery`对象的环境中调用的，因此`this`引用的可能是一个包装了一个或多个DOM元素的对象。无论实际匹配的元素有多少，所有方法都必须以适当的方式运行。一般来说，方法应该调用`this.each()`来迭代匹配的元素，然后依次操作每个元素。

方法应该返回`jQuery`对象以保持连缀能力。如果匹配的对象集合被修改，那么应该通过调用`.pushStack()`创建一个新的`jQuery`对象，而且应该返回这个新对象。如果返回的值不是`jQuery`对象，必须明确地加以说明。

方法定义必须以分号结尾，以便代码压缩程序能够正确地解析相应的文件。

4. 文档的格式

文件中内置的文档应该以`ScriptDoc`格式在前面添加每个函数或方法的定义。要了解有关`ScriptDoc`格式的说明，请参考<http://www.scriptdoc.org/>。

10.5 小结

在本书的最后这一章中，我们看到`jQuery`核心提供的功能并没有限制这个库的能力。众多可

用的插件充分地扩展了jQuery的功能，而且，我们也可以方便地创建自己的插件，进一步拓展这个库的能力。

本章开始，我们介绍了用于度量和操作元素大小的Dimensions插件，讨论了同HTML表单进行交互时可以派上用场的Form插件。而且，也展示了通过Interface插件能够创建的各种用户界面元素。

然后，我们学习了如何创建包含各种功能的自定义插件，包括使用jQuery库的全局函数、定义操作DOM元素的jQuery对象的新方法、以新的方式增强查找DOM元素的选择符表达式，以及自定义修改动画样式的缓动函数等。

通过有效地使用这些工具，我们可以把jQuery，以及我们自己的JavaScript代码，塑造成任何想要的形式。



附录 A

在线资源

*I can't remember what I used to know
Somebody help me now and let me go*

—Devo,

“Deep Sleep”

下面列出的在线资源提供了在本书内容之外深入学习jQuery、JavaScript以及Web开发的起点。由于因特网上的高品质资源实在太多，本附录不可能全部罗列这些资源。此外，其他出版物中也可能会提供这里没有提到的有价值的信息。

A.1 jQuery 文档

jQuery Wiki

位于jquery.com上的文档是以维基形式存在的，这意味着文档的内容可以由公众编辑。下面这个网站上包括完整的jQuery API、教程、入门指南、插件仓库等：

<http://docs.jquery.com/>

jQuery API

在jQuery.com上，有两个地方可以找到API——文档区域和分页API浏览器。

jQuery.com的文档区域不仅包含jQuery的方法，而且包含jQuery的全部选择符表达式：

<http://docs.jquery.com>Selectors>

<http://docs.jquery.com/>

<http://jquery.com/api>

jQuery API浏览器

Jörn Zaefferer整理了一个方便地查看jQuery API的树形浏览器，包含搜索和按字母或按类别排序功能：

<http://jquery.bassistance.de/api-browser/>

Visual jQuery

Yehuda Katz设计的另一个API浏览器既漂亮又实用。而且，还能从中查到许多jQuery插件的方法：

<http://www.visualjquery.com/>

Web开发者博客

Sam Collet在他的博客上维护了一个全面的jQuery文档列表，包括可下载的版本和备忘表(cheat sheet)：

<http://webdevel.blogspot.com/2007/01/jquery-documentation.html>

A.2 JavaScript 参考

Mozilla开发者中心

这个网站包含全面的JavaScript参考、学习JavaScript编程的指南、实用工具的链接等：

<http://developer.mozilla.org/en/docs/JavaScript>

Dev.Opera

虽然Opera的这个针对开发者的网站主要面向它自己的浏览器平台，但其中包含了许多有用的JavaScript文章：

<http://dev.opera.com/articles/>

Quirksmode

Peter-Paul Koch的Quirksmode网站是非常棒的资源，通过这个网站可以了解浏览器在实现JavaScript功能及许多CSS属性方面的差异：

<http://www.quirksmode.org/>

JavaScript Toolbox

Matt Kruse的JavaScript Toolbox是一个朴素的JavaScript库的大型分类网站，其中也提供了有关JavaScript最佳实践的建议和Web上各种经过审查的JavaScript资源的集合：

