

Learning jQuery 1.3

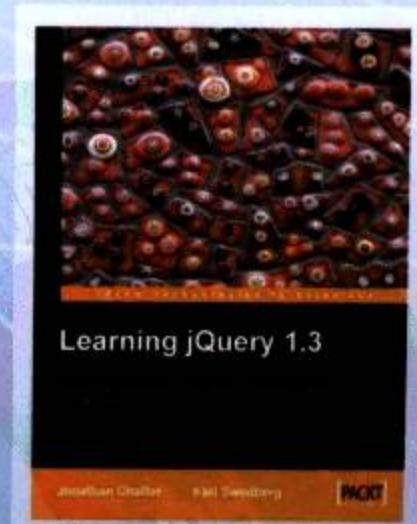
Better Interaction Design and Web Development with Simple JavaScript Techniques

jQuery基础教程（第2版）

[美] Jonathan Chaffer 著
Karl Swedberg

李松峰 卢玉平 译

- 《jQuery基础教程》升级版，涵盖jQuery 1.3
- Amazon全五星盛誉
- jQuery官方网站推荐



人民邮电出版社
POSTS & TELECOM PRESS

“本书是学习jQuery的最佳途径。”

—Slashdot.org

“本书和jQuery框架本身一样优雅和简洁。作者恰到好处地讲述了如何使用jQuery加快RIA开发。”

—Ajaxian.com

Learning jQuery 1.3

Better Interaction Design and Web Development with Simple JavaScript Techniques

jQuery基础教程（第2版）

jQuery是功能强大却又简洁明快的轻量级JavaScript库，出自名家之手。因为在DOM操作、事件处理、Ajax特性以及动画特效等方面的出色表现，它迅速从众多JavaScript库中脱颖而出，成为一颗闪亮的明星。

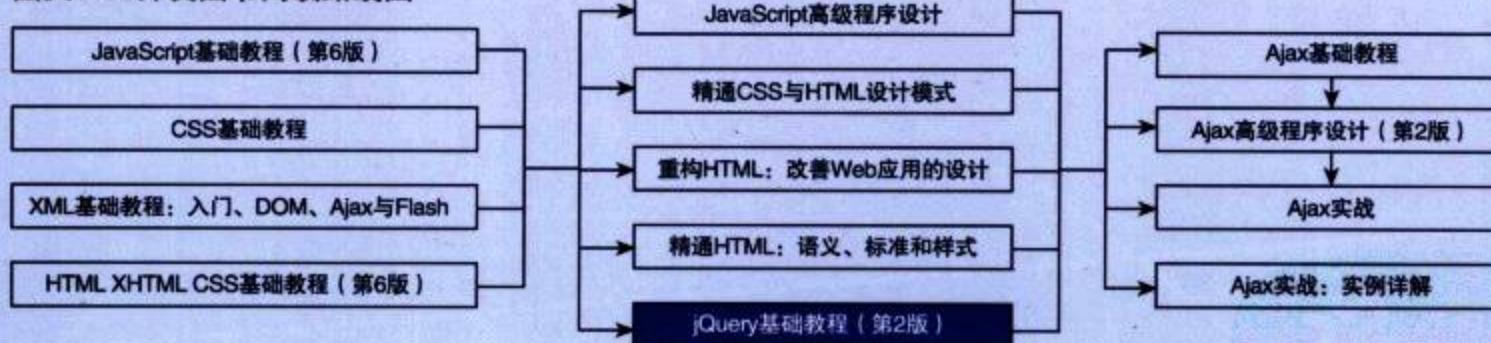
本书作为《jQuery基础教程》的升级版，涵盖了jQuery 1.3的各种新特性，特别是新增了介绍jQuery UI（jQuery官方用户界面插件库）的内容。本书源自著名jQuery资源网站LearningjQuery.com，深得业界好评。从中你不仅能够学到jQuery的基本知识，还能领略大量堪称典范的实例和JavaScript编程最佳实践。值得一提的是，本版新增的附录D分门别类地列出了所有jQuery API，为高效使用jQuery提供了方便。

本书注重理论与实践相结合，适合初中级Web开发人员阅读和参考。

Jonathan Chaffer 资深Web专家，Structure互动公司CTO。著名jQuery资源网站LearningjQuery.com创始人之一。他还是著名的开源CMS项目Drupal的核心开发人员，开发了广受欢迎的Content Construction Kit（内容构建工具包）模块并大幅修改了菜单系统。

Karl Swedberg 资深Web程序员，曾在微软工作，目前就职于Structure互动公司。著名jQuery资源网站LearningjQuery.com创始人之一。

图灵Web开发图书阅读路线图



本书相关信息请访问：图灵网站 <http://www.turingbook.com>

读者/作者热线：(010)51095186

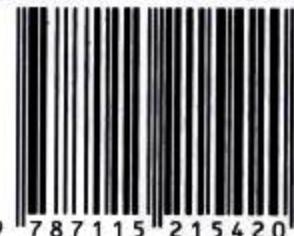
反馈/投稿/推荐信箱：contact@turingbook.com

分类建议 计算机/网络开发/程序设计

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-21542-0



ISBN 978-7-115-21542-0

定价：49.00元

TURING 图灵程序员设计丛书 Web 开发系列

Learning jQuery 1.3

Better Interaction Design and Web Development with Simple JavaScript Techniques

jQuery基础教程 (第2版)

[美] Jonathan Chaffer 著
Karl Swedberg
李松峰 卢玉平 译

人民邮电出版社
北京



图书在版编目 (CIP) 数据

jQuery 基础教程：第 2 版 / (美) 查弗
(Chaffer, J.), (美) 斯威德伯格 (Swedberg, K.) 著；
李松峰, 卢玉平译. — 2 版. — 北京：人民邮电出版社，
2009.11

(图灵程序设计丛书)

书名原文：Learning jQuery 1.3: Better
Interaction Design and Web Development with Simple
JavaScript Techniques
ISBN 978-7-115-21542-0

I. ①j… II. ①查… III. ①JAVA 语言—程序设计—
教材 IV. ①TP312

中国版本图书馆CIP数据核字 (2009) 第175880号

内 容 提 要

本书作为《jQuery 基础教程》的升级版，涵盖了 jQuery 1.3 的全部新特性，特别是新增了介绍 jQuery UI (jQuery 官方用户界面插件库) 的内容。本书前 6 章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、效果、DOM 操作、AJAX 支持等。随后 3 章从理论到实践，通过表格操作、构建功能型表单、实现滑移和翻转效果等实例，深入浅出地讲解了如何创造性地运用 jQuery 提供的丰富而强大的 API。本书最后两章专门介绍了如何使用和编写 jQuery 插件。值得一提的是，本版新增的附录 D 分门别类地列出了所有 jQuery API，为高效使用 jQuery 提供了方便。

本书注重理论与实践相结合，适合初中级 Web 开发人员阅读和参考。

图灵程序设计丛书

jQuery基础教程 (第2版)

◆ 著 [美] Jonathan Chaffer Karl Swedberg
译 李松峰 卢玉平
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

北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：21.5
字数：508千字 2009年11月第2版
印数：8 001—12 000册 2009年11月北京第1次印刷

ISBN 978-7-115-21542-0

定价：49.00元

版 权 声 明

Copyright © 2009 Packt Publishing. First published in the English language under the title *Learning jQuery 1.3: Better Interaction Design and Web Development with Simple JavaScript Techniques*.

Simplified Chinese-language edition copyright © 2009 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

从重印次数以及我博客上的留言来看，本书的上一版《jQuery基础教程》确实受到了广大读者的热烈欢迎。

《jQuery基础教程》出版时jQuery的主版本号为1.1，目前的最新版本是1.3。简略来说，1.2版（2007年9月）主要是去掉了对XPath选择符的多余支持，并推出了jQuery UI。而1.3版（2009年1月）的主要变化则是使用了全新的选择符引擎Sizzle (<http://sizzlejs.com/>)，大幅提升了库的整体性能。

本书这一新版本就是基于当前最新的jQuery 1.3编写和修订的。与上一版相比，本书新增了一章内容（全书达到了11章）和一个附录（详见目录）。另外，大部分章节都补充了很多关键的新内容。整体更新量大约占全书的三分之一以上。对于这个新版本，译者的总体感觉：一是内容质量有了普遍的提升，不仅是修正了上一版中几乎所有的错误，而且补充了很多应该了解和掌握的重要JavaScript知识点；二是内容更加与时俱进，通过重写相关的示例，这一版把jQuery 1.3中的新特性讲解得更加淋漓尽致；三是附加值更大了，一方面是重写了上一版“附录C JavaScript闭包”中的示例，另一方面又新增了“附录D 快速参考”。

值得一提的就是这个附录D，基本上是*Learning jQuery*（本书上一版英文名称）姊妹篇*jQuery Reference Guide*（即“jQuery参考指南”）的一个“浓缩”。这个快速参考囊括了选择符表达式、DOM遍历方法、事件方法、效果方法、DOM操作方法、AJAX方法、其他方法等七大主题。具有一定jQuery开发经验的读者，完全可以将这个附录作为简明的jQuery中文API参考使用，避免通过电脑查询的麻烦。

感谢武卫东老师和傅志红老师对本书提出的修改意见，感谢杨爽同学对全书最后审定和出版付出的努力。

最后，希望读者能够一如既往地喜欢这本jQuery的经典教程！并且，能够通过本书领会到jQuery乃至富Web应用程序开发的真谛。

2009年9月

本书上一版译者序

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1秒钟后，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.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库和框架致力于解决的问题，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、隐含迭代、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皆然不同的理念，处处匠心独运，别出心裁——具体细节，请参考本书第1章。

《jQuery基础教程》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“JavaScript闭包”是名副其实的“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。

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。感谢图灵俱乐部“明月星光”网友的热心建言。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，请务必本着“治病救人”的白求恩精神，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@gmail.com

2008年2月于北京

序

得知Karl Swedberg和Jonathan Chaffer共同编写这本jQuery教程，我深感荣幸。作为第一本jQuery图书，它为其他jQuery——实际上，也为其他JavaScript——图书，树立了一个新标杆。第一版自面世以来，始终高居最畅销JavaScript图书榜首，究其原因，概源自其内在的高品质和对细节的关注。

我尤其高兴，是Karl和Jonathan共同执笔撰写了这本书，因为我对他们非常了解，知道他们是写这方面图书的最佳人选。作为jQuery开发团队的核心人员，我在过去的几年间对Karl有了充分的了解，特别是对他编写本书的情况十分熟悉。看看最终作品就会知道，作为开发人员和曾经的英文教师，由他来完成这个写书任务简直是老天的巧妙安排。

我还曾有机会与他们两位谋面——对于从事分布式开源项目工作的我们来说，这种见面机会算是极为难得的。当然，他们目前依旧是jQuery社区的中坚分子。

jQuery社区中有许许多多不同的人在使用jQuery，其中包括设计人员、开发人员、有编程经验的人和没有编程经验的人。即使是jQuery团队内部，也有很多不同背景的人为这个项目的发展提供各自的建议。来自五湖四海的jQuery用户都有着同一个目标，即我们这个由开发人员和设计人员组成的社区，其宗旨就是让JavaScript开发变得越来越简单。

此时此刻，重申开源项目是社区导向的，或者说开源项目的目标就是帮助新用户快速上手，好像总有几分陈词滥调的意味。然而，这个宗旨对jQuery而言绝非表面上做做姿态，其理念恰恰正是项目本身绵绵不绝的动力源泉。在jQuery团队中，除了维护核心代码的人，实际上还有更多的人在负责管理社区、撰写文档和编写插件。虽然库本身的稳定性至关重要，但代码背后的社区也绝对不容忽视。一个项目是等闲平庸、举步维艰，还是能处处满足甚至超出用户的期许，可以说完全取决于社区。

我们如何运营项目，用户如何使用我们的代码，是jQuery与大多数开源项目（以及大多数JavaScript库）的根本差异所在。jQuery项目及其社区是具有高度智慧的。我们深知是什么让jQuery带给了用户不同的编程体验，并且也在竭尽全力把这些知识和智慧传递给我们的用户。

袖手旁观永远不会理解jQuery社区，只有参与其中，潜心钻研，才能获得切身体验。我们衷心希望本书读者有朝一日都能够加入jQuery社区。无论是加入我们的论坛、邮件列表还是博客，jQuery社区都能为你更好地利用jQuery提供各方面帮助。

对我个人而言，jQuery绝不仅仅就是一些代码块那么简单，它是这几年来，为了让这个库更有价值，社区成员日积月累的所有经验的大汇聚。其中蕴涵着一次次惊心动魄的起起落落，一次

次开发过程中的奋斗挣扎，当然还有看着它不断成长和成功带来的喜悦。它贴近用户和团队成员，反映他们的需求，并且日益成长完善。

我一开始看到这本书将jQuery作为一个统一的工具来讨论时，第一感觉是书中介绍的jQuery跟我印象中汇聚各种经验的jQuery不太一样，但吃惊之余，更多的还是心潮澎湃。能够看到别人通过学习、理解进而塑造出的jQuery，作为项目创始人而言，其创造之乐也莫过如此了！

我决不是唯一超越工具—使用者关系层面去欣赏jQuery的人。我不确定能否准确地罗列出原因，但我已经多次看到这样的场面——当用户恍然领悟到jQuery的效力时，他们的脸上会情不自禁地流露出会心的微笑。

还有一个特别的时刻，也只有jQuery用户才能体会到——有一天，他们会突然意识到自己使用的工具，实际上远不是一个简单的工具，他们将顿悟原来可以彻底换个思维方式来编写动态Web应用程序。想想吧，那个时刻将会多么美妙，而我认为这绝对是jQuery项目最大的价值所在。

希望手捧本书的读者朋友，也能够体验到那美妙的时刻。

John Resig

jQuery创建人，Mozilla公司技术推广专家，畅销书*Pro JavaScript Techniques*作者^①

① 中文版《精通JavaScript》已经由人民邮电出版社出版。——译者注

前　　言

2005年, JavaScript神童John Resig(现在Mozilla公司工作)利用业余时间编写了一个JavaScript库。他受该领域的先驱人物Dean Edwards和Simon Willison等人的启发,为这个库编写了很多函数,利用这些函数能够以编程方式快速查找网页中的元素,并为这些元素指定行为。2006年1月,当他首次发布这个项目时,其中已经包含了DOM操作和基本的动画功能。他把这个项目命名为jQuery,意在强调其查找或“查询”网页元素,并通过JavaScript操作这些元素的核心用途。此后短短几年间, jQuery的功能越来越丰富,性能逐步提升,同时也被因特网上一些最有名的站点广泛采用。虽然Resig依旧在领导该项目的开发,但jQuery作为一个真正开源的项目,已经拥有了一个足以傲视群雄的、由顶尖JavaScript开发人员组成的核心团队,以及一个数千名开发人员组成的活跃社区。

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。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,数百种为扩展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,同时给出需要在实际开发中反复用到的jQuery库特性的简明参考。

本书内容

第1章将引领读者对jQuery有个大概的了解。这一章先简单介绍jQuery及其用途,然后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。

第2章讲述如何通过jQuery中的选择符表达式及DOM遍历方法,在页面中的任何地方找到想要的元素。这一章将展示如何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,还会介绍如何以不唐突的方式添加事件(甚至在页面加载完成之前)。此外,这一章还将深入更高级的主题,例如事件冒泡、委托和命名空间。

第4章介绍通过jQuery实现动画的技术,我们将学会隐藏、显示和移动页面元素,获得爽心

悦目的效果。

第5章讲述如何通过命令改变页面。这一章讲述的是动态修改HTML文档结构及其内容的技术。

第6章讨论通过jQuery轻松访问服务器端功能的各种方法，而且不用像过去那样笨拙地刷新页面。

接下来3章（第7章、第8章、第9章）主要以实例为主，即在前几章内容的基础上，创建常见问题的稳健jQuery解决方案。

第7章讲述排序、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，将设计一个具有自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章介绍如何在显示页面元素时增强它们的美感和实用性。其中，动态显示和隐藏信息的方式既可以是自动的，也可以是用户控制的。

第10章和第11章的主题是jQuery库的第三方扩展，将向读者展示扩展这个库的各种方式。

第10章介绍Form插件和官方用户界面插件集合jQuery UI。同时，还将介绍到哪里寻找其他流行的jQuery插件并了解它们的功能。

第11章将讨论如何利用jQuery强大的扩展能力，从头开发自己的插件。不仅包括创建自己的实用函数，还有添加jQuery对象方法、添加自定义选择符表达式，等等。

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将帮助读者理解闭包——什么是闭包，怎么利用闭包。

附录D提供了jQuery的简明参考，包括所有方法和选择符表达式。在实际开发中，明确自己目标的情况下，通过这个简单明了的附录，能够方便快捷地找到正确的方法和选择符。

阅读本书要求

要在学习本书过程中，同步编写和运行书中的示例，需要有：

- 一个简单的文本编辑器；
- 一个浏览器，如Mozilla Firefox、Apple Safari或Microsoft Internet Explorer；
- jQuery源文件（1.3.1或更高版本），下载地址为<http://jquery.com/>。

此外，要运行第6章中的AJAX示例，还需要配置支持PHP的服务器。

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JavaScript编程知识和HTML及CSS基础知识，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。

正文中提到的代码如下所示：“通过使用`include`指令可以包含其他上下文”。

代码段版式如下所示：

```
<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>
```

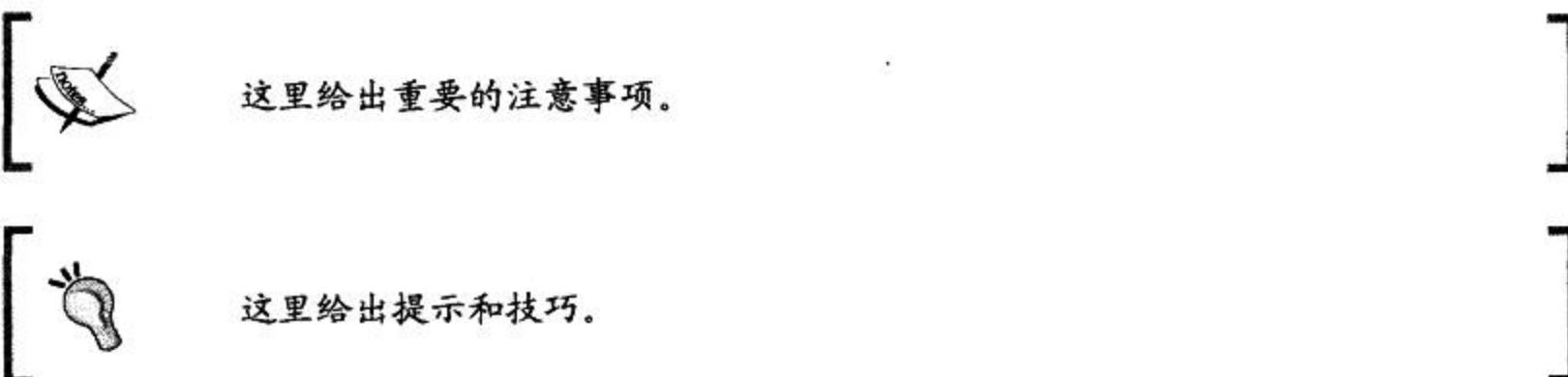
当需要读者特别注意代码块中的某一部分时，相关的代码行或项将以粗体印刷：

```
\$(document).ready(function() {
  \$('a[href^=mailto:]').addClass('mailto');
  \$('a[href$=.pdf]').addClass('pdflink');
  \$('a[href^=http][href*=henry]')
  .addClass('henrylink');
});
```

命令行输入和输出将按照如下样式印刷：

```
outerFn():
Outer function
Inner function
```

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），正文中会表示为：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。



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feedback@packtpub.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/files/code/6705_Code.zip，可以直接下载本书示例代码。^①

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questions@packtpub.com，我们会尽力解决。

^① 本书示例代码也可从图灵网站（www.turingbook.com）本书主页上免费注册下载。——编者注



书号：978-7-115-21166-8

精通Dojo



书号：978-7-115-18915-8

JavaScript实战



书号：978-7-115-20071-6

JavaScript袖珍速查手册

- 简明扼要，易于查询
- 涵盖130多种JavaScript典型编程任务
- 丰富精彩的代码，可立即用于实战

内容简介

JavaScript 是 Web 开发的必备技术之一。本书从 JavaScript 开发中总结提炼出 130 多种典型的任务，主要涵盖了 JavaScript 基础知识、常用语句、图像与动画、CSS、DOM、Ajax 和 Web 服务等内容，给出了精炼的代码，并提供简明的说明。本书篇幅短小，内容简洁实用，易于查询，可以使你的开发工作如虎添翼。

本书适合各层次 Web 开发人员参考和使用。

媒体评论

“最佳 JavaScript 袖珍手册！代码和阐释都极为精彩。”

——Amazon 读者评论



书号：978-7-115-20490-5

JavaScript基础教程 (第7版)

- 经典JavaScript入门书，涵盖Ajax
- 透彻讲解Web开发相关技术
- 让你体验轻松实用的学习方式

内容简介

在主流计算平台全面转向 Web 的今天，JavaScript 理所当然地成为广大开发人员必须熟练掌握的一项基本技术。

本书是已被奉为经典的 JavaScript 入门书，以易学便查、图文并茂、循序渐进和善于用常见任务讲解语言知识而著称，讲述 JavaScript 编程的必知必会知识同时，兼顾了 DOM、XML、Ajax 等重要的相关技术内容。多年来，本书不断重印改版，原版累计销售已经超过 150 000 册。第 6 版中文版出版后也多次重印，广受国内读者好评。

第 7 版增加了更多 Ajax 设计示例和现代编程技巧，以 YUI 为例讲述了如何高效运用 JavaScript 库，并在主流浏览器的最新版本中测试了全书示例。通过本书，你可以轻松而迅速地掌握 JavaScript Web 开发的基本技能，并掌握 Web 开发的最佳实践，领悟其中真谛。

媒体评论

如果你要学习 JavaScript，本书绝对不容错过。

——JavaScript.about.com

一本伟大的 JavaScript 入门书！适合初级程序员和 Web 设计师。

——Amazon.com



书号：978-7-115-19622-4

深入浅出Ext JS

- 涵盖Ext JS 3.0新特性
- Ext JS专家力作，示例丰富，完美结合理论和实践
- Ajax中国、Dojo中国、开源人、一起Ext 4大网站联袂推荐

内容简介

以用户为中心的时代，应用的界面外观正在变得越来越重要。然而，很多程序员都缺乏美术功底，要开发出界面美观的应用实属不易。Ext JS 的出现，为广大程序员们解决了这一难题。它有丰富多彩的界面和强大的功能，是开发具有炫丽外观的 RIA 应用的最佳选择。

本书是国内 Ext JS 先驱者的智慧结晶。内容翔实、示例丰富，包含大量示例代码，可操作性极强。不仅全面地阐述了 Ext JS 的基础知识，而且深入浅出地讲解了 Ext JS 开发中的高级技巧，同时还给出了大量专家级的建议。通过学习本书，读者将感受到 Ext JS 的无穷魅力和 Ajax 技术带来的完美体验。

本书适合所有有一定 CSS 和 HTML 基础的开发者阅读。

媒体评论

“本书是非常急缺的全面讲解 Ext JS 的图书，弥补了 Ajax 社区的一大空白。”

——Ajax 中国 (www.okajax.com)，国内权威的 Ajax 专业门户

“要想全面了解和掌握 Ext JS，强烈推荐阅读《深入浅出 Ext JS》。从各方面来看，这本书都非常优秀！”

——Dojo 中国 (www.dojochina.com)，Dojo 中文官方网站

本书样章发布后好评如潮，经仔细研读，果然名副其实。内容翔实，示例丰富，叙述风趣幽默，可操作性极强……Ext JS 开发者必备！

——一起 Ext (www.17ext.com)，著名 Ext JS 中文社区

“本书即将出版的消息在开源社区内受到了广泛关注，我们翘首以盼，期待这一场盛宴！”

——开源人 (<http://www.vifir.com/>)，专注于为开源人士服务的互动平台



书号：978-7-115-19599-9

jQuery 实战

- jQuery之父强烈推荐
- Amazon五星盛誉图书
- 深入剖析jQuery内部工作机制
- 提升Web开发效率的捷径

内容简介

jQuery 是目前应用最广泛的优秀开源 JavaScript/Ajax 框架之一，已经成为微软 ASP.NET、Visual Studio 和诺基亚 Web Run Time 等主流开发平台的组成部分。借助 jQuery 的魔力，数十行 JavaScript 代码可以被神奇地压缩成区区几行，多少 Web 开发人员在那一瞬间深深地迷恋上了这个方便快捷、功能完备的利器。

本书是带领你自如驾驭 jQuery 的导航者，替你肃清学习和编程路上的各种障碍。在这里，你不仅能深入学习 jQuery 的各种特性和技巧，还能领略到 jQuery 的内部工作机制和插件体系结构以及背后的各種策略和理论，学会怎样与其他工具和框架交互。有了 jQuery 和这本书，你不需要再费心劳力地纠缠于各种高深复杂的 JavaScript 技巧，只使用层叠样式表、XHTML 以及普通的 JavaScript 知识，就能直接操作页面元素，实现更快速更高效的 Web 开发。

媒体评论

“本书令我惊喜……这是一部深入透彻的著作，jQuery 项目本身都从中获益匪浅。相信它将成为你学习和使用 jQuery 的理想资源。”

——jQuery 之父 John Resig，《精通 JavaScript》一书作者
“本书堪与 jQuery 本身相媲美——快速、实用、高效。”

——Eric Pascarello，《Ajax 实战》一书作者

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 jQuery项目历史	3	
1.4 第一个jQuery驱动的页面	4	
1.4.1 下载jQuery	4	
1.4.2 建立HTML文档	4	
1.4.3 编写jQuery代码	7	
1.4.4 最终结果	9	
1.5 小结	9	
第2章 选择符	10	
2.1 DOM	10	
2.2 工厂函数\$()	11	
2.3 CSS选择符	11	
2.4 属性选择符	14	
2.5 自定义选择符	15	
2.5.1 每隔一行为表格添加样式	16	
2.5.2 基于表单的选择符	18	
2.6 DOM遍历方法	18	
2.6.1 为特定单元格添加样式	19	
2.6.2 连缀	20	
2.7 访问DOM元素	21	
2.8 小结	21	
第3章 事件	22	
3.1 在页面加载后执行任务	22	
3.1.1 代码执行的时机选择	22	
3.1.2 基于一个页面执行多个脚本	23	
3.1.3 缩短代码的简写方式	24	
3.1.4 与其他库共存	24	
3.2 简单的事件	25	
3.2.1 简单的样式转换器	25	
3.2.2 简写的事件	31	
3.3 复合事件	32	
3.3.1 显示和隐藏高级特性	32	
3.3.2 突出显示可单击的项	34	
3.4 事件的旅程	35	
3.5 通过事件对象改变事件的旅程	37	
3.5.1 事件目标	38	
3.5.2 停止事件传播	39	
3.5.3 默认操作	39	
3.5.4 事件委托	40	
3.6 移除事件处理程序	42	
3.6.1 事件的命名空间	42	
3.6.2 重新绑定事件	43	
3.7 模仿用户操作	44	
3.8 小结	47	
第4章 效果	49	
4.1 修改内联CSS	49	
4.2 基本的隐藏和显示	53	
4.3 效果和速度	54	
4.3.1 指定显示速度	55	
4.3.2 淡入和淡出	55	
4.4 复合效果	56	
4.5 创建自定义动画	57	
4.5.1 切换淡入淡出	58	
4.5.2 创建多个属性的动画	58	
4.6 并发与排队效果	61	
4.6.1 处理一组元素	61	
4.6.2 处理多组元素	63	

4.6.3 回调函数	65
4.6.4 简单概括	67
4.7 小结	67
第5章 DOM 操作.....	68
5.1 操作属性	68
5.1.1 非 class 属性	68
5.1.2 深入理解\$()工厂函数	70
5.2 插入新元素	72
5.3 移动元素	73
5.3.1 标注、编号和链接到上下文	76
5.3.2 插入脚注	78
5.4 包装元素	79
5.5 复制元素	80
5.5.1 连同事件一起复制	82
5.5.2 通过复制创建突出引用	82
5.5.3 通过 CSS 使突出引用偏离正文	82
5.5.4 回到代码中	83
5.5.5 修饰突出引用	85
5.6 DOM 操作方法的简单归纳	87
5.7 小结	87
第6章 AJAX.....	88
6.1 基于请求加载数据	88
6.1.1 追加 HTML	89
6.1.2 操作 JavaScript 对象	92
6.1.3 加载 XML 文档	98
6.2 选择数据格式	101
6.3 向服务器传递数据	102
6.3.1 执行 GET 请求	102
6.3.2 执行 POST 请求	105
6.3.3 序列化表单	106
6.4 关注请求	108
6.5 AJAX 和事件	111
6.6 安全限制	111
6.7 其他工具	114
6.7.1 低级 AJAX 方法	114
6.7.2 修改默认选项	114
6.7.3 部分加载 HTML 页面	115
6.8 小结	117
第7章 表格操作.....	118
7.1 排序和分页	119
7.1.1 服务器端排序	119
7.1.2 JavaScript 排序	120
7.1.3 服务器端分页	134
7.1.4 JavaScript 分页	136
7.1.5 完成的代码	140
7.2 修改表格外观	142
7.2.1 突出显示行	142
7.2.2 工具提示条	149
7.2.3 折叠和扩展	153
7.2.4 筛选	155
7.2.5 完成的代码	159
7.3 小结	162
第8章 构建功能型表单.....	163
8.1 改进基本的表单	163
8.1.1 渐进增强表单样式	163
8.1.2 根据条件显示的字段	169
8.1.3 表单验证	171
8.1.4 复选框操作	178
8.1.5 完成的代码	180
8.2 提升紧凑的表单	183
8.2.1 字段的占位符文本	183
8.2.2 AJAX 自动完成	186
8.2.3 完成的代码	193
8.3 操作数字型表单数据	195
8.3.1 购物车表格结构	195
8.3.2 拒绝非数字输入	199
8.3.3 数字计算	199
8.3.4 删除商品	205
8.3.5 修改送货信息	208
8.3.6 完成的代码	211
8.4 小结	213
第9章 滑移和翻转.....	214
9.1 标题新闻翻转效果	214
9.1.1 设置页面	214
9.1.2 取得新闻源	216
9.1.3 设置翻转效果	219

9.1.4 标题新闻翻转函数	220	第 11 章 开发插件	273
9.1.5 悬停时暂停	222	11.1 添加新的全局函数	273
9.1.6 从不同的域中取得新闻源	224	11.1.1 添加多个函数	273
9.1.7 附加的内部渐变效果	226	11.1.2 关键所在	274
9.1.8 完成的代码	228	11.1.3 创建实用方法	275
9.2 图像传送带	229	11.2 添加 jQuery 对象方法	276
9.2.1 设置页面	230	11.2.1 对象方法的环境	276
9.2.2 通过单击滑移图像	233	11.2.2 方法连缀	278
9.2.3 放大图像	239	11.3 DOM 遍历方法	280
9.2.4 完成的代码	250	11.4 添加新的简写方法	283
9.3 小结	253	11.5 方法的参数	286
第 10 章 使用插件	254	11.5.1 简单参数	287
10.1 查找插件和帮助	254	11.5.2 参数映射	288
10.2 使用插件	254	11.5.3 默认参数值	289
10.3 Form 插件	255	11.5.4 回调函数	290
10.4 jQuery UI 插件库	257	11.5.5 可定制的默认值	291
10.4.1 效果	257	11.6 添加选择符表达式	292
10.4.2 交互组件	259	11.7 共享插件	295
10.4.3 部件	261	11.7.1 命名约定	295
10.4.4 jQuery UI ThemeRoller	263	11.7.2 别名 \$ 的使用	295
10.5 其他插件	264	11.7.3 方法接口	295
10.5.1 表单类	264	11.7.4 文档格式	296
10.5.2 表格类	266	11.8 小结	296
10.5.3 图像类	268	附录 A 在线资源	297
10.5.4 亮盒及模态对话框	269	附录 B 开发工具	303
10.5.5 图表类	270	附录 C JavaScript 闭包	307
10.5.6 事件类	272	附录 D 快速参考	318
10.6 小结	272		

第1章

jQuery入门



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。

下面先来看一看jQuery能为我们做什么。

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。

- **取得文档中的元素。**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。
- **改变文档的内容。**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。

- **响应用户的交互操作。**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中内置了一批淡入、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。
- **支持扩展。**为了避免特性蠕变（feature creep）^②，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。

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

② 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

口 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在其多数方法中采用了一种被称作连缀（chaining）^①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 jQuery 项目历史

本书内容涵盖了jQuery 1.3.x的功能和语法，这也本书写作时jQuery的最新版本。虽然这个项目发展至今，其遵循的承诺未曾改变，即为查找和操作页面中的元素，提供一种便捷手段，但某些语法细节和特性则发生了变化。本节有关jQuery项目历史的简单概述，仅涉及不同版本间最重要的变化。

- 公众开发阶段：2005年8月，John Resig最先提议改进Prototype的“Behaviour”库，随之而来的新框架于2006年1月14日正式以jQuery的名称发布。
- jQuery 1.0（2006年8月）：该库的第一个稳定版本，已经具有了对CSS选择符、事件处理和AJAX交互的稳健支持。
- jQuery 1.1（2007年1月）：这一版大幅简化了API。许多较少使用的方法被合并，减少了需要掌握和解释的方法数量。
- jQuery 1.1.3（2007年7月）：这次小版本变化包含了对jQuery选择符引擎执行速度的显著提升。从这个版本开始，jQuery的性能达到了Prototype、Mootools以及Dojo等同类JavaScript库的水平。
- jQuery 1.2（2007年9月）：这一版去掉了对XPath选择符的支持，原因是相对于CSS语法它已经变得多余了。这一版能够支持对效果的更灵活定制，而且借助新增的命名空间事件，也使插件开发更容易。
- jQuery UI（2007年9月）：这个新的插件套件是作为曾经流行但已过时的Interface插件的替代项目而发布的。jQuery UI中包含大量预定义好的部件（widget），以及一组用于构建高级元素（例如可拖放的界面元素）的工具。
- jQuery 1.2.6（2008年5月）：这一版主要是将Brandon Aaron开发的流行的Dimensions插件的功能移植到了核心库中。

^① 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

- **jQuery 1.3 (2009年1月)**: 这一版使用了全新的选择符引擎Sizzle，库的性能也因此有了极大提升。这一版正式支持事件委托特性。
- **jQuery 1.3.2 (2009年2月)**: 这次小版本升级进一步提升了库的性能，例如改进了`:visible/:hidden`选择符、`.height()/width()`方法的底层处理机制。另外，也支持查询的元素按文档顺序返回。



jQuery 以前版本的发行版说明可以在官方站点查到，网址为 http://docs.jquery.com/History_of_jQuery。

1.4 第一个jQuery驱动的页面

了解jQuery能够提供的丰富特性之后，我们可以来看一看这个库的实际应用了。

1.4.1 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第一手资源。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。而在正式发布的页面中，则可以使用压缩版。

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。

1.4.2 建立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，以及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。

```
<!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>
```

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;  
}  
.highlight {  
    font-style: italic;  
    border: 1px solid #888;  
    padding: 0.5em;  
    margin: 0.5em 0;  
    background-color: #ffc;  
}
```

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。



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。



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。

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。

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.

YKCOWREBBAJ
 sevol yhile eht dna .giltib sawT
 :ebaw eht ni elbmig dna eryg siD
 .sevogob eht erew yemim iA
 .abengluo ehter 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 1-1

1.4.3 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('highlight');
});
```

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`highlight`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

^① 即类为`.poem-stanza`的文档部分。——译者注

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，如果将这行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 highlightPoemStanzas() {
    $('.poem-stanza').addClass('highlight');
}
```

然后，需要修改HTML的`<body>`标签，将这个函数附加给事件：

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

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(highlightPoemStanzas);
```

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`highlightPoemStanzas()`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('highlight');
});
```

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特性，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。

1.4.4 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。

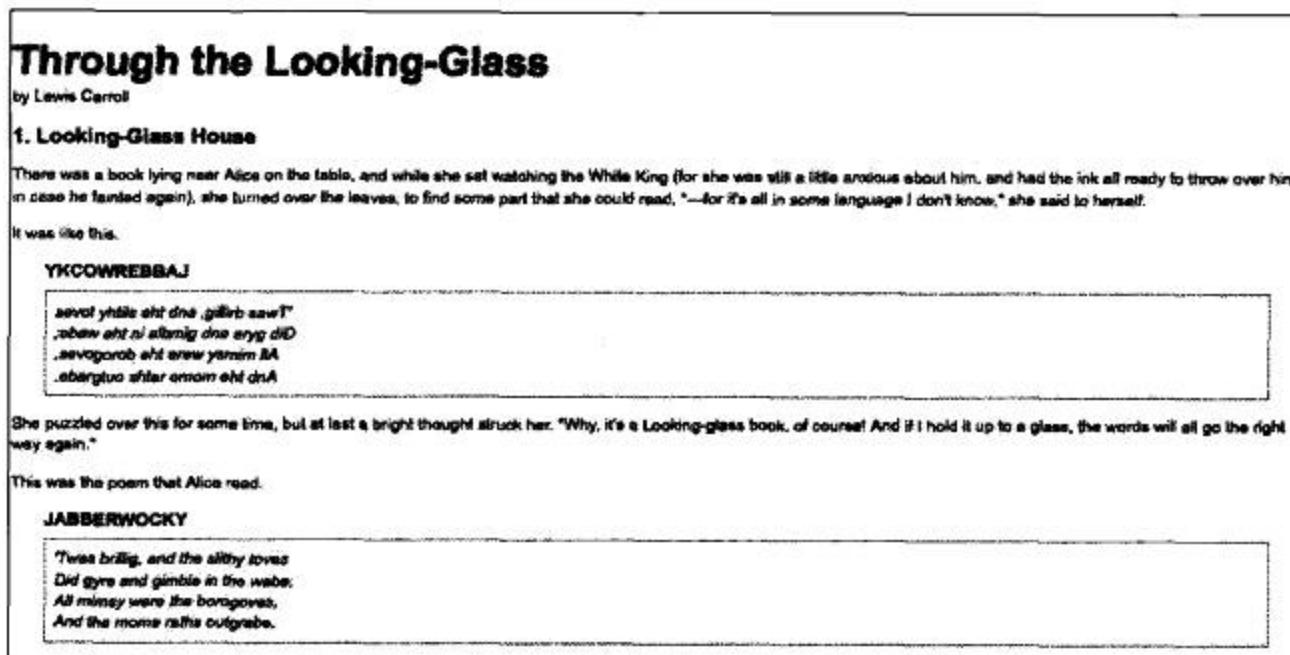


图 1-2

由于JavaScript插入了`emphasized`类，页面中的两节诗歌文本变成了斜体，并且被包含在方框中。这些样式来源于`alice.css`样式表，而样式生效则有赖于JavaScript代码插入的`highlight`类。

1.5 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。



jQuery利用了CSS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集合。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最强大的特性之一就是它能够简化在DOM中选择元素的任务。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（但并不是它唯一的子元素）。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`<p>`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始终都会被包装在一个jQuery对象中。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，无论使用哪种类型的选择符，都要从一个美元符号和一对圆括号开始：`$()`。所有能在样式表中使用的选择符，都能放到这个圆括号中的引号内。随后，我们就可以对匹配的元素集合应用jQuery方法。

让jQuery与其他JavaScript库有效地协同

 在jQuery中，美元符号\$只不过标识符jQuery的“别名”。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。

有3种基本的选择符：**标签名、ID和类**。这些选择符可以单独使用，也可以与其他选择符组合使用。表2-1展示了这3种基本的选择符。

表2-1 基本的选择符

选择符	CSS	jQuery	说明
标签名	P	<code>\$('p')</code>	取得文档中所有的段落
ID	#some-id	<code>\$('#some-id')</code>	取得文档中ID为some-id的一个元素
类	.some-class	<code>\$('.some-class')</code>	取得文档中类为some-class的所有元素

第1章曾经提到过，在将方法连缀到\$()工厂函数后面时，包装在jQuery对象中的元素会被自动、隐式地循环遍历。换句话说，这样就避免了使用for循环之类的显示迭代（这种迭代在DOM脚本编程中非常常见）。

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中的几乎所有选择符，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。

 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="/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 II</li>
      </ul>
    </li>
  </ul>
</li>
```