<http://www.javascripttoolbox.com/>

A.3 JavaScript 代码压缩程序

Packer

这是由Dean Edwards创建的JavaScript压缩器/模糊器，jQuery的源代码就是通过它来压缩的。这个压缩程序提供了基于Web的界面，也可以免费下载。经过压缩后的代码会有效地减少文件大小，而增加的执行时间则很少：

<http://dean.edwards.name/packer/>
<http://dean.edwards.name/download/#packer>

JSMin

由Douglas Crockford创建的JSMin是一个筛选程序，用于从JavaScript文件中删除注释和不必要的空格。这个程序通常能够使文件大小减半，从而节省下载时间：

<http://www.crockford.com/javascript/jsmin.html>

Pretty Printer

这个工具会“修饰”经过压缩的JavaScript，能够尽可能地恢复换行及缩进。而且，该程序也为修整结果提供了很多选项：

<http://www.prettyprinter.de/>

A.4 (X)HTML 参考

W3C超文本标记语言主页

W3C（World Wide Web Consortium，万维网联盟）颁布了(X)HTML标准，而其HTML主页也是学习各种规范和指导方针的一个不错的起点：

<http://www.w3.org/MarkUp/>

A.5 CSS 参考

W3C层叠样式表主页

W3C的CSS主页中包含教程、规范、测试套件以及其他资源的链接：

<http://www.w3.org/Style/CSS/>

Mezzoblue CSS Cribsheet

Dave Shea提供这个有帮助的CSS“备忘录”(cribsheet)是为了让设计过程更加容易，同时也为你在遇到麻烦时提供一份快捷的参考：

<http://mezzoblue.com/css/cribsheet/>

Position Is Everything

这个网站中包含了一份浏览器中存在的CSS bug的目录，并解释了如何战胜这些bug：

<http://www.positioniseverything.net/>

A.6 XPath 参考

W3C XML Path语言1.0版规范

虽然jQuery对XPath的支持是有限的，但W3C的XPath规范对想要学习更多XPath选择符的人

仍然是很有用的：

<http://www.w3.org/TR/xpath>

TopXML XPath参考

TopXML网站为想学习XPath的人提供了实用的按坐标轴、节点测试及函数分类的图表：

<http://www.topxml.com/xsl/XPathRef.asp>

MSDN XPath参考

MSDN (Microsoft Developer Network, 微软开发者网络) 网站中也包含XPath语法和函数的信息：

<http://msdn2.microsoft.com/en-us/library/ms256115.aspx>^①。

A.7 有用的博客

jQuery官方博客

John Resig及其他供稿人会在这个官方博客中发布与新版本及项目组其他创新有关的文章，偶尔也会发布教程和某些声明：

<http://jquery.com/blog/>

Learning jQuery

Karl Swedberg、Jonathan Chaffer^②、Brandon Aaron等人维护的一个包括jQuery教程、例子及公告的博客：

<http://www.learningjquery.com/>

Jack Slocum的Blog

Jack Slocum是流行的JavaScript库EXT的作者，他在这个博客中记录了自己的日常工作和有关JavaScript编程的经验：

<http://www.jackslocum.com/blog/>

Web Standards with Imagination

Dustin Diaz的这个博客以Web设计和开发方面的文章见长，主要侧重于JavaScript：

<http://www.dustindiaz.com/>

Snook

Jonathan Snook的全面的编程/Web开发博客：

① 中文版URL为<http://msdn2.microsoft.com/en-us/library/ms256115.aspx>。——译者注

② 这两位是本书的作者。——译者注

<http://snook.ca/>

I Can't

以下由Christian Heilmann维护的3个网站包含JavaScript和Web开发方面的博客、示例代码和长篇文章：

<http://icant.co.uk/>
<http://www.wait-till-i.com/>
<http://www.onlinetools.org/>

DOM Scripting

Jeremy Keith的博客专为那本流行的DOM脚本编程的图书拾遗补缺^①——是有关不唐突的JavaScript的绝好资源：

<http://domscripting.com/blog/>

As Days Pass By

Stuart Langridge会在这里与大家分享浏览器DOM的高级用途：

<http://www.kryogenix.org/code/browser/>

A List Apart

A List Apart探索Web内容的设计、开发和内涵，特别关注Web标准和最佳实践：

<http://www.alistapart.com/>

Particletree

Chris Campbell、Kevin Hale和Ryan Campbell共同启动的这个博客，提供了涉及Web开发方方面面的有价值的信息：

<http://particletree.com/>

The Strange Zen of JavaScript

Scott Andrew LePera的博客专注于Web应用程序的实际应用，内容涉及有关JavaScript的陷阱(quirk)、告诫、技巧、珍闻及名言录：

<http://jszen.blogspot.com/>

A.8 使用jQuery的Web开发框架

很多开源项目的开发者们在了解jQuery之后，都把这个JavaScript库整合到了他们自己的系统中。以下是较早采用jQuery的部分框架的列表。

^① 这本书中文版即人民邮电出版社出版的《JavaScript DOM编程艺术》。——译者注

- Drupal: <http://drupal.org/>
- Joomla Extensions: <http://extensions.joomla.org/>
- Pommo: <http://pommo.org/>
- SPIP: <http://www.spip.net/>
- Textpattern: <http://textpattern.com/>
- Trac: <http://trac.edgewall.org/>
- WordPress: <http://wordpress.org/>

要了解更全面的列表, 请访问以下Sites Using jQuery页面:

http://docs.jquery.com/Sites_Using_jQuery



附录 B

开发工具

*When a problem comes along
You must whip it*

—Devo,
“Whip It”

虽然文档能够帮我们发现JavaScript应用程序中的问题，但一套优秀的软件开发工具也是不可替代的。好在，有很多现成的软件包可以用来检查和调试JavaScript代码，而且其中多数都可以免费使用。

B.1 针对 Firefox 的工具

Mozilla Firefox是大多数Web开发者的首选浏览器，因而也拥有最广泛和最受好评的开发工具。

Firebug

Firefox的Firebug扩展对jQuery开发来说是必不可少的：

<http://www.getfirebug.com/>

Firebug主要具有以下功能。

- 极佳的DOM查看器，可以用来方便地查找文档片段的名称和选择符。
- CSS操作工具，可以用来查明页面的样式来源，并且可以直接修改CSS样式。
- 交互性的JavaScript控制台。
- JavaScript调试程序，可以用来监视变量和跟踪代码执行。

Web Developer Toolbar

Web Developer Toolbar不仅涵盖了Firebug中DOM检视的功能，还包含一些完成常见任务的工具，例如cookie操作、表单检查和页面缩放等。还可以使用这个工具条方便快捷地禁用网站中的JavaScript，以确保网站中的功能对缺乏JavaScript的用户能够平稳退化：

<http://chrисpederick.com/work/web-developer/>

Venkman

Venkman是Mozilla项目中官方的JavaScript调试程序。这个程序提供了一个故障诊断环境，让人很容易联想到以其他语言编写的GDB（GNU Project Debugger）调试程序。

<http://www.mozilla.org/projects/venkman/>

正则表达式测试器

在JavaScript中用编写匹配字符串的正则表达式需要一些技巧。通过这个Firefox扩展，可以在输入搜索文本的界面中方便地测试正则表达式：

<http://sebastianzartner.ath.cx/new/downloads/RExT/>

B.2 针对IE的工具

许多网站在IE和其他浏览器中都会有不一致的表现，因此拥有针对这个平台的调试工具也很重要。

Microsoft Internet Explorer Developer Toolbar

这个开发者工具条主要提供了一个网页的DOM树视图。通过这个视图，不仅能形象地查找元素，而且还能动态修改CSS规则。此外，这个工具条还提供了其他辅助开发功能，例如用于度量页面元素大小的尺子等：

<http://www.microsoft.com/downloads/details.aspx?FamilyID=e59c3964-672d-4511-bb3e-2d5e1db91038>

Microsoft Visual Web Developer

微软的Visual Studio程序包也可以用来检查和调试JavaScript代码：

<http://msdn.microsoft.com/vstudio/express/vwd/>

请按照以下链接中的指示，可以交互地运行这个调试程序的免费版本（Visual Web Developer Express）：

<http://www.berniecode.com/blog/2007/03/08/how-to-debug-javascript-with-visual-web-developer-express/>