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。

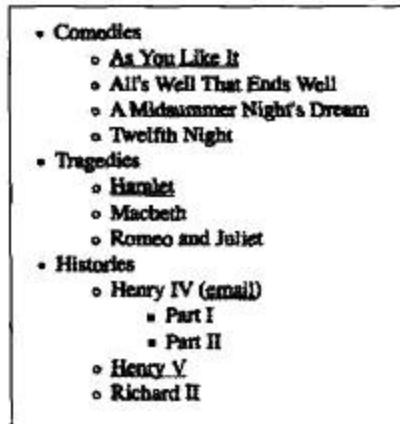


图 2-1

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()`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。

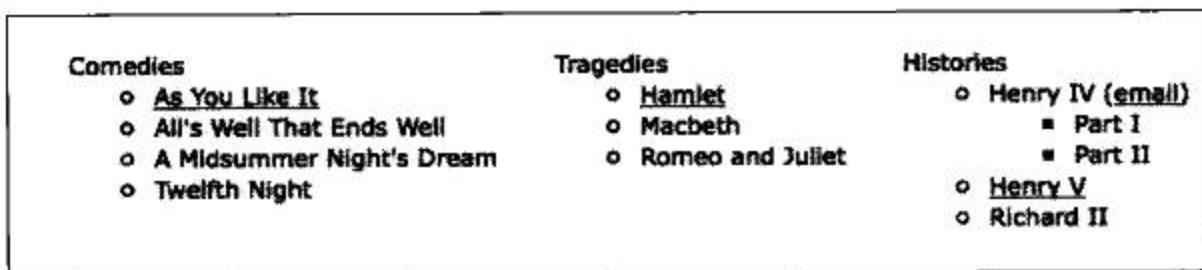


图 2-2

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中定义的浅灰的背景颜色。此时的嵌套列表如图2-3所示。

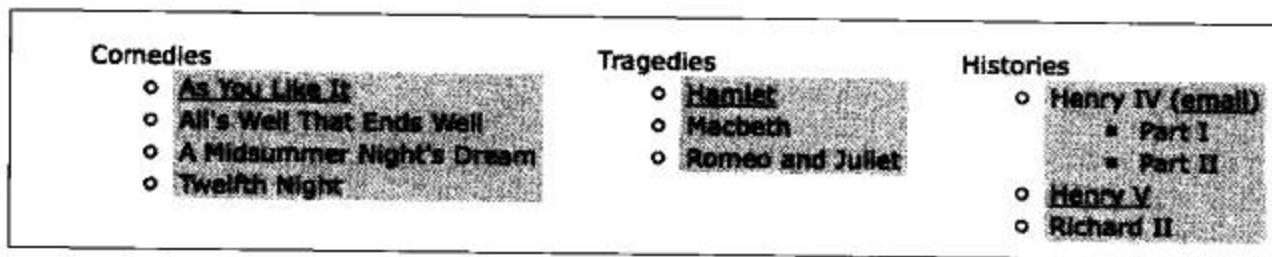


图 2-3

2.4 属性选择符

属性选择符是CSS选择符中特别有用的一类选择符。顾名思义，属性选择符通过HTML元素的属性选择元素，例如链接的title属性或图像的alt属性。例如，要选择带有alt属性的所有图像元素，可以使用以下代码：

```
$('img[alt]')
```

 在1.2版之前，jQuery使用XPath（XML Path Language，XML路径语言）语法实现其属性选择符，同时还包含了很多其他XPath选择符。虽然这些基本的XPath选择符已经从核心jQuery库中去掉了，但以插件形式仍然可以使用它们，有关信息请参见 <http://plugins.jquery.com/project/xpath/>。

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(*)表示位于字符串中任意位置的值，使用感叹号(!)表示相反的值。

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: #00c;  
}  
amailto {  
    background: url(images/mail.png) no-repeat right top;  
    padding-right: 18px;  
}  
a.pdflink {  
    background: url(images/pdf.png) no-repeat right top;  
    padding-right: 18px;  
}  
a.henrylink {  
    background-color: #fff;  
    padding: 2px;  
    border: 1px solid #000;  
}
```

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

要为所有电子邮件链接添加类，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');
});
```

要为所有指向PDF文件的链接添加类，需要使用美元符号（\$）而不是脱字符号（^）。这是因为我们要选择所有href属性以.pdf结尾的链接：

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

属性选择符也可以组合使用。例如，可以为href属性既以http开头且任意位置又包含henry的所有链接添加一个henrylink类：

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

在把这3个类应用到3种类型的链接之后，应该看到如图2-4所示的结果。

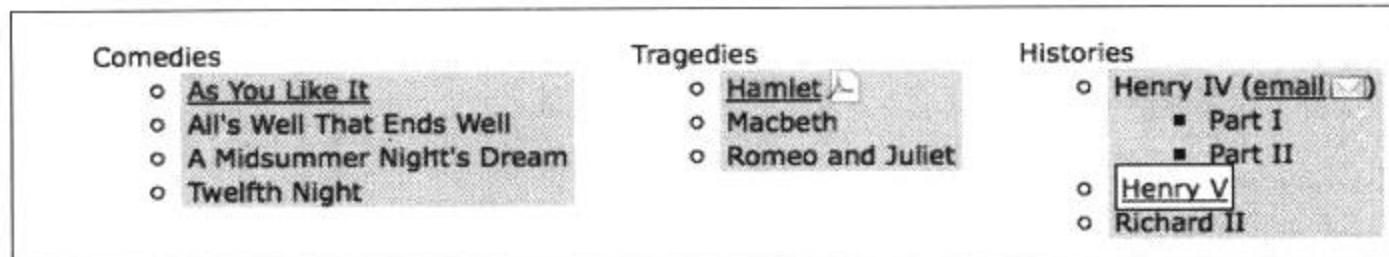


图 2-4

我们注意到，Hamlet链接右侧有一个PDF图标，email链接旁边有一个信封图标，而Henry V链接则带有白色背景和黑色边框。

2.5 自定义选择符

除了各种CSS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（在此，其实也可以使用`'div:first-child'`来代替）。

2.5.1 每隔一行为表格添加样式

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>
    <td>Comedy</td>
    <td></td>
  </tr>
  <tr>
    <td>All's Well that Ends Well</td>
    <td>Comedy</td>
    <td>1601</td>
  </tr>
  <tr>
    <td>Hamlet</td>
    <td>Tragedy</td>
    <td>1604</td>
  </tr>
  <tr>
    <td>Macbeth</td>
    <td>Tragedy</td>
    <td>1606</td>
  </tr>
  <tr>
    <td>Romeo and Juliet</td>
    <td>Tragedy</td>
    <td>1595</td>
  </tr>
  <tr>
    <td>Henry IV, Part I</td>
    <td>History</td>
    <td>1596</td>
  </tr>
  <tr>
    <td>Henry V</td>
    <td>History</td>
    <td>1599</td>
  </tr>
</table>
```

可以在样式表中为所有表格行添加一种样式，然后再为偶数行定义一个`alt`类。

```
tr {
```

```
background-color: #fff;  
}  
.alt {  
background-color: #ccc;  
}
```

最后编写jQuery代码，将这个类添加到表格中的偶数行：

2

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

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 2-6

必须注意, :contains()选择符区分大小写。换句话说, 使用不带大写“H”的\$('td:contains(henry)'), 不会选择任何单元格。

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行条纹和突出显示效果。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。

2.5.2 基于表单的选择符

在操作表单时, jQuery的自定义选择符也可以简化选择元素的任务。表2-2列出了其中一些选择符。

表2-2 选择符

选 择 符	匹 配
:text、:checkbox、 :radio、:image、 :submit、:reset、 :password、:file	type属性值为选择符名称(不包括冒号)的输入元素, 例如<input type="text">
:input	输入字段、文本区、选择列表和按钮元素
:button	按钮元素或type属性值为button的输入元素
:enabled	启用的表单元素
:disabled	禁用的表单元素
:checked	勾选的单选按钮或复选框
:selected	选择的选项元素

与其他选择符类似, 组合使用表单选择符可以更有针对性。例如, 使用\$('radio:checked')可以选择所有选中的单选按钮(而不是复选框), 而使用\$('password, :text:disabled')则可以选择所有密码输入字段和禁用的文本输入字段。可见, 即便是使用自定义选择符, 也可以按照基本的CSS语法来定义匹配的元素列表。

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的结果一样。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大的, 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('alt');`，可以通过`.filter()`方法重写成下面这样：

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

而且，这两种取得元素的方式在很大程度上可以互为补充。同样，`.filter()`的功能也十分强大，因为它可以接受函数参数。通过传入的函数，可以执行复杂的测试，以决定相应元素是否应该保留在匹配的集合中。例如，假设我们要为所有外部链接添加一个类。jQuery中没有针对这种需求的选择符。如果没有筛选函数，就必须显式地遍历每个元素，对它们单独进行测试。但是，有了下面的筛选函数，就仍然可以利用jQuery的隐式迭代能力，保持代码的简洁：

```
$('a').filter(function() {
    return this.hostname && this.hostname != location.hostname;
}).addClass('external');
```

第2行代码可以筛选出符合下面两个条件的`<a>`元素：

(1) 必须包含一个带有域名(`this.hostname`)的`href`属性。这个测试可以排除`mailto`及类似链接。

(2) 链接指向的域名(还是`this.hostname`)必须不等于(`!=`)页面当前所在域的名称(`location.hostname`)。

更准确地说，`.filter()`方法会迭代所有匹配的元素，基于每个元素测试函数的返回值。如果函数返回`false`，从匹配集合中删除相应元素；如果返回`true`，则保留相应元素。

下面，我们再通过前面添加了条纹效果的表格，来演示一些遍历方法的其他用途。

2.6.1 为特定单元格添加样式

此前，我们已经为所有包含文本Henry的单元格添加了`highlight`类。如果想改为给每个包含Henry的单元格的下一个单元格添加样式，可以将已经编写好的选择符作为起点，然后简单地连缀一个`.next()`方法即可：

```
$(document).ready(function() {
    $('td:contains(Henry)').next().addClass('highlight');
});
```

表格现在的效果如图2-7所示。

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 2-7

.next()方法只选择下一个最接近的同辈元素。要想突出显示Henry所在单元格后面的全部单元格，可以使用.nextAll()方法。

```
$(document).ready(function() {
    $('td:contains(Henry)').nextAll().addClass('highlight');
});
```

 有读者可能已经猜到了，.next()和.nextAll()方法分别有一个对应方法，即.prev()和.prevAll()。此外，.siblings()能够选择处于相同DOM层次的所有其他元素，无论这些元素处于当前元素之前还是之后。

要在这些单元格中再包含原来的单元格（即包含Henry的那个单元格），可以添加.andSelf()方法：

```
$(document).ready(function() {
    $('td:contains(Henry)').nextAll().andSelf().addClass('highlight');
});
```

事实上，要选择同一组元素，可以采用的选择符和遍历方法的组合很多。例如，下面就是选择所有包含Henry的单元格所在行的另一种方式：

```
$(document).ready(function() {
    $('td:contains(Henry)').parent().children().addClass('highlight');
});
```

这种组合方式没有遍历同辈元素，而是通过.parent()方法在DOM中上溯一层到达|，然后再通过.children()选择该行的所有单元格。
| |

2.6.2 连缀

刚刚介绍的遍历方法组合展示了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。jQuery的这种连缀能力不仅有助于保持代码简洁，而且在替代组合重新指定选择符时，还有助于提升脚本性能。

在使用连缀时，为照顾到代码的可读性，还可以把一行代码分散到几行来写。例如，一组连缀的方法可以写成1行（限于版面，印成两行）：

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

也可以写成7行：

```
($('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
```

不可否认，这个例子中展示的迂回曲折的DOM遍历过程几近荒谬。我们当然不建议读者使用如此复杂的连缀方式，因为还有更简单、更直接的方法。这个例子的用意只是演示一下连缀为我们带来的极大灵活性。

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——通过DOM元素的属性。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：

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

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的包装并直接露出节点列表，而方括号中的索引（这里的0）则相当于从中取出了原本的DOM元素。

2.8 小结

通过本章介绍的技术，读者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`，或高级的CSS选择符`:nth-child()`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。

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技术和本书后面讨论的一些技巧派上用场。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而言都是可以操作的，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。



为了保证JavaScript代码执行以前页面已经应用了样式，最好是在`<head>`元素中把`<link rel="stylesheet">`标签放在`<script>`标签前面。

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。



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（或者是使用jQuery为`load`事件设置处理程序）。这两种机制能够和平共存。

3

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。



这里在将函数指定为处理程序时，省略了后面的圆括号，只使用了函数名。如果带着圆括号，函数会被立即调用；没有圆括号，函数名就只是函数的标识符，可以用于在将来再调用函数。

在只有一个函数的情况下，这样做没有什么问题。但是，假设我们又定义了第二个函数：

```
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^①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。而且，这些函数会按照注册它们的顺序依次执行^②。



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^③，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。像 `document.addEventListener()` 和 `document.attachEvent()` 这种特定于浏览器的方法，也能够提供类似功能。但是，jQuery 则可以让我们不必考虑浏览器不一致性而完成这一任务。

3.1.3 缩短代码的简写方式

前面提到的 `$(document).ready()` 结构，实际上是在基于 `document` 这个 DOM 元素构建而成的 jQuery 对象上，调用了 `.ready()` 方法。`$()` 工厂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，但本书作者推荐使用较长的形式，因为较长的形式能够更清楚地表明代码在做什么。

3.1.4 与其他库共存

在某些情况下，可能有必要在同一个页面中使用多个 JavaScript 库。由于很多库都使用 `$` 标识符（因为它简短方便），因此就需要一种方式来避免名称冲突。

① 即每次调用 `$(document).ready()` 方法。——译者注

② 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

③ 关于这个 `addLoadEvent()` 方法，读者可以参考 <http://simonwillison.net/2004/May/26/addLoadEvent/>。——译者注

为解决这个问题，jQuery提供了一个名叫`.noConflict()`的方法，调用该方法可以把对`$`标识符的控制权交还给其他库。使用`.noConflict()`方法的一般模式如下：

```
<script src="prototype.js" type="text/javascript"></script>
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
    jQuery.noConflict();
</script>
<script src="myscript.js" type="text/javascript"></script>
```

首先，包含jQuery之外的库（这里是Prototype）。然后，包含jQuery库，取得对`$`的使用权。接着，调用`.noConflict()`方法让出`$`，以便将控制权交还给最先包含的库（Prototype）。这样，就可以在自定义脚本中使用两个库了——但是，在需要使用jQuery方法时，必须记住要用jQuery而不是`$`来调用。

在这种情况下，还有一个在`.ready()`方法中使用`$`的技巧。我们传递给它的回调函数可以接收一个参数——jQuery对象本身。利用这个参数，可以重新命名jQuery为`$`，而不必担心造成冲突：

```
jQuery(document).ready(function($) {
    // 在这里，可以正常使用$!
});
```

或者，也可以使用刚刚介绍的简写语法：

```
jQuery(function($) {
    // 使用$的代码
});
```

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。

^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

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-default">
    Default
  </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。

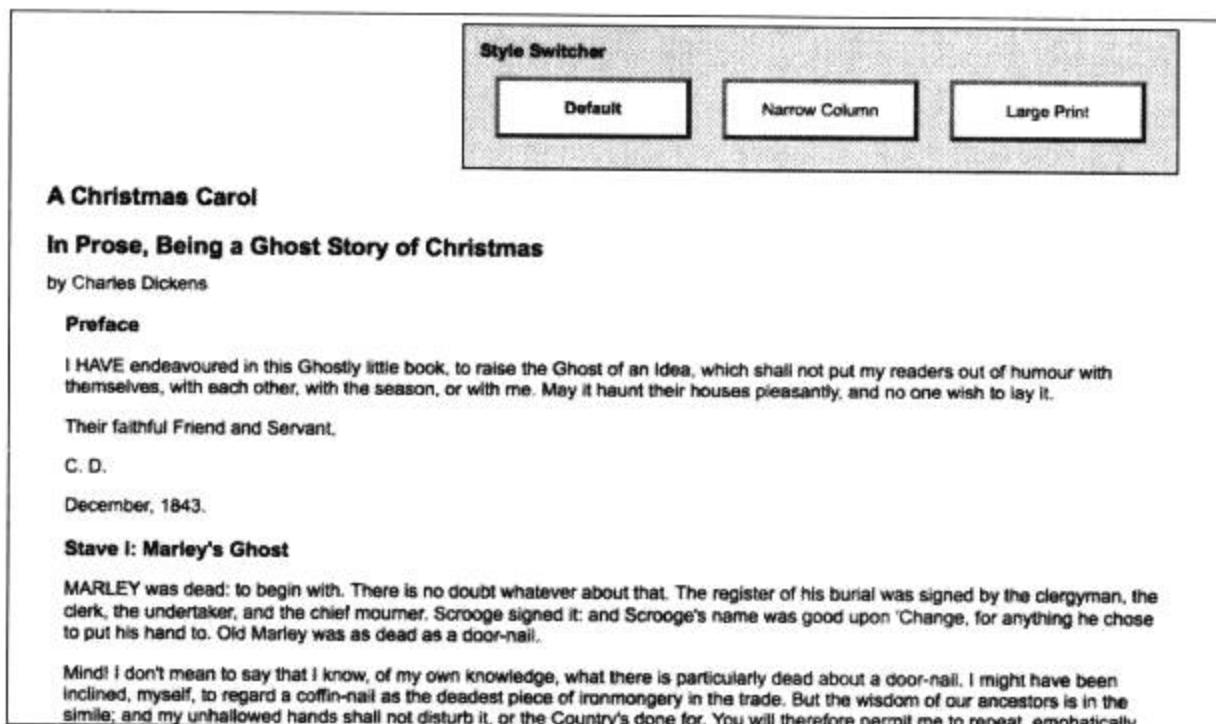


图 3-1

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。按照第2章介绍的知识，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（而不是像我们到目前为止看到的那样在页面加载后执行）。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。

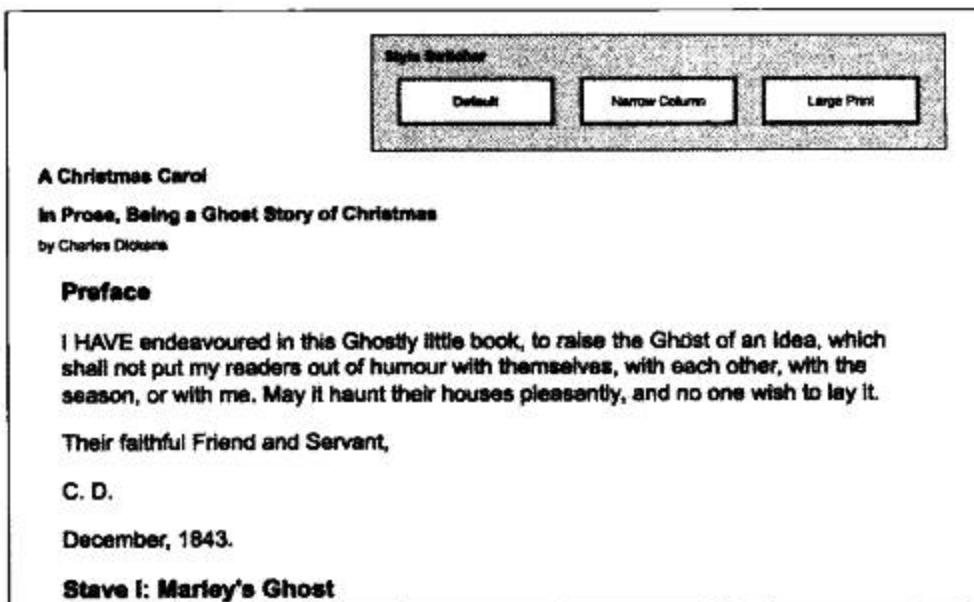


图 3-2

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（Default和Narrow），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-default').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。

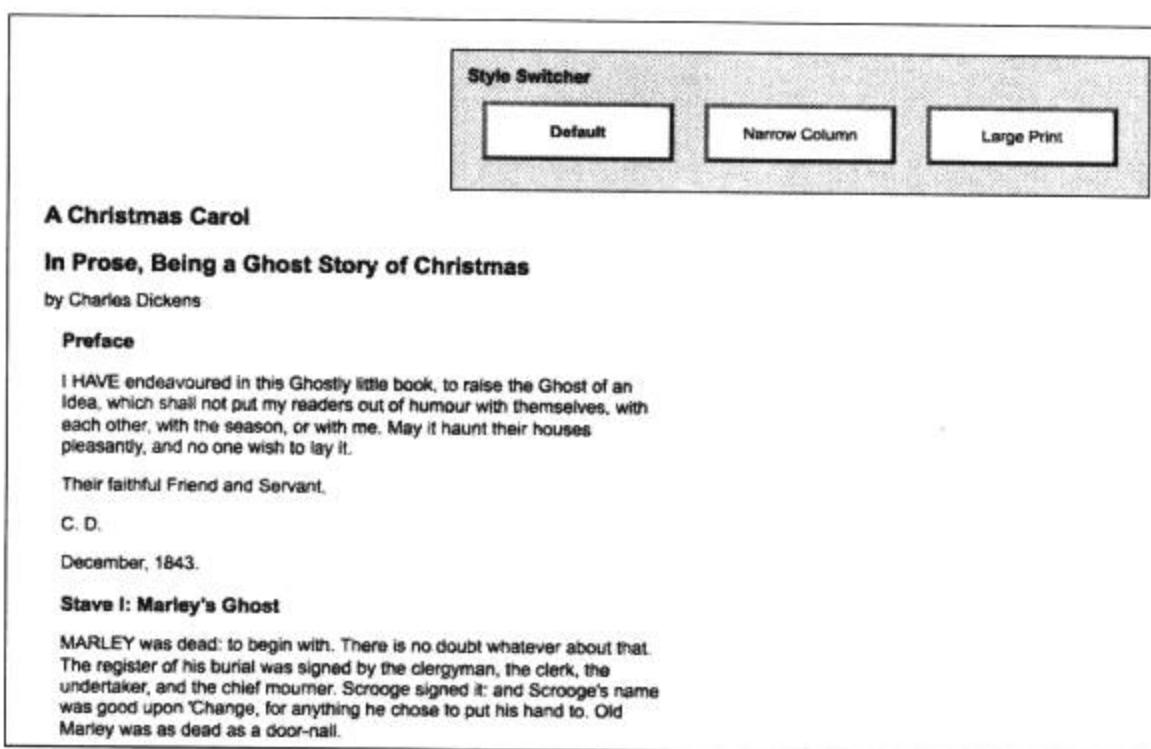


图 3-3

单击Default按钮可以从<body>标签中同时移除两个类，让页面恢复为初始状态。

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。

^① 即允许向\$()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

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-default').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');
  });
  $('#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。

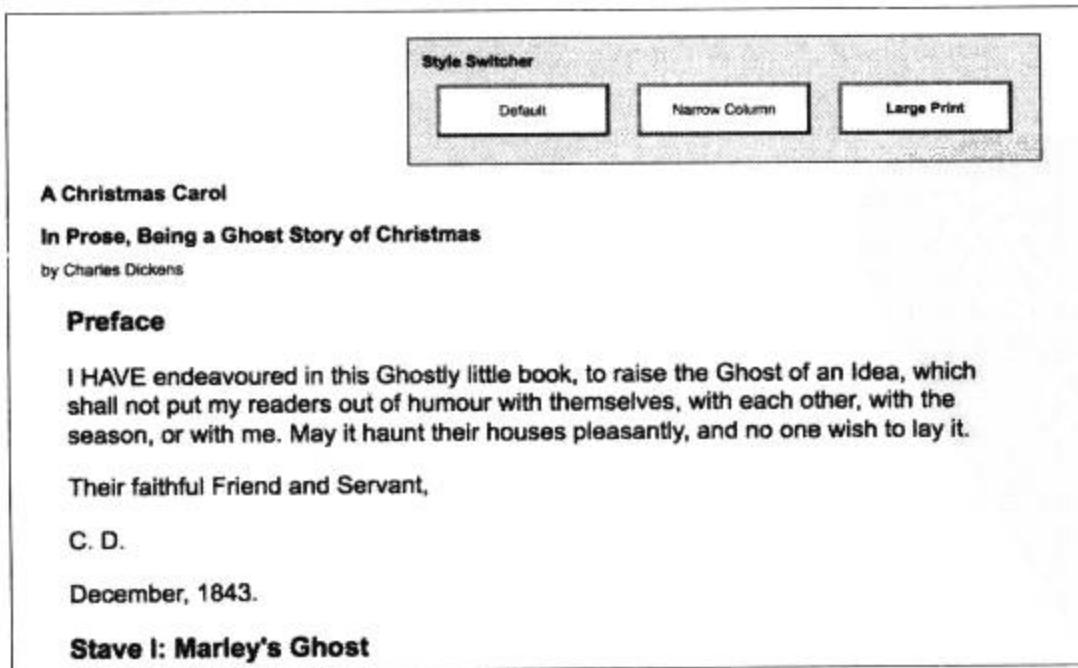


图 3-4

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

```
$(document).ready(function() {
    $('#switcher-default').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

我们刚才的代码优化实际上是在做重构——修改已有代码，以更加高效和简洁的方式实现相同任务。为寻找进一步重构的机会，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-default').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');
    });
});
```

注意，为了适应更通用的移除类的操作，我们对操作顺序作了小小的调整——先执行`.remove-`

`Class()`, 以便它不会撤销几乎同时执行的`.addClass()`。



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。

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。

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');
  });
});
});
```

jQuery为标准的DOM事件都提供了相应的简写事件方法：

- | | |
|-----------------------------------|------------------------------------|
| <input type="checkbox"/> blur | <input type="checkbox"/> mousedown |
| <input type="checkbox"/> change | <input type="checkbox"/> mousemove |
| <input type="checkbox"/> click | <input type="checkbox"/> mouseout |
| <input type="checkbox"/> dblclick | <input type="checkbox"/> mouseover |
| <input type="checkbox"/> error | <input type="checkbox"/> mouseup |
| <input type="checkbox"/> focus | <input type="checkbox"/> resize |
| <input type="checkbox"/> keydown | <input type="checkbox"/> scroll |
| <input type="checkbox"/> keypress | <input type="checkbox"/> select |
| <input type="checkbox"/> keyup | <input type="checkbox"/> submit |
| <input type="checkbox"/> load | <input type="checkbox"/> unload |

这些简写的方法能够把一个事件处理程序绑定到同名事件上面。

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。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

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

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方法（根据不同的参数类型区分），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。第一次在元素上单击会调用第一个函数，第二次单击则会触发第二个函数，以此类推。当两个函数都被调用一次后，循环又会从第一个函数开始。有了.toggle()方法，实现我们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');
    });
});
```

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

而第二次单击则又恢复了它们的可见性，如图3-6所示。

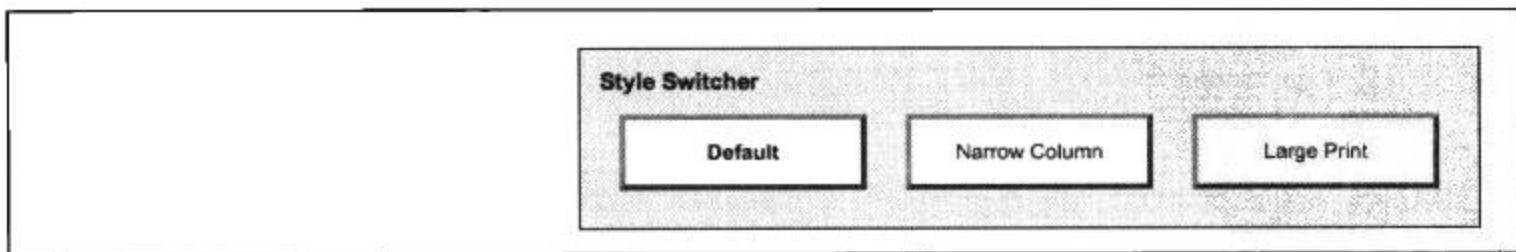


图 3-6

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或更多种不同的操作而言，`.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了一些提示——按钮（即`<div>`元素）实际上都是活动的，随时等待用户操作。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指针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^③，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，通过JavaScript来改变元素的样式——事实上是可以执行任意操作。

同`.toggle()`方法一样，`.hover()`方法也接受两个函数参数。但在这里，第一个函数会在鼠标指针进入被选择的元素时执行，而第二个函数会在鼠标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，当鼠标指针悬停在任何

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

③ 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

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。

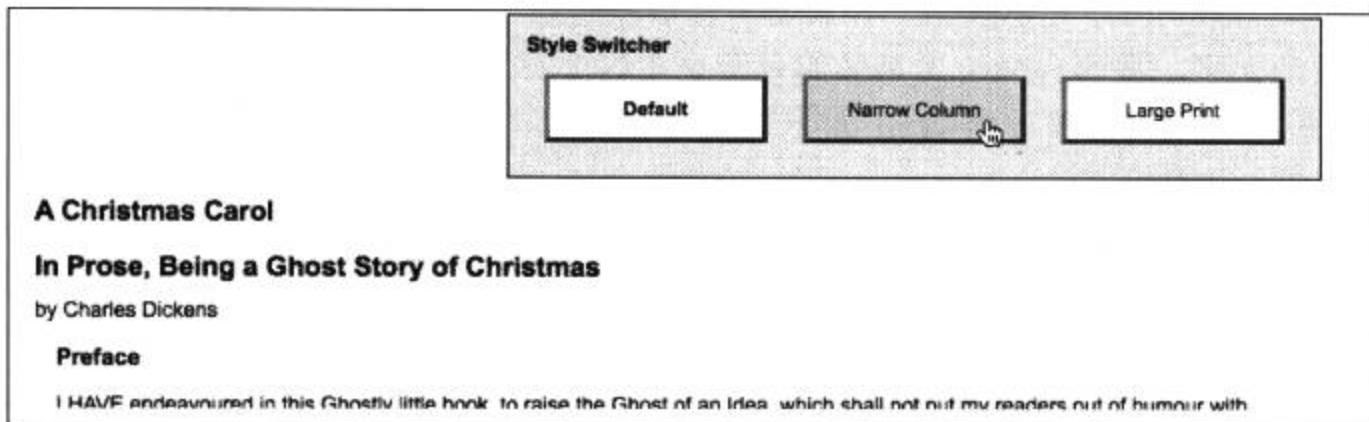


图 3-7

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。

3.4 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。

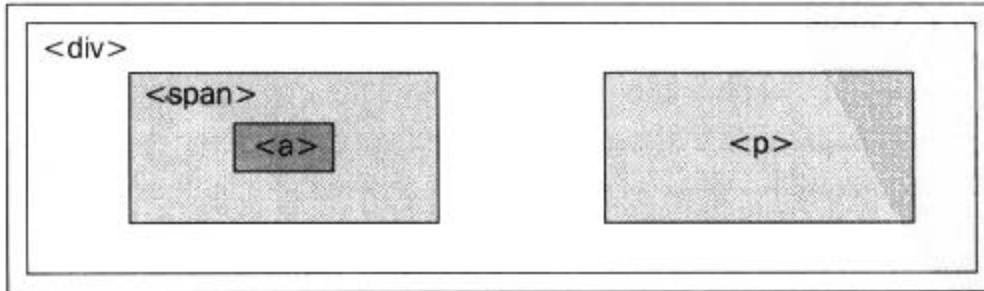


图 3-8

从逻辑上看，任何事件都可能会有多个元素负责响应。举例来说，如果单击了页面中的链接元素，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

于用户鼠标指针之下。而< p >元素则与这次交互操作无关。

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< span >，最后是< a >，如图3-9所示。

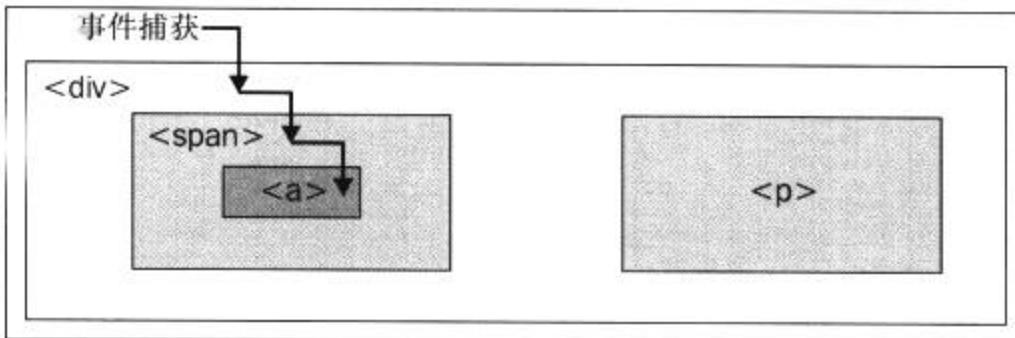


图 3-9

 从技术上说，在浏览器对事件捕获的实现中，每个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< span >和< div >，如图3-10所示。

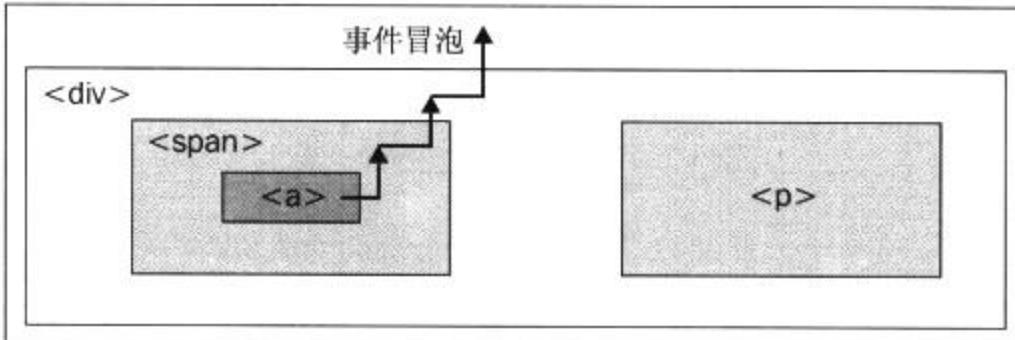


图 3-10

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，事件要从一般元素到具体元素逐层捕获，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个阶段。

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

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

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addEventListerner()中将最后一个参数设置为true。——译者注

段注册事件处理程序^①。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。

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

添加了一个mouseout事件处理程序。当用户的鼠标指针退出这个

时，会按照预期运行mouseout处理程序。因为这个过程发生在顶层元素上，所以其他元素不会取得这个事件。但是，当指针从元素上退出时，元素也会取得一个mouseout事件。然后，这个事件会向上冒泡到和

，从而触发上述的事件处理程序。这种冒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，这可能会意味着突出显示会过早地消失^②。

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。



如果只想在鼠标指针进入或离开某个元素时执行操作，而不是响应这两个事件，则可以绑定jQuery的mouseenter和mouseleave事件，这样也可以避免冒泡的问题。不过，由于此类事件通常都需要配对使用，因此多数时候还是选择.hover()更合适。

刚才介绍的mouseout的问题说明了限制事件作用域的必要性。虽然.hover()可以处理这种特殊情况，但在其他情况下，我们可能还需要从空间（阻止事件发送到某些元素）和时间（阻止事件在某个时间段发送）上限制某个事件。

3.5 通过事件对象改变事件的旅程

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

元素：

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

① 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

② 这里所谓的突出显示过早地消失，是假设在表现按钮的

元素中也包含一个元素。这样，当鼠标指针离开但尚未离开

（按钮）时，由于事件冒泡，

会继之后也取得mouseout事件，因而会导致按钮上的突出显示提前消失。——译者注

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

```

$( '#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。但同时也造成了一个问题，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，即事件首先被按钮处理，然后又沿着DOM树向上传递，直至到达`<div id="switcher">`激活事件处理程序并隐藏按钮。

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被调用的事件处理程序。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：

```

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

```

3.5.1 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（`<h3>`）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。

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

3.5.2 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^①。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) {
    $('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。

3.5.3 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^②。

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

① 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

② 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

上同时调用`.stopPropagation()`和`.preventDefault()`的一种简写方式。

 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。

3.5.4 事件委托

事件冒泡并不总是带来问题，也可以利用它为我们带来好处。事件委托就是利用冒泡的一项高级技术。通过事件委托，可以借助一个元素上的事件处理程序完成很多工作。

 jQuery 1.3 引入了一对新方法——`.live()`和`.die()`。这两个方法执行的任务与`.bind()`和`.unbind()`相同，但它们在后台使用事件委托为我们带了本节后面介绍的好处。这两个方法的文档可以在<http://docs.jquery.com/Events/live>查到。

在我前面的例子中，只有3个`<div class="button">`元素注册了单击处理程序。假如我们想为更多元素注册处理程序怎么办？这种情况比我们想象的更常见。例如，有一个显示信息的大型表格，每一行都有一项需要注册单击处理程序。虽然不难通过隐式迭代来指定所有单击处理程序，但性能可能会很成问题，因为循环是由jQuery在内部完成的，而且要维护所有处理程序也需要占用很多内存。

为解决这个问题，可以只在DOM中的一个祖先元素上指定一个单击处理程序。由于事件会冒泡，未遭拦截的单击事件最终会到达这个祖先元素，而我们可以在此时再作出相应处理。

下面我们就以样式转换器为例（尽管其中的按钮数量还不至于使用这种方法），说明如何使用这种技术。通过前面的学习我们知道，当发生单击事件时，可以使用`event.target`属性检查鼠标指针下方是什么元素。

```
$document.ready(function() {
  $('#switcher').click(function(event) {
    if ($(event.target).is('.button')) {
      $('body').removeClass();
      if (event.target.id == 'switcher-narrow') {
        $('body').addClass('narrow');
      }
      else if (event.target.id == 'switcher-large') {
        $('body').addClass('large');
      }
      $('#switcher .button').removeClass('selected');
      $(event.target).addClass('selected');
      event.stopPropagation();
    }
  });
});
```

这里使用了一个新方法，即`.is()`。这个方法接收一个选择符表达式（第2章介绍过），然后用选择符来测试当前的jQuery对象。如果集合中至少有一个元素与选择符匹配，`.is()`返回`true`。在这个例子中，`$(event.target).is('.button')`测试被单击的元素是否有一个`button`类。如果是，则继续执行以前编写的那些代码——但有一个明显不同，即此时的关键字`this`引用的是`<div id="switcher">`。换句话说，如果现在需要访问被单击的按钮，每次都必须通过`event.target`来引用。



3 要测试元素是否包含某个类，也可以使用另一个简写方法`.hasClass()`。不过，`.is()`方法则更灵活一些，它可以测试任何选择符表达式。

然而，以上代码还有一个不期而至的连带效果。当按钮被单击时，转换器会折叠起来，就像使用`.stopPropagation()`之前看到的效果一样。用于切换转换器可见性的处理程序，现在被绑定到了按钮上面。因此，阻止事件冒泡并不会影响切换发生。要解决这个问题，可以去掉对`.stopPropagation()`的调用，然后添加另一个`.is()`测试：

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

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

虽然这个例子的代码显得稍微复杂了一点，但随着带有事件处理程序的元素数量增多，使用事件委托终究还是正确的技术。



读者在本章后面可以看到，事件委托在另外一些情况下也很有用，例如通过DOM操作方法添加新元素（第5章）或在执行AJAX请求（第6章）时。

3.6 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，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，即当Narrow Column或Large Print按钮被选中时，单击样式转换器的背景区域不应该引发任何操作。为此，可以在非默认样式转换按钮被单击时，调用`.unbind()`方法移除折叠处理程序。

```
$(document).ready(function() {
    $('#switcher').click(function(event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    });
    $('#switcher-narrow, #switcher-large').click(function() {
        $('#switcher').unbind('click');
    });
});
```

现在，如果单击Narrow Column按钮，样式转换器（`<div>`）上的单击处理程序就会被移除。然后，再单击背景区域将不会导致它折叠起来。但是，按钮本身的作用却失效了！由于为使用事件委托而重写了按钮处理代码，因此按钮本身也带有样式转换器（`<div>`）的单击事件处理程序。换句话说，在调用`$('#switcher').unbind('click')`时，会导致按钮上绑定的两个事件处理程序都被移除。

3.6.1 事件的命名空间

显然，应该让对`.unbind()`的调用更有针对性，以避免把注册的两个单击处理程序全都移除。达成目标的一种方式是使用事件命名空间，即在绑定事件时引入附加信息，以便将来识别特定的处理程序。要使用命名空间，需要退一步使用绑定事件处理程序的非简写方法，即`.bind()`方法本身。

我们为`.bind()`方法传递的第一个参数，应该是处理程序对应的JavaScript事件名称。不过，在此可以使用一种特殊的语法形式，即对事件加以细分。

```
$(document).ready(function() {
    $('#switcher').bind('click.collapse', function(event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    });
    $('#switcher-narrow, #switcher-large').click(function() {
        $('#switcher').unbind('click.collapse');
    });
});
```

对于事件处理系统而言，后缀`.collapse`是不可见的。换句话说，这里仍然会像编写`.bind('click')`一样，让注册的函数响应单击事件。但是，通过附加的命名空间信息，则可以解除对这个特定处理程序的绑定，同时不影响为按钮注册的其他单击处理程序。



稍后读者就会看到，还有另一种可以让`.unbind()`调用更有针对性的方式。不过，事件命名空间却是很有用的工具。第11章将会介绍到，在创建插件时使用这个工具会特别方便。

3

3.6.2 重新绑定事件

现在单击Narrow Column或Large Print按钮，会导致样式转换器的折叠功能失效。可是，我们希望该功能在单击Default按钮时再恢复。为此，应该在Default按钮被单击时，重新绑定事件处理程序。

首先，应该为事件处理程序起个名字，以便多次使用：

```
$(document).ready(function() {
    var toggleStyleSwitcher = function(event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    };
    $('#switcher').bind('click.collapse', 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。

而且，我们知道`.bind()`的第二个参数是一个函数引用。在此需要强调一点，使用命名函数时，必须省略函数名称后面的圆括号。圆括号会导致函数被调用，而非被引用。

在函数有了名字之后，将来就可以再次绑定而无需重新定义它了：

```
$(document).ready(function() {
    var toggleStyleSwitcher = function(event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    };
    $('#switcher').bind('click.collapse', toggleStyleSwitcher);
    $('#switcher-narrow, #switcher-large').click(function() {
        $('#switcher').unbind('click.collapse');
    });
    $('#switcher-default').click(function() {
        $('#switcher')
```

```

        .bind('click.collapse', 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。这似乎就意味着，如果连续多次单击Normal按钮，则可能会绑定多个toggleStyleSwitcher处理程序，因而造成单击

时产生不正常的行为。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。

使用命令函数还有另外一个好处，即不必再使用事件命名空间。因为.unbind()可以将这个命名函数作为第二个参数，结果只会解除对特定处理程序的绑定。

```

$(document).ready(function() {
    var toggleStyleSwitcher = function(event) {
        if (!$(event.target).is('.button')) {
            $('#switcher .button').toggleClass('hidden');
        }
    };

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

```

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。

3.7 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。如果我们想向禁用JavaScript的用户隐藏一些内容，以实现平稳退化，那么这就是一种非常合适的方式。

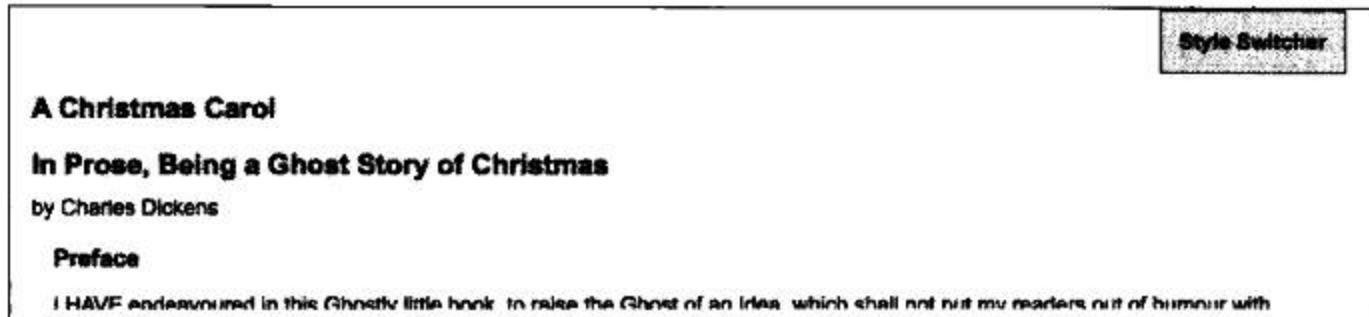


图 3-11

.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();
});
```

键盘事件

作为另一个例子，我们还可以向样式转换器中添加键盘快捷方式。当用户输入每种显示样式的第一个字母时，可以让页面像响应按钮被单击一样作出响应。要实现这种功能，需要先了解键盘事件，键盘事件与鼠标事件稍有不同。

键盘事件可以分为两类：直接对键盘按键给出响应的事件（keyup和keydown）和对文本输入给出响应的事件（keypress）。输入一个字母的事件可能会对应着几个按键，例如输入大写的X要同时按Shift和X键。虽然各种浏览器的具体实现有所不同（毫不奇怪），但有一条实践经验还是比较可靠的：如果想知道用户按了哪个键，应该侦听keyup或keydown事件；如果想知道用户输入的是什么字符，应该侦听keypress事件。对于这里想要实现的功能而言，我们只想知道用户什么时候按下了D、N或L键，因而就要使用keyup。

接下来，需要确定哪个元素应该侦听这个事件。相对于可以通过鼠标指针确定事件目标的鼠标事件而言，这个细节就没有那么明显了。事实上，键盘事件的目标是当前拥有键盘焦点的元素。元素的焦点可能会在几种情况下转移，包括单击鼠标和按下Tab键。并非所有元素都可以获得焦点，只有那些默认情况下具有键盘驱动行为的元素，如表单字段、链接，以及指定了tabIndex属性的元素才可以获得焦点。

对于眼前的例子来说，哪个元素获得焦点其实并不重要，我们只想让转换器在用户按下某个键时能够有所反应。这一次，又可以利用事件冒泡了——因为可以假设所有键盘事件最终都会冒

泡到document元素，所以可以把keyup事件直接绑定到该元素。

最后，需要在keyup处理程序被触发时知道用户按下了哪个键。此时可以检查相应的事件对象，事件对象的.keyCode属性包含着被按下的那个键的标识符。对于字母键而言，这个标识符就是相应大写字母的ASCII值。因此，可以根据这个值来触发对应的按钮单击事件。

```
$ (document).ready(function() {
  $(document).keyup(function(event) {
    switch (String.fromCharCode(event.keyCode)) {
      case 'D':
        $('#switcher-default').click();
        break;
      case 'N':
        $('#switcher-narrow').click();
        break;
      case 'L':
        $('#switcher-large').click();
        break;
    }
  });
});
```

这样，按下这3个键中的任何一个，都会模拟鼠标对相应按钮的单击——假如键盘事件没有被某些特性（例如Firefox的“在输入时搜索文本”功能）所截取的话。

除了使用.trigger()模拟单击外，下面我们再深入一步，看一看怎样把相关代码提取到一个函数中，以便更多处理程序（click和keyup）可以调用它。尽管在眼前的例子中必要性不大，但这种技术确实有利于消减冗余代码。

```
$ (document).ready(function() {
  // 在样式转换器按钮上启用悬停效果。
  $('#switcher .button').hover(function() {
    $(this).addClass('hover');
  }, function() {
    $(this).removeClass('hover');
  });

  // 让样式转换器能够扩展和折叠。
  var toggleStyleSwitcher = function(event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  };
  $('#switcher').click(toggleStyleSwitcher);
  // 模拟一次单击，以便开始时处理折叠状态。
  $('#switcher').click();
  // setBodyClass()用于修改页面样式。
  // 样式转换器的状态也会被更新。
  var setBodyClass = function(className) {
    $('body').removeClass();
```

```

$( 'body' ).addClass( className );
$( '#switcher .button' ).removeClass('selected');
$( '#switcher-' + className ).addClass('selected');
if (className == 'default') {
    $( '#switcher' ).click(toggleStyleSwitcher);
}
else {
    $( '#switcher' ).unbind('click', toggleStyleSwitcher);
    $( '#switcher .button' ).removeClass('hidden');
}
};

// 当按钮被单击时调用setBodyClass()。
$( '#switcher' ).click(function(event) {
    if ($(event.target).is('.button')) {
        if (event.target.id == 'switcher-default') {
            setBodyClass('default');
        }
        if (event.target.id == 'switcher-narrow') {
            setBodyClass('narrow');
        }
        else if (event.target.id == 'switcher-large') {
            setBodyClass('large');
        }
    }
});
;

// 当按下相应按键时调用setBodyClass()。
$(document).keyup(function(event) {
    switch (String.fromCharCode(event.keyCode)) {
        case 'D':
            setBodyClass('default');
            break;
        case 'N':
            setBodyClass('narrow');
            break;
        case 'L':
            setBodyClass('large');
            break;
    }
});
});

```

3.8 小结

在本章中，我们讨论的特性主要实现以下功能。

- 使用`.noConflict()`方法实现同一个页面中多个JavaScript库的和平共存。
- 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。
- 实现事件委托，减少页面中必须绑定的事件处理程序数量。
- 调用`.unbind()`移除停用的事件处理程序。
- 利用事件命名空间分离相关的事件处理程序，以便于分组处理。
- 通过`.trigger()`引发执行绑定的事件处理程序。
- 使用键盘事件处理程序通过`keyup()`响应用户的按键操作。

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。



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):

```
.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。

 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。

^① 此处的“映射”为JavaScript中的对象字面量。下同。——译者注

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，我们仍以第3章的样式转换器为例，但这次使用的HTML稍有不同：

```
<div id="switcher">
  <div class="label">Text Size</div>
  <button id="switcher-default">Default</button>
  <button id="switcher-large">Bigger</button>
  <button id="switcher-small">Smaller</button>
</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.</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。

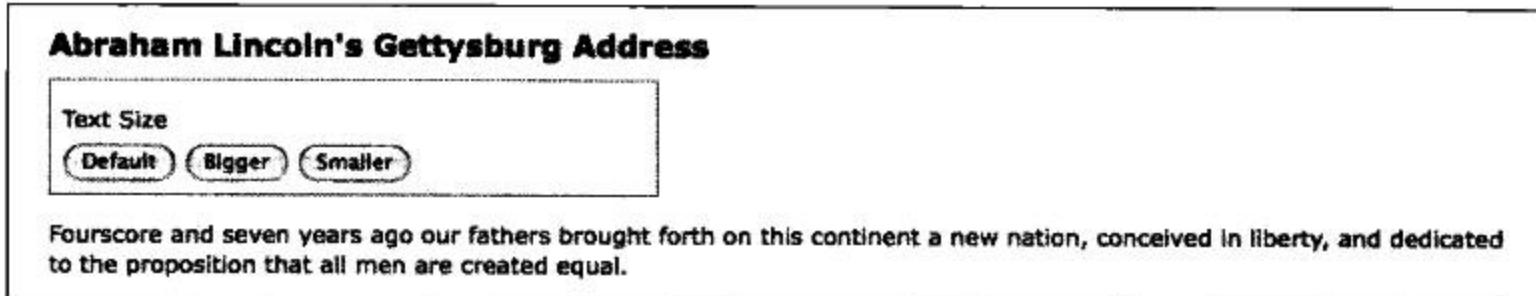


图 4-1

在这个版本的样式转换器中，使用了`<button>`元素。单击Bigger和Smaller按钮，会增大`<div class="speech">`中文本的字体大小，而单击Default按钮，则会把`<div class="speech">`中文本的字体重置为初始大小。

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为基础，按照一个设定的系数（例如40%）来递增字体大小。

同以前一样，我们的代码仍然是从`$(document).ready()`和`$('#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，由于返回的值中包含数字值及其单位（px），需要去掉单位部分才能执行计算。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() {  
    var $speech = $('div.speech');  
    $('#switcher-large').click(function() {  
        var num = parseFloat($speech.css('fontSize'), 10);  
    });  
});
```

`$ (document).ready()` 中的第一行代码把`<div class="speech">`保存到一个变量中。注意变量名`$speech`中的`$`。由于`$`是JavaScript变量中合法的字符，因此可以利用它来提醒自己该变量中保存着一个jQuery对象。

在`.click`处理程序中，通过`parseFloat()`函数只取得字体大小属性中的数值部分。`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

更好的方案是把对这两个按钮的单击操作，通过

中的`<button>`元素组合到一个`.click()`处理程序中。在查找到数值后，再根据用户单击的按钮ID来决定使用乘法还是除法。下面就是相应的代码：

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

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。

如果提供一种方式能够返回字体大小的初始值当然更好了。为了让用户能够做到这一点，可以在DOM就绪后立即把字体大小保存在一个变量中。然后，当用户单击Default按钮时，再使用这个变量的值。虽然可以通过再添加一个`else if`语句来处理这次单击，但此时改用`switch`语句应该更合适。

```
$(document).ready(function() {
    var $speech = $('div.speech');
    var defaultSize = $speech.css('fontSize');
    $('#switcher button').click(function() {
        var num = parseFloat($speech.css('fontSize'), 10);
        switch (this.id) {
            case 'switcher-large':
                num *= 1.4;
                break;
            case 'switcher-small':
                num /= 1.4;
                break;
            default:
                num = parseFloat(defaultSize, 10);
        }
        $speech.css('fontSize', num + 'px');
    });
});
```

在此，仍然是检查`this.id`的值并据以改变字体大小，但如果它的值既不是`'switcher-large'`也不是`'switcher-small'`，那么就应该使用默认的初始字体大小。

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。



要了解有关`display`属性及其不同的值在网页中的视觉表现，请访问 Mozilla Developer Center 的相关页面，网址为 <https://developer.mozilla.org/en/CSS/display/>，或到 <https://developer.mozilla.org/samples/cssref/display.html> 查看相关示例。

`.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，再添加一个新段落，然后在第一个段落末尾加上一个“read more”链接：

```
<div id="switcher">
  <div class="label">Text Size</div>
  <button id="switcher-default">Default</button>
  <button id="switcher-large">Bigger</button>
  <button id="switcher-small">Smaller</button>
</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>
  <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>
<a href="#" class="more">read more</a>
</div>

```

当DOM就绪时，隐藏第二个段落：

```

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

```

现在的讲话（speech）文本看起来如图4-3所示。

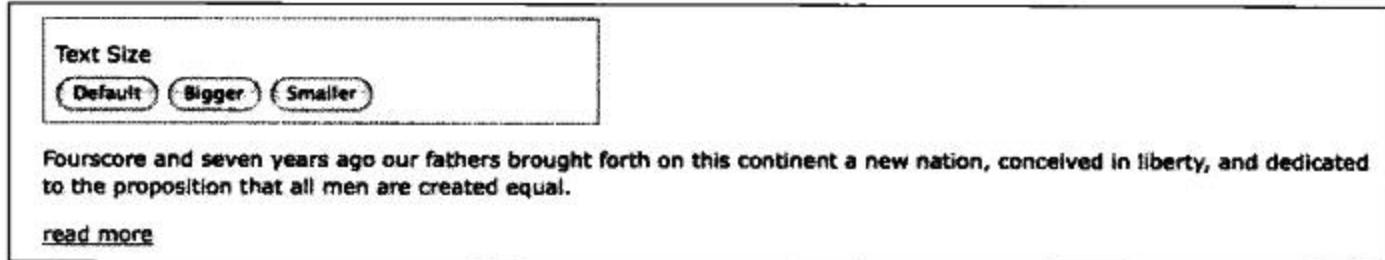


图 4-3

然后，当用户单击第一段末尾read more链接时，就会隐藏该链接同时显示第二个段落：

```

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

```

注意，这里使用了return false来避免链接的默认操作，此时的讲话文本如图4-4所示。

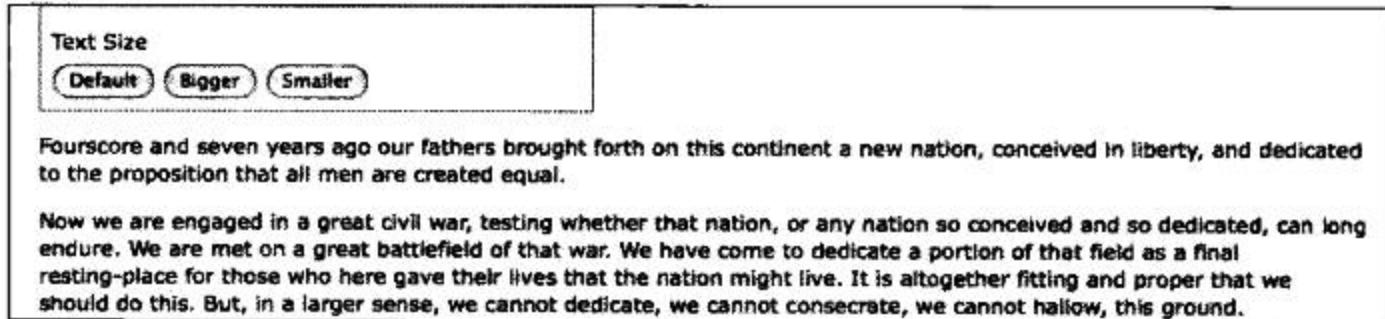


图 4-4

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（或更准确地说，一个时长）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。

4.3.1 指定显示速度

对于jQuery提供的任何效果，都可以指定3种预设的速度参数：`'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《林肯盖提斯堡演说辞》（Lincoln's Gettysburg Address）的第2段指定显示速度：

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

当我们在效果完成大约一半时捕获段落的外观时，会看到类似图4-5所示的结果。

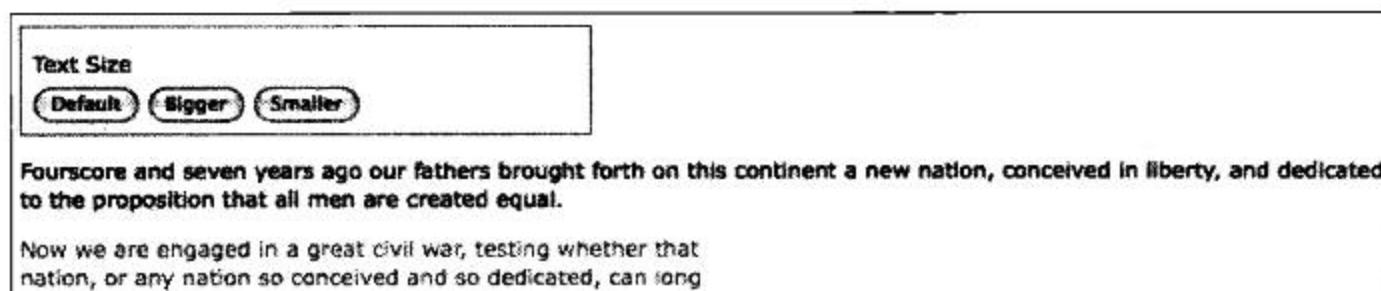


图 4-5

4.3.2 淡入和淡出

虽然使用`.show()`和`.hide()`方法在某种程度上可以创造漂亮的效果，但其效果有时候也可能会显得比较过分。考虑到这一点，jQuery也提供了两个更为精细的内置动画方法。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();
    $('a.more').click(function() {
        $('p:eq(1)').fadeIn('slow');
        $(this).hide();
        return false;
    });
});
```

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。

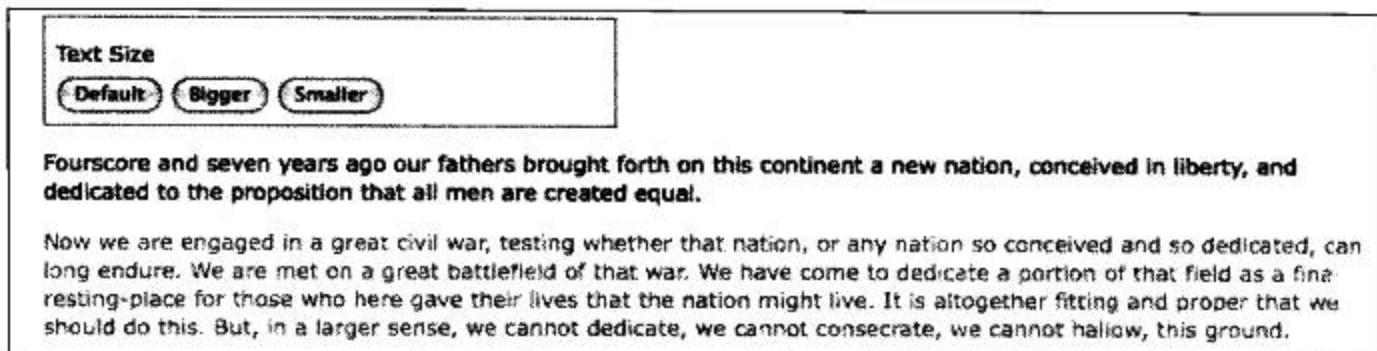


图 4-6

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 复合效果

有时候，我们需要切换某些元素的可见性，而不像前面例子中那样只把它们显示出来。要实现切换，可以先检查匹配元素的可见性，然后再添加适当的方法。在此，仍然以淡入淡出效果为例，可以把示例脚本修改为如下所示：

```
$(document).ready(function() {
    var $firstPara = $('p:eq(1)');
    $firstPara.hide();
    $('a.more').click(function() {
        if ($firstPara.is(':hidden')) {
            $firstPara.fadeIn('slow');
            $(this).text('read less');
        } else {
            $firstPara.fadeOut('slow');
            $(this).text('read more');
        }
        return false;
    });
});
```

与我们在本章前面所做的一样，首先缓存选择符以避免重复遍历DOM。而且，这里也不再隐藏被单击的链接，而是修改它的文本。

使用`if else`语句切换元素的可见性是非常自然而然的方式。但通过jQuery实现复合效果，却不一定非要使用这个条件语句（尽管在这个例子中，需要条件语句来修改链接的文本）。jQuery提供了一个`.toggle()`方法，该方法的作用类似于`.show()`和`.hide()`方法，而且与它们一样的是，`.toggle()`方法也可以带或不带速度参数。另一个复合方法是`.slideToggle()`，该方法通过逐渐增加或减少元素高度来显示或隐藏元素。以下是使用`.slideToggle()`方法的脚本：

```
$(document).ready(function() {
    var $firstPara = $('p:eq(1)');
```

```

$firstPara.hide();
$('a.more').click(function() {
    $firstPara.slideToggle('slow');
    var $link = $(this);
    if ( $link.text() == "read more" ) {
        $link.text('read less');
    } else {
        $link.text('read more');
    }
    return false;
});
});

```

这一次，`$(this)`有可能会被重复，出于性能和可读性考虑，我们把它保存在了`$link`变量中。同样，条件语句检查的是链接的文本而非第二个段落的可见性，因为我们只利用它来修改文本。

4.5 创建自定义动画

除了预置的效果方法外，jQuery还提供了一个强大的`.animate()`方法，用于创建控制更加精细的自定义动画。`.animate()`方法有两种形式，第一种形式接收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——既可以预置的字符串，也可以是一个毫秒数值。
- (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。

把这4个参数放到一起，结果如下所示：

```

.animate({property1: 'value1', property2: 'value2'},
    speed, easing, function() {
        alert('The animation is finished.');
    }
);

```

第二种形式接收两个参数，一个属性映射和一个选项映射。

```
.animate({properties}, {options})
```

实际上，这里的第二个参数是把第一种形式的第2~4个参数封装在了另一个映射中，同时又添加了两个选项。考虑到可读性并调整了换行之后，调用第二种形式的方法的代码如下：

```

.animate({
    property1: 'value1',
    property2: 'value2'
}, {
    duration: 'value',
    easing: 'value',
    complete: function() {
        alert('The animation is finished.');
    }
});

```

```

},
queue: boolean,
step: callback
});

```

现在，我们使用第一种形式的`.animate()`方法，但在本章后面介绍排队效果时会使用其第二种形式。

4.5.1 切换淡入淡出

在讨论复合效果时，读者是否注意到并非所有方法都有对应的切换方法呢？确实如此。虽然滑动方法有对应的`.slideToggle()`，但却没有与`.fadeIn()`和`.fadeOut()`对应的`.fadeToggle()`方法！好在，通过`.animate()`方法可以轻松实现切换淡入淡出的动画。在此，我们把前面例子中调用`.slideToggle()`方法的代码替换了自定义动画代码：

```

$(document).ready(function() {
  $('p:eq(1)').hide();
  $('a.more').click(function() {
    $('p:eq(1)').animate({opacity: 'toggle'}, 'slow');
    var $link = $(this);
    if ($link.text() == "read more") {
      $link.text('read less');
    } else {
      $link.text('read more');
    }
    return false;
  });
});

```

通过这个例子可以看出，`.animate()`方法针对CSS属性提供了方便简写值——'show'、'hide'和'toggle'，以便在简写方法不适用时提供另一种简化任务的方式。

4.5.2 创建多个属性的动画

使用`.animate()`方法可以同时修改多个CSS属性。例如，要在切换第二个段落时，创建一个同时具有滑动和淡入淡出效果的动画，只需在`.animate()`方法的属性映射参数中添加一个`height`属性值-对即可：

```

$(document).ready(function() {
  $('p:eq(1)').hide();
  $('a.more').click(function() {
    $('p:eq(1)').animate({
      opacity: 'toggle',
      height: 'toggle'
    },
    'slow');
    var $link = $(this);
    if ($link.text() == "read more") {

```

```

        $link.text('read less');
    } else {
        $link.text('read more');
    }
    return false;
});
});

```

此外，不仅可以在简写效果方法中使用样式属性，也可以使用其他属性，如：left、top、fontSize、margin、padding和borderWidth。还记得改变演讲段落文本大小的脚本吗？要实现同样的文本大小变化动画，只要把.css()方法替换成.animate()方法即可：

```

$(document).ready(function() {
    var $speech = $('div.speech');
    var defaultSize = $speech.css('fontSize');
    $('#switcher button').click(function() {
        var num = parseFloat($speech.css('fontSize'), 10);
        switch (this.id) {
            case 'switcher-large':
                num *= 1.4;
                break;
            case 'switcher-small':
                num /= 1.4;
                break;
            default:
                num = parseFloat(defaultSize, 10);
        }
        $speech.animate({fontSize: num + 'px'},
                       'slow');
    });
});

```

再使用其他属性，则可以创造出更复杂的效果。例如，可以在把某个项从页面左侧移动到右侧的同时，让该项的高度增加20像素并使其边框宽度增加到5像素。

下面，我们就把这个效果应用于

盒子。图4-7显示了应用效果之前的画面。

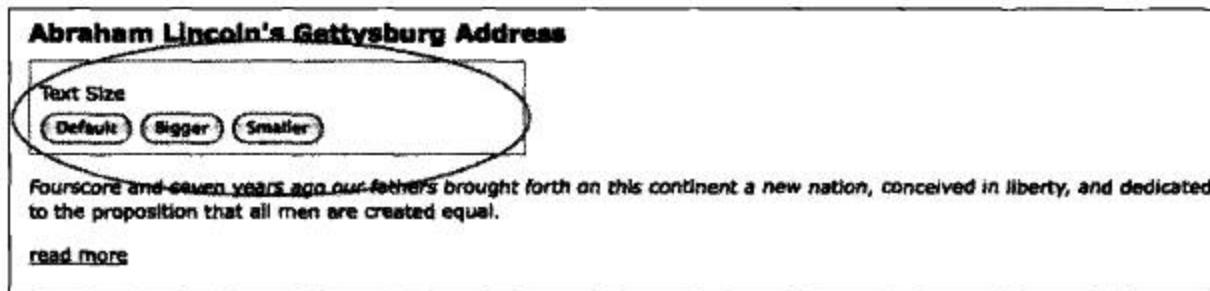


图 4-7

在可变宽度的布局中，需要计算盒子在与页面右侧对齐之前应该移动的距离。假设段落宽度为100%，可以从段落宽度中减去Text Size盒子的宽度。虽然jQuery的.width()方法是执行这种计

算的便捷方法，但该方法不考虑左右内边距和左右边框的宽度。不过从jQuery1.2.6起，我们就可以使用`.outerWidth()`方法了；这样，就可以避免手工计算和添加内边距及边框宽度的麻烦。对于这个例子而言，我们打算通过单击按钮上面的Text Size文本来触发动画。以下就是这个例子的代码：

```
$ (document).ready(function() {
    $('div.label').click(function() {
        var paraWidth = $('div.speech p').outerWidth();
        var $switcher = $(this).parent();
        var switcherWidth = $switcher.outerWidth();
        $switcher.animate({left: paraWidth - switcherWidth,
                           height: '+=20px', borderWidth: '5px'}, 'slow');
    });
});
```

注意，`height`属性的像素值前面有`+/-`运算符。这个表达式（jQuery 1.2中新增）表示一个相对值，即动画的效果不是把高度增加到20像素，而是在当前高度基础上再增加20像素。

此时的代码虽然能够增加相应`<div>`的高度，并加宽其边框，但还不能改变其`left`位置属性。因此，还需要通过修改CSS来支持对位置属性的修改。

通过CSS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。



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;
}
```

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

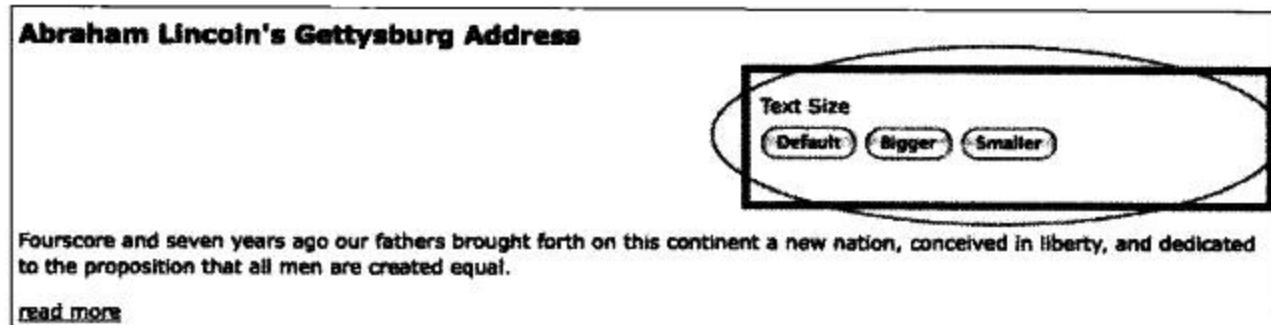


图 4-8

4.6 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。

4.6.1 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，我们仍以移动Text Size盒子、增加其高度、加宽其边框为例。不过，这次我们相继地执行这三个效果——很简单，只要把它们分别放在`.animate()`方法中并连缀起来即可：

```
$('document').ready(function() {
    $('div.label').click(function() {
        var paraWidth = $('div.speech p').outerWidth();
        var $switcher = $(this).parent();
        var switcherWidth = $switcher.outerWidth();
        $switcher
            .animate({left: paraWidth - switcherWidth},
                     'slow')
            .animate({height: '+=20px'}, 'slow')
            .animate({borderWidth: '5px'}, 'slow');
    });
});
```

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`<div id="switcher">`上的效果进行排队：

- (1) 通过`.fadeTo()`将其不透明度减退为0.5。
- (2) 通过`.animate()`将其移动到右侧。
- (3) 通过`.fadeTo()`将其渐增回完全不透明。
- (4) 通过`.slideUp()`隐藏它。
- (5) 通过`.slideDown()`再将其显示出来。

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() {
        var paraWidth = $('div.speech p').outerWidth();
        var $switcher = $(this).parent();
        var switcherWidth = $switcher.outerWidth();
        $switcher
            .fadeTo('fast', 0.5)
            .animate({
                'left': paraWidth - switcherWidth
            }, 'slow')
```

```

    .fadeTo('slow',1.0)
    .slideUp('slow')
    .slideDown('slow');
})
);
}
);

```

不过，要是想在这个

不透明度减退至一半的同时，把它移动到右侧应该怎么办呢？如果两个动画以相同速度执行，则可以简单地把它们组合到一个.animate()方法中。但这个例子中的.fadeTo()使用的速度字符串是'fast'，而向右移动的动画使用的速度字符串是'slow'。在这种情况下，第二种形式的.animate()方法又可以派上用场了：

```

$(document).ready(function() {
    $('#div.label').click(function() {
        var paraWidth = $('#div.speech p').outerWidth();
        var $switcher = $(this).parent();
        var switcherWidth = $switcher.outerWidth();
        $switcher
            .fadeTo('fast',0.5)
            .animate({
                'left': paraWidth - switcherWidth
            }, {duration: 'slow', queue: false})
            .fadeTo('slow',1.0)
            .slideUp('slow')
            .slideDown('slow');
    });
});

```

第二个参数（即选项映射）包含了queue选项，把该选项设置为false即可让当前动画与前一个动画同时开始。

有关为一组元素应用排队效果的最后一个需要注意的问题，就是排队不能自动应用给其他的非效果方法，如.css()。下面，假设我们想在.slideUp()执行后，但在.slideDown()执行前，把

的背景颜色修改为红色，可以尝试像下面这样做：

```

$(document).ready(function() {
    $('#div.label').click(function() {
        var paraWidth = $('#div.speech p').outerWidth();
        var $switcher = $(this).parent();
        var switcherWidth = $switcher.outerWidth();
        $switcher
            .fadeTo('fast',0.5)
            .animate({
                'left': paraWidth - switcherWidth
            }, 'slow')
            .fadeTo('slow',1.0)
            .slideUp('slow')
            .css('backgroundColor','#f00')
            .slideDown('slow');
    });
});

```

然而，即使把修改背景颜色的代码放在连缀序列中正确的位置上，它也会在单击后立即执行。把非效果方法添加到队列中的一种方式，就是使用`.queue()`方法。以下就是在这个例子中使用`.queue()`方法的代码：

```
$('document').ready(function() {
    $('div.label').click(function() {
        var paraWidth = $('div.speech p').outerWidth();
        var $switcher = $(this).parent();
        var switcherWidth = $switcher.outerWidth();
        $switcher
            .fadeTo('fast', 0.5)
            .animate({
                'left': paraWidth - switcherWidth
            }, 'slow')
            .fadeTo('slow', 1.0)
            .slideUp('slow')
            .queue(function() {
                $switcher
                    .css('backgroundColor', '#f00')
                    .dequeue();
            })
            .slideDown('slow');
    });
});
```

像这样传递一个回调函数，`.queue()`方法就可以把该函数添加到相应元素的效果队列中。在这个函数内部，我们把背景颜色设置为红色，然后又添加了一个对应的`.dequeue()`方法。添加的这个`.dequeue()`方法可以让队列在中断的地方再接续起来，然后再与后续的`.slideDown('slow')`连缀在一起。如果在此不使用`.dequeue()`方法，动画就会中断。



要了解有关`.queue()`和`.dequeue()`方法的更多信息，读者可以参考<http://docs.jquery.com/Effects>。

在下面讨论多组元素的效果之后，我们会介绍另一种向队列中添加非效果方法的方式。

4.6.2 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《林肯盖茨堡演说辞》剩余的部分添加到HTML中，并把它们分成两段：

```
<div id="switcher">
    <div class="label">Text Size</div>
    <button id="switcher-default">Default</button>
    <button id="switcher-large">Bigger</button>
    <button id="switcher-small">Smaller</button>
```

```

</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>
    <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>
    <a href="#" class="more">read more</a>
    <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，我们为第三段和第四段分别添加1像素宽的边框和灰色的背景。同时，在DOM就绪时立即隐藏第4段：

```

$(document).ready(function() {
    $('p: eq(2)').css('border', '1px solid #333');
    $('p: eq(3)').css('backgroundColor', '#ccc').hide();
});

```

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(2)').css('border', '1px solid #333')
    .click(function() {
      $(this).slideUp('slow')
      .next().slideDown('slow');
    });
  $('p:eq(3)').css('backgroundColor', '#ccc').hide();
});

```

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4-9所示，可以证实，它们确实是同时发生的。

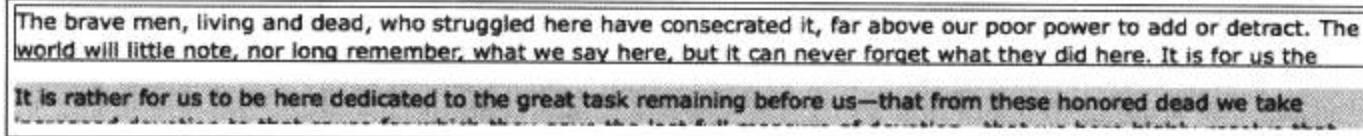


图 4-9

原来可见的第三个段落，正处于向上滑到一半的状态；与此同时，原来隐藏的第四个段落，正处于向下滑到一半的状态。

4.6.3 回调函数

为了对不同元素上的效果实现排队，jQuery为每个效果方法都提供了回调函数。同我们在事件处理程序和`.queue()`方法中看到的一样，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(2)').css('border', '1px solid #333')
    .click(function() {
      $(this).next().slideDown('slow', function() {
        // 这里的代码会在第三个段落的滑下效果结束后执行。
      });
    });
  $('p:eq(3)').css('backgroundColor', '#ccc').hide();
});

```

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 \$thirdPara = \$(this)。

这样，无论是在回调函数的外部还是内部，\$thirdPara引用的都是第三个段落。下面是使用了新变量之后的代码：

```
$(document).ready(function() {
    var $thirdPara = $('p:eq(2)');
    $thirdPara
        .css('border', '1px solid #333')
        .click(function() {
            $(this).next().slideDown('slow', function() {
                $thirdPara.slideUp('slow');
            });
        });
    $('p:eq(3)').css('backgroundColor', '#ccc').hide();
});
```

在.slideDown()的回调函数内部使用\$thirdPara有赖于闭包。我们将在附录C中详细地讨论这个重要但又不太好理解的闭包。

这次效果中途的屏幕截图4-10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。

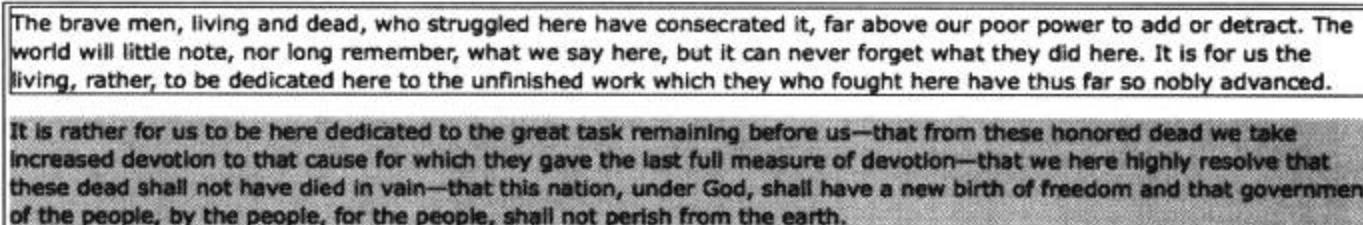


图 4-10

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接近一系列效果结束时改变背景颜色的问题了。这次，我们不像前面那样使用.queue()方法，而是使用回调函数：

```
$(document).ready(function() {
    $('#div.label').click(function() {
        var paraWidth = $('#div.speech p').outerWidth();
        var $switcher = $(this).parent();
        var switcherWidth = $switcher.outerWidth();
        $switcher
            .fadeTo('slow', 0.5)
            .animate({
                'left': paraWidth - switcherWidth
            }, 'slow')
            .fadeTo('slow', 1.0)
            .slideUp('slow', function() {
                $switcher
                    .css('backgroundColor', '#f00');
            });
    });
});
```

```
        })
        .slideDown('slow');
    });
});
});
```

同前面一样，`<div id="switcher">`的背景颜色在它滑上之后滑下之前，变成了红色。

4.6.4 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）——除非`queue`选项值为`false`。

(2) 多组元素上的效果：

- ◆ 默认情况下是同时发生的。
- ◆ 当在另一个效果方法或者在`.queue()`方法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，读者应该能够使用`.css()`或`.animate()`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。

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——就酷似变魔术。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。当然，也可以使用`.attr()`和`.removeAttr()`来修改`class`属性——但这种情况下，应该优先使用专门的`.addClass()`和`.removeClass()`方法，因为它们能够正确处理一个元素有多个类（例如`<div class="first second">`）的情况。

5.1.1 非 `class` 属性

有些属性如果不通过jQuery并不容易操作，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>
<!-- . . . code continues . . . -->
</div>

```

对于以上HTML，我们可以循环遍历

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'});
});

```

这种技术之所以奏效，是因为我们希望新的rel属性在每个链接中都具有相同的值。不过，更常见的情况是为每个元素添加或修改的属性都必须具有不同的值。例如，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，对第一个链接它的值是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`，对于第一个链接而言，其`id`值就是`wikilink-0`，对于第二个链接就是`wikilink-1`，然后依此类推。



实际上，在此也可以利用为`.attr()`方法传递第二个函数来实现隐式迭代，就像我们在第2章通过`.filter()`方法的参数实现隐式迭代一样（要了解更详细的信息，请参考 <http://docs.jquery.com/Attributes/attr#keyfn>）。不过，这里使用`.each()`方法似乎更方便一些。

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'
        });
    });
});
```

注意，这里我们把`$(this)`保存在了变量`$thisLink`中，原因就是为了多次使用它。在设置了这3属性之后，其中一个（第一个）链接的HTML现在应该如下所示：

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

5.1.2 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。

5

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.