DebugBar

DebugBar提供了一个DOM观察器和一个用于调试的JavaScript控制台：

<http://www.debugbar.com/>

Drip

JavaScript代码造成的内存泄漏可能会在IE中导致性能及稳定性问题。Drip有助于检测和隔离

这些内存问题：

<http://Sourceforge.net/projects/ieleak/>

要了解造成IE内存泄漏的常见原因，请参考附录C。

B.3 针对 Safari 的工具

Safari是开发平台中的新秀，但针对代码在这款浏览器中与在其他浏览器中的差异，也有一些可用的工具。

Web Inspector

Safari的Nightly Build中包含了检查个别页面元素及集合信息的能力，特别是能够检查出应用到每个元素上面的CSS规则：

<http://trac.webkit.org/projects/webkit/wiki/Web%20Inspector>

Drosera

Drosera针对Safari的JavaScript调试程序和另外一个WebKit驱动的应用程序。这个调试程序支持断点、变量监视和交互性的控制台。

<http://trac.webkit.org/projects/webkit/wiki/Drosera>

B.4 其他工具

Firebug Lite

尽管Firebug扩展自身局限于Firefox浏览器，但通过在网页中包含Firebug Lite脚本则可以再现其某些功能。通过这个模拟Firebug的代码包，可以在任何浏览器中调用`console.log()`而不会引发JavaScript错误：

<http://www.getfirebug.com/lite.html>

TextMate jQuery包

这个扩展针对流行的Mac OS X文本编辑器提供了对jQuery方法和选择符的语法高亮、方法完成及在代码中查看简单API的功能。这个包也兼容Windows平台中的E文本编辑器：

<http://www.learningjquery.com/2006/09/textmate-bundle-for-jquery>

Charles

当开发AJAX主导的应用程序时，实际地观察在浏览器和服务器之间传输的数据非常有必要。Web调试代理Charles可以显示这两点间的所有通信过程，包括正常的Web请求、HTTPS通信、Flash远程通信和AJAX响应等：

<http://www.xk72.com/charles/>

Aptana

这个基于Java的Web开发IDE（Integration Development Environment，集成开发环境）是免费的，而且是跨平台的。除了提供标准和高级的代码编辑功能之外，它还整合了完整版的jQuery API文档。

<http://www.aptana.com/>

附录 C

JavaScript闭包

*Let's close our eyes together
Now can you see how good it's going to be?*

—Devo,
“Pink Jazz Trancers”

在本书中，我们看到过很多以函数作为参数的jQuery方法。在我们所举的例子中，也曾经反复地创建、调用和传递函数。虽然我们平时只需粗略地了解JavaScript的内部工作机制，就可以这样使用函数，但是，如果缺乏对这个语言特性的深入理解，那么这些操作的负面作用也会时不时给我们带来意想不到的结果。在本附录中，我们再额外探讨一种深奥（也很流行）的函数类型，这就是闭包。

C.1 内部函数

能够跻身支持内部函数声明的编程语言行列，对JavaScript来说应该算是一种幸运。许多传统的编程语言（例如C），都会把全部函数集中在顶级作用域中。而支持内部函数的语言，则允许开发者在必要的地方集合小型实用函数，以避免对命名空间的干扰。

所谓内部函数，就是定义在另一个函数中的函数。例如：

```
function outerFun() {  
    function innerFun() {  
        alert('hello');  
    }  
}
```

innerFun()就是一个被包含在outerFun()作用域中的内部函数。这意味着，在outerFun()内部调用innerFun()是有效的，而在outerFun()外部调用innerFun()则是无效的。下列代码会导致一个JavaScript错误：

```
function outerFun() {  
    function innerFun() {  
        alert('hello');  
    }  
}  
innerFun();
```

不过，为了触发警告框的显示，可以在outerFun()内部调用innerFun()：

```
function outerFun() {
    function innerFun() {
        alert('hello');
    }
    innerFun();
}
outerFun();
```