But this rule applies not always to the Tradesman, 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....

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

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() {
    $('back to top</a>'\)
        .insertAfter\('div.chapter p'\);
    \$\('<a id="top"></a></a>'\);
}\);
```

通过`.after()`方法也能完成与`.insertAfter()`相同任务，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。

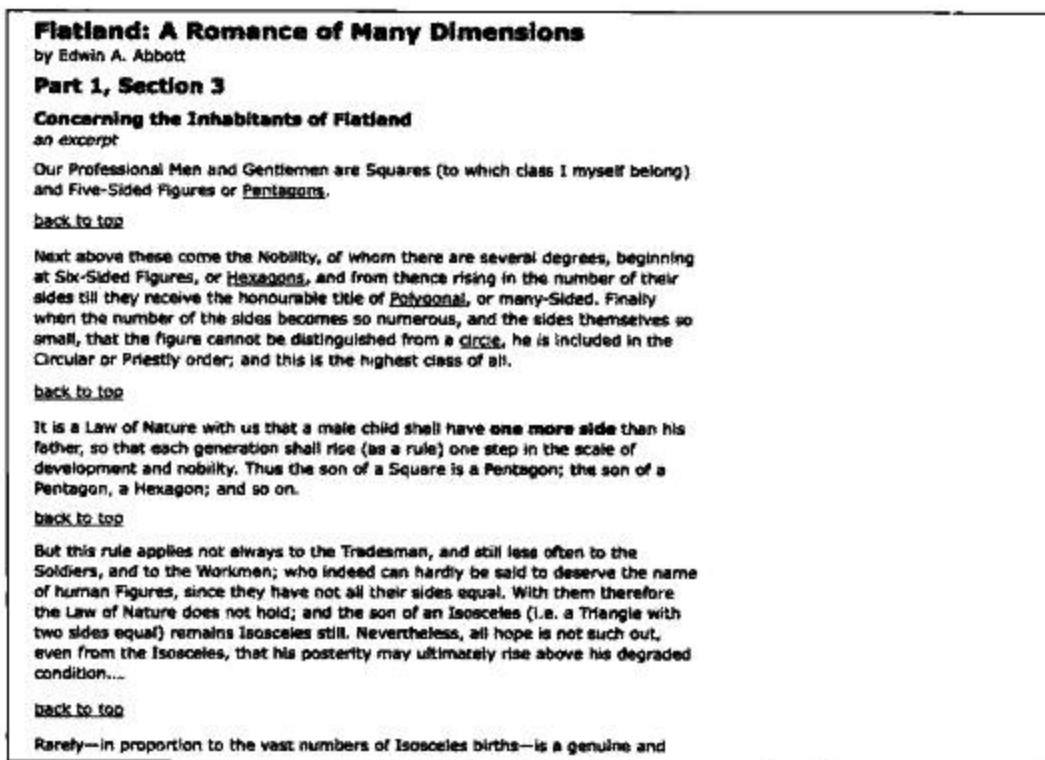


图 5-2

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。

The screenshot shows a portion of the 'Flatland: A Romance of Many Dimensions' page. At the top, it says 'Flatland: A Romance of Many Dimensions' by Edwin A. Abbott. Below that is 'Part 1, Section 3' and 'Concerning the Inhabitants of Flatland'. An excerpt from the text follows:

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 Tradesman, 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. "What need of a

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—, in proportion to the vast numbers of Isosceles
births—, is a genuine and certifiable Equal-Sided
Triangle produced from Isosceles parents. <span
class="footnote">"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 Equalsidedness
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.</span> 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. <span class="footnote">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.</span>

</p>
<p>How admirable is the Law of Compensation! <span
class="footnote">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!</span>
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.&hellip;

</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。

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

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');
});
```

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;
    margin: 1em 0;
}
.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>').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

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>').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>'].join('')
})
);
}
);

```

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，第一行代码是`$(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()`方法圆括号中连接数组，是一个上标形式的链接，将被插入到相应的脚注(``)前面。把这个链接插入到DOM中之后，可以看到类似下面的标记代码：

```

<a href="#foot-note-1" id="context-1" class="context"><sup>1</sup></a>

```

乍一看，读者可能会觉得连接数组的语法有点陌生，因此我们花点时间来分析一下。在`.before()`方法的圆括号中，我们首先输入一对方括号——`[]`，也就是数组直接量。方括号中的每个元素都后跟一个逗号（注意，最后一个元素除外）。考虑到可读性，我们把每个元素都独自放在了一行。然后，在数组一经创建时，就通过JavaScript方法`.join()`把数组转换成一个字符串。`.join()`方法以一个空字符串（由一对单引号表示）作为参数，因为我们不想在由数组项生成的HTML输出中添加任何字符。

注意代码中的`index+1`。因为计数起点是0，开始时加1即表示让第一个链接的`href`属性值为`#footnote-1`，`id`属性值为`#context-1`，而实际的链接文本为1。在此，`href`属性的值特别重要，这个值必须与脚注的`id`属性值相同（当然，不包括#）。

事实上，不使用连接数组，而使用一个相对较长的拼接字符串也可以达到同样的目的：

```

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

```

不过在这个例子中，使用数组技术编写代码似乎更容易。

 Web 上有很多人已经讨论过连接数组与拼接字符串之间的性能差别。其中比较深入的一篇文章讨论了一些使用这两种技术的基准测试，文章链接为 <http://www.sitepen.com/blog/2008/05/09/string-performance-an-analysis/>。

然而，多数情况下，这两种技术的差异微乎其微。如果确实需要考虑脚本性能，那么在其他方面采取措施可能会更有成效（例如前面讨论的“缓存”选择符。）

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。

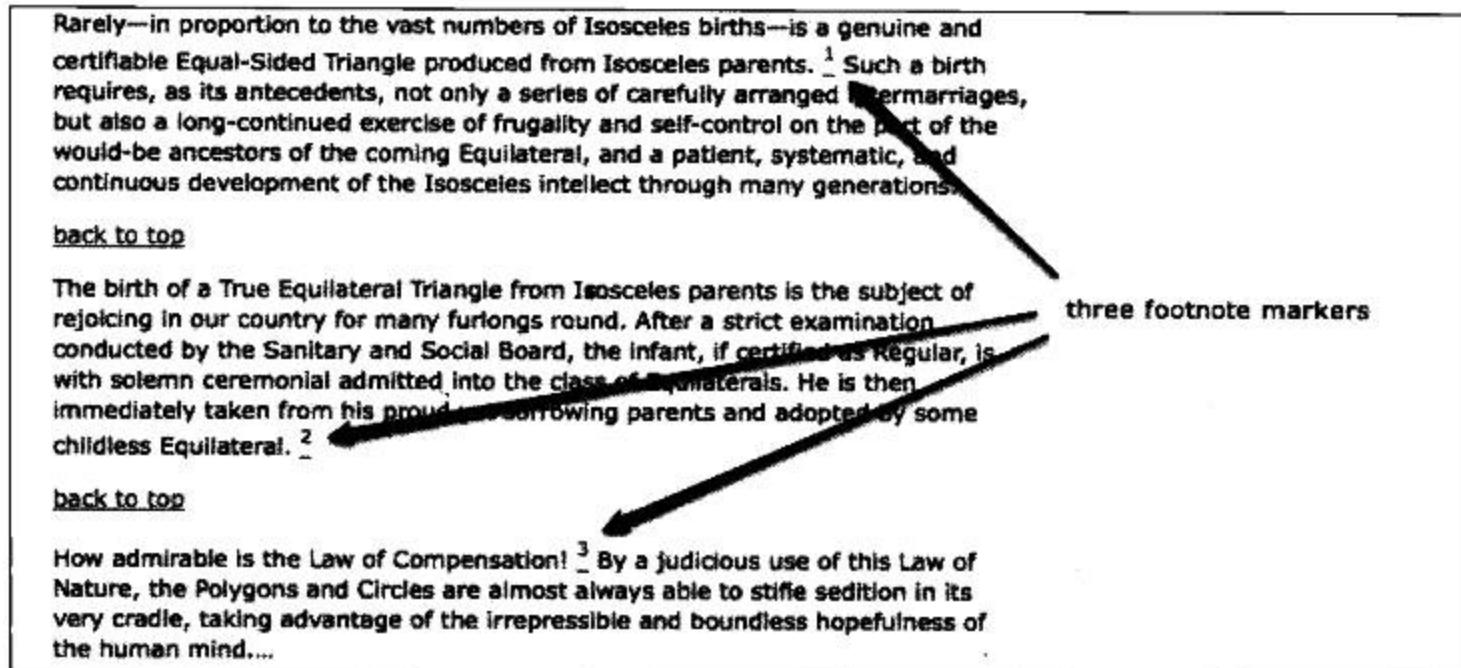


图 5-7

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 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>']
        ].join(''))
  )
  .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 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>']
            ).join('')
    })
    .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。

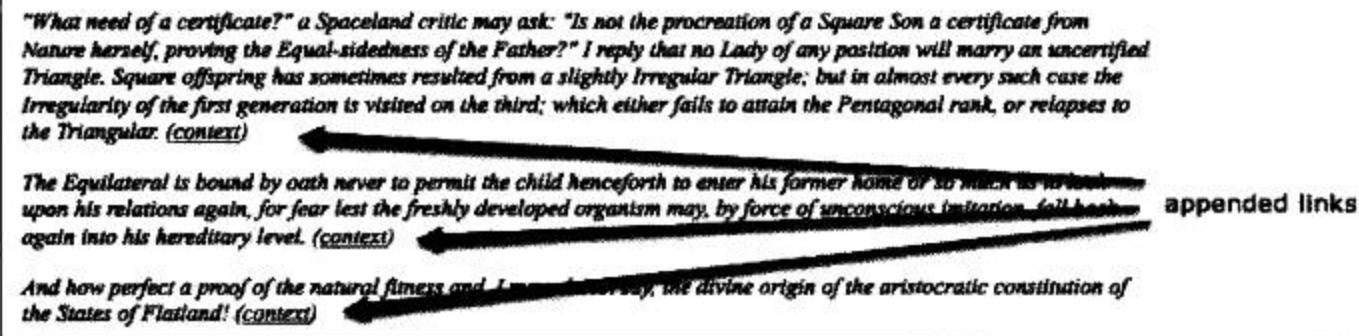


图 5-8

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``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,
                 '>'],
                 index+1,
                 '" id="context-', index+1,
                 '" class="context">',
                 '<sup>' + (index+1) + '</sup>',
                 '</a>']
            ).join('')
    })
    .appendTo('#notes')
    .append(' &ampnbsp(<a href="#context-' + (index+1) +
            '">context</a>)' );
});
});

```

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：

```

    index+1,
    '" id="context-' + index+1 +
    '" class="context">',
    '<sup>' + (index+1) + '</sup>',
    '</a>'
  ].join('')
)
.appendTo('#notes')
.append('  (<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 lest 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。



jQuery还提供了另外两个包装元素的方法：`.wrapAll()`和`.wrapInner()`。更多信息请参见 <http://docs.jquery.com/Manipulation/wrapAll> 和 <http://docs.jquery.com/Manipulation/wrapInner>。

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

5

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 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

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 连同事件一起复制

在默认情况下，`.clone()`方法不会复制匹配的元素或其后代元素中绑定的事件。不过，可以为这个方法传递一个布尔值参数，将这个参数设置为`true`，就可以连同事件一起复制，即`.clone(true)`。这样一来，就可以避免像第3章讨论的那样，每次复制之后还要手工重新绑定事件的麻烦。

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。

5.5.3 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在DOM中（绝对或相对）定位的最近祖先元素的上方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 $145\text{px} + 5\text{px} + 10\text{px}$ ，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 。

5.5.4 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，我们同样把需要多次用到的选择符表达式保存在变量中，以提升性能和可读性。

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。

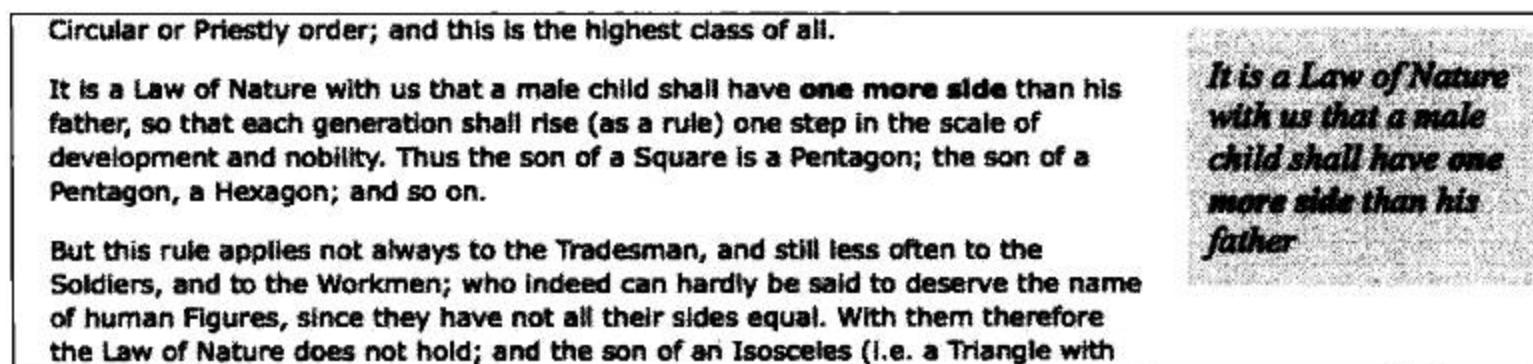


图 5-12

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。

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。

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。

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 Tradesman, 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](#)

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.²

[back to top](#)

*It is a Law of Nature
... that a male child
shall have one more
side than his father*

*The birth of a True
Equilateral Triangle
from Isosceles
parents is the subject
of rejoicing in our
country*

当然，还必须修改CSS来处理新添加的<div>，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，我们将以前应用给的一些规则转移到了<div class="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。

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](#)

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.²

[back to top](#)

How admirable is the Law of Compensation!³ By a judicious use of this Law of

*It is a Law of Nature
... that a male child
shall have one more
side than his father*

*The birth of a True
Equilateral Triangle
from Isosceles
parents is the
subject of rejoicing
in our country*

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HTML中创建新元素，使用`$()`工厂函数。

(2) 要在每个匹配的元素中插入新元素，使用：

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

(3) 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()`

(4) 要在每个匹配的元素外部插入新元素，使用：

- ◆ `.wrap()`
- ◆ `.wrapAll()`
- ◆ `.wrapInner()`

(5) 要用新元素或文本替换每个匹配的元素，使用：

- ◆ `.html()`
- ◆ `.text()`
- ◆ `.replaceAll()`
- ◆ `.replaceWith()`

(6) 要移除每个匹配的元素中的元素，使用：

- ◆ `.empty()`

(7) 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。



近几年，根据使用的技术来评价站点的做法日益流行。在用于描述新Web应用程序的众多热门字眼中，“AJAX驱动”最为引人注目。为某个站点加上AJAX的标签，其含义可谓丰富，因为这个术语本身涵盖的是一组相关的能力和技术概念。

从技术上说，AJAX表示的是一组首字母缩写词——Asynchronous JavaScript and XML（异步JavaScript和XML）。一个AJAX解决方案中涉及如下技术：

- 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，如HTML或JSON；
- 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。

假设我们要创建一个页面，用以显示字典中的词条，词条按照英文首字母分组。那么，定义页面内容区的HTML代码可以像下面这样：

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



当然，真正的实现应该本着渐进增强的原则，让页面在没有 JavaScript 的情况下照样可以使用。但我们在乎这里为保持例子简单，没有为链接添加有意义的地址；后面会通过 jQuery 为其添加行为。

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。第二个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>

```

这个页面的HTML代码中还包含更多词条。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.

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。通常，我们把这种文件叫做片段；它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 a').click(function() {
        $('#dictionary').load('a.html');
        return false;
    });
});
```

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。

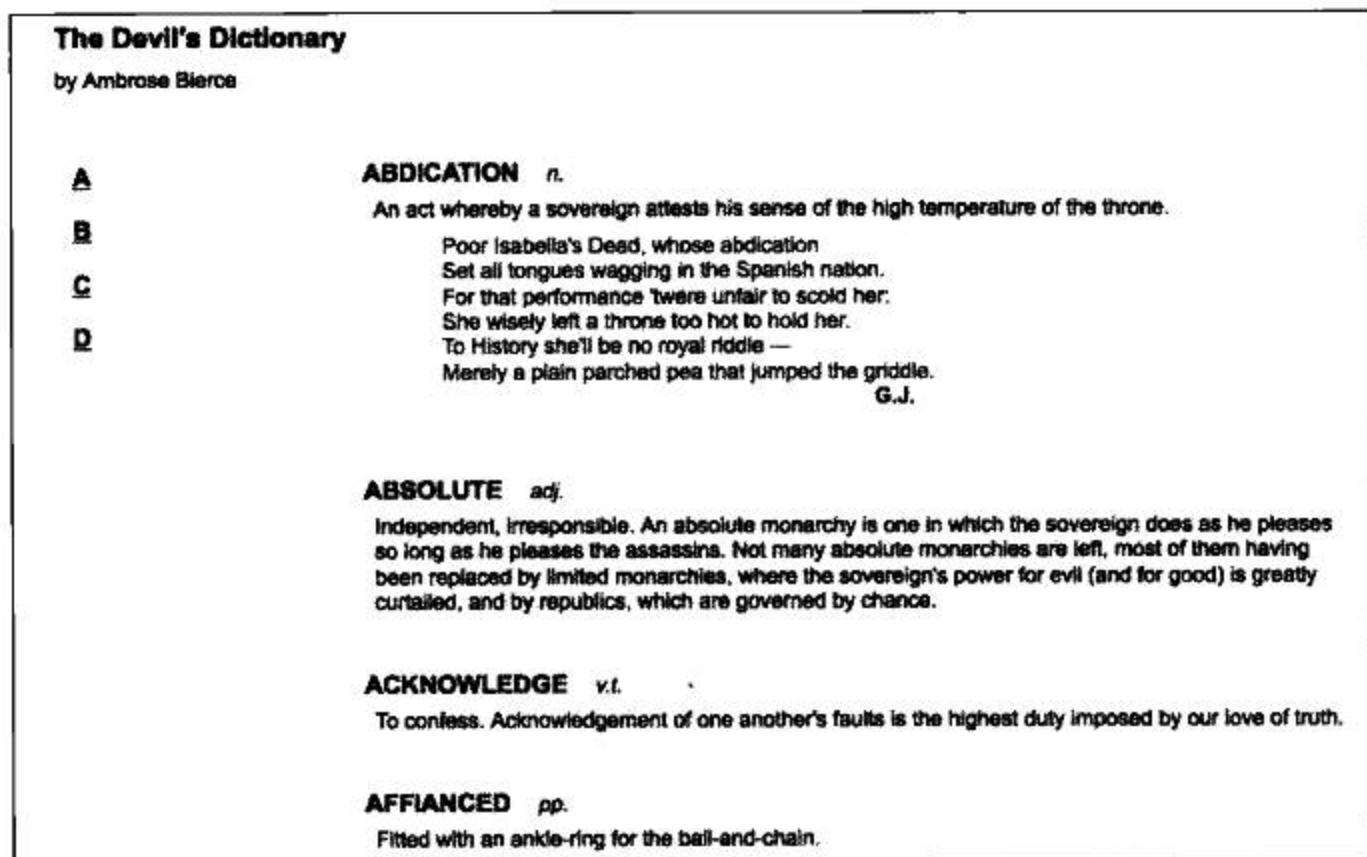


图 6-3

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。

测试这个例子：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 a').click(function() {
    $('#dictionary').load('a.html');
    alert('Loaded!');
    return false;
});
});
```

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。因为JavaScript通常以同步（synchronous）方式执行代码，即严格按照顺序逐行执行。

然而，当我们在运行中的服务器上测试上面这些代码时，由于网络延迟，警告框很可能先于加载完成就出现了。这是因为所有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。

使用jQuery的选择符可以遍历和操作取得的HTML结构，但前提是必须先把HTML结构插入到文档中。而且，采用JavaScript内置的数据格式也会减少编码量。

1. 取得JavaScript对象

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"
  ]
}
```

^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



要了解JSON有哪些优势，都有哪些语言支持这种数据格式，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，这个文件开头部分的代码如下：

```
[  
 {  
   "term": "BACCHUS",  
   "part": "n.",  
   "definition": "A convenient deity invented by the...",  
   "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..."  
 },  
 {  
   "term": "BEARD",  
   "part": "n.",  
   "definition": "The hair that is commonly cut off by..."  
 },
```

6

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（由jQuery库定义的`jQuery`或`$`对象）的方法定义的，也就是说，它不是个别jQuery对象实例（即通过`$()`函数创建的对象）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 a').click(function() {
    $.getJSON('b.json');
    return false;
});
});
```

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 a').click(function() {
        $.getJSON('b.json', function(data) {
        });
        return false;
    });
});
```

我们在此使用了匿名函数作为回调函数，这在jQuery代码中很常见，主要是为了保持代码简洁。当然，命名函数同样也可以作为回调函数。

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 a').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);
            });
        });
        return false;
    });
});
```

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。



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 a').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'];
        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);
      });
    });
    return false;
  });
});

```

6



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。

^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

编写完这些代码后，就可以单击下一个B链接来验证我们的成果了，如图6-4所示。

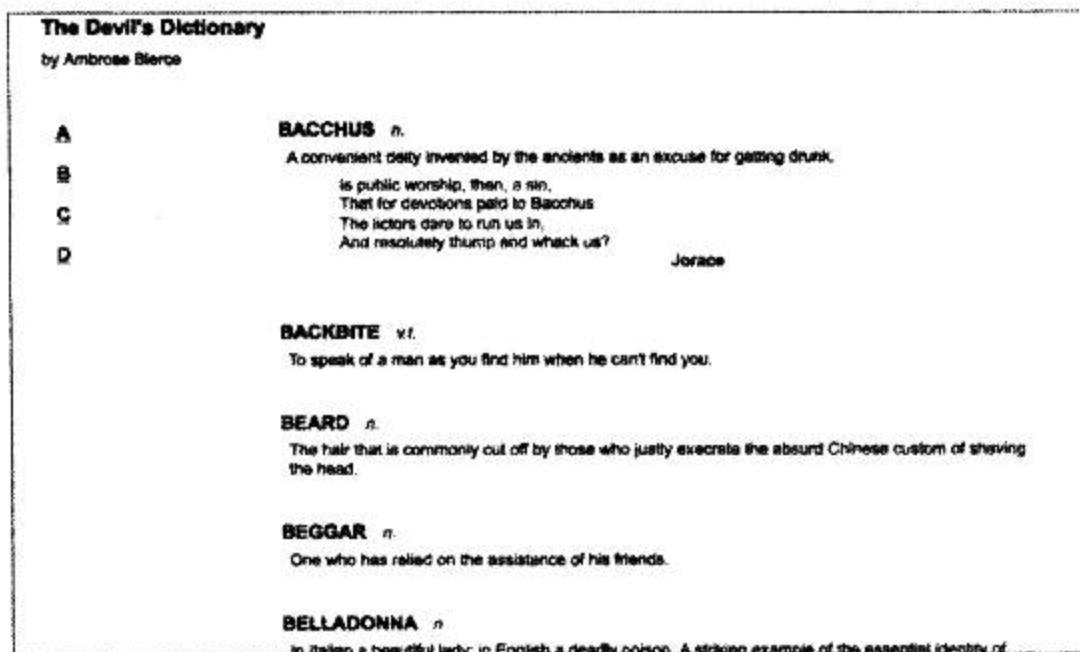


图 6-4

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 a').click(function() {
    $.getScript('c.js');
    return false;
  });
});
```

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..." },
  {
    "term": "CANNIBAL",
    "part": "n.",
```

```

    "definition": "A gastronome of the old school who..."}
},
{
  "term": "CHILDHOOD",
  "part": "n.",
  "definition": "The period of human life intermediate..."
},
{
  "term": "CLARIONET",
  "part": "n.",
  "definition": "An instrument of torture operated by..."
},
{
  "term": "COMFORT",
  "part": "n.",
  "definition": "A state of mind produced by..."
},
{
  "term": "CORSAIR",
  "part": "n.",
  "definition": "A politician of the seas."
}
];
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C链接，应该会看到我们预期的结果，如图6-5所示。

The Devil's Dictionary
by Ambrose Bierce

A

C

CALAMITY n.
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.

CANNIBAL n.
A gastronome of the old school who preserves the simple tastes and adheres to the natural diet of the pre-pork period.

CHILDHOOD n.
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.

CLARIONET n.
An instrument of torture operated by a person with cotton in his ears. There are two instruments that are worse than a clarinet — two clarinets.

COMFORT n.
A state of mind produced by contemplation of a neighbor's uneasiness.

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="DEFAME" part="v.t.">
        <definition>
            To lie about another. To tell the truth about another.
        </definition>
    </entry>
    <entry term="DEFENCELESS" part="adj.">
        <definition>
            Unable to attack.
        </definition>
    </entry>
    <entry term="DELUSION" part="n.">
        <definition>
            The father of a most respectable family, comprising
            Enthusiasm, Affection, Self-denial, Faith, Hope,
            Charity and many other goodly sons and daughters.
        </definition>
        <quote author="Mumfrey Mappel">
            <line>All hail, Delusion! Were it not for thee</line>
            <line>The world turned topsy-turvy we should see;
                </line>
            <line>For Vice, respectable with cleanly fancies,
                </line>
            <line>Would fly abandoned Virtue's gross advances.
                </line>
        </quote>
    </entry>
    <entry term="DIE" part="n.">
        <definition>
            The singular of "dice." We seldom hear the word,
            because there is a prohibitory proverb, "Never say
            die." At long intervals, however, some one says: "The
            die is cast," which is not true, for it is cut. The
            word is found in an immortal couplet by that eminent
            poet and domestic economist, Senator Depew:
        </definition>
        <quote>
            <line>A cube of cheese no larger than a die</line>
            <line>May bait the trap to catch a nibbling mie.</line>
        </quote>
    </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 a').click(function() {
        $.get('d.xml', function(data) {

    });
    return false;
});
});
```

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 a').click(function() {
        $.get('d.xml', function(data) {
            $('#dictionary').empty();
            $(data).find('entry').each(function() {
                var $entry = $(this);
                var html = '<div class="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));
            });
        });
        return false;
    });
});
```

这样，当单击D链接时，也可以得到预期的效果，如图6-6所示。

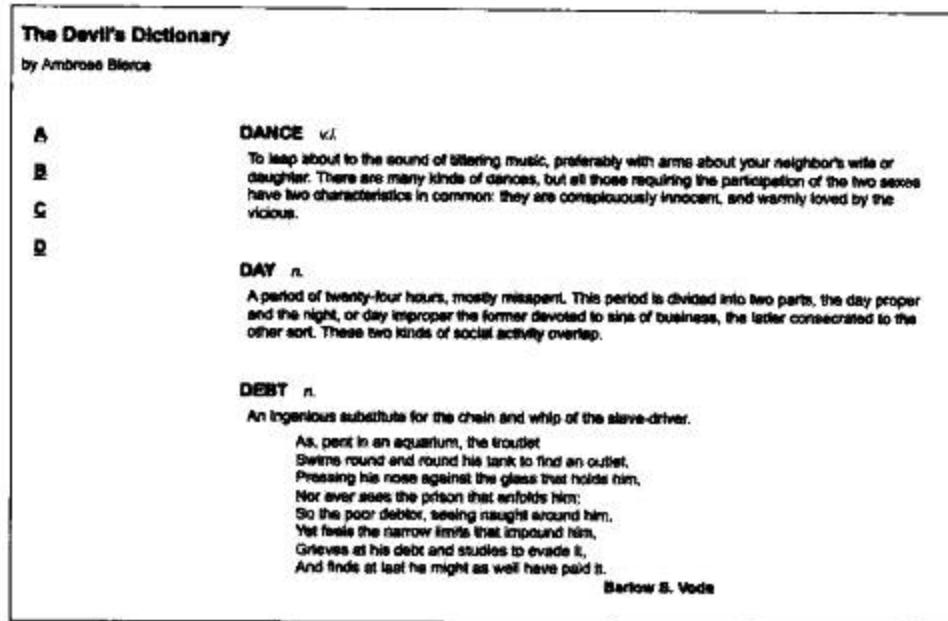


图 6-6

这是我们已知的DOM遍历方法的新用途，而且，jQuery对CSS选择符支持的灵活性由此也可见一斑。CSS的选择符语法一般适合美化HTML页面，位于标准.css文件中的选择符，如div和body等标签名都是为了在HTML中找到内容。然而，jQuery可以使用任意XML标签名，如这里的entry和definition，就和使用标准HTML标签一样方便。

而且，jQuery内部先进的选择符引擎，对于在更复杂的情况下查找XML文档中的元素同样很有帮助。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:has(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。

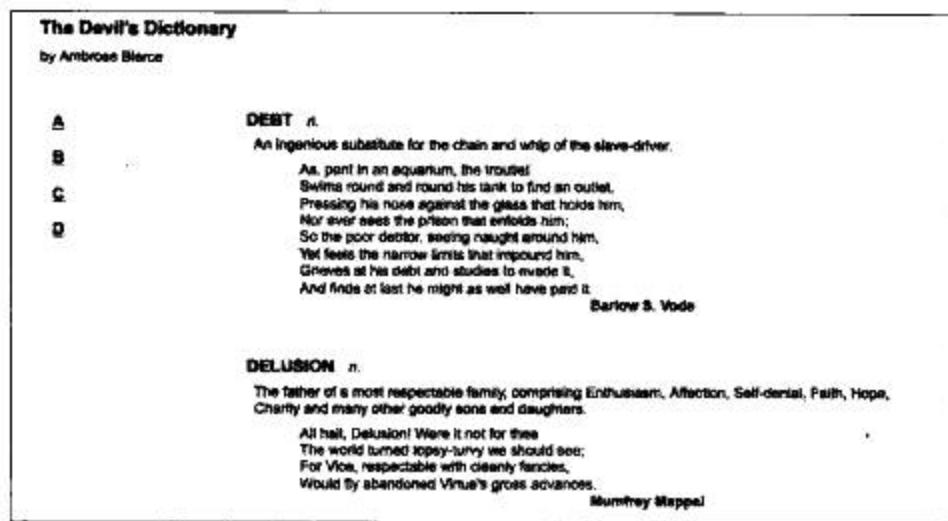


图 6-7

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。

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

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充分体现。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。



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
        disguises 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
$term = strtoupper($_REQUEST['term']);
if (isset($entries[$term])) {
    $entry = $entries[$term];

    $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'])) {
        $html .= '<div class="quote">';
        foreach ($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。

EAVESDROP

vi.
 Secretly to overhear a catalogue of the crimes and vices of another or yourself.
 A lady with one of her ears applied
 To an open keyhole heard, inside,
 Two female gossips in converse free —
 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!"
 Gopete Sherany

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。

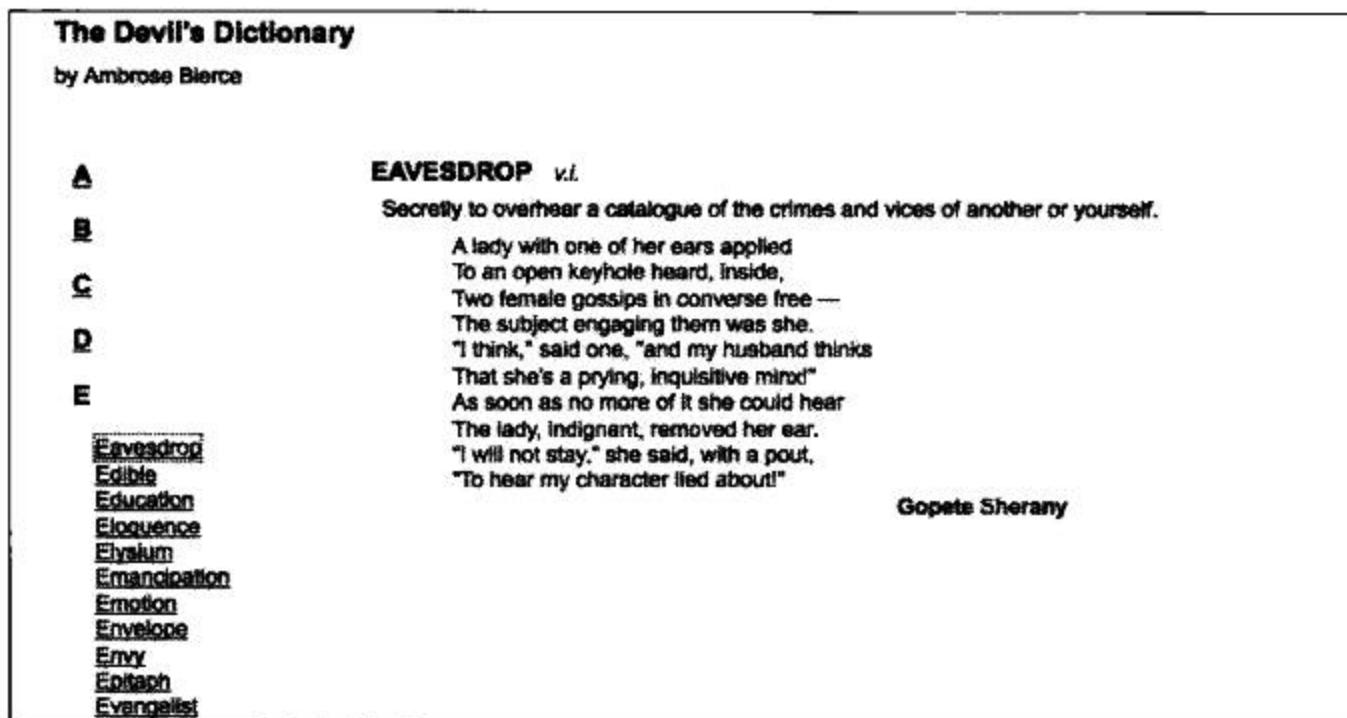


图 6-9

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（这也是一种渐进增强的做法）。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。