这种技术特别适合于小型、单用途的函数。例如，递归但却带有非递归API包装的算法通常最适合通过内部函数来表达。

C.2 伟大的逃脱

在函数引用参与进来之后，问题就变得复杂了。有些语言，比如Pascal，允许通过内部函数实现代码隐藏，而这些函数因此也会永远被埋没在它们的父函数中。然而，JavaScript则允许开发者像传递任何类型的数据一样传递函数。也就是说，JavaScript中的内部函数能够逃脱定义它们的外部函数。

逃脱的方式有很多种。例如，可以将内部函数指定给一个全局变量：

```
var globVar;

function outerFun() {
    function innerFun() {
        alert('hello');
    }
    globVar = innerFun;
}
outerFun();
globVar();
```

在函数定义之后调用outerFun()会修改全局变量globVar，此时它引用的是innerFun()。这意味着，后面调用globVar()的操作就如同调用innerFun()一样，也会导致显示警告框。但此时在outerFun()外部直接调用innerFun()仍然会导致错误！这是因为虽然内部函数通过把引用保存在全局变量中实现了逃脱，但这个函数的名字仍然被截留在outerFun()的作用域中。

另外，也可以通过在父函数中返回值来营救出内部函数的引用：

```
function outerFun() {
    function innerFun() {
        alert('hello');
    }
    return innerFun ;
}
var globVar = outerFun();
globVar();
```

这里，并没有在outerFun()内部修改全局变量，而是从outerFun()中返回了一个对innerFun()的引用。通过调用outerFun()能够取得这个引用，而且，这个引用可以保存在变量中，也可以自己调用自己，从而触发警告框。

这种即使在离开函数作用域的情况下，仍然能够通过引用调用内部函数的事实，意味着只要存在调用这些内部函数的可能，JavaScript就需要保留被引用的函数。而且，JavaScript运行时程序需要跟踪引用这个内部函数的所有变量，直至最后一个变量废弃，JavaScript的垃圾收集器才能外面释放相应的内存空间。

C.3 变量作用域

内部函数当然也可以拥有自己的变量，只不过这些变量都被限制在内部函数的作用域中：

```
function outerFun() {
    function innerFun() {
        var innerVar = 0;
        innerVar++;
        alert(innerVar);
    }
    return innerFun;
}
```

每当通过引用或其他方式调用这个内部函数时，都会创建一个新的innerVar变量，然后递增，最后显示：

```
var globVar = outerFun();
globVar(); // 警告框中显示：1
globVar(); // 警告框中显示：1
var innerVar2 = outerFun();
innerVar2(); // 警告框中显示：1
innerVar2(); // 警告框中显示：1
```

内部函数可以像其他函数一样引用全局变量：

```
var globVar = 0;
function outerFun() {
    function innerFun() {
        globVar++;
        alert(globVar);
    }
    return innerFun;
}
```

现在，每次调用内部函数都会持续地递增这个全局变量的值：

```
var globVar = outerFun();
globVar(); // 警告框中显示：1
globVar(); // 警告框中显示：2
var globVar2 = outerFun();
```

```
globVar2(); // 警告框中显示: 3
globVar2(); // 警告框中显示: 4
```

但是，如果这个变量是父函数的局部变量又会怎样呢？因为内部函数会继承父函数的作用域，所以内部函数也可以引用这个变量：

```
function outerFun() {
    var outerVar = 0;
    function innerFun() {
        outerVar++;
        alert(outerVar);
    }
    return innerFun;
}
```

这一次，对内部函数的调用会产生有意思的行为：

```
var globVar = outerFun();
globVar(); // 警告框中显示: 1
globVar(); // 警告框中显示: 2
var globVar2 = outerFun();
globVar2(); // 警告框中显示: 1
globVar2(); // 警告框中显示: 2
```

我们看到了前面两种情况合成的效果。通过每个引用调用innerFun()都会独立地递增innerVar。也就是说，第2次调用outerFun()没有继续沿用innerVar的值，而是在第2次函数调用的作用域中创建并绑定了一个新的innerVar的实例。结果，就造成了在上面的调用之后继续调用globVar()会导致警告框中显示3，而再次调用globVar2()也会导致警告框中显示3。这两个计数器完全是无关的。