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。

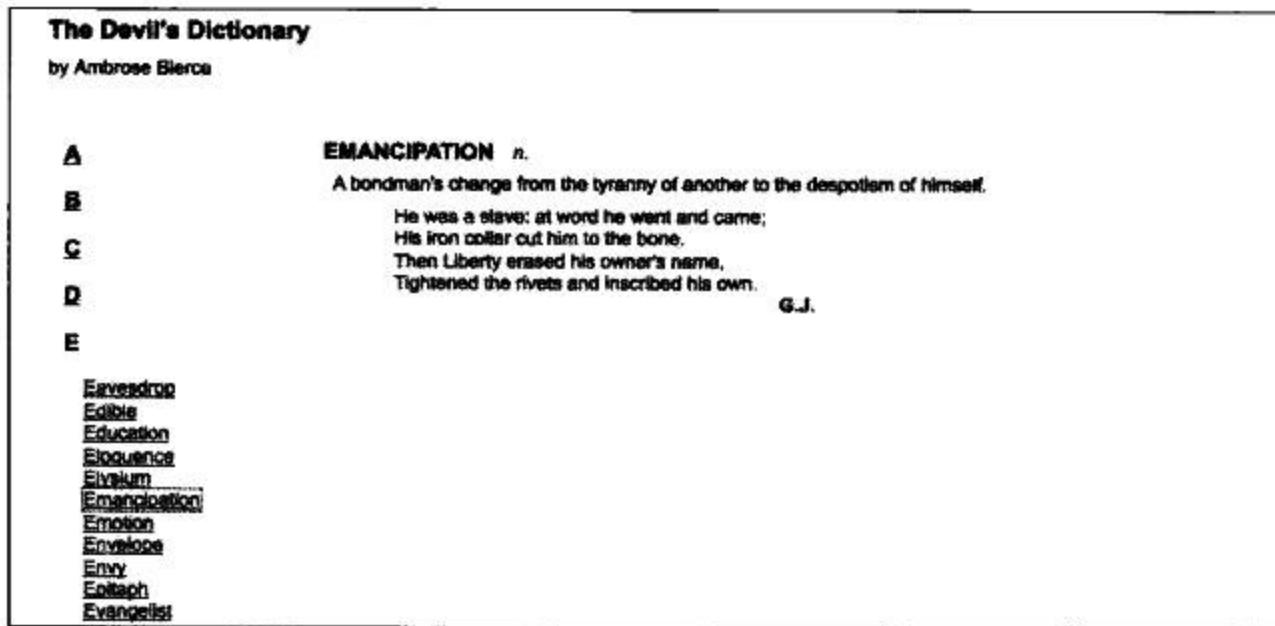


图 6-10

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">
    <h3>F</h3>
    <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。特别是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).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。

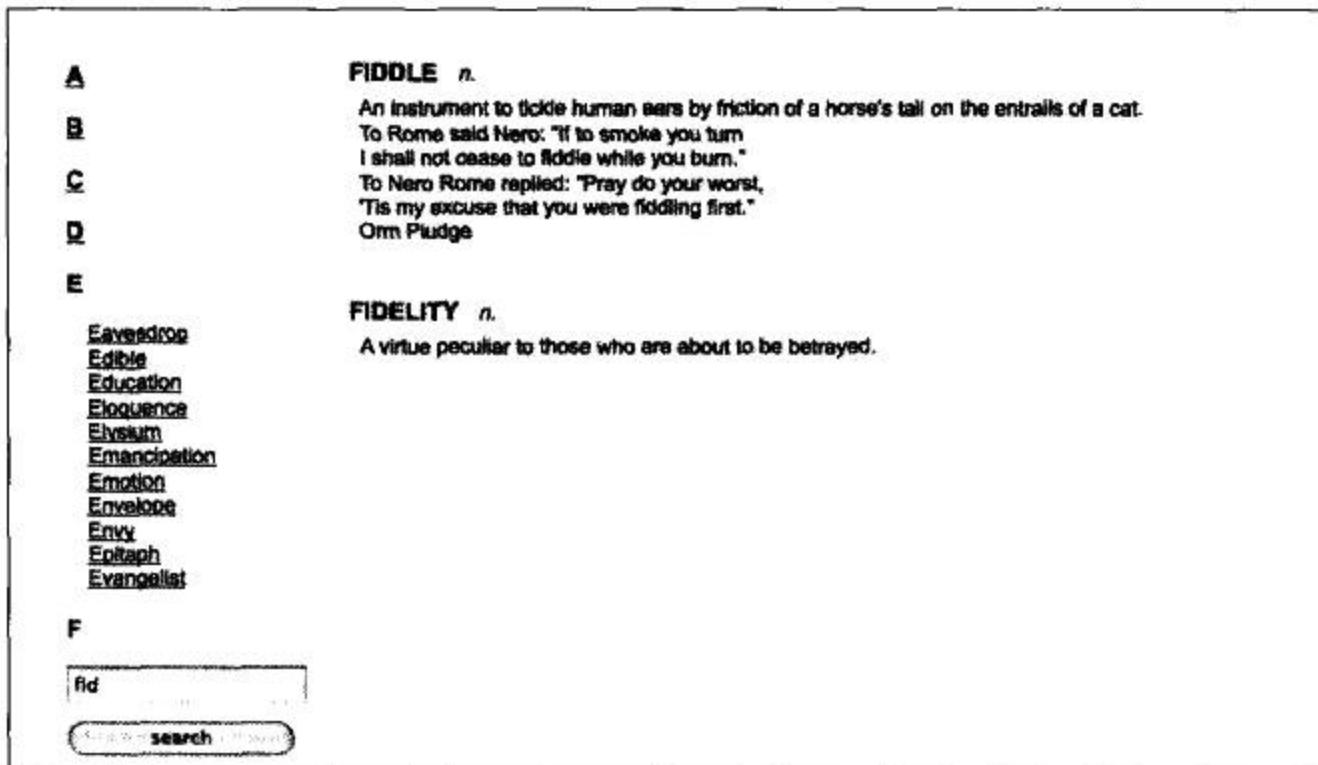


图 6-11

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>
```

反馈信息就是一个HTML片段，比如可以包含用作动态指示器（throbber）的一幅动态GIF图像。此时，可以在CSS文件中添加一些简单的样式，这样当显示该信息时，页面的外观如图6-12所示。

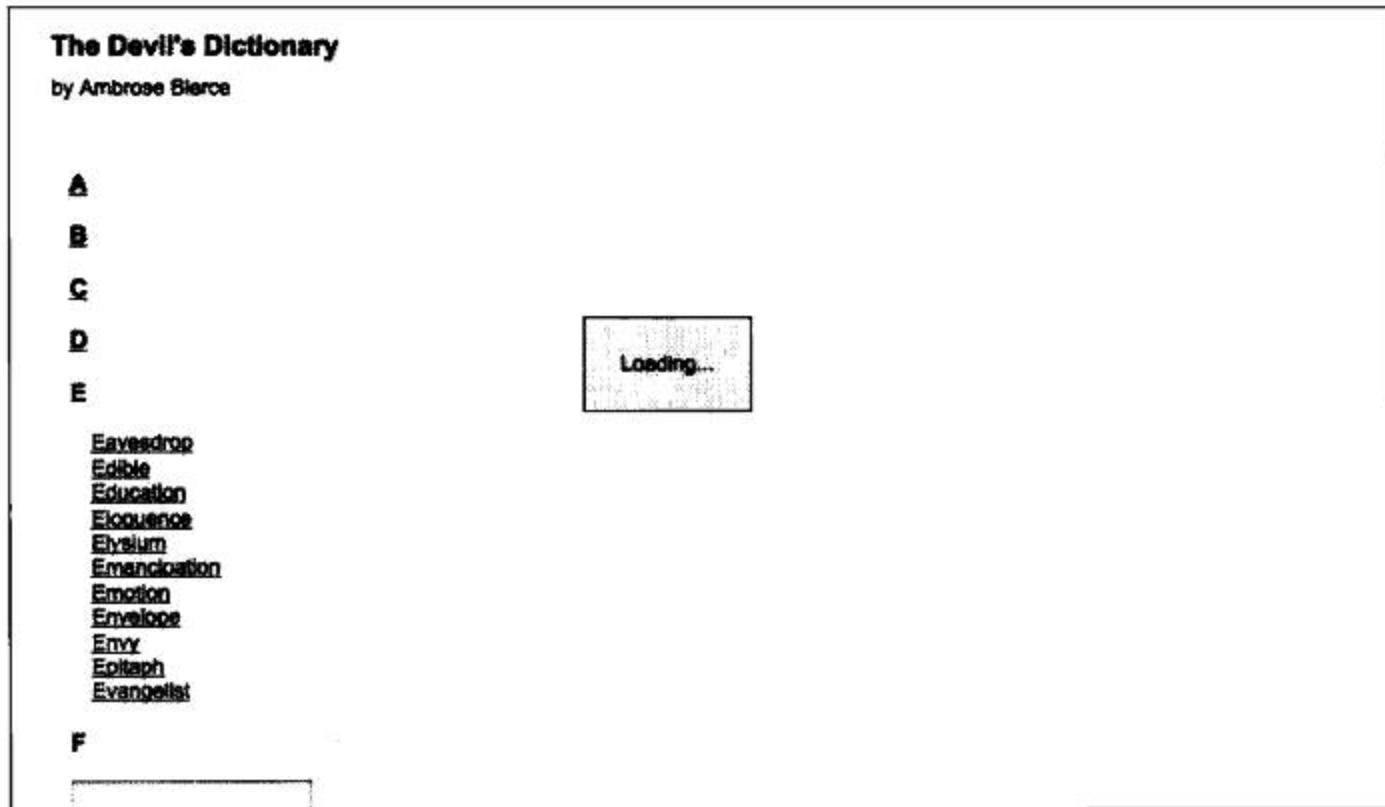


图 6-12

但是，依照渐进增强的原则，我们并没有把HTML标记直接放在页面中。因为这段信息只能在JavaScript有效的情况下使用，所以应该通过jQuery来插入它：

```
$(document).ready(function() {
  $('<div id="loading">Loading...</div>')
  .insertBefore('#dictionary')
});
```

而在CSS文件中，我们为这个

添加了一条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() {
  $('<div id="loading">Loading...</div>')
  .insertBefore('#dictionary')
  .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() {
    $('<div id="loading">Loading...</div>')
        .insertBefore('#dictionary')
        .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。链接A上的.load()和链接B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，这些稍后再作介绍。

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 a').click(function() {
        $('#dictionary').load('a.html');
        return false;
    });
});
```

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 a').click(function() {
        $('#dictionary').hide().load('a.html', function() {
            $(this).fadeIn();
        });
        return false;
    });
});
```

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

假设我们想让字典词条名来控制后面解释的可见性，即单击词条名可以显示或隐藏相应的解释。使用到目前为止介绍的技术，实现这一点应该很简单：

```
$(document).ready(function() {
    $('.term').click(function() {
        $(this).siblings('.definition').slideToggle();
    });
});
```

当词条被单击时，该元素找到类名中包含`definition`的相邻节点，并根据情况滑入或滑出这些节点。

虽然看起来一切正常，但在现有代码基础上单击不会有什么结果。因为在添加`click`处理程序的时候，词条还没有被添加到文档中。而且即使已经把`click`处理程序添加到词条元素，只要一单击其他字母，这些处理程序仍然会丢失绑定。

这是通过AJAX生成页面内容时的一个常见问题。对此，一种常见的解决方案就是在页面内容更新时重新绑定处理程序。但这样做的结果相当繁琐，因为哪怕页面的DOM结构有一点点变化，都会调用绑定处理程序的代码。

另外一种值得推荐的做法是第3章介绍的**事件委托**。在此，事件委托的本质就是把事件处理程序绑定到一个祖先元素，而这个祖先元素始终不变。对于这个例子而言，我们可以使用`.live()`方法把`click`处理程序绑定到`document`，以这种方式来捕获单击事件：

```
$(document).ready(function() {
    $('.term').live('click', function() {
        $(this).siblings('.definition').slideToggle();
    });
});
```

`.live()`方法告诉浏览器密切注意页面上发生的任何单击事件。当（且仅当）被单击的元素与`.term`选择符匹配时，才会执行事件处理程序。这样，无论单击哪个词条，都可以正常切换相应的解释，即使对应的解释内容是通过AJAX后来添加到文档中的也没有问题。

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，那我们就必须聪明一些。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');
```

实际上，`$.getScript()`方法在检测到其URL参数中包含远程主机时，就会自动采用这种技术；也就是说，该方法已经替我们想到了这一点。

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。

使用 JSONP 加载远程数据

使用`<script>`标签从远程获取JavaScript文件的思路，可以变通为从其他服务器取得JSON文件。不过，这样需要对服务器上的JSON文件稍加修改。在实现这一技术的众多解决方案中，jQuery直接支持的是JSONP（JSON with Padding，填充式JSON）。

JSONP的格式是把标准JSON文件包装在一对圆括号中，圆括号又前置一个任意字符串。这个字符串，即所谓的P（Padding，填充），由请求数据的客户端来决定。而且，由于有一对圆括号，因此返回的数据在客户端可能会导致一次函数调用，或者是为某个变量赋值——取决于客户端请求中发送的填充字符串。

^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

用PHP在服务器端实现对JSONP的支持非常简单：

```
<?php
print($_GET['callback'] . ('. $data .'));
?>
```

这里，\$data是一个包含JSON文件字符串表示的变量。调用这段脚本时，从客户端请求中取得的callback查询字符串参数，会被添加到包含JSON数据文件的圆括号前面。

为演示这一技术，需要稍微修改一下前面的JSON示例，以便调用这个远程数据源。\$.getJSON()函数利用了一个特殊的占位符?来实现了这一点。

```
$(document).ready(function() {
    var url = 'http://examples.learningjquery.com/jsonp/g.php';
    $('#letter-g a').click(function() {
        $.getJSON(url + '?callback=?', 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'];
                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);
            });
        });
        return false;
    });
});
```

正常情况下，我们是不能从远程服务器（这个例子中的examples.learningjquery.com）取得JSON数据的。但是，由于远程文件经过设置以JSONP格式提供数据，因此通过在URL后面添加一个查询字符串，并使用?作为callback参数的占位符就可以获得数据。请求返回之后，

jQuery会为我们替换?、解析结果并通过data参数将数据传入成功函数。结果就好像是在处理本地JSON数据一样。

注意，前面提到的安全注意事项在这里也适用，即服务器返回的任何结果都将在用户的计算机中执行。因此，应该只针对来自可信任站点的数据使用JSONP技术。

6.7 其他工具

jQuery的AJAX工具箱中包含的工具非常丰富，前面我们介绍的只是其中一小部分。鉴于有用的工具确实很多，下面我们仅就定制AJAX通信过程中较为重要的一些工具给出概述。

6.7.1 低级AJAX方法

前面已经介绍了一些用于启动AJAX通信的方法。但在内部，jQuery会把这些方法都映射为\$.ajax()全局函数的一种变体。这个函数不针对任何特定的AJAX通信类型，而是接收一个选项映射参数，并根据该参数来决定相应的行为。

我们介绍的第一个例子，是使用\$('#dictionary').load('a.html')加载HTML片段。同样的操作如果使用\$.ajax()来实现，应该如下所示：

```
$.ajax({
  url: 'a.html',
  type: 'GET',
  dataType: 'html',
  success: function(data) {
    $('#dictionary').html(data);
  }
});
```

在此，需要明确指定请求方法、返回的数据类型以及如何处理得到的数据。显然，这需要更多的编码工作量；可是，多一些工作量也能带来更大的灵活性。使用低级的\$.ajax()函数时，可以获得下列特殊的好处。

- 避免浏览器缓存来自服务器的响应。非常适合服务器动态生成数据的情况。
- 可以注册独立的回调函数，如针对请求完成、针对发生错误，或者针对任何情况。
- 抑制正常情况下所有AJAX交互都可以触发的全局处理程序（例如通过\$.ajaxStart()注册的处理程序）。
- 在远程主机需要认证的情况下，可以提供用户名和密码。

要了解如何利用上述及其他特性，请参考本书作者的另一本书*jQuery Reference Guide*或参考在线API (<http://docs.jquery.com/Ajax/jQuery.ajax>)。

6.7.2 修改默认选项

使用\$.ajaxSetup()函数可以修改调用AJAX方法时每个选项的默认值。这个函数与\$.ajax()接受相同的选项映射参数，之后的所有AJAX请求都将使用传递给该函数的选项——除非明确覆盖。

```

$.ajaxSetup({
    url: 'a.html',
    type: 'POST',
    dataType: 'html'
});
$.ajax({
    type: 'GET',
    success: function(data) {
        $('#dictionary').html(data);
    }
});

```

这里的操作与前面使用`$.ajax()`时实现的操作相同，不过由于已经通过`$.ajaxSetup()`为请求指定了默认的URL，因此调用`$.ajax()`时就不再需要再指定该选项了。相对而言，虽然已经把`type`参数的默认值指定为`POST`，但在`$.ajax()`调用中仍然可以覆盖这个值，将其修改为`GET`。

6.7.3 部分加载 HTML 页面

本章讨论的第一种，也是最简单的一种AJAX技术，就是取得并将HTML片段插入到当前页面中。不过，有时候服务器提供的页面中虽然包含我们需要的部分，但该部分之外的HTML却不是我们所需要的。当遇到这种服务器不能提供适当的数据格式的情况时，也可以在客户端求助于jQuery。

如果在本章第一个例子中，我们需要的字典解释包含在如下所示的完整的HTML页面中：

```

<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>The Devil's Dictionary: H</title>
    <link rel="stylesheet" href="dictionary.css"
          type="text/css" media="screen" />
</head>
<body>
    <div id="container">
        <div id="header">
            <h2>The Devil's Dictionary: H</h2>
            <div class="author">by Ambrose Bierce</div>
        </div>

        <div id="dictionary">
            <div class="entry">
                <h3 class="term">HABEAS CORPUS</h3>
                <div class="part">n.</div>
                <div class="definition">
                    A writ by which a man may be taken out of jail
                    when confined for the wrong crime.
                </div>
            </div>
        </div>
    </div>
</body>

```

```

</div>

<div class="entry">
    <h3 class="term">HABIT</h3>
    <div class="part">n.</div>
    <div class="definition">
        A shackle for the free.
    </div>
</div>
</div>
</body>
</html>

```

那么，可以通过前面编写的代码把整个文档都加载到页面中：

```

$(document).ready(function() {
    $('#letter-h a').click(function() {
        $('#dictionary').load('h.html');
        return false;
    });
});

```

不过产生的结果并不理想，因为文档中包含不需要的内容，如图6-13所示。

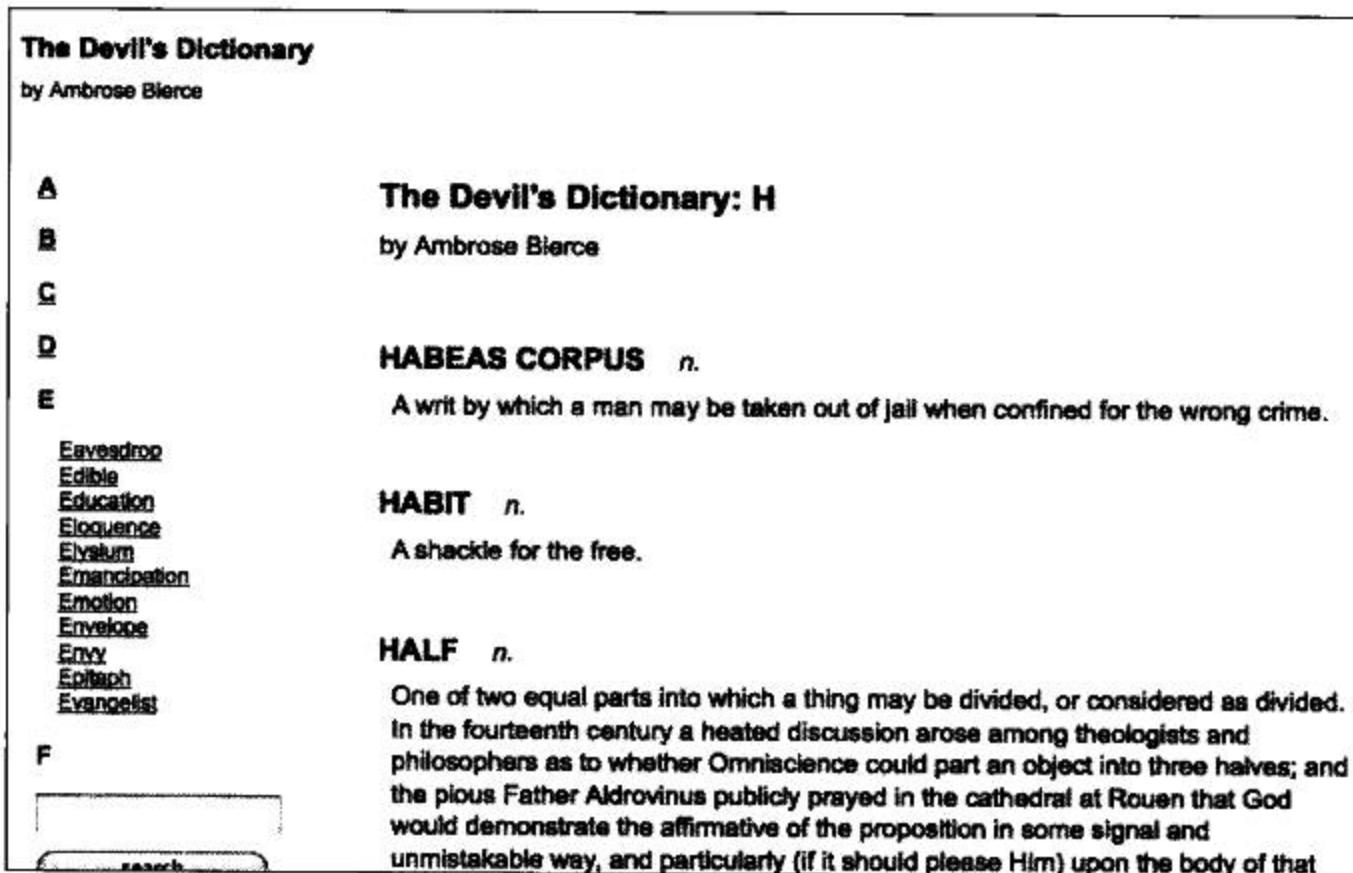


图 6-13

要去掉页面中多余的内容，可以利用`.load()`的一些新特性——在指定要加载文档的URL时，也可以提供一个jQuery选择符表达式。如果指定了这个表达式，`.load()`方法就会利用它查

找加载文档的匹配部分。最终，只有匹配的部分才会被插入到页面中。具体来说，我们可以利用这个技术，只取得文档中的词条部分，然后插入到页面中：

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

这样，文档中无关的部分已经从页面中去掉了，如图6-14所示。

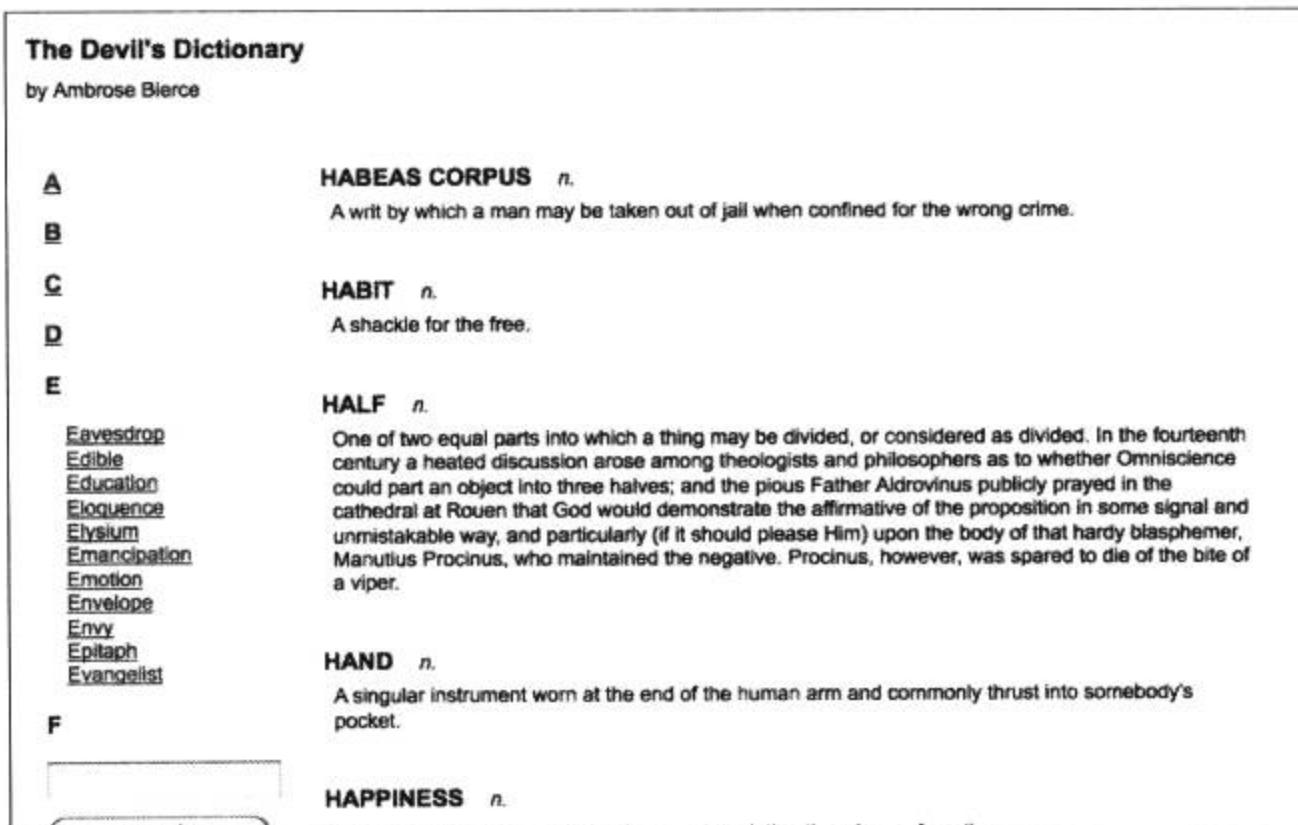


图 6-14

6.8 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。jQuery能够为我们提供帮助的地方还不止于此，后续几章还会讨论jQuery插件提供的很多特性。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



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，即以真实例子为出发点，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。

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，是如何对表格这种作为表格式数据（tabular data）语义容器的元素应用jQuery，使其更具有可读性、可用性，也更具有感染力。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/>）。

本章将讨论如下内容：

- 排序
- 分页
- 突出显示表格行
- 工具提示条
- 折叠和扩展行
- 筛选

7.1 排序和分页

操作表格式数据的两种最常见任务就是排序和分页。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很有价值的。然而，这个有用的操作实现起来也是最有技巧性的。

首先，我们来看一看如何实现对表格数据排序，或者说如何将表格数据按照对用户最有价值的顺序排列。

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（`<th>`）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">
  <thead>
    <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>
  </thead>
  <tbody>
    ...
  </tbody>
</table>
```

7

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 th a').click(function() {
    $('#my-data tbody').load($(this).attr('href'));
    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。在通过jQuery利用AJAX发送页面请求时，jQuery会将HTTP头部X-Requested-With设置为XMLHttpRequest，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元素自身，而不是包含表格的整个页面。这样，在得到响应后，客户端代码就可以直接用它来代替原有的元素。

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，结果如图7-1所示。

```
<table class="sortable">
  <thead>
    <tr>
      <th></th>
      <th>Title</th>
      <th>Author(s)</th>
      <th>Publish 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>
    </tr>
  </tbody>
</table>
```

```

<td>Douglas Paterson</td>
<td>Dec 2006</td>
<td>$40.49</td>
</tr>
</tbody>
</table>

```

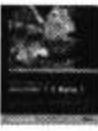
	Title	Author(s)	Publish Date	Price
	Building Websites with Joomla! 1.5 Beta 1	Hagen Graf	Feb 2007	\$40.49
	Learning Mambo: A Step-by-Step Tutorial to Building Your Website	Douglas Paterson	Dec 2006	\$40.49
	Moodle E-Learning Course Development	William Rice	May 2006	\$35.99
	AJAX and PHP: Building Responsive Web Applications	Cristian Darie, Mihai Bucica, Filip Cherecheș-Toşa, Bogdan Brinzărea	Mar 2006	\$31.49
	OpenVPN: Building and Integrating Virtual Private Networks	Markus Feilner	May 2006	\$53.99

图 7-1

我们的目标是把这个表格的列标题转换成按钮，以便按照相应的列对数据进行排序。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7-2所示。

使用这些行组标签，就可以方便地选择和操作数据行，同时不会影响表格标题。

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>

```

	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 Budica, Filip Chereches-Toga, Bogdan Brinzarea	Mar 2006	\$31.49
	OpenVPN: Building and Integrating Virtual Private Networks	Markus Feilner	May 2006	\$53.99

图 7-2

● 使用JavaScript对数组排序

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，而且可以接受一个比较函数作为参数。作为参数的比较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。

例如，有下面这个简单的数字数组：

```
var arr = [52, 97, 3, 62, 10, 63, 64, 1, 9, 3, 4];
```

通过调用`arr.sort()`可以对该数组进行排序。排序后，数组中项的顺序如下所示：

```
[1, 10, 3, 3, 4, 52, 62, 63, 64, 9, 97]
```

在默认情况下，数组项就是按照我们看到的这种以字典排序（按照字母先后顺序）的方式重排。然而，对于数字数组来说，按照数字大小排序显然更有意义。为此，可以为`.sort()`方法传递一个比较函数：

```
arr.sort(function(a,b) {
  if (a < b)
    return -1;
  if (a > b)
    return 1;
  return 0;
});
```

这个函数会在a应该排在前面时返回一个负数，在b应该排在前面时返回一个正数，而在两项的先后顺序无所谓的情况下返回零。传递这个函数之后，`.sort()`方法就可以正确地对数字项进行排序了：

```
[1, 3, 3, 4, 9, 10, 52, 62, 63, 64, 97]
```

● 使用比较函数对表格行排序

我们最初的排序程序如下所示：

```
$('document').ready(function() {
    $('table.sortable').each(function() {
        var $table = $(this);
        $('th', $table).each(function(column) {
            var $header = $(this);
            if ($header.is('.sort-alpha')) {
                $header.addClass('clickable').hover(function() {
                    $header.addClass('hover');
                }, function() {
                    $header.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()`。

有了包含DOM节点的数组后，就可以对其排序了，不过首先必须编写一个适当的比较函数。我们打算根据相应单元格中的文本内容来对表格行排序，因此比较函数应该比较单元格的文本内容。由于在内部的`.each()`调用中捕获到了列索引，所以我们知道要查看哪个单元格。之所以要把文本转换成大写，是因为JavaScript中的字符串比较区分大小写，而我们希望排序结果不区分大小写。为了减少冗余计算，我们把需要比较的键值（大写的文本内容）保存在变量中，比较它们，根据上面讨论的规则返回正数、负数或者零。

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7-3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 Bucica, Filip Chereches-Toşa, Bogdan Brinzarea	Mar 2006	\$31.49
	Alfresco Enterprise Content Management Implementation	Munwar Shariff	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 Poduval, The Hoa Nguyen, Yves Coene, Jeremy Bolle, Stany Blanvalet, Markus Zim, Matjaz Juric, Sean Carey, Michael Cardella, Kevin Geminic, Praveen Ramachandran	Jul 2006	\$40.49

图 7-3

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) {
            var $header = $(this);
            if ($header.is('.sort-alpha')) {
                $header.addClass('clickable').hover(function() {
                    $header.addClass('hover');
                }, function() {
                    $header.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7-4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 Danie, Mihai Bucica, Filip Chereches-Toşa, Bogdan Brinzarea	Mar 2006	\$31.49
		Alfresco Enterprise Content Management Implementation	Munwar Shariff	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 Poduval, The Hoa Nguyen, Yves Coene, Jeremy Boile, Stany Blanvalet, Markus Zirm, Matjaz Juric, Sean Carey, Michael Cardella, Kevin Geminiuc, Praveen Ramachandran	Jul 2006	\$40.49

图 7-4

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`。将jQuery对象作为返回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11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) {
    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;
});

```

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。



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。

7

除了使用expando属性之外，jQuery还提供了另一种数据存储机制——使用`.data()`方法可以设置或取得与页面元素关联的任何信息，而使用`.removeData()`方法则可以删除以这种方式保存的信息。

```

$.each(rows, function(index, row) {
    $(row).data('sortKey', $(row).children('td')
        .eq(column).text().toUpperCase());
});
rows.sort(function(a, b) {
    if ($(a).data('sortKey') < $(b).data('sortKey'))
        return -1;
    if ($(a).data('sortKey') > $(b).data('sortKey'))
        return 1;
    return 0;
});
$.each(rows, function(index, row) {

```

```

$stable.children('tbody').append(row);
$(row).removeData('sortKey');
});

```

有时候，使用`.data()`方法而不是`expando`属性会更方便一些，毕竟我们多数情况下操作的都是jQuery对象而不是直接操作DOM节点。而且，使用`.data()`方法还可以在IE下避免内存泄漏问题。不过，为了让读者熟悉对DOM节点和jQuery对象之间操作的转换，同时也作为一个参考，我们在这个例子中仍然使用`expando`属性。

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>

```

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7-5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 Atwal	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, Henrette Baumann, Patrick Grassie	Sep 2005	\$31.49

图 7-5

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，则首先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;
```

这里使用的正则表达式用于移除非数字和小数点的前导字符，然后把结果传给parseFloat()。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：

```
jQuery.fn.alternateRowColors = function() {  
    $('tbody tr:odd', this)  
        .removeClass('even').addClass('odd');  
    $('tbody tr:even', this)
```

```

.removeClass('odd').addClass('even');
return this;
};

$(document).ready(function() {
  $('table.sortable').each(function() {
    var $table = $(this);
    $table.alternateRowColors();
    $('th', $table).each(function(column) {
      var $header = $(this);
      var findSortKey;
      if ($header.is('.sort-alpha')) {
        findSortKey = function($cell) {
          return $cell.find('.sort-key')
            .text().toUpperCase()
            + ' ' + $cell.text().toUpperCase();
        };
      }
      else if ($header.is('.sort-numeric')) {
        findSortKey = function($cell) {
          var key = $cell.text().replace(/[^d.]/g, '');
          key = parseFloat(key);
          return isNaN(key) ? 0 : key;
        };
      }
      else if ($header.is('.sort-date')) {
        findSortKey = function($cell) {
          return Date.parse('1 ' + $cell.text());
        };
      }
      if (findSortKey) {
        $header.addClass('clickable').hover(function() {
          $header.addClass('hover');
        }, function() {
          $header.removeClass('hover');
        }).click(function() {
          var rows = $table.find('tbody > tr').get();
          $.each(rows, function(index, row) {
            var $cell = $(row).children('td').eq(column);
            row.sortKey = findSortKey($cell);
          });
          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();
});
}
});
```

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7-6和图7-7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

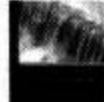
#	Title	Author(s)	Publish Date	Price
	User Training for Busy Programmers	William Rice	Jun 2005	\$11.69
	Creating your MySQL Database: Practical Design Tips and Techniques	Marc Delisle	Nov 2006	\$17.99
	Pluggable Authentication Modules: The Definitive Guide to PAM for Linux SysAdmins and C Developers	Kenneth Geissshirt	Jan 2007	\$17.99
	Invision Power Board 2: A User Guide	David Mytton	Jul 2005	\$22.49

图 7-7

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，因而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');
```

以上代码首先移除所有单元格的sorted类，然后又将该类添加到用于排序的列中。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7-8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 Atwal	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, Henriette Baumann, Patrick Grassle	Sep 2005	\$31.49

图 7-8

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 -sortDirection;
if (a.sortKey > b.sortKey) return sortDirection;
```

如果sortDirection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：

```
jQuery.fn.alternateRowColors = function() {
  $('tbody tr:odd', this)
    .removeClass('even').addClass('odd');
  $('tbody tr:even', this)
    .removeClass('odd').addClass('even');
```

```

        return this;
    };

$(document).ready(function() {
    $('table.sortable').each(function() {
        var $table = $(this);
        $table.alternateRowColors();
        $('th', $table).each(function(column) {
            var $header = $(this);
            var findSortKey;
            if ($header.is('.sort-alpha')) {
                findSortKey = function($cell) {
                    return $cell.find('.sort-key').text().toUpperCase()
                        + ' ' + $cell.text().toUpperCase();
                };
            }
            else if ($header.is('.sort-numeric')) {
                findSortKey = function($cell) {
                    var key = $cell.text().replace(/[^^\d.]/g, '');
                    key = parseFloat(key);
                    return isNaN(key) ? 0 : key;
                };
            }
            else if ($header.is('.sort-date')) {
                findSortKey = function($cell) {
                    return Date.parse('1 ' + $cell.text());
                };
            }
            if (findSortKey) {
                $header.addClass('clickable').hover(function() {
                    $header.addClass('hover');
                }, function() {
                    $header.removeClass('hover');
                }).click(function() {
                    var sortDirection = 1;
                    if ($header.is('.sorted-asc')) {
                        sortDirection = -1;
                    }
                    var rows = $table.find('tbody > tr').get();
                    $.each(rows, function(index, row) {
                        var $cell = $(row).children('td').eq(column);
                        row.sortKey = findSortKey($cell);
                    });
                    rows.sort(function(a, b) {
                        if (a.sortKey < b.sortKey) return -sortDirection;
                        if (a.sortKey > b.sortKey) return sortDirection;
                        return 0;
                    });
                    $.each(rows, function(index, row) {

```

```
$table.children('tbody').append(row);
row.sortKey = null;
});
$table.find('th').removeClass('sorted-asc')
.removeClass('sorted-desc');
if (sortDirection == 1) {
    $header.addClass('sorted-asc');
}
else {
    $header.addClass('sorted-desc');
}
$table.find('td').removeClass('sorted')
.filter(':nth-child(' + (column + 1) + ')')
.addClass('sorted');
$table.alternateRowColors();
});
}
});
});
```

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7-9所示。

Title	Author(s)	Publish Date	Price
 Microsoft AJAX C# Essentials: Building Responsive ASP.NET 2.0 Applications	Cristian Darie, Bogdan Brinzarea	Mar 2007	\$31.99
 MediaWiki Administrators' Tutorial Guide	Mizanur Rahman	Mar 2007	\$31.99
 CherryPy Essentials: Rapid Python Web Application Development	Sylvain Hellegouarch	Mar 2007	\$31.99
 Visual SourceSafe 2005 Software Configuration Management in Practice	Alexandru Serban	Feb 2007	\$44.99

图 7-9

7.1.3 服务器端分页

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。

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，有一个8行2列的表格，开始时按照第一列排序。如果随后再按照第二列排序，则很多行都会改变位置，如图7-10所示。

A	4	E	1
B	5	C	2
C	2	G	3
D	7	A	4
E	1	B	5
F	8	H	6
G	3	D	7
H	6	F	8

果是数据集中的前4行，如图7-12所示。

A	4
B	5
C	2
D	7

按第一列排序

E	1
C	2
G	3
A	4

按第二列排序

图 7-12

7.1.4 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').hide()
            .slice(currentPage * numPerPage,
                   (currentPage + 1) * numPerPage)
            .show();
    });
});
```

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，首先要隐藏所有行，然后再选择当前页中的行，最后显示选择的这些行。这里使用的`.slice()`方法的作用类似于同名的数组方法，它负责把选择的行缩减为给定的两个位置之间的元素。

在编写这种代码时最容易出错的地方，就是正确地表述传递给`.slice()`方法的表达式。在此，我们需要确定当前页面开头行和结尾行的索引。对于开头行的索引，只需用每页行数乘以当前页数即可。而用当前页数加1，然后再乘以行数，则可以得到下一页开头行的索引；`.slice()`方法会取得到第二个参数表示的索引为止，但不包含该索引处的所有行。

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——用于导航到不同数据页的一组链接。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：

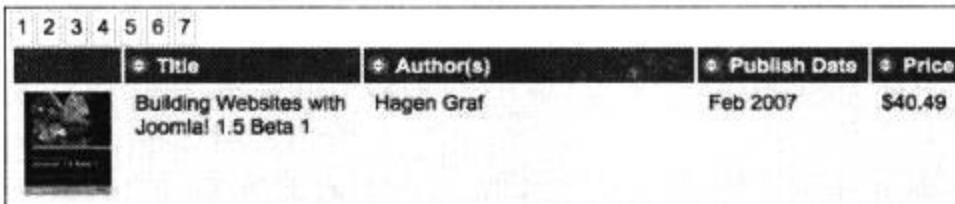
```

var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('

</div>');
for (var page = 0; page < numPages; page++) {
    $('</span>').text(page + 1)
        .appendTo($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 7-13 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	

图 7-13

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').hide()
                .slice(currentPage * numPerPage,
                    (currentPage + 1) * numPerPage)
                .show();
        };
        var numRows = $table.find('tbody tr').length;
        var numPages = Math.ceil(numRows / numPerPage);
        var $pager = $('

</div>');
        for (var page = 0; page < numPages; page++) {
            $('</span>').text(page + 1)
                .click(function() {
                    currentPage = page;
                    repaginate();
                }).appendTo($pager).addClass('clickable');
        }
        $pager.insertBefore($table);
    });
});


```

这些代码多数是有效的，即当页面加载时和单击其中任何链接时都会调用新的repaginate()函数。可是，单击每个链接打开的分页中都没有数据行，如图7-14所示。



图 7-14

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自定义事件数据，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：

```
$('<span class="page-number"></span>').text(page + 1)
  .bind('click', {newPage: page}, function(event) {
    currentPage = event.data['newPage'];
    repaginate();
}).appendTo($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7-15所示。

◆ Title	◆ Author(s)	◆ Publish Date	◆ Price
	PHPEclipse: A User Guide	Shu-Wai Chow	Feb 2006 \$31.49
	Alfresco Enterprise Content Management Implementation	Munwar Shariff	Jan 2007 \$53.99
	Smarty PHP Template Programming and Applications	Joao Prado Maia, Hasin Hayder, Lucian Gheorghe	Apr 2006 \$35.99
	JasperReports for Java Developers	David Heffelfinger	Aug 2006 \$40.49

图 7-15

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：

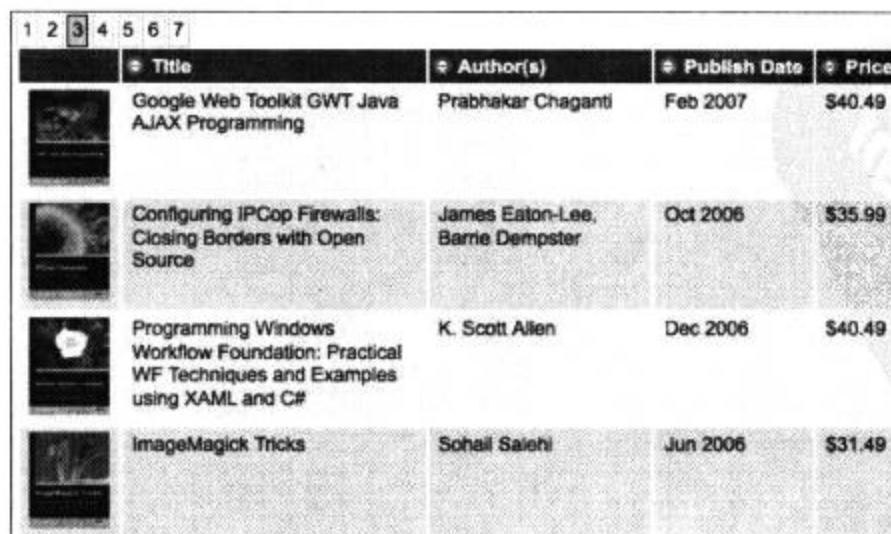
```

$(document).ready(function() {
    $('table.paginated').each(function() {
        var currentPage = 0;
        var numPerPage = 10;
        var $table = $(this);
        var repaginate = function() {
            $table.find('tbody tr').hide()
                .slice(currentPage * numPerPage,
                    (currentPage + 1) * numPerPage)
                .show();
        };
        var numRows = $table.find('tbody tr').length;
        var numPages = Math.ceil(numRows / numPerPage);
        var $pager = $('

</div>');
        for (var page = 0; page < numPages; page++) {
            $('</span>').text(page + 1)
                .bind('click', {newPage: page}, function(event) {
                    currentPage = event.data['newPage'];
                    repaginate();
                    $(this).addClass('active')
                        .siblings().removeClass('active');
                }).appendTo($pager).addClass('clickable');
        }
        $pager.insertBefore($table)
            .find('span.page-number:first').addClass('active');
    });
});


```

这样，分页指示器上就会突出地显示出当前页码了，如图7-16所示。



The screenshot shows a table with a header row containing columns for Title, Author(s), Publish Date, and Price. Below the header are four data rows, each with a small thumbnail image on the left and detailed information on the right. Above the table is a navigation bar with page numbers 1, 2, 3, 4, 5, 6, and 7. The number 3 is highlighted with a black border, indicating it is the current page.

#	Title	Author(s)	Publish Date	Price
1	Google Web Toolkit GWT Java AJAX Programming	Prabhakar Chaganti	Feb 2007	\$40.49
2	Configuring IPCop Firewalls: Closing Borders with Open Source	James Eaton-Lee, Barrie Dempster	Oct 2006	\$35.99
3	Programming Windows Workflow Foundation: Practical WF Techniques and Examples using XAML and C#	K. Scott Allen	Dec 2006	\$40.49
4	ImageMagick Tricks	Sohail Salehi	Jun 2006	\$31.49

图 7-16

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').hide()
        .slice(currentPage * numPerPage,
            (currentPage + 1) * numPerPage)
        .show();
});
```

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。

7.1.5 完成的代码

完成后的全部排序和分页代码如下所示：

```
jQuery.fn.alternateRowColors = function() {
    $('tbody tr:odd', this)
        .removeClass('even').addClass('odd');
    $('tbody tr:even', this)
        .removeClass('odd').addClass('even');
    return this;
};

$(document).ready(function() {
    $('table.sortable').each(function() {
        var $table = $(this);
        $table.alternateRowColors();
```

^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

```

$( 'th', $table ).each(function(column) {
    var $header = $(this);
    var findSortKey;
    if ($header.is('.sort-alpha')) {
        findSortKey = function($cell) {
            return $cell.find('.sort-key').text().toUpperCase()
                + ' ' + $cell.text().toUpperCase();
        };
    }
    else if ($header.is('.sort-numeric')) {
        findSortKey = function($cell) {
            var key = $cell.text().replace(/[^^\d.]*/, '');
            key = parseFloat(key);
            return isNaN(key) ? 0 : key;
        };
    }
    else if ($header.is('.sort-date')) {
        findSortKey = function($cell) {
            return Date.parse('1 ' + $cell.text());
        };
    }
    if (findSortKey) {
        $header.addClass('clickable').hover(function() {
            $header.addClass('hover');
        }, function() {
            $header.removeClass('hover');
        }).click(function() {
            var sortDirection = 1;
            if ($header.hasClass('.sorted-asc')) {
                sortDirection = -1;
            }
            var rows = $table.find('tbody > tr').get();
            $.each(rows, function(index, row) {
                var $cell = $(row).children('td').eq(column);
                row.sortKey = findSortKey($cell);
            });
            rows.sort(function(a, b) {
                if (a.sortKey < b.sortKey) return -sortDirection;
                if (a.sortKey > b.sortKey) return sortDirection;
                return 0;
            });
            $.each(rows, function(index, row) {
                $table.children('tbody').append(row);
                row.sortKey = null;
            });
            $table.find('th').removeClass('sorted-asc')
                .removeClass('sorted-desc');
            if (sortDirection == 1) {
                $header.addClass('sorted-asc');
            }
            else {
        
```

```

        $header.addClass('sorted-desc');
    }
    $table.find('td').removeClass('sorted')
      .filter(':nth-child(' + (column + 1) + ')')
      .addClass('sorted');
    $table.alternateRowColors();
    $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').hide()
        .slice(currentPage * numPerPage,
               (currentPage + 1) * numPerPage)
        .show();
    });
    var numRows = $table.find('tbody tr').length;
    var numPages = Math.ceil(numRows / numPerPage);
    var $pager = $('

</div>');
    for (var page = 0; page < numPages; page++) {
      $('</span>').text(page + 1)
        .bind('click', {newPage: page}, function(event) {
          currentPage = event.data['newPage'];
          $table.trigger('repaginate');
          $(this).addClass('active')
            .siblings().removeClass('active');
        }).appendTo($pager).addClass('clickable');
    }
    $pager.insertBefore($table)
      .find('span.page-number:first').addClass('active');
  });
});
});


```

7.2 修改表格外观

前面我们已经介绍了一些对表格中的数据进行排序，以便用户查找想要的信息的方式。不过，在分页或排序之后，仍然会有一些数据不容易被发现。此时，除了从显示表格行的顺序和数量方面为用户提供辅助之外，还可以从这些行的外观上面作文章。

7.2.1 突出显示行

通过突出显示表格行来引导用户的视线，是一种提示哪些数据重要的有效方式。要展示突出显示表格的方式，必须有一个表格作为示例。这次，我们使用一个包含新数据项的表格，它比前

面的表格稍微复杂一点——除了包含主标题行之外，这个表格还包含了一些用作子标题的行。这个示例表格的HTML结构如下所示：

```
<table>
  <thead>
    <tr>
      <th>Date</th>
      <th>Headline</th>
      <th>Author</th>
      <th>Topic</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th colspan="4">2008</th>
    </tr>
    <tr>
      <td>Sep 28</td>
      <td>jQuery, Microsoft, and Nokia</td>
      <td>John Resig</td>
      <td>third-party</td>
    </tr>
    ...
    <tr>
      <td>Jan 15</td>
      <td>jQuery 1.2.2: 2nd Birthday Present</td>
      <td>John Resig</td>
      <td>release</td>
    </tr>
  </tbody>
  <tbody>
    <tr>
      <th colspan="4">2007</th>
    </tr>
    <tr>
      <td>Dec 8</td>
      <td>jQuery Plugins site updated</td>
      <td>Mike Hostetler</td>
      <td>announcement</td>
    </tr>
    ...
    <tr>
      <td>Jan 11</td>
      <td>Selector Speeds</td>
      <td>John Resig</td>
      <td>source</td>
    </tr>
  </tbody>
  ...
</table>
```

注意，这个表格中包含多个`<tbody>`元素，而且这也是用于对行进行分组的一种有效的HTML标记方式。在此，我们把部分标题分别放在了各自的组中，并使用`<th>`标记以示区分。应用一些基本的CSS样式之后，这个表格将如图7-17所示。

Date	Headline	Author	Topic
2008			
Sep 28	jQuery, Microsoft, and Nokia	John Resig	third-party
Aug 31	jQuery Conference 2008 Agenda	Rey Bango	conference
Aug 29	jQuery.com Site Redesign	John Resig	announcement
Aug 15	Registration Open for jQuery Conference 2008	Karl Swedberg	conference
Jul 14	jQuery UI 1.5.2	Paul Bakaus	release
Jun 26	jQuery UI 1.5.1	Paul Bakaus	release
Jun 26	jQuery Camp 2008 Announced	Rey Bango	conference
Jun 9	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	release
Jun 4	jQuery 1.2.6: Events 100% faster	John Resig	release
Mar 7	jQuery UI Worldwide Sprint: March 14-15	Richard Worth	conference
Feb 8	jQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	release
Jan 23	jQuery UI and beyond: The jQuery-Liferay partnership	Paul Bakaus	announcement
Jan 15	jQuery 1.2.2: 2nd Birthday Present	John Resig	release
2007			
Dec 8	jQuery Plugins site updated	Mike Hostetler	announcement
Dec 6	Flot, a new plotting plugin for jQuery	Bradley Sepos	plug-in
Nov 2	Google Using jQuery	Rey Bango	third-party

图 7-17

1. 行条纹效果

在本章前面以及第2章，我们已经看到过突出显示行的简单例子了。为表格行应用条纹效果，是在多列间精确引导用户视线的一种常用方法。

如前所见，实现最简单的行条纹效果的方式，就是分别为奇偶行应用不同的类：

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

尽管对简单的表格结构来说，以上代码可以胜任，但如果表格中包含我们不想应用条纹效果的行（例如示例表格中用于表示年份的子标题行），则基本的奇偶模式就不适用了。例如，假设2006年这一行被标记为even（偶数行），则该行前后的两行一定都是odd（奇数行），但基于此应用样式的结果却是不可取的，如图7-18所示。

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

图 7-18

为此，可以利用我们在第2章介绍的`:nth-child()`伪类，来保证从年份行开始重新应用交替样式模式，如图7-19所示。

```

$(document).ready(function() {
    $('table.striped tr:nth-child(odd)').addClass('odd');
    $('table.striped tr:nth-child(even)').addClass('even');
});

```

January, 2006			
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

图 7-19

这样，每一组行的第一行都会被标记为odd；不过，子标题行并没有被排除在外。因此，还需要再使用`:has()`伪类来排除子标题行，结果如图7-20所示：

```

$(document).ready(function() {
    $('table.striped tr:not(:has(th)):odd').addClass('odd');
    $('table.striped tr:not(:has(th)):even').addClass('even');
});

```

January, 2006			
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

图 7-20

虽然排除了子标题行，但行组是以odd还是以even行开头，却要取决于对前一行添加了哪个类。要解决这两个问题确实需要一些技巧性，而使用`.each()`来引入显式迭代则是一个很直观的做法：

```

$(document).ready(function() {
    $('table.striped tbody').each(function() {
        $(this).find('tr:not(:has(th)):odd').addClass('odd');
        $(this).find('tr:not(:has(th)):even').addClass('even');
    });
});

```

现在，每个行组都相互没有依赖地应用了条纹效果，而且子标题行也被排除在外了，如图7-21所示。

January, 2006			
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

图 7-21

2. 高级行条纹效果

针对奇偶行的操作已经为我们奠定了学习更复杂技术的基础。对于数据量特别大的表格而

言，即使是交替变换行的颜色也不会产生多大辅助作用。此时，以更多的行数为单位交替变换颜色会更好一些。下面，我们就以前面的表格为例，以3行为单位交替应用不同的颜色。

第2章曾经介绍了通过`.filter()`方法以非常灵活的方法选择页面元素的技术。我们知道，`.filter()`不仅可以接受一个选择符表达式，也可以接收一个筛选函数，例如：

```
$(document).ready(function() {
    $('table.striped tbody').each(function() {
        $(this).find('tr:not(:has(th))').filter(function(index) {
            return (index % 6) < 3;
        }).addClass('odd');
    });
});
```

这几行代码实现了很多操作，因此下面我们就来逐行分析它。

首先，像前面一样使用`.each()`方法按照行组来分割要操作的表格行。我们希望3行条纹效果在每个子标题之后都重新开始，因此这样就可以每次只处理一个行组。然后，像在上一个例子中一样，使用`.find()`找到所有不包含`<th>`元素（因此不包含子标题）的表格行。

然后，需要选择当前集合中的前3个元素，再跳过3个元素，依此类推。这时就需要用到`.filter()`方法了，其筛选函数接受的参数是匹配元素集合中项的索引。这个索引也就是要处理的表格部分中的行号。此时，当且仅当筛选函数返回`true`的情况下，当前元素才会保留在集合中。

求模运算符（%）为我们提供了必要的信息。表达式`index % 6`的结果是用行号除以6得到的余数，如果余数是0、1或2，则把相应行标记为`odd`；如果余数是3、4或5，则把相应行标记为`even`。

不过，上面的代码只会标记`odd`行。为了应用`even`类，可能需要编写另一个对应的筛选函数，或者也可以发挥一点创造力：

```
$(document).ready(function() {
    $('table.striped tbody').each(function() {
        $(this).find('tr:not(:has(th))').addClass('even')
            .filter(function(index) {
                return (index % 6) < 3;
            }).removeClass('even').addClass('odd');
    });
});
```

在此，我们首先为所有行都应用了`even`类，然后在需要添加`odd`类时再移除`even`类。于是，表格就按照我们的期望以3行交替一次样式的形式出现了。而且，针对表格中的每个部分都会重新开始应用样式，如图7-22所示。

3. 交互式突出显示行

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`Author`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;
}
```

Date	Headline	Author	Topic
2006			
Sep 28	jQuery, Microsoft, and Nokia	John Resig	third-party
Aug 31	jQuery Conference 2006 Agenda	Rey Bango	conference
Aug 29	jQuery.com Site Redesign	John Resig	announcement
Aug 15	Registration Open for jQuery Conference 2006	Karl Swedberg	conference
Jul 14	jQuery UI 1.5.2	Paul Bakaus	release
Jun 26	jQuery UI 1.5.1	Paul Bakaus	release
Jun 26	jQuery Camp 2006 Announced	Rey Bango	conference
Jun 9	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	release
Jun 4	jQuery 1.2.6: Events 100% faster	John Resig	release
Mar 7	jQuery UI Worldwide Sprint: March 14-15	Richard Worth	conference
Feb 8	jQuery 1.2.3: A/R, Namespacing, and UI Alpha	John Resig	release
Jan 23	jQuery UI and beyond: The jQuery-Liferay partnership	Paul Bakaus	announcement
Jan 15	jQuery 1.2.2: 2nd Birthday Present	John Resig	release
2007			
Dec 8	jQuery Plugins site updated	Mike Hostetler	announcement
Dec 6	Fiot, a new plotting plugin for jQuery	Bradley Sepos	plug-in
Nov 2	Google Using jQuery	Rey Bango	third-party
Sep 17	jQuery UI: Interactions and Widgets	John Resig	announcement
Sep 10	jQuery 1.2: jQuery.extend("Awesome")	John Resig	release
Sep 6	jQueryCamp '07 (Boston)	John Resig	conference
Aug 24	jQuery 1.1.4: Faster, More Tests, Ready for 1.2	John Resig	release
Jul 17	SF jQuery Meetup and Aux Experience	John Resig	conference

图 7-22

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使用.click()方法为其添加行为：

```
$ (document).ready(function() {
    var $authorCells = $('table.striped td:nth-child(3)');
    $authorCells.click(function() {
        // 突出显示行的代码。
    });
});
```

7

我们注意到，选择符表达式中包含了:nth-child(n)伪类选择符，该选择符指向作者(Author)信息所在的第三列。为防止以后表格结构有较大变化，我们想让常量3只在代码中的一个地方出现，以便于更新。同时，考虑到效率，我们又把选择符的结果保存在变量\$authorCells中，而不是每次在需要时再重复使用选择符。



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：

^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

```

$(document).ready(function() {
    var $authorCells = $('table.striped td:nth-child(3)');
    $authorCells.click(function() {
        var authorName = $(this).text();
        $authorCells.each(function(index) {
            if (authorName == $(this).text()) {
                $(this).parent().toggleClass('highlight');
            }
        });
    });
});

```

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。当我们单击第二个作者的名字时，这些代码并没有像我们想象的那样，把突出显示转换到包含第二个作者的行，而是为包含第二个作者的行也添加了highlight类。为了避免这种行为，可以向代码中添加一条else语句，移除被单击的作者名字所在行之外其他行的highlight类：

```

$(document).ready(function() {
    var $authorCells = $('table.striped td:nth-child(3)');
    $authorCells.click(function() {
        var authorName = $(this).text();
        $authorCells.each(function(index) {
            if (authorName == $(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7-23所示。

Date	Headline	Author	Topic
2008			
Sep 28	jQuery, Microsoft, and Nokia	John Resig	third-party
Aug 31	jQuery Conference 2008 Agenda	Rey Bango	conference
Aug 29	jQuery.com Site Redesign	John Resig	announcement
Aug 15	Registration Open for jQuery Conference 2008	Karl Swedberg	conference
Jul 14	jQuery UI 1.5.2	Paul Bakaus	release
Jun 26	jQuery UI 1.5.1	Paul Bakaus	release
Jun 26	jQuery Camp 2008 Announced	Rey Bango	conference
Jun 9	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	release
Jun 4	jQuery 1.2.6: Events 100% faster	John Resig	release
Mar 7	jQuery UI Worldwide Sprint: March 14-15	Richard Worth	conference
Feb 8	jQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	release
Jan 23	jQuery UI and beyond: The jQuery-Liferay partnership	Paul Bakaus	announcement
Jan 15	jQuery 1.2.2: 2nd Birthday Present	John Resig	release
2007			
Dec 8	jQuery Plugins site updated	Mike Hostetler	announcement
Dec 6	Flot, a new plotting plugin for jQuery	Bradley Sebos	plug-in
Nov 2	Google Using jQuery	Rey Bango	third-party
Sep 17	jQuery UI: Interactions and Widgets	John Resig	announcement

图 7-23

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。

7.2.2 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，以便当用户的鼠标指针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() {
    var $authorCells = $('table.striped td:nth-child(3)');
    $authorCells
        .addClass('clickable')
        .click(function() {
            var authorName = $(this).text();
            $authorCells.each(function(index) {
                if (authorName == $(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生什么事情。就页面用户而言，他可能只知道单击会触发另一种行为，例如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。为了给我们的例子添加一些可用性，可以让用户鼠标指针悬停于Author单元格上时，显示一个工具提示条。工具提示条的文本可以告诉用户单击之后会发生什么，例如可以将消息文本设置为“Highlight all articles by Rey Bango”（单击突出显示Rey Bango的所有文章）。可以将这些文本包含在

中，然后再把这个

添加到body中。而我们可以在脚本中通过\$tooltip变量来引用这个新创建的元素。

```
var $tooltip = $('

</div>').appendTo('body');


```

围绕这个工具提示条，必须重复执行3个基本操作：

- (1) 当鼠标指针悬停在交互元素上面时，显示工具提示条；
- (2) 当鼠标指针离开交互元素时，隐藏工具提示条；
- (3) 当鼠标指针在交互元素上移动时，重新定位工具提示条。

我们会先针对每个操作编写一个函数，然后再通过jQuery将这几个函数与浏览器事件关联起来。

下面首先来编写positionTooltip()函数，当鼠标指针移动到Author单元格上时需要引用这个函数：

```
var positionTooltip = function(event) {
    var tPosX = event.pageX;
    var tPosY = event.pageY + 20;
    $tooltip.css({top: tPosY, left: tPosX});
};
```

这里使用了event对象的pageX和pageY属性来设置工具提示条的left和top位置。在响应mousemove等鼠标事件并调用这个函数时，event.pageX和event.pageY会为我们提供鼠标指针所在位置的坐标值，因此tPosX和tPosY指定的就是位于鼠标指针位置以下20像素的点。

接下来要编写的是showTooltip()函数，该函数用于把工具提示条显示在屏幕上。

```
var showTooltip = function(event) {
    var authorName = $(this).text();
    $tooltip
        .text('Highlight all articles by ' + authorName)
        .show();
    positionTooltip(event);
};
```

showTooltip()函数的代码也非常简单：先把当前单元格的内容提取为字符串（也就是作者的名字），并把该字符串设置为工具提示条的内容，然后再显示工具提示条。

显示工具提示条之后，再调用positionTooltip()函数将它放在适当的位置上。由于此时已经将工具提示条添加到了body元素，因此还需要通过一些CSS来让它浮动在页面以上正确的层次上：

```
#tooltip {
    position: absolute;
    z-index: 2;
    background: #efd;
    border: 1px solid #ccc;
    padding: 3px;
}
```

最后，我们来编写简单的hideTooltip函数：

```
var hideTooltip = function() {
    $tooltip.hide();
};
```

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 $authorCells = $('table.striped td:nth-child(3)');
    var $tooltip = $('

</div>').appendTo('body');
    var positionTooltip = function(event) {
        var tPosX = event.pageX;
        var tPosY = event.pageY + 20;
        $tooltip.css({top: tPosY, left: tPosX});
    };
    var showTooltip = function(event) {


```

```

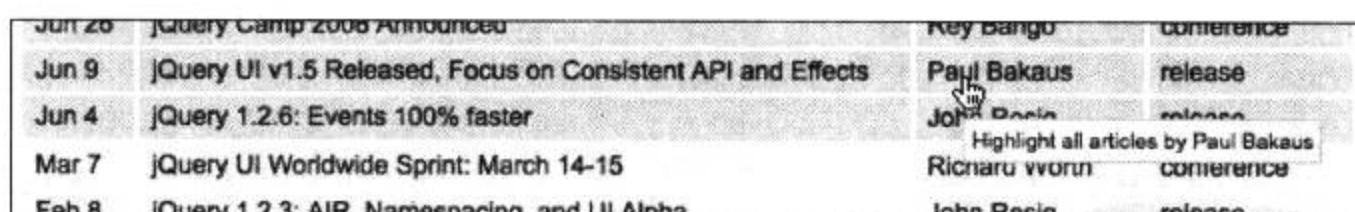
var authorName = $(this).text();
$tooltip
    .text('Highlight all articles by ' + authorName)
    .show();
positionTooltip(event);
};

var hideTooltip = function() {
    $tooltip.hide();
};

$authorCells
    .addClass('clickable')
    .hover(showTooltip, hideTooltip)
    .mousemove(positionTooltip)
    .click(function(event) {
        var authorName = $(this).text();
        $authorCells.each(function(index) {
            if (authorName == $(this).text()) {
                $(this).parent().toggleClass('highlight');
            }
            else {
                $(this).parent().removeClass('highlight');
            }
        });
    });
});

```

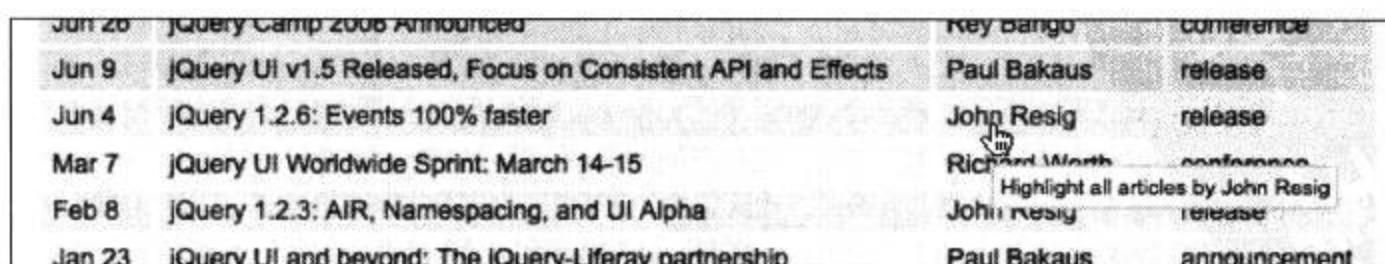
我们注意到，`.hover()`和`.mousemove()`方法的参数引用了在其他地方定义的函数。因此，必须省略调用函数时才需要在后面添加的一对括号。现在，当鼠标指针移动到作者单元格上面时，工具提示条就会显示出来，而且它能够随着鼠标指针移动而移动，并在鼠标指针离开单元格时消失，如图7-24所示。



Jun 26	jQuery Camp 2008 Announced	Key Bang	conference
Jun 9	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	release
Jun 4	jQuery 1.2.6: Events 100% faster	John Resig	release
Mar 7	jQuery UI Worldwide Sprint: March 14-15	Richard Wetherholt	conference
Feb 8	iQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	release

图 7-24

现在的问题是，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7-25所示。



Jun 26	jQuery Camp 2008 Announced	Key Bang	conference
Jun 9	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	release
Jun 4	jQuery 1.2.6: Events 100% faster	John Resig	release
Mar 7	jQuery UI Worldwide Sprint: March 14-15	Richard Wetherholt	conference
Feb 8	iQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	release
Jan 23	iQuery UI and beyond: The iQuery-Liferay partnership	Paul Bakaus	announcement

图 7-25

这里，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|元素包含highlight类，就使用Unhighlight来代替Highlight：
| |

```
var action = 'Highlight';
if ($(this).parent().is('.highlight')) {
    action = 'Unhighlight';
}
$tooltip
    .text(action + ' all articles by ' + authorName)
    .show();
```

虽然这样能够确保工具提示条的文本在鼠标指针进入单元格时有意义，但同样也需要在鼠标单击时重新生成其文本内容。为此，需要在click事件处理程序内部调用showTooltip()函数：

```
$(document).ready(function() {
    var $authorCells = $('table.striped td:nth-child(3)');
    var $tooltip = $('

</div>').appendTo('body');

    var positionTooltip = function(event) {
        var tPosX = event.pageX;
        var tPosY = event.pageY + 20;
        $tooltip.css({top: tPosY, left: tPosX});
    };

    var showTooltip = function(event) {
        var authorName = $(this).text();
        var action = 'Highlight';
        if ($(this).parent().is('.highlight')) {
            action = 'Unhighlight';
        }
        $tooltip
            .text(action + ' all articles by ' + authorName)
            .show();
        positionTooltip(event);
    };

    var hideTooltip = function() {
        $tooltip.hide();
    };

    $authorCells
        .addClass('clickable')
        .hover(showTooltip, hideTooltip)
        .mousemove(positionTooltip)
        .click(function(event) {
            var authorName = $(this).text();
            $authorCells.each(function(index) {
                if (authorName == $(this).text()) {
                    $(this).parent().toggleClass('highlight');
                }
            })
        })
    });
});


```

```

    else {
        $(this).parent().removeClass('highlight');
    }
});
showTooltip.call(this, event);
});
});

```

通过使用JavaScript内置的call()函数，就可以在作者单元格的click事件处理程序的作用域中，调用并运行showTooltip()函数。这样一来，在执行showTooltip()期间，this关键字引用的才是正确的对象（当前的作者单元格）。

现在，当鼠标指针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7-26所示。

Sep 28	jQuery, Microsoft, and Nokia	John Resig	third-party
Aug 31	jQuery Conference 2008 Agenda	Rey Bangs	conference
Aug 29	jQuery.com Site Redesign	John Resig	announcement
Aug 15	Registration Open for jQuery Conference 2008	Kyle Simpson	Unhighlight all articles by Rey Bangs
Jul 14	jQuery UI 1.5.2	Paul Bakaus	release
Jun 26	jQuery UI 1.5.1	Paul Bakaus	release
Jun 26	jQuery Chrome 2008 Announcement	Rey Bangs	conference

图 7-26

7.2.3 折叠和扩展

在把大量数据分成多个部分的情况下，如果能够隐藏与当前部分无关的信息会非常有用。前面示例中使用的新闻表格是以年份对行分组的。如果能够折叠（或隐藏）一年的新闻，就不必过多滚动页面即可查看到表格中包含的全部数据。

要实现新闻表格的折叠功能，首先需要创建用来触发此行为的页面元素。可折叠项的一个标准界面元素就是减号图标，而对应的加号图标可以用来扩展项。下面，我们将本着渐进增强的原则，通过JavaScript来插入这两个图标：

```

$(document).ready(function() {
    var collapseIcon = '../images/bullet_toggle_minus.png';
    var collapseText = 'Collapse this section';
    var expandIcon = '../images/bullet_toggle_plus.png';
    var expandText = 'Expand this section';
    $('table.collapsible tbody').each(function() {
        var $section = $(this);
        $('

```

函数开头的几个变量中既保存了图标的位置，也保存了它们的替代文本。这样不仅便于后面的代码引用，也可以确保修改起来简单。接着，我们使用.each()循环插入了图像，而通过把<tbody>元素的引用保存在\$section变量中，也为后面引用<tbody>提供了便利。

接下来，需要让这两个图标能够分别触发表格行的折叠和扩展行为。新添加的clickable类用于为用户提供反馈，另一个添加到`<tbody>`的类则用于代码识别该元素包含的行当前是否可见。

```
$('document').ready(function() {
    var collapseIcon = '../images/bullet_toggle_minus.png';
    var collapseText = 'Collapse this section';
    var expandIcon = '../images/bullet_toggle_plus.png';
    var expandText = 'Expand this section';
    $('table.collapsible tbody').each(function() {
        var $section = $(this);
        $('').attr('src', collapseIcon)
            .attr('alt', collapseText)
            .prependTo($section.find('th'))
            .addClass('clickable')
            .click(function() {
                if ($section.is('.collapsed')) {
                    $section.removeClass('collapsed')
                        .find('tr:not(:has(th))').fadeIn('fast');
                    $(this).attr('src', collapseIcon)
                        .attr('alt', collapseText);
                }
                else {
                    $section.addClass('collapsed')
                        .find('tr:not(:has(th))').fadeOut('fast');
                    $(this).attr('src', expandIcon)
                        .attr('alt', expandText);
                }
            });
    });
});
});
```

在单击事件发生时，代码将执行下列操作：

- (1) 添加或移除`<tbody>`元素的`collapsed`类，以便识别相应表格区域的可见状态；
- (2) 找到不包含相应部分标题以下的所有行，通过淡入/淡出动画效果来显示或隐藏这些行；
- (3) 切换图标的当前状态，即修改其`src`和`alt`属性，以反映单击它以后将会触发的操作。

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7-27所示。

Jan 23	jQuery UI and beyond: The jQuery-Liferay partnership	John Resig	release
Jan 15	jQuery 1.2.2: 2nd Birthday Present	John Resig	announcement
往 2007			
往 2006			
Dec 27	The Path to 1.1	John Resig	source
Dec 18	Meet The People Behind jQuery	John Resig	announcement
Dec 13	Holman and understand jQuery	John Resig	tutorial

图 7-27

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。如果想要动画效果，也可以使用.fadeIn()和.fadeOut()方法。

7.2.4 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，需要创建放置筛选链接的元素。根据渐进增强原则，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 = $('

</div>');
                $('

### </h3>') .text('Filter by ' + $(this).text() + ':') .appendTo($filters); $filters.insertBefore($table); } }); }); });


```

7

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7-28所示。

Date	Headline	Author	Topic	Filter by Topic:
2008				
Sep 28	jQuery, Microsoft, and Nokia	John Resig	third-party	
Aug 31	jQuery Conference 2008 Agenda	Rey Bango	conference	
Aug 29	jQuery.com Site Redesign	John Resig	announcement	
Aug 15	Registration Open for jQuery Conference 2008	Karl Swedberg	conference	
Jul 14	iQuery UI 1.5.2	Paul Bakaus	release	

图 7-28

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链接。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：

```
$('document').ready(function() {
    $('table.filterable').each(function() {
        var $table = $(this);

        $table.find('th').each(function(column) {
            if ($(this).is('.filter-column')) {
                var $filters = $('

</div>');
                $('

### </h3>') .text('Filter by ' + $(this).text() + ':') .appendTo($filters); var keywords = ['conference', 'release']; $.each(keywords, function(index, keyword) { $(' </div>').text(keyword) .bind('click', {key: keyword}, function(event) { $('tr:not(:has(th))', $table).each(function() { var value = $('td', this).eq(column).text(); if (value == event.data['key']) { $(this).show(); } else { $(this).hide(); } }); $(this).addClass('active') .siblings().removeClass('active'); }).addClass('clickable').appendTo($filters); }); $filters.insertBefore($table); } }); }); });


```

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筛选链接。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，以避免意外的闭包导致的问题。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。由于我们使用的行选择符会排除包含`<th>`的行，因此不必担心也会将子标题隐藏起来。

现在这两个链接都实现了相应的筛选功能，如图7-29所示。

- 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 = {};
$table.find('td:nth-child(' + (column + 1) + ')')
.each(function() {
  keywords[$(this).text()] = $(this).text();
});

```

Date	Headline	Author	Topic	Filter by Topic:
≡ 2008				
Aug 31	jQuery Conference 2008 Agenda	Rey Bango	conference	<input checked="" type="checkbox"/> conference
Aug 15	Registration Open for jQuery Conference 2008	Karl Swedberg	conference	<input type="checkbox"/> release
Jun 26	jQuery Camp 2008 Announced	Rey Bango	conference	
Mar 7	jQuery UI Worldwide Sprint: March 14-15	Richard Worth	conference	
≡ 2007				

图 7-29

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。最重要的是，每个键只能有一个值，而且键也始终唯一。
 - 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7-30所示。

Date	Headline	Author	Topic	Filter by Topic:
≡ 2008				
Sep 28	jQuery, Microsoft, and Nokia	John Resig	third-party	<input checked="" type="checkbox"/> third-party
≡ 2007				
Nov 2	Google Using jQuery	Rey Bango	third-party	<input type="checkbox"/> conference
Feb 20	jQuery and Jack Slocum's Ext	John Resig	third-party	<input type="checkbox"/> announcement
≡ 2006				
Oct 25	Friends of Firefox - Mozilla Utilizes jQuery	Will Jessup	third-party	<input type="checkbox"/> release
≡ 2005				
Dec 14	Google Homepage API	John Resig	third-party	<input type="checkbox"/> plug-in
Dec 12	Sparklines with Javascript and Canvas	John Resig	third-party	<input type="checkbox"/> standards

图 7-30

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链接，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。如图7-31所示。

Date	Headline	Author	Topic	Filter by Topic:
2008				
Sep 28	jQuery, Microsoft, and Nokia	John Resig	third-party	<input checked="" type="checkbox"/> all
Aug 31	jQuery Conference 2008 Agenda	Rey Bango	conference	<input type="checkbox"/> third-party
Aug 29	jQuery.com Site Redesign	John Resig	announcement	<input type="checkbox"/> conference
Aug 15	Registration Open for jQuery Conference 2008	Karl Swedberg	conference	<input type="checkbox"/> announcement
Jul 14	jQuery UI 1.5.2	Paul Bakaus	release	<input type="checkbox"/> release
Jun 26	jQuery UI 1.5.1	Paul Bakaus	release	<input type="checkbox"/> plug-in
Jun 26	jQuery Camp 2008 Announced	Rey Bango	conference	<input type="checkbox"/> standards
Jun 9	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	release	<input type="checkbox"/> documentation
Jun 4	jQuery 1.2.6: Events 100% faster	John Resig	release	<input type="checkbox"/> tutorial
				<input type="checkbox"/> miscellaneous
				<input type="checkbox"/> source

图 7-31

3. 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。

- 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。此时，可以使用jQuery伪类:`:visible`来正确地收集带有条纹效果的行。在作出这一修改后，接下来应该准备为表格行应用条纹效果的代码，同时创建一个自定义的事件类型，以便从其他地方调用这些代码——就跟我们将排序和分页功能整合到一起时一样。

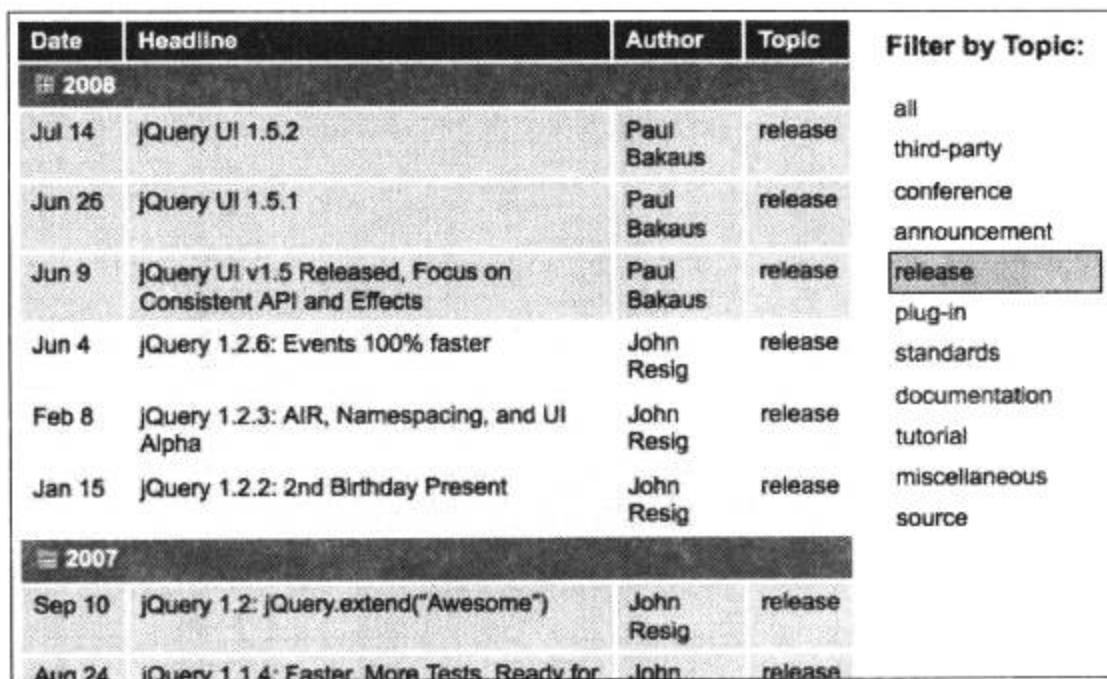
```
$ (document).ready(function() {
    $('table.striped').bind('stripe', function() {
        $('tbody', this).each(function() {
            $(this).find('tr:visible:not(:has(th))')
                .removeClass('odd').addClass('even')
                .filter(function(index) {
                    return (index % 6) < 3;
                }).removeClass('even').addClass('odd');
        });
        }).trigger('stripe');
    });
});
```