当内部函数在定义它的作用域的外部被引用时，就创建了该内部函数的一个闭包。在这种情况下，我们称不是内部函数局部变量的变量为**自由变量**，称外部函数的调用环境为**封闭闭包的环境**。从本质上讲，如果内部函数引用了位于外部函数中的变量，相当于授权该变量能够被延迟使用。因此，当外部函数调用完成后，这些变量的内存不会被释放，因为闭包仍然需要使用它们。

C.4 闭包之间的交互

当存在多个内部函数时，很可能会出现意料之外的闭包。假设我们又定义了一个递增函数，这个函数中的增量为2：

```
function outerFun() {
    var outerVar = 0;
    function innerFun() {
        outerVar++;
        alert(outerVar);
    }
    function innerFun2() {
        outerVar = outerVar + 2;
        alert(outerVar);
    }
}
```

```

        }
        return {'innerFun': innerFun, 'outerFun2': outerFun2};
    }
}

```

这里，我们通过映射返回两个内部函数的引用（这也示范了内部函数的引用逃脱父函数的另一种方式）。可以通过返回的引用调用任何一个内部函数：

```

var globVar = outerFun();
globVar.innerFun(); // 警告框中显示: 1
globVar.innerFun2(); // 警告框中显示: 3
globVar.innerFun(); // 警告框中显示: 4
var globVar2 = outerFun();
globVar2.innerFun(); // 警告框中显示: 1
globVar2.innerFun2(); // 警告框中显示: 3
globVar2.innerFun(); // 警告框中显示: 4

```

这两个内部函数引用了同一个局部变量，因此它们共享同一个封闭环境。当innerFun()为outerVar递增1时，就为调用innerFun2()设置了outerVar的新的起点值。同样，我们也看到对outerFunc()的后续调用还会创建这些闭包的新实例，同时也会创建相应的新封闭环境。面向对象编程的爱好者们会注意到，这在本质上是创建了一个新对象，自由变量就是这个对象的实例变量，而闭包就是这个对象的实例方法。而且，这些变量也是私有的，因为不能在封装它们的作用域外部直接引用这些变量，从而确保了面向对象的数据专有特性。

C.5 jQuery 中的闭包

我们曾经介绍过的jQuery库中的许多方法都至少要接受一个函数作为参数。为方便起见，我们通常都在这种情况下使用匿名函数，以便在必需时再定义函数的行为。但是，这也意味着我们很少在顶级命名空间中定义函数；也就是说，这些函数都是内部函数，而内部函数很容易就会变成闭包。

C.5.1 `$(document).ready()` 的参数

我们使用jQuery编写的几乎全部代码都要放在作为`$(document).ready()`参数的一个函数内部。这样做是为了保证在代码运行之前DOM已经就绪，而DOM就绪通常是运行jQuery代码的一个必要条件。当创建了一个函数并把它传递给`.ready()`之后，这个函数的引用就会被保存为全局jQuery对象的一部分。在稍后的某个时间——当DOM就绪时，这个引用就会被调用。

由于我们通常把`$(document).ready()`放在代码结构的顶层，因而这个函数不会成为闭包。但是，我们的代码通常都是在这个函数内部编写的，所以这些代码都处于一个内部函数中：

```

$(document).ready(function() {
    var readyVar = 0;
    function outerFun() {
        function innerFun() {
            readyVar++;
            alert(readyVar);
        }
    }
})

```

```

        }
        return innerFun;
    }
    var readyVar2 = outerFun();
    readyVar2();
});

```

这看上去同前面全局变量的例子有些类似，只不过它们同其他的代码一样都被包装在对`$(document).ready()`的一次调用中。这意味着`readyVar`不是全局变量，而是匿名函数的局部变量。但是，变量`readyVar2`则取得了对一个闭包的引用，而闭包的环境中封闭着`readyVar`。

把大多数jQuery代码都放在一个函数体中是很有用的，因为这样可以避免某些命名空间冲突。例如，正是这个特性可以使我们通过调用`jQuery.noConflict()`为其他库释放简写方式`$`，但我们仍然能够定义在`$(document).ready()`中使用的局部简写方式。

C.5.2 事件处理程序

`$(document).ready()`结构通常用于包装其他的jQuery代码，包括事件处理程序的赋值。因为处理程序是函数，它们也就变成了内部函数；而且，因为这些内部函数会被保存并在以后调用，于是它们也变成了闭包。以一个简单的单击处理程序为例：

```

$(document).ready(function() {
    var readyVar = 0;
    $('.trigger').click(function() {
        readyVar++;
        alert(readyVar);
    });
});

```

由于变量`readyVar`是在`.ready()`处理程序中声明的，所以它只对位于这个块中的jQuery代码有效，对`.ready()`处理程序外部的代码无效。然而，这个变量可以被`.click()`处理程序中的代码引用，在这个例子中`.click()`应用程序会递增并通过警告框显示该变量。由于创建了闭包，每次单击按钮都会引用`readVar`的同一个实例。也就是说，警告框会显示持续递增的一组值，而不是每次都显示1。

事件处理程序同其他函数一样，也能够共享它们的封闭环境：

```

$(document).ready(function() {
    var readyVar = 0;
    $('.add').click(function() {
        readyVar++;
        alert(readyVar);
    });
    $('.subtract').click(function() {
        readyVar--;
        alert(readyVar);
    });
});

```

因为这两个函数引用的是同一个变量，所以两个按钮的递增和递减操作会影响同一个值，而不是各自独立的值。

这些例子都和我们常规的jQuery代码一样使用了匿名函数。但是，这不会影响到闭包的构建。例如，我们可以编写一个匿名函数，报告jQuery对象中每个项的索引：

```
$(document).ready(function() {
    $('li').each(function(index) {
        $(this).click(function() {
            alert(index);
        });
    });
});
```

由于最里面的函数是在`.each()`回调函数中定义的，因而以上代码实际上创建了同存在的列表项一样多的函数。这些函数分别作为一个单击处理程序被添加给了相应的列表项。而且，由于`.each()`回调函数拥有参数`index`，所以在这些函数的封闭环境中都有各自的`index`变量。这就如同把单击处理程序的代码写成一个命名函数：

```
$(document).ready(function() {
    $('li').each(function(index) {
        function clickHandler() {
            alert(index);
        }
        $(this).click(clickHandler);
    });
});
```

只不过使用匿名函数的版本更短一些而已。然而，这个命名函数的位置也是很重要的：

```
$(document).ready(function() {
    function clickHandler() {
        alert(index);
    }
    $('li').each(function(index) {
        $(this).click(clickHandler);
    });
});
```

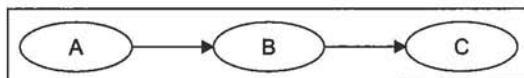
这个版本会导致无论单击哪个列表项都会触发一个JavaScript错误，因为在`clickHandler()`的封闭环境中找不到`index`。此时，`index`仍然是一个自由变量，但它在这个环境中没有定义。

C.6 内存泄漏的风险

JavaScript使用一种称为垃圾收集的技术来管理分配给它的内存。这与C这样的低级语言不同，C要求程序员明确地预定内存空间，并在这些内存不再使用时释放它们。其他语言，比如Objective-C，实现了一个引用计数系统来辅助程序员完成这些工作。通过这个引用计数系统，程序员能够了解到有多少个程序块使用了一个特定的内存段，因而可以在不需要时清除这些内存

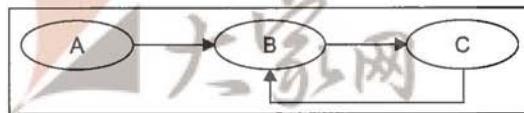
段。另一方面，JavaScript是一种高级语言，它一般是通过后台来维护这种计数系统。

当JavaScript代码生成一个新的内存驻留项时（比如一个对象或函数），系统就会为这个项留出一块内存空间。因为这个对象可能会被传递给很多函数，并且会被指定给很多变量，所以很多代码都会指向这个对象的内存空间。JavaScript会跟踪这些指针，当最后一个指针废弃不用时，这个对象占用的内存会被释放。以下面的指针链接为例：