这些实现筛选的代码在每次发生筛选操作时，都会调用`$table.trigger('stripe')`。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7-32所示。

- 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。



The screenshot shows a table with columns: Date, Headline, Author, Topic, and Filter by Topic. The table has two sections: "2008" and "2007". A sidebar on the right lists filter options: all, third-party, conference, announcement, release (which is selected), plug-in, standards, documentation, tutorial, miscellaneous, and source. The table rows are as follows:

Date	Headline	Author	Topic
Jul 14	jQuery UI 1.5.2	Paul Bakaus	release
Jun 26	jQuery UI 1.5.1	Paul Bakaus	release
Jun 9	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	release
Jun 4	jQuery 1.2.6: Events 100% faster	John Resig	release
Feb 8	jQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	release
Jan 15	jQuery 1.2.2: 2nd Birthday Present	John Resig	release
Sep 10	jQuery 1.2: jQuery.extend("Awesome")	John Resig	release
Aug 24	iQuery 1.1.4: Faster, More Tests, Ready for	John	release

图 7-32

要处理后一种情况，可以改变显示和隐藏表格行的方式。如果使用类来标识哪些行应该隐藏，就不必再明确调用`.hide()`和`.show()`了。因此，可以用`.addClass('filtered')`和`.removeClass('filtered')`来代替`.hide()`和`.show()`，并辅之以与这个类对应的CSS规则。这样既实现了隐藏和显示表格行的操作，同时也能保持实现折叠的代码简单明了。在移除这个类并把行折叠起来后，这些就不会意外地显示了。

引入`filtered`这个新类也有助于我们解决另一个方向上的问题，即在需要扩展表格区域时，可以测试`filtered`类是否存在，如果存在则跳过相应的行而不显示它们。而执行这个测试类的操作，仅需在扩展代码包含的选择符表达式中添加`:not(.filtered)`即可。

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。

7.2.5 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() {
    $('table.striped').bind('stripe', function() {
        $('tbody', this).each(function() {
            $(this).find('tr:visible:not(:has(th))')
                .removeClass('odd').addClass('even')
                .filter(function(index) {
                    return (index % 6) < 3;
                }).removeClass('even').addClass('odd');
        });
    });
});
```

```
}).trigger('stripe');
});

$(document).ready(function() {
    var $authorCells = $('table.striped td:nth-child(3)');
    var $tooltip = $('

</div>').appendTo('body');

    var positionTooltip = function(event) {
        var tPosX = event.pageX;
        var tPosY = event.pageY + 20;
        $tooltip.css({top: tPosY, left: tPosX});
    };

    var showTooltip = function(event) {
        var authorName = $(this).text();
        var action = 'Highlight';
        if ($(this).parent().is('.highlight')) {
            action = 'Unhighlight';
        }
        $tooltip
            .text(action + ' all articles by ' + authorName)
            .show();
        positionTooltip(event);
    };

    var hideTooltip = function() {
        $tooltip.hide();
    };

    $authorCells
        .addClass('clickable')
        .hover(showTooltip, hideTooltip)
        .mousemove(positionTooltip)
        .click(function(event) {
            var authorName = $(this).text();
            $authorCells.each(function(index) {
                if (authorName == $(this).text()) {
                    $(this).parent().toggleClass('highlight');
                } else {
                    $(this).parent().removeClass('highlight');
                }
            });
            showTooltip.call(this, event);
        });
});

$(document).ready(function() {
    var collapseIcon = '../images/bullet_toggle_minus.png';
    var collapseText = 'Collapse this section';
    var expandIcon = '../images/bullet_toggle_plus.png';
    var expandText = 'Expand this section';


```

```

    $('table.collapsible tbody').each(function() {
        var $section = $(this);
        $('<img />').attr('src', collapseIcon)
            .attr('alt', collapseText)
            .prependTo($section.find('th'))
            .addClass('clickable')
            .click(function() {
                if ($section.is('.collapsed')) {
                    $section.removeClass('collapsed')
                        .find('tr:not(:has(th)):not(.filtered)')
                        .fadeIn('fast');
                    $(this).attr('src', collapseIcon)
                        .attr('alt', collapseText);
                }
                else {
                    $section.addClass('collapsed')
                        .find('tr:not(:has(th))')
                        .fadeOut('fast', function() {
                            $(this).css('display', 'none');
                        });
                    $(this).attr('src', expandIcon)
                        .attr('alt', expandText);
                }
                $section.parent().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 = $('<div class="filters"></div>');
                $('<h3></h3>')
                    .text('Filter by ' + $(this).text() + ':')
                    .appendTo($filters);
                $('<div class="filter">all</div>').click(function() {
                    $table.find('tbody tr').removeClass('filtered');
                    $(this).addClass('active')
                        .siblings().removeClass('active');
                    $table.trigger('stripe');
                }).addClass('clickable active').appendTo($filters);

                var keywords = {};
                $table.find('td:nth-child(' + (column + 1) + ')')
                    .each(function() {
                        keywords[$(this).text()] = $(this).text();
                    });
            }
        });
    });
});

```

```

$.each(keywords, function(index, keyword) {
    $('<div class="filter"></div>').text(keyword)
        .bind('click', {key: keyword}, function(event) {
            $('tr:not(:has(th))', $table).each(function() {
                var value = $('td', this).eq(column).text();
                if (value == event.data['key']) {
                    $(this).removeClass('filtered');
                }
                else {
                    $(this).addClass('filtered');
                }
            });
            $(this).addClass('active')
                .siblings().removeClass('active');
            $table.trigger('stripe');
        }).addClass('clickable').appendTo($filters);
    });
    $filters.insertBefore($table);
}
});
});
});
});

```

7.3 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

8

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 改进基本的表单

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。为了示范如何围绕表单实现渐进增强，本节将创建一个联系表单，然后再通过jQuery来从外观和行为两方面对其加以改进。

8.1.1 渐进增强表单样式

首先，我们先为表单增添一些艺术美感。在JavaScript无效的情况下，表单的第一个控件组（fieldset）如图8-1所示。

The screenshot shows a web form with a title 'Personal Info'. It contains two input fields for 'First Name' and 'Last Name', both marked as required. Below these is a question 'How would you like to be contacted? (choose at least one method)'. There are three checkbox options: 'by E-Mail', 'by Phone', and 'by Fax', each paired with a text input field. A note next to each checkbox specifies: '(required when corresponding checkbox checked)'.

图 8-1

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。

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>
    </ol>

```

```
<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>
</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>`。虽然控件组中的元素没有“标准的”元素结构，但有序列表在某种程度上却可以在联系表单中表达相应的语义。

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。

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（如果不是要求感观的满足）。

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>`。要实现这一点，使用jQuery的`.replaceWith()`方法是非常直观的：

```
$(document).ready(function() {
  $('legend').each(function(index) {
    $(this).replaceWith('<h3>' + $(this).text() + '</h3>');
  });
});
```

注意，这里不能依赖于jQuery的隐式迭代。因为除了要替换每个元素之外，还必须插入相应元素的文本内容。为此，这里使用`.each()`方法，以便通过`$(this)`取得相应元素的文本内容。

现在，当在样式表中为这个`<h3>`元素应用蓝色的背景和白色文本颜色后，表单的第一个控件组将如图8-2所示。

还有一种方法，无需替换`<legend>`元素，即将其内容包装在一对``标签中：

```
$(document).ready(function() {
  $('legend').wrapInner('<span></span>');
});
```

图 8-2

在<legend>中插入标签与将<legend>替换为<h3>相比，至少有两个好处：一是对于无法使用JavaScript的屏幕阅读器用户可以保留<legend>的语义；二是可以减少要编写的脚本量。但也有一个缺点，即增加了为标题应用样式的难度。因为这样一来，至少要同时设置<fieldset>和的position属性，还要设置<fieldset>的padding-top及的width属性：

```
fieldset {
    position: relative;
    padding-top: 1.5em;
}

legend span {
    position: absolute;
    width: 100%;
}
```

无论是替换表单中的<legend>元素，还是向元素中插入标签，总之都实现了我们想要的样式。接下来该着手清理必填的提示信息了。

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).end()
        .end()
        .filter('.conditional')
        .next('span').text(conditionalFlag);
});
```

使用.end()可以进一步扩展连缀的方法以便继续操作同一组元素，同时也避免了重复创建对象或遍历DOM。每个.end()方法负责从当前状态向后退一步，返回最后一个遍历方法执行之

前的那一组元素。上面的代码在同一行中使用两个`.end()`方法：第一个`.end()`恢复到`.filter('.required')`匹配的元素组，第二个`.end()`则恢复到`$('form :input')`取得的元素组。因此，在通过`.filter('.conditional')`选择带有`class="conditional"`属性的元素时，针对的是表单中的所有元素。

现在，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。为此，还可以进一步扩展连缀的方法：

```
$(document).ready(function() {
    var requiredFlag = ' * ';
    var conditionalFlag = ' ** ';

    $('form :input')
        .filter('.required')
        .next('span').text(requiredFlag).end()
        .prev('label').addClass('req-label').end()
    .end()
    .filter('.conditional')
        .next('span').text(conditionalFlag);
});
```

鉴于把这么多方法连缀在一起会造成阅读困难，因此必须在代码中使用统一的换行和缩进。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。

The screenshot shows a 'Personal Info' form with a dark header bar. Below it, there are two input fields: 'First Name' and 'Last Name', each followed by an asterisk (*) indicating a required field. Below these is a question: 'How would you like to be contacted? (choose at least one method)'. Underneath this question are three groups of checkboxes and input fields. Each group consists of a checkbox followed by a label ('by E-Mail', 'by Phone', or 'by Fax') and an input field, all preceded by a double asterisk (**).

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 conditionalFlag = ' ** ';

    var requiredKey = $('input.required:first')
        .next('span').text();
    var conditionalKey = $('input.conditional:first')
```

```

    .next('span').text();

    requiredKey = requiredFlag +
        requiredKey.replace(/^\\((.+))$/,'$1');
    conditionalKey = conditionalFlag +
        conditionalKey.replace(/^\\((.+))$/,'$1');

// .....省略的代码
});

```

前面两行新增的代码声明了变量`requiredKey`和`conditionalKey`，它们用于保存每个字段的提示信息。接下来的两行代码则修改了这两个变量：把相应的标志、文本连接了起来，同时去掉了圆括号。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。

● 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'.replace(/regular-expression/, 'replacement')
```

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。

● 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 conditionalFlag = ' ** ';

    var requiredKey = $('input.required:first')
        .next('span').text();
    var conditionalKey = $('input.conditional:first')
        .next('span').text();

```

```

requiredKey = requiredFlag +
    requiredKey.replace(/^\((.+)\)\$/,'$1');
conditionalKey = conditionalFlag +
    conditionalKey.replace(/^\((.+)\)\$/,'$1');

$(<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。

The screenshot shows a web form with the following structure:

- Header:** * required
** required when corresponding checkbox checked
- Section:** Personal Info
- Fields:**
 - First Name: [Input Field]
 - Last Name: [Input Field]
 - How would you like to be contacted? (choose at least one method):
 - by E-Mail: [Input Field]
 - by Phone: [Input Field]
 - by Fax: [Input Field]

图 8-4

为第一个字段组编写的jQuery代码差不多就要完成了。

8.1.2 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').next('span').andSelf().hide();
});

```

图8-5就是对这个字段组精简后的界面。

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').next('span').andSelf().hide()
    .end().end()
    .each(function() {
        var $thisInput = $(this);
        var $thisFlag = $thisInput.next('span');
        $thisInput.prev('label').find(':checkbox')
        .click(function() {
            // 省略的代码……
        });
    });
});
```

以上代码同样使用了两个.end()方法，这次是为了让.each()方法直接作用于\$('input.conditional')选择符匹配的元素。

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').next('span').andSelf().hide()
    .end().end()
    .each(function() {
        var $thisInput = $(this);
        var $thisFlag = $thisInput.next('span');
        $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`）。

现在所剩的只有两个操作了：

(1) 在页面加载完成后，保证复选框处于未被选中状态，因为浏览器会在页面刷新后自动保留表达元素的状态。

(2) 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').next('span').andSelf().hide()
    .end().end()
    .each(function() {
        var $thisInput = $(this);
        var $thisFlag = $thisInput.next('span');
        $thisInput.prev('label').find(':checkbox')
        .attr('checked', false)
        .click(function() {
            if (this.checked) {
                $thisInput.show();
                $thisFlag.show();
                $(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。

8.1.3 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。

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')
    .attr('checked', false)
    .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).hasClass('required')) {
            var $listItem = $(this).parents('li:first');
            if (this.value == '') {
                var errorMessage = 'This is a required field';
                $('')
                    .addClass('error-message')
                    .text(errorMessage)
                    .appendTo($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).hasClass('required')) {
            var $listItem = $(this).parents('li:first');
            if (this.value == '') {
                var errorMessage = 'This is a required field';
                if ($(this).hasClass('conditional')) {
                    errorMessage += ', when its related ' +
                        'checkbox is checked';
                }
                $ ('<span></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 web form titled "Personal Info". It contains fields for "First Name" (with value "Mark") and "Last Name" (with value "Turner"). Below these is a question "How would you like to be contacted? (choose at least one method)". 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" appears over the "by E-Mail" checkbox. Another identical tooltip message appears over the "by Phone" 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).hasClass('required')) {
    var $listItem = $(this).parents('li:first');
    if (this.value == '') {
        var errorMessage = 'This is a required field';
        if ($(this).hasClass('conditional')) {
            errorMessage += ', when its related checkbox
                            is checked';
        }
        $('</span>')
            .addClass('error-message')
            .text(errorMessage)
            .appendTo($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。

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.id == 'email') {
        var $listItem = $(this).parents('li:first');
        if ($(this).is(':hidden')) {
            this.value = '';
        }
        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。
 - 测试电子邮件字段是否处于隐藏状态。如果是（即对应的复选框未被选中），该字段值被设置为空字符串。这样，前面用到的警告类和错误消息移除功能，也会应用于电子邮件字段。
 - 使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})$/
```

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)
```

3. 最终检查

现在，对联系表单进行验证的代码差不多完成了。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：`<div id="submit-message">`，我们会在下一步添加这个元素。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。

Please correct errors with 3 fields

Send

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 list = [];
      $('.warning label').each(function() {
        list.push($(this).text());
      });
      $('

</div>')
        .attr({
          'id': 'submit-message',
          'class': 'warning'
        })
        .append('Please correct errors with the following ' +
               numWarnings + ' fields:<br />')
        .append('&bull; ' + list.join('<br />&bull; '))
        .insertBefore('#send');
      return false;
    };
  });
});


```

首先，需要在代码中添加把list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list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list数组中一个独立的元素。

接着，我们又修改了前面创建的<div id="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。

Please correct errors with the following 3 fields:

- First Name
- Last Name
- by E-Mail

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。

8.1.4 复选框操作

联系表单中还包含Miscellaneous部分，其中是要求用户选择“How did you discover us（知道我们的方式）”的一组复选框，如图8-10所示。

Miscellaneous

How did you discover us? (Check all that apply)

- Magazine
- Website
- Television
- Movie
- School
- Mom
- Billboard
- Graffiti
- Detritus
- Hate Mail

图 8-10

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。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中再放入和相应的文本，最后再把这个- 元素及其后代元素添加到- 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个checkall类，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').addClass('checkall');  
});
```

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。

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)

- check all
- Magazine
- Website
- Television
- Movie
- School
- Mom
- Billboard
- Graffiti
- Detritus
- Hate Mail

图 8-11

Miscellaneous

How did you discover us? (Check all that apply)

- un-check all
- Magazine
- Website

图 8-12

8.1.5 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。
    $('legend').each(function(index) {
        $(this).replaceWith('<h3>' + $(this).text() + '</h3>');
    });

    var requiredFlag = ' * ';
    var conditionalFlag = ' ** ';
    var requiredKey = $('input.required:first')
                    .next('span').text();
    var conditionalKey = $('input.conditional:first')
                    .next('span').text();
});
```

```

requiredKey = requiredFlag +
    requiredKey.replace(/^\((.+)\)\$/,'$1');
conditionalKey = conditionalFlag +
    conditionalKey.replace(/^\((.+)\)\$/,'$1');

$('.<p></p>')
    .addClass('field-keys')
    .append(requiredKey + '<br />')
    .append(conditionalKey)
    .insertBefore('#contact');

$('form :input')
    .filter('.required')
        .next('span').text(requiredFlag).end()
        .prev('label').addClass('req-label').end()
    .end()
    .filter('.conditional')
        .next('span').text(conditionalFlag);

// 条件文本输入字段，复选框切换。
$('input.conditional').next('span').andSelf().hide()
.end().end()
.each(function() {
    var $thisInput = $(this);
    var $thisFlag = $thisInput.next('span');
    $thisInput.prev('label').find(':checkbox')
        .attr('checked', false)
        .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).hasClass('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';
    }
    $('<span></span>')
      .addClass('error-message')
      .text(errorMessage)
      .appendTo($listItem);
    $listItem.addClass('warning');
}
}

if (this.id == 'email') {
  var $listItem = $(this).parents('li:first');
  if ($(this).is(':hidden')) {
    this.value = '';
  }
  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');
  }
}
});

// 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());
    });
    $('<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;
  }
});

```

```

    );
})

// 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').addClass('checkall');
});
}

```

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<http://plugins.jquery.com/project/validate>。

8.2 提升紧凑的表单

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；有时候，提升这种紧凑的表单，其作用甚至会大于增强整页的表单。

8.2.1 字段的占位符文本

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。

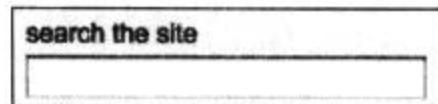


图 8-13

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，我们可以仅在JavaScript有效时，通过为搜索表单添加一个类，利用CSS把标签定位在字段上方：

```
$(document).ready(function() {
    var $search = $('#search').addClass('overlabel');
});
```

新增的一行代码为搜索表单添加了一个类，同时把选择符查询的结果保存在了变量中，以便后面引用。而样式表则通过overlabel类为标签添加样式：

```
.overlabel {
    position: relative;
}
.overlabel label {
    position: absolute;
    top: 6px;
    left: 3px;
    color: #999;
    cursor: text;
}
```

添加类之后，不仅标签被移动到了适当的位置，而且颜色变灰后的文本显然也具有了占位符文本的样子，如图8-14所示。

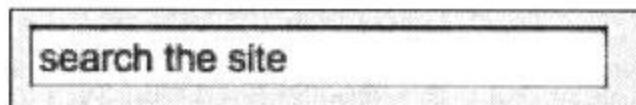


图 8-14

效果还不错，不过存在两个问题：

- (1) 标签文本会导致用户看不清自己在输入框中填写的内容，如图8-15所示；
- (2) 用户只能通过按Tab键把光标定位在输入框中，由于标签盖住输入框，因此用户不能通过单击鼠标输入文本。

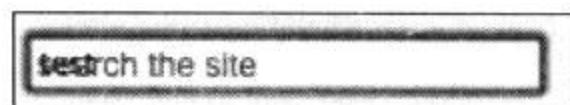
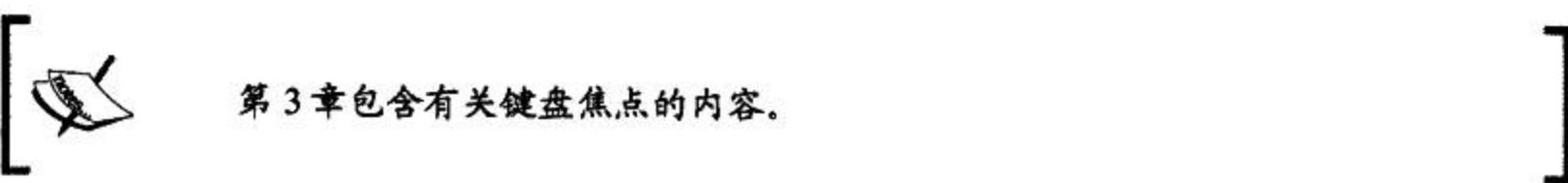


图 8-15

为解决第一个问题，需要在输入框获得焦点时隐藏标签文本。然后，当输入框失去焦点时再显示标签文本——前提是用户没有在输入框填写内容。



第3章包含有关键焦点的内容。

在输入框获得焦点时隐藏标签文本很简单：

```
$(document).ready(function() {
  var $search = $('#search').addClass('overlabel');
  var $searchInput = $search.find('input');
  var $searchLabel = $search.find('label');

  $searchInput
    .focus(function() {
      $searchLabel.hide();
    })
    .blur(function() {
      if (this.value == '') {
        $searchLabel.show();
      }
    });
});
```

这样，当用户在输入框中填写内容时，标签文本就会被隐藏起来，如图8-16所示。

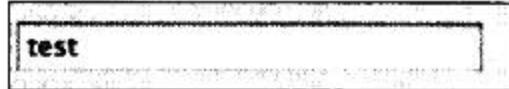


图 8-16

接下来的第二个问题也就很容易解决了。可以在隐藏标签文本的同时，利用对标签的单击来触发输入框的focus事件，从而把用户光标定位在输入框内：

```
$(document).ready(function() {
  var $search = $('#search').addClass('overlabel');
  var $searchInput = $search.find('input');
  var $searchLabel = $search.find('label');

  $searchInput
    .focus(function() {
      $searchLabel.hide();
    })
    .blur(function() {
      if (this.value == '') {
        $searchLabel.show();
      }
    });
});

$searchLabel.click(function() {
  $searchInput.trigger('focus');
});
```

最后，还需要处理当页面刷新后输入框内保留文本的情况——与在本章前面的表单验证示例中处理条件输入框类似。此时，如果输入框中有文本，则隐藏标签：

```
$(document).ready(function() {
    var $search = $('#search').addClass('overlabel');
    var $searchInput = $search.find('input');
    var $searchLabel = $search.find('label');

    if ($searchInput.val()) {
        $searchLabel.hide();
    }

    $searchInput
        .focus(function() {
            $searchLabel.hide();
        })
        .blur(function() {
            if (this.value == '') {
                $searchLabel.show();
            }
        });
    });

    $searchLabel.click(function() {
        $searchInput.trigger('focus');
    });
});
```

使用标签而不直接在搜索框中插入默认值的做法有一个优点，即可以地将相同的脚本应用到任何文本输入框，而不必担心会与验证脚本冲突。

8.2.2 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主题，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jQuery UI插件中的自动完成部件（widget），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部件作为添加更复杂功能的起点。相关网址为 <http://ui.jquery.com>。

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。

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。脚本中的字符串操作（如`str_replace()`和`implode()`）可以确保脚本输出适当格式的JSON数据，从而避免解析过程中发生JavaScript错误。

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': '../search/autocomplete.php',
            'data': {'search-text': $('#search-text').val()},
            'dataType': 'json',
            'type': 'GET',
            '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。

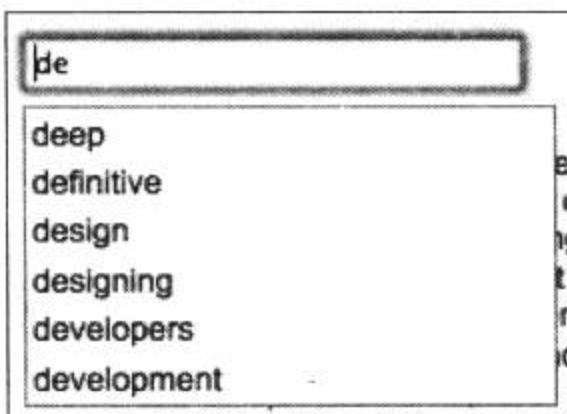


图 8-17

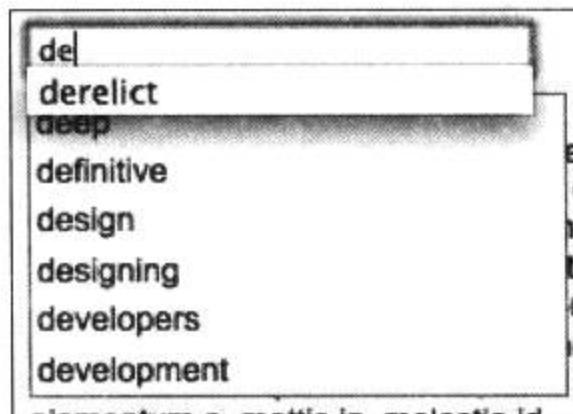


图 8-18

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，因为这么做会禁用浏览器的自动完成功能，但又没有提供替代功能。为此，我们还是要通过脚本来设置这个属性：

```
$('#search-text').attr('autocomplete', 'off')
```

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指针选定一个建议项：

```
'success': function(data) {
    if (data.length) {
        $autocomplete.empty();
        $.each(data, function(index, term) {
            $('- </li>').text(term)
                .appendTo($autocomplete);
        });
    }
}

```

```

.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。

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': '../search/autocomplete.php',
    'data': {'search-text': $('#search-text').val()},
    'dataType': 'json',
  })
  .done(function(data) {
    var items = data.items;
    var term = $('#search-text').val();
    var selectedIndex = items.findIndex(item => item.value === term);
    setSelectedItem(selectedIndex);
  });
});

```

```

'type': 'GET',
'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。最后，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。

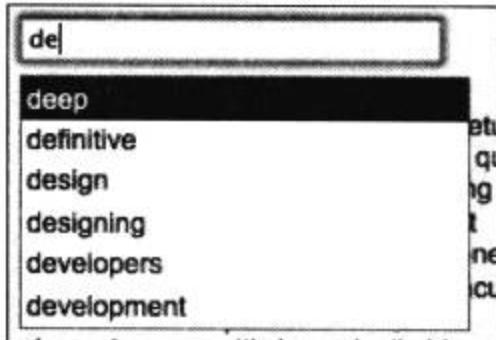


图 8-19

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。

5. 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': '../search/autocomplete.php',
  'data': {'search-text': $('#search-text').val()},
  'dataType': 'json',
  'type': 'GET',
  '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。

6. 将建议项插入到字段中

接下来需要处理回车键（或Mac中的return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。

7. 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. 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。

8.2.3 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 $search = $('#search').addClass('overlabel');
    var $searchInput = $search.find('input');
    var $searchLabel = $search.find('label');

    if ($searchInput.val()) {
        $searchLabel.hide();
    }

    $searchInput
        .focus(function() {
            $searchLabel.hide();
        })
        .blur(function() {
            if (this.value == '') {
                $searchLabel.show();
            }
        });
    $searchLabel.click(function() {
        $searchInput.trigger('focus');
    });

    var $autocomplete = $('

</ul>');
    $autocomplete.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': '../search/autocomplete.php',
                'data': {'search-text': $('#search-text').val()},
                'dataType': 'json',
                'type': 'GET',
                '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);
});
});

```

8.3 操作数字型表单数据

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。

8.3.1 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
    </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	\$35.99	\$71.98
Creating your MySQL Database	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，我们采取一个折中的方案，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，选择符就会变得更简单。

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:nth-child(even)').addClass('alt');
});
```

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	\$35.99	\$71.98
Creating your MySQL Database	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

8.3.2 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：

```
$('td.quantity input').keypress(function(event) {
    if (event.which && (event.which < 48 ||
        event.which > 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。

8.3.3 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：

```
Var $quantities = $('td.quantity input');
$quantities.change(function() {
    var totalQuantity = 0;
    $quantities.each(function() {
        var quantity = parseInt(this.value);
        totalQuantity += quantity;
    });
    $('tr.shipping td.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	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	15	\$2 per item	\$10.00
Total			\$172.13

Recalculate **Place Order**

图 8-22

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：

```
$('#cart tbody tr').each(function() {
    var quantity = parseInt($('td.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：

```
$(td.quantity input').change(function() {
    var totalQuantity = 0;
    $('#cart tbody tr').each(function() {
        var price = parseFloat($('td.price', this).text()
            .replace(/[^.\d.]*/, ''));
        price = isNaN(price) ? 0 : price;
        var quantity =
            parseInt($('td.quantity input', this).val());
        var cost = quantity * price;
        $('td.cost', this).text('$' + cost);
    });
});
```

```

        totalQuantity += quantity;
    });
    $('tr.shipping td.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	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

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	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	43	\$2 per item	\$10.00
Total			\$172.13
Recalculate		Place Order	

图 8-24

没错，总费用应该是\$1079.70，但这里显示的是\$1079.7。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	10	\$17.99	\$179.89999999999996
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：

```
$('#cart tbody tr').each(function() {
    var price = parseFloat($('td.price', this).text()
        .replace(/[^^\d.]*/, ''));

    price = isNaN(price) ? 0 : price;
    var quantity = parseInt($('td.quantity input', this).val());
    var cost = quantity * price;
    $('td.cost', this).text('$' + 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	\$1079.70
Creating your MySQL Database	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

在经过一系列数学运算后，浮点数值舍入造成的误差，即使.toFixed()也无法掩盖。在较大的应用程序中，处理货币计算的最可靠方式，就是以整数形式保存分值（而不是元），然后只在显示货币值时再添加小数点。

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）：

```
$('td.quantity input').change(function() {
    var totalQuantity = 0;
    var totalCost = 0;
    $('#cart tbody tr').each(function() {
        var price = parseFloat($('td.price', this).text()
            .replace(/[^\\d.*]/, ''));

        price = isNaN(price) ? 0 : price;
        var quantity =
            parseInt($('td.quantity input', this).val());
        var cost = quantity * price;
        $('td.cost', this).text('$' + cost.toFixed(2));
        totalQuantity += quantity;
        totalCost += cost;
    });
    $('tr.shipping td.quantity').text(String(totalQuantity));
    $('tr.subtotal td.cost').text('$' +
        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	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
		<input type="button" value="Recalculate"/>	<input type="button" value="Place Order"/>

图 8-27

4. 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($('tr.tax td.price').text()) / 100;
var tax = Math.ceil(totalCost * taxRate * 100) / 100;
$('tr.tax td.cost').text('$' + 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	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	\$14.00
Total			\$214.27
		Recalculate	Place Order

图 8-28

5. 最后的调整

对我们的例子而言，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：

```
$('tr.shipping td.quantity').text(String(totalQuantity));
var shippingRate = parseFloat($('tr.shipping td.price')
    .text().replace(/[^d.]*/, '')); 
var shipping = totalQuantity * shippingRate;
$('tr.shipping td.cost').text('$' + shipping.toFixed(2));
totalCost += shipping;
```

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形式显示`totalCost`：

```
$('tr.total td.cost').text('$' + totalCost.toFixed(2));
```

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：

```
$('#recalculate').hide();
```

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，结果如图8-29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	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	\$14.00
Total			\$214.27
		Place Order	

图 8-29

8.3.4 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8-30所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$26.99	\$26.99
Smarty PHP Template Programming and Applications	1	\$35.99	\$71.98
Creating your MySQL Database	2	\$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
Place Order			

图 8-30

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 = $('

```

```

'src': '../images/cross.png',
'alt': 'remove from cart',
'title': 'remove from cart',
'class': 'clickable'
}).click(function() {
  $(this).parents('tr').find('td.quantity input')
    .val(0);
});

```

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8-31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	\$35.99
Creating your MySQL Database	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	\$14.00
Total			\$214.27

图 8-31

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 = $('

```

现在总金额会随着单击删除按钮而更新了，如图8-32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	2	\$17.99	\$35.98
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.94
Tax		6%	\$9.18
Shipping	6	\$2 per item	\$12.00
Total			\$174.12

图 8-32

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8-33所示）：

```
$deleteButton = $('').attr({
  'width': '16',
  'height': '16',
  'src': '../images/cross.png',
  'alt': 'remove from cart',
  'title': 'remove from cart',
  'class': 'clickable'
}).click(function() {
  $(this).parents('tr').find('td.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	2	\$17.99	\$35.98
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
<hr/>			\$152.94
Subtotal			\$152.94
Tax		6%	\$9.18
Shipping	6	\$2 per item	\$12.00
<hr/>			\$174.12
<hr/>			Place Order

图 8-33

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。与此同时，还需要修改代码以确保在交替选择行时忽略隐藏的行。可是，即便过滤掉所有隐藏的行，仍然不能使用:even选择符，因为要通过它为所有作为其父元素第偶数个子元素的可见行应用alt类。下面，我们就来看一看，当包含4行的表格中的第二个行被隐藏时，下列修改后的代码会导致什么结果：

```
$('#cart tbody tr').removeClass('alt')
  .filter(':visible:nth-child(even)').addClass('alt');
```

得到的标记如下所示（有精简）：

```
<tr> . . . </tr>
<tr style="display: none"> . . . </tr>
<tr> . . . </tr>
<tr class="alt"> . . . </tr>
```

表格中的3行可见，但只有第四行包含alt类。实际上，在此我们需要的是:visible:odd选择符表达式，该表达式会从移除隐藏行之后的所有行中，隔一行选择一行（这也说明了从基于1

索引的选择符到基于0索引的选择符的转换)。同第2章一样,如果表格包含多个`<tbody>`元素,使用`:odd`或`:even`会导致无法预料的结果,但当前示例没有这个问题。修改了选择符之后,新函数的代码应该如下所示:

```
var stripe = function() {
    $('#cart tbody tr').removeClass('alt')
        .filter(':visible:odd').addClass('alt');
};

stripe();
```

接着,我们就可以在移除一行之后调用这个函数了:

```
$deleteButton = $('').attr({
    'width': '16',
    'height': '16',
    'src': '../images/cross.png',
    'alt': 'remove from cart',
    'title': 'remove from cart',
    'class': 'clickable'
}).click(function() {
    $(this).parents('tr').find('td.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8-34所示。

Item	Quantity	Price	Total
Building Telephony Systems With Asterisk	3	\$26.99	\$80.97
Creating your MySQL Database	2	\$17.99	\$35.98
Drupal: Creating Blogs, Forums, Portals, and Community Websites	1	\$35.99	\$35.99
Subtotal			\$152.94
Tax		6%	\$9.18
Shipping	6	\$2 per item	\$12.00
Total			\$174.12
Place Order			

图 8-34

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。

8.3.5 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8-35所示。

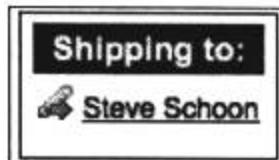


图 8-35

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;
    });
});
```

PHP页面（shipping.php）在接收到\$.get()方法发送的请求之后，可以在输出大部分页面标记之前测试服务器端变量\$_SERVER['HTTP_X_REQUESTED_WITH']存在与否，如果存在则只返回完整页面的一个片段——表单。

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8-36所示。

First name:	Steve
Last name:	Schoon
Address:	52 Rodeo Drive
City:	Beverly Hills
State:	CA
ZIP:	90210
Gift message: Enjoy these great books! You'll love the cliff-hangers!	

图 8-36

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 = $(this).serialize();
        $.post('shipping.php', postData);
        return false;
    });
});
```



jQuery 的 Form 插件提供更可靠的`.serialize()`方法。这个插件（可以从<http://www.malsup.com/jquery/form/>下载）适用于大多数通过AJAX提交表单的情形。

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 = $(this).serialize();
        $.post('shipping.php', postData, function(data) {
            $('#shipping form').remove();
            $(data).appendTo('#shipping');
        });
        return false;
    });
});
```

但是，这些代码还起作用！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，在创建了表单之后，在`$.get()`方法的回调函数中绑定`saveShipping()`函数。同样地，还需要在DOM就绪时和在`$.post()`方法的回调函数中重新创建了Edit shipping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);
        });
    };
});
```

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;
};

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。

8.3.6 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80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() {
    var stripe = function() {
        $('#cart tbody tr').removeClass('alt')
            .filter(':visible:odd').addClass('alt');
    };
    stripe();

    $('#recalculate').hide();

    $('.quantity input').keypress(function(event) {
        if (event.which && (event.which < 48 ||
                           event.which > 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(), 10);
            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(/[^0-9]/g, ''));

var shipping = totalQuantity * shippingRate;
$('.shipping .cost').text('$' + shipping.toFixed(2));
totalCost += shipping;
$('.total .cost').text('$' + totalCost.toFixed(2));
});

$('<th> </th>')
.insertAfter('#cart thead th:nth-child(2)');
$('#cart tbody tr').each(function() {
    $deleteButton = $('
```

```
var saveShipping = function() {
    var postData = $(this).serialize();
    $.post('shipping.php', postData, function(data) {
        $('#shipping form').remove();
        $(data).appendTo('#shipping');
        $('#shipping-name').click(editShipping);
    });
    return false;
};
$('#shipping-name').click(editShipping);
});
```

8.4 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。

滑移和翻转



我们已经介绍了一些根据需要隐藏和显示信息的方式，例如可折叠的信息窗格。不过，有时候在把内容移入或移出视图时，我们想让读者看到更引人入胜的动画效果，如传送带、循环播放、滑移和翻转。这些动画效果的共同特点，就是能够快速切换多个数据段，而切换方式既能吸引眼球，又能令人过目不忘。

在实例部分的最后一章，也就是本章中，我们将探讨如何创建这些高超的动画，同时将它们与AJAX和CSS技术巧妙整合起来，为用户提供超爽的视觉体验。

本章将详细介绍两个相对复杂的例子——标题新闻翻转效果和图像传送带。通过这两个例子，读者可以学习到如何：

- 实现元素的位置动画
- 解析XML文档
- 利用部分透明技术制作高级样式效果
- 从不同域取得信息
- 创建水平滚动的用户界面元素
- 标记层的重叠与覆盖
- 放大图像

9.1 标题新闻翻转效果

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。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。整体效果就是一条没有尽头的新闻纸，在页面上不停旋转。

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：

```
<h3>Recent News</h3>
<div id="news-feed">
  <a href="news/index.html">News Releases</a>
</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，如图9-1所示。

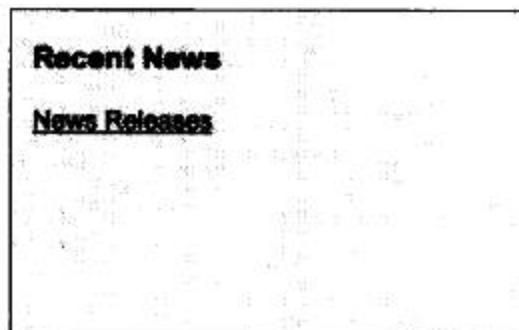


图 9-1

这是我们按照平衡退化原则准备的后备力量，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`<div>`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，当把#news-feed的overflow属性设置为hidden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。