图中的对象A有一个属性指向B，而B也有一个属性指向C。即使当前作用域中只有对象A有效，但由于指针的关系所有3个对象都必须保留在内存中。当离开A的当前作用域时（例如代码执行到声明A的函数的末尾处），垃圾收集器就可以释放A占用的内存。此时，由于没有什么指向B，因此B可以释放，最后，C也可以释放。

然而，当对象间的引用关系变得复杂时，处理起来也会更加困难：



这里，我们又为对象C添加了一个引用B的属性。在这种情况下，当A释放时，仍然有来自C的指针指向B。这种引用循环需要由JavaScript进行特殊的处理，但必须考虑到整个循环与作用域中的其他变量已经处于隔离状态。

C.6.1 意外的引用循环

闭包可能会导致在不经意间创建引用循环。因为函数是必须保存在内存中的对象，所以位于函数封闭环境中的所有变量也需要保存在内存中：

```

function outerFun() {
    var outerVar = {};
    function innerFun() {
        alert(outerVar);
    };
    outerVar.innerFun = innerFun;
    return innerFun;
}

```

这里创建了一个名为outerVar的对象，该对象在内部函数innerFun()中被引用。然后，为outerVar创建了一个指向innerFun()的属性，之后返回了innerFun()。这样就在innerFun()上创建了一个引用outerVar的闭包，而outerVar又引用了innerFun()。但是，也可能会出现比这种情况更隐蔽的引用循环：

```

function outerFun() {

```

```

var outerVar = {};
function innerFun() {
    alert('hello');
};
outerVar.innerFun = innerFun;
return innerFun;
};

```

这里我们修改了innerFun()，使它不再引用outerVar。但是，这样做仍然没有断开循环。即使innerFun()不再引用outerVar，outerVar也仍然位于innerFun()的封闭环境中。由于闭包的原因，位于outerFun()中的所有变量都隐含地被innerFun()所引用。因此，闭包会使意外地创建这些引用循环变得易如反掌。

C.6.2 IE 中的内存泄漏问题

上述这些情况通常不是什么问题，因为JavaScript能够检测到这些情况并在它们孤立时将其清除。然而，IE中存在一种难以处理的引用循环问题。当一个循环中同时包含DOM元素和常规JavaScript对象时，IE无法释放任何一个对象——因为这两类对象是由不同的内存管理程序负责管理的。换句话说，除非关闭浏览器，否则这种循环在IE中永远得不到释放。为此，随着时间的推移，这可能会导致大量内存被无效地占用。导致这种循环的一个常见原因是简单的事件处理程序：

```

$(document).ready(function() {
    var div = document.getElementById('foo');
    div.onclick = function() {
        alert('hello');
    }
});

```

当指定单击事件处理程序时，就创建了一个在其封闭的环境中包含div变量的闭包。而且，现在的div也包含一个指向闭包的引用。这样，就导致了在IE中即使离开当前页面也不会释放这个循环。

C.6.3 好消息

下面，我们通过常规的jQuery结构来编写同样的代码：

```

$(document).ready(function() {
    var $div = $('#foo');
    $div.click(function() {
        alert('hello');
    });
});

```

即使此时仍然会创建一个闭包，并且也会导致同前面一样的循环，但这里的代码却不会使IE发生内存泄漏。由于jQuery考虑到了内存泄漏的潜在危害，所以它会手动释放自己指定的所有事件处理程序。只要坚持使用jQuery的事件绑定方法，就无需为这种特定的常见原因导致的内存泄漏而担心。

但是，这并不意味着我们完全脱离了险境。当对DOM元素进行其他操作时，仍然要处处留心。只要是将JavaScript对象指定给DOM元素，就可能在IE中导致内存泄漏。jQuery只是有助于减少发生这种情况的可能性。

C.7 结束语

JavaScript闭包是一种强大的语言特性。通过使用这个语言特性来隐藏变量，可以避免覆盖其他地方使用的同名变量。由于jQuery经常依赖于把函数作为方法的参数，所以在编写jQuery代码时也会经常在不经意间创建闭包。理解闭包有助于编写出更有效也更简洁的代码，如果再加上一些小心并利用好jQuery内置的安全措施，则可以有效地防止闭包可能引发的内存泄漏问题。