将新闻条目的position属性设置为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() {
    var $container = $(this);
    $container.empty();
  });
});

```

正常情况下，在使用`.each()`方法时，一般都是要迭代一组较大的元素。不过，这里的选择符`#news-feed`查找的是ID，因此只有两种可能的结果。第一种，`$()`工厂函数会生成一个与带有`news-feed` ID的唯一元素匹配的jQuery对象；第二种，`$()`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`<div>`，以便盛放新内容，结果如图9-2所示。



图 9-2

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。前面曾看到过，通过这个方法的成功处理程序可以操作来自远程服务器的内容。换句话说，取得的新闻源将以XML格式传入这个处理程序中并被解析。然后，就可以使用jQuery的选择符引擎来操作解析后的XML数据。

```

$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
    $.get('news/feed.xml', function(data) {
      $('rss item', data).each(function() {
        // 这里的操作新闻标题的代码。
      });
    });
  });
});

```



要了解有关`$.get()`及其他AJAX方法的更多信息，请参考第6章。

接下来，需要把新闻的每一部分组织起来，得到一个有用的HTML标记块。在此，仍然可以使用`.each()`遍历新闻源中的所有项，并基于其中的信息构建新闻条目的链接：

```
$ (document).ready(function() {
    $('#news-feed').each(function() {
        var $container = $(this);
        $container.empty();
        $.get('news/feed.xml', function(data) {
            $('rss item', data).each(function() {
                var $link = $('</a>');
                .attr('href', $('link', this).text())
                .text($('title', this).text());
                var $headline = $('

#### </h4>').append($link); $(' </div>') .append($headline) .appendTo($container); }); }); }); });


```

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>`元素并将其放在一个`<h4>`元素中。再把每个新闻条目放到`<div id="news-feed">`中——不过，为了能看到当前代码执行的结果（如图9-3所示），这里没有为包含`<div>`添加`headline`类。

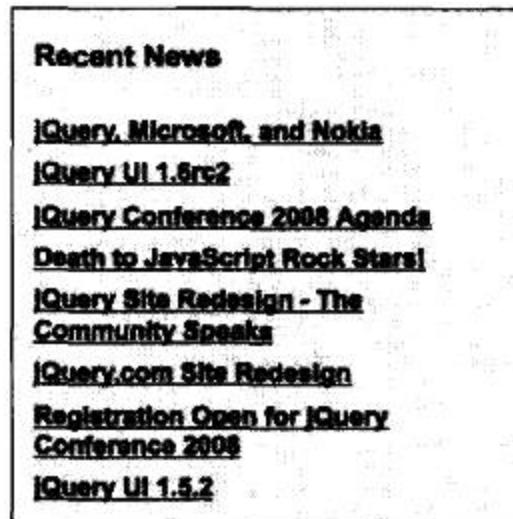


图 9-3

除了新闻标题之外，我们还想多显示一些与新闻相关的辅助性信息。具体来说，还要取得发表日期和新闻摘要并将它们显示出来。

```
$ (document).ready(function() {
    $('#news-feed').each(function() {
        var $container = $(this);
        $container.empty();
        $.get('news/feed.xml', function(data) {
            $('rss item', data).each(function() {
```

```
var $link = $('</a>')
    .attr('href', $('link', this).text())
    .text($('title', this).text());
var $headline = $('

#### </h4>').append($link) var pubDate = new Date( $('pubDate', this).text()); var pubMonth = pubDate.getMonth() + 1; var pubDay = pubDate.getDate(); var pubYear = pubDate.getFullYear(); var $publication = $(' </div>') .addClass('publication-date') .text(pubMonth + '/' + pubDay + '/' + pubYear); var $summary = $(' </div>') .addClass('summary') .html($('description', this).text()); $(<div></div>) .append($headline, $publication, $summary) .appendTo($container); });


```

RSS源中的日期信息是以RFC 822格式提供的，其中包含日期、时间以及时区信息（例如Sun, 28 Sep 2008 18:01:55 +0000）。这种格式看起来并不十分直观，因此需要使用JavaScript内置的Date对象将其转换成更简单的表现形式（如9/28/2008）。

新闻摘要比较容易提取和格式化。不过，值得一提的是，在我们这个示例RSS源中，描述（`description`）标签中已经包含了一些HTML实体。为保证jQuery不会自动转义这些实体，需要使用`.html()`方法（而非`.text()`方法）将这些描述标签包含的内容插入到页面中。

在创建了上述新元素之后，我们又通过`.append()`方法将它们插入到文档中。在此，我们还利用了这个方法的一个新特性，即在为它传递多个参数时，所有参数会按照顺序被加入文档。现在的结果如图9-4所示。

现在，每条新闻的标题、日期、链接及摘要信息已经各就各位。接下来就剩使用`.addClass('headline')`为它们添加`headline`类了（添加`headline`类之后，所有新闻都会根据前面CSS样式的设置而隐藏起来），而我们也应该继续考虑如何为标题添加翻转动画效果了。

- Recent News**
 - jQuery, Microsoft, and Nokia**
9/28/2008
We have two pieces of fantastic, albeit serendipitous, news today: Both Microsoft and Nokia are taking the major step of adopting jQuery as part of their official application development platform.
 - jQuery UI 1.8rc2**
9/19/2008
Hey everyone. I'm glad to announce

图 9-4

9.1.3 设置翻转效果

由于可见的新闻条目会随时间推移而变化，因此需要一种方式来跟踪哪条新闻可见及其所在位置。首先，要设置两个变量，一个记录当前可见的标题新闻^①，另一个记录刚刚滚动出视图的标题新闻。在初始状态下，这两个变量的值都是0。

```
var currentHeadline = 0, oldHeadline = 0;
```

然后，要设置所有标题新闻的初始位置。还记得吗？我们在样式表中把标题新闻的top属性设置为比它们容器的height大10像素的值，而且，由于容器的overflow属性被设置为hidden，因而就隐藏了这些标题新闻。为了后面方便，我们要把这个属性值保存在一个变量中，以便当把某个标题新闻滚动出可见区域后，再把它重设回原来隐藏它的位置。

```
var hiddenPosition = $container.height() + 10;
```

我们也希望在页面加载后，立即显示第一个标题新闻，因此可以吧它的top属性设置为0：

```
$('.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9-5所示。

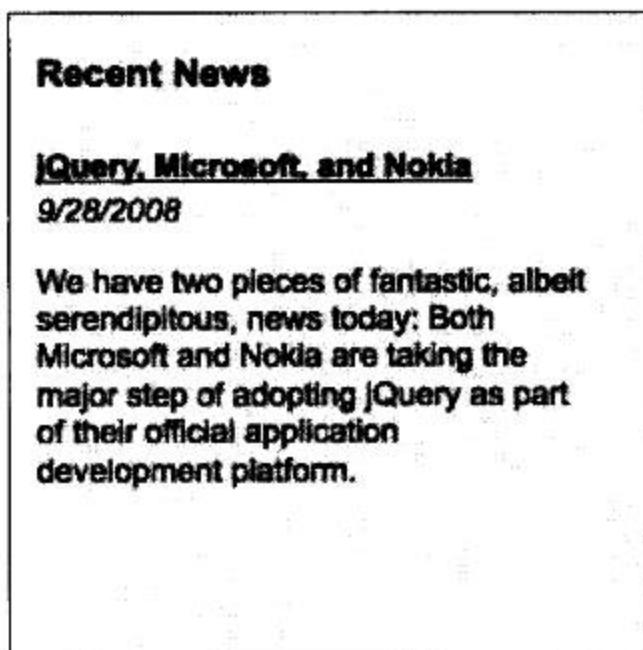


图 9-5

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 pause;
```

现在还不必为pause设置值，该变量的值将在每次翻转发生时设置。但是，为了避免与同名的全局变量发生冲突，必须要使用var来声明局部变量。

^① 这里的“标题新闻”指的是与.headline匹配的元素，包含标题、日期、链接和摘要。下同。——译者注

9.1.4 标题新闻翻转函数

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。为了在需要时重用代码，下面要为这个任务定义一个函数。

首先，来看一看如何更新跟踪哪个标题新闻当前可见的变量。使用模运算符（%）可以方便地“消掉”标题新闻数量。每调用一次这个函数，我们就为oldHeadline的值加上1，然后再对headlineCount执行求模运算。从而保证currentHeadline变量中始终保存正确的标题新闻索引。

 要了解有关模运算符的更详细的讨论，请参见第7章中使用相同技术为表格行应用不同的颜色的内容。

此外，还需要更新oldHeadline的值，从而确保正确操作移到可见区域外的标题新闻。

```
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，添加将旧标题新闻移出视图的动画。然后，再插入将新标题新闻滑入视图的动画。

```
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() {
            pause = setTimeout(headlineRotate, 5000);
        });
    oldHeadline = currentHeadline;
};
```

上述两种动画都是通过操作新闻条目的top属性实现的。前面为了隐藏这些标题新闻，我们将它们的top属性设置为hiddenPosition（一个比容器高度大一些的值）。只要把top属性修改为0，相应的标题新闻就会进入视图，而将该属性再修改为-hiddensPostion即可将相应标题新闻移出视图。

 第4章曾介绍过这个top是一个CSS定位属性，该属性只有在元素的position属性值为absolute或relative的情况下才有效。

在移入移出标题新闻的代码中，还需要分别定义回调函数，以便在动画完成时采取必要操作。当旧标题新闻滑出视图后，要把其top属性重置为hiddenPosition，从而做好再次进入视图的准备。当移入新标题新闻的动画结束后，需要确定后续要显示新闻队列中的哪一条新闻。整个过程可以通过JavaScript的setTimeout()函数来控制，这个函数会在指定的时间间隔后调用传入的函数。在此，我们想每隔5秒（5000毫秒）调用一次headlineRotate()函数。

这样，就构成了一个活动循环：一个动画结束后，另一个动画又处于待执行状态。剩下的就是启动这个活动循环（即第一次调用headlineRotate()函数）了。于是，我们通过另一个setTimeout()调用，实现在了RSS源被加载5秒后，触发第一个动画。以下就是实现这个标题新闻翻转效果的代码：

```

$(document).ready(function() {
    $('#news-feed').each(function() {
        var $container = $(this);
        $container.empty();
        $.get('news/feed.xml', function(data) {
            $('rss item', data).each(function() {
                var $link = $('</a>');
                .attr('href', $('link', this).text())
                .text($('title', this).text());
                var $headline = $('

#### </h4>').append($link); var pubDate = new Date($('pubDate', this).text()); var pubMonth = pubDate.getMonth() + 1; var pubDay = pubDate.getDate(); var pubYear = pubDate.getFullYear(); var $publication = $(' </div>') .addClass('publication-date') .text(pubMonth + '/' + pubDay + '/' + pubYear); var $summary = $(' </div>') .addClass('summary') .html($('description', this).text()); $(' </div>') .addClass('headline') .append($headline, $publication, $summary) .appendTo($container); }); var currentHeadline = 0, oldHeadline = 0; var hiddenPosition = $container.height() + 10; $('div.headline').eq(currentHeadline).css('top', 0); var headlineCount = $('div.headline').length; var pause; var headlineRotate = function() { currentHeadline = (oldHeadline + 1) * headlineCount; $('div.headline').eq(oldHeadline).animate( {top: -hiddenPosition}, 'slow', function() { $(this).css('top', hiddenPosition); }); oldHeadline++; if (oldHeadline < headlineCount) { setTimeout(headlineRotate, 5000); } }; }); });


```

```
        });
        $('div.headline').eq(currentHeadline).animate(
            {top: 0}, 'slow', function() {
                pause = setTimeout(headlineRotate, 5000);
            });
        oldHeadline = currentHeadline;
    );
    pause = setTimeout(headlineRotate, 5000);
});
});
```

在以上代码执行的过程中，可以看到一条新闻从顶部滑出，另一条新闻从底部滑入的效果（如图9-6所示）。

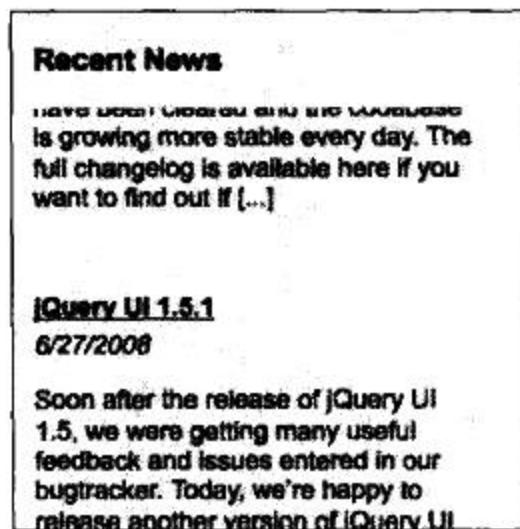


图 9-6

9.1.5 悬停时暂停

尽管标题新闻翻转的功能完全实现了，但我们还必须提到一个重要的可用性问题——当用户想单击某个标题新闻中的链接时，如果该标题新闻滚动出了可见区域，那么用户就必须等到整整一组新闻全部循环完毕。如果当用户把鼠标指针放到标题新闻中的任意位置上，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。

```
$container.hover(function() {
  clearTimeout(pause);
}, function() {
  pause = setTimeout(headlineRotate, 250);
});
```

当鼠标指针进入标题新闻区域后，第一个`.hover`处理程序会调用JavaScript的`clearTimeout`函数，取消正在运行的计时器，从而阻止再次调用`headlineRotate`函数。当鼠标指针离开时，第二个`.hover()`处理程序会重新启动计时器，从而在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，如果用户的鼠标指针反复在这个

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：多个标题新闻会相互层叠在可见区域中，如图9-7所示。

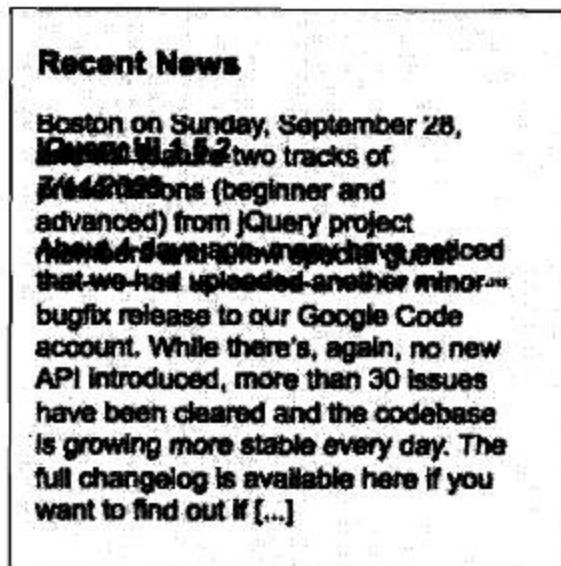


图 9-7

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。在这个条件语句里，首先将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;
                pause = setTimeout(headlineRotate, 5000);
            });
        oldHeadline = currentHeadline;
    }
};
```

新添加的这些代码有效地增强了标题新闻翻转效果。重复的mouseover-mouseout行为不会

再导致标题新闻相互堆积的结果。不过，这个重复行为仍然给我们留下了一个小问题：后续的标题新闻会按照不同的步调出现，两三个标题新闻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，如果用户的鼠标指针离开这个

时，现有的计时器还没有完成，就会导致同时存在多个计时器。因而，我们还需要再添加一个安全装置，即以pause变量为标志，表示另一个动画是否即将开始。为此，需要在计时器被清除或当一个动画完成时，把这个变量设置为false。然后，通过检查该变量的值来保证在现有的计时器没有完成之前，不设置新计时器：

```
var headlineRotate = function() {
    if (!rotateInProgress) {
        rotateInProgress = true;
        pause = 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 (!pause) {
                    pause = setTimeout(headlineRotate, 5000);
                }
            });
        oldHeadline = currentHeadline;
    }
};

if (!pause) {
    pause = setTimeout(headlineRotate, 5000);
}

$container.hover(function() {
    clearTimeout(pause);
    pause = false;
}, function() {
    if (!pause) {
        pause = setTimeout(headlineRotate, 250);
    }
});
```

最终，我们构建的标题新闻翻转效果能够经受住鼠标指针的各种不正常行为。

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。第6章曾介绍过，AJAX请求不能（这是一条强制性规则）向提供其所在页面的域外部发送请求。当时，我们讨论过如何使用JSONP数据格式来绕过这一限制。不过，在此我们假设不能改变数据来源，因而就需要另外寻找解决方案。

为了通过AJAX取得其他域中的文件，可以通过一些服务器端代码作为请求代理，让JavaScript认为要取得的XML文件就在我们自己的服务器上（虽然它实际上来自其他服务器）。为此，需要编写一个简单的PHP脚本，把新闻源文件读取到我们的服务器上来，并以该数据作为对jQuery脚本请求的响应。这个PHP脚本文件名为feed.php，可以像前面取得feed.xml一样调用它：

```
$.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，然后再把相应内容输出到浏览器：

```
<?php
    header('Content-Type: text/xml');
    print file_get_contents('http://jquery.com/blog/feed');
?>
```

注意，这里需要明确把页面的内容类型设置为text/xml，以便jQuery可以把它作为正常的静态XML文档获取并分析。



由于安全方面的原因，一些Web主机提供商不允许使用PHP的file_get_contents()函数取得远程文件。如果是这样，可以使用替代方案，如使用cURL库。有关cURL库的更多信息，请参考<http://wiki.dreamhost.com/CURL>。

添加加载进度指示器

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。我们应该告诉用户正在执行加载操作。为此，可以在发送AJAX请求前向页面中添加一幅进度指示器图像。

```
var $loadingIndicator = $('')
    .attr({
        'src': 'images/loading.gif',
        'alt': 'Loading. Please wait.'
    })
    .addClass('news-wait')
    .appendTo($container);
```

然后，在\$.get()函数的成功回调函数的第一行代码中，通过一个简单的命令移除这幅图像：

```
$loadingIndicator.remove();
```

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9-8所示。

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。

Recent News

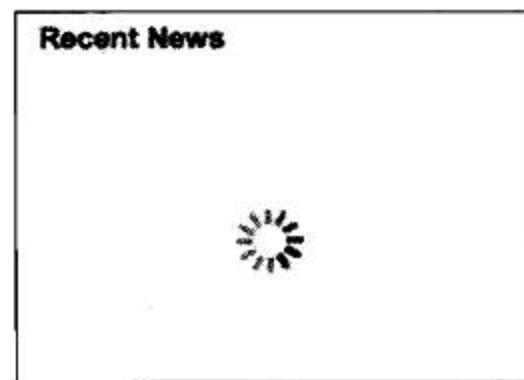


图 9-8



使用<http://ajaxload.info/>提供的服务，可以轻松地创建用作AJAX加载进度指示器的GIF图像。

9.1.7 附加的内部渐变效果

在完成标题新闻翻转效果之前，我们再修饰最后一笔——让标题新闻中的文本呈现出逐渐淡出到背景中的效果。这种效果可以称为渐变淡入，即位于效果区的文本内容上方完全不透明，而下方则是透明的。

然而，单独的文本元素不可能同时存在多个透明度不同的区域。为了模拟这种效果，可以用一系列元素盖住效果区，同时为每个元素应用不同的透明度。所有这些

“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。这样就可以让用户的眼睛“相信”文本是逐渐淡入的，他们不会想到文本上又放了一系列元素。

接下来，需要创建

元素。要知道所需元素的数量，首先应该确定整个效果区的像素高度。在这里，我们希望效果区高度是

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 $container = $(this);
        $container.empty();

        var fadeHeight = $container.height() / 4;
        for (var yPos = 0; yPos < fadeHeight; yPos += 2) {
            $('<div></div>')
                .addClass('fade-slice')
                .appendTo($container);
        }
    });
});
```

这样就有了25个“切片”（每个高度为2像素，总共构成了50像素高的渐变区域），不过它们现在还不分彼此地堆积在容器上方呢。为了实现我们想要的效果，必须为每个“切片”指定不同的位置和透明度。而使用迭代变量yPos就可以精确地计算出每个元素的opacity和top属性值了：

```
$(document).ready(function() {
    $('#news-feed').each(function() {
        var $container = $(this);
        $container.empty();

        var fadeHeight = $container.height() / 4;
        for (var yPos = 0; yPos < fadeHeight; yPos += 2) {
            $('<div></div>').css({
```

```
        opacity: yPos / fadeHeight,  
        top: $container.height() - fadeHeight + yPos  
    }) .addClass('fade-slice') .appendTo($container);  
}  
});  
});
```

整个计算过程不太好理解，因此我们把每次循环得到的数值列在一个表格中（如图9-9所示）。其中，`opacity`值逐渐增大（即不透明度越来越高），而`top`值从渐变区域的顶部（150）开始，一直增长到容器的高度值。

yPos	opacity	top
0	0	150
2	0.04	152
4	0.08	154
6	0.12	156
8	0.16	158
...		
40	0.80	190
42	0.84	192
44	0.88	194
46	0.92	196
48	0.96	198

图 9-9

因为最后一个

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，页面标题新闻区域中的文本在从容器底部向上滚动时，就会显示出从不透明到透明的优美的渐变效果，如图9-10所示。

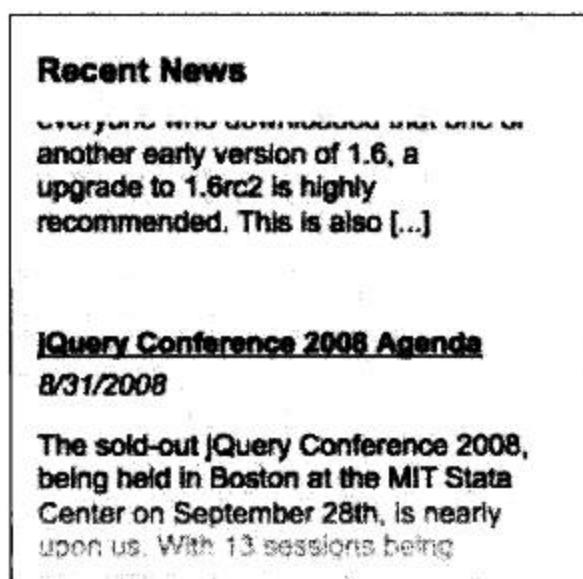


图 9-10 \

9.1.8 完成的代码

现在，第一个标题新闻翻转效果的例子就完成了。新闻条目取自远程服务器，经过格式化，以动画方式按既定的时间间隔滑入、滑出视图，而且还带有漂亮的样式：

```
$ (document).ready(function() {
    $('#news-feed').each(function() {
        var $container = $(this);
        $container.empty();

        var fadeHeight = $container.height() / 4;
        for (var yPos = 0; yPos < fadeHeight; yPos += 2) {
            $('

</div>').css({
                opacity: yPos / fadeHeight,
                top: $container.height() - fadeHeight + yPos
            }).addClass('fade-slice').appendTo($container);
        }

        var $loadingIndicator = $('')
            .attr({
                'src': 'images/loading.gif',
                'alt': 'Loading. Please wait.'
            })
            .addClass('news-wait')
            .appendTo($container);

        $.get('news/feed.php', function(data) {
            $loadingIndicator.remove();
            $('rss item', data).each(function() {
                var $link = $('</a>')
                    .attr('href', $('link', this).text())
                    .text($('title', this).text());
                var $headline = $('

#### </h4>').append($link); var pubDate = new Date($('pubDate', this).text()); var pubMonth = pubDate.getMonth() + 1; var pubDay = pubDate.getDate(); var pubYear = pubDate.getFullYear(); var $publication = $(' </div>') .addClass('publication-date') .text(pubMonth + '/' + pubDay + '/' + pubYear); var $summary = $(' </div>') .addClass('summary') .html($('description', this).text()); $(' </div>') .addClass('headline') .append($headline, $publication, $summary) .appendTo($container); }); }); var currentHeadline = 0, oldHeadline = 0;


```

```

var hiddenPosition = $container.height() + 10;
$('div.headline').eq(currentHeadline).css('top', 0);
var headlineCount = $('div.headline').length;
var pause;
var rotateInProgress = false;
var headlineRotate = function() {
    if (!rotateInProgress) {
        rotateInProgress = true;
        pause = 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 (!pause) {
                    pause = setTimeout(headlineRotate, 5000);
                }
            });
        oldHeadline = currentHeadline;
    }
};
if (!pause) {
    pause = setTimeout(headlineRotate, 5000);
}
$container.hover(function() {
    clearTimeout(pause);
    pause = false;
}, function() {
    if (!pause) {
        pause = setTimeout(headlineRotate, 250);
    }
});
});
});
});

```

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="images/covers/large/1847190871.jpg"
        title="Community Server Quickly">
      
      <span class="price">$35.99</span>
    </a>
    <a href="images/covers/large/1847190901.jpg"
        title="Deep Inside osCommerce: The Cookbook">
      
      <span class="price">$44.99</span>
    </a>
    <a href="images/covers/large/1847190979.jpg"
        title="Learn OpenOffice.org Spreadsheet Macro
          Programming: OOoBasic and Calc automation">
      
      <span class="price">$35.99</span>
    </a>
    <a href="images/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="images/covers/large/1847191002.jpg"
        title="Google Web Toolkit GWT Java AJAX Programming">
      
      <span class="price">$40.49</span>
    </a>
```

```

<a href="images/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120像素）、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9-11所示。

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);
});
```



图 9-11

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，这样就把它们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。

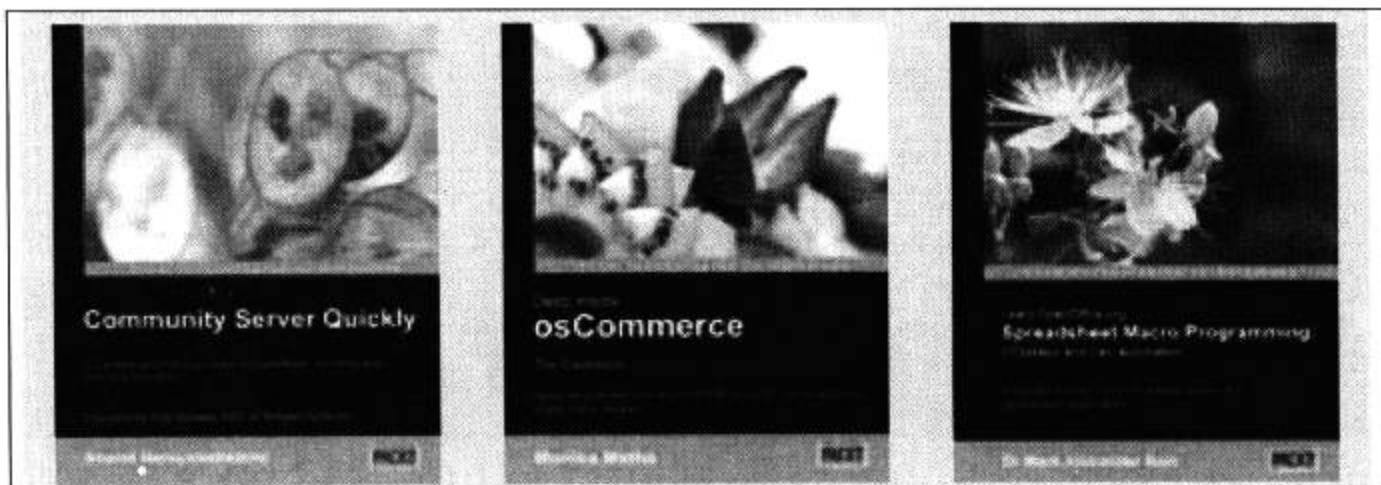


图 9-12

9.2.2 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——稍后我们就会看到，这一点很重要。

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（通过.css('left',1000)）。然后，再把位于队尾的图像（5号）转移到队列的前头（使用.prependTo()），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，函数中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9-13所示。



图 9-13

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。

1. 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);
};

9
```

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(-spacing)。然后，再把它连同其他图像一起滑进视图，如图9-14所示。

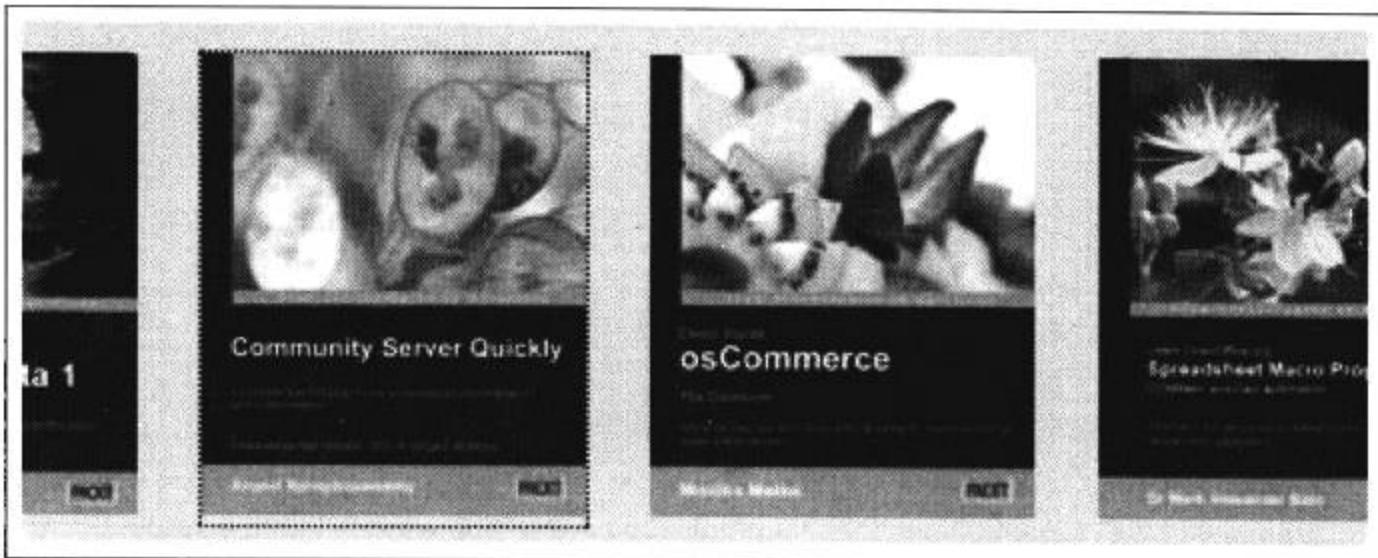


图 9-14

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。

单击最右侧的图像也会执行一组类似的动画效果，顺序相反。这一次，是最左侧的图像被移出视图，而且必须在动画完成并调用setUpCovers()之前将其移动到队尾。然后，新的、最右侧的图像必须在相应动画开始前移动到位(spacing * 3)。

2. 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，可以在用户鼠标指针悬停到图像上时，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9-15所示。



图 9-15

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个图标的HTML元素，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)
    .css('display', 'none');
var $rightRollover = $('')
    .attr('src', 'images/right.gif')
    .addClass('control')
    .css('opacity', 0.6)
    .css('display', 'none');
var $enlargeRollover = $('')
    .attr('src', 'images/enlarge.gif')
    .addClass('control')
    .css('opacity', 0.6)
    .css('display', 'none');
```

有读者可能注意到了，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)
        .css('display', 'none');
}
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;
}
```

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位置上以显示它们：

```
var setUpCovers = function() {
    var $covers = $('#featured-books .covers a');
    $covers.unbind('click mouseenter mouseleave');

    // 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。这里，我们使用了.unbind()方法的另一个特性，即通过以空格分隔事件名称，可以一次性解除多个事件处理程序的绑定。

为什么是mouseenter和mouseleave呢？在调用.hover()方法时，jQuery内部会绑定两个独立的事件。其中，将第一个函数绑定到mouseenter事件，将第二个函数绑定到mouseleave事件。因此，要想移除通过.hover()绑定的事件处理程序，需要解除对mouseenter和mouseleave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9-16所示。



图 9-16

9.2.3 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。



有很多支持在页面上显示信息的jQuery插件。其中比较流行的有FancyBox、ShadowBox、Thickbox、SimpleModal和jqModal。有关使用插件的更多信息，请参考第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9-17所示。

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：

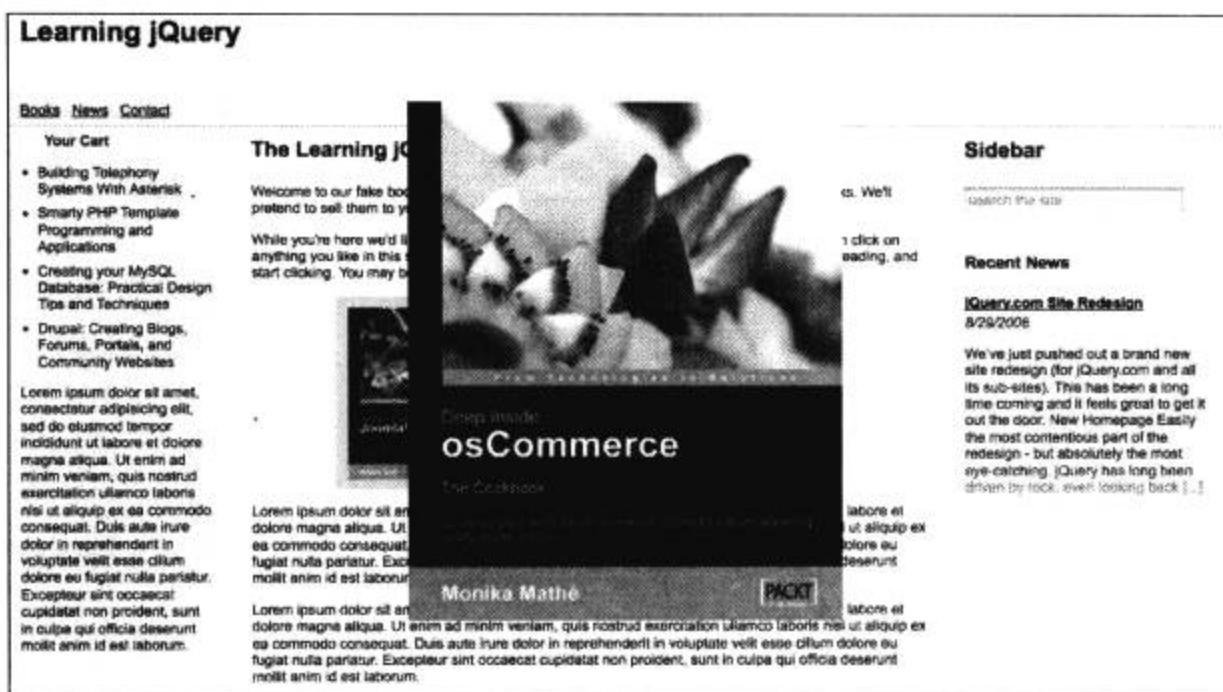


图 9-17

// 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。

2. 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给出提示。

为此，需要在大幅封面图像上添加一个关闭（Close）按钮作为标记。同创建前面使用的单个元素（即保证只显示一次的元素）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9-18所示。

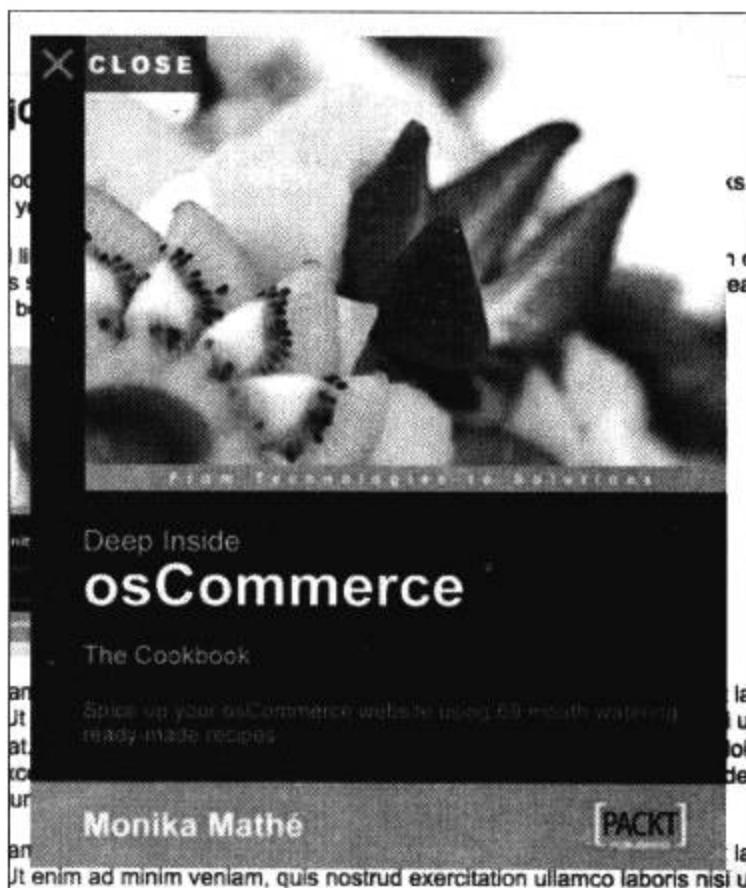


图 9-18

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();
});
$closeButton
.css({
  'left': ($('body').width() - 360) / 2,
  'top' : 100
})
.show()
.click(function() {
  $enlargedCover.click();
});
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。

3. 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，关键字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9-19所示，价格标记被定位在了放大图像的右上角。



图 9-19

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。

4. 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。为了改进这一点，可以使用jQuery内置的动画能力，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。计算这个位置通常需要使用纯JavaScript执行一些巧妙的DOM遍历，但jQuery再一次为我们提供了快捷方法。jQuery的`.offset()`方法返回的对象包含元素相对于页面的`left`和`top`坐标值。然后，我们可以把图像的`width`和`height`插入到这个对象中，从而得到一个包含位置数据小信息包。

```
var startPos = $(this).offset();
startPos.width = $(this).width();
startPos.height = $(this).height();
```

在这些位置信息的基础上再计算目标坐标就简单多了。同样，也可以把目标坐标集中放在一个对象中。

```
var endPos = {};
endPos.width = startPos.width * 3;
endPos.height = startPos.height * 3;
endPos.top = 100;
endPos.left = ($('#body').width() - endPos.width) / 2;
```

现在，我们可以把这个对象作为CSS属性的映射，将它们传递给`.css()`或`.animate()`方法。

```
$enlargedCover.attr('src', $(this).attr('href'))
.css(startPos)
.show()
.animate(endPos, 'normal', function() {
    $enlargedCover
        .one('click', function() {
            $closeButton.unbind('click').hide();
            $priceBadge.hide();
            $enlargedCover.fadeOut();
        });
    $closeButton
        .css({
            'left': endPos.left,
            'top' : endPos.top
        })
        .show()
        .click(function() {
            $enlargedCover.click();
        });
    $priceBadge
        .css({
            'right': endPos.left,
            'top' : endPos.top
        })
        .text(price)
        .show();
});
```

注意，不能在动画完成之前放置关闭按钮和价格标记，因此还要把相关代码移至`.animate()`方法的回调函数中。此外，可以利用这个机会重用前面计算的放大后封面的位置信息，针对这两个元素来简化对`.css()`的调用。

这样，就实现了从小封面到大封面的平滑变换效果（如图9-20至图9-23所示）。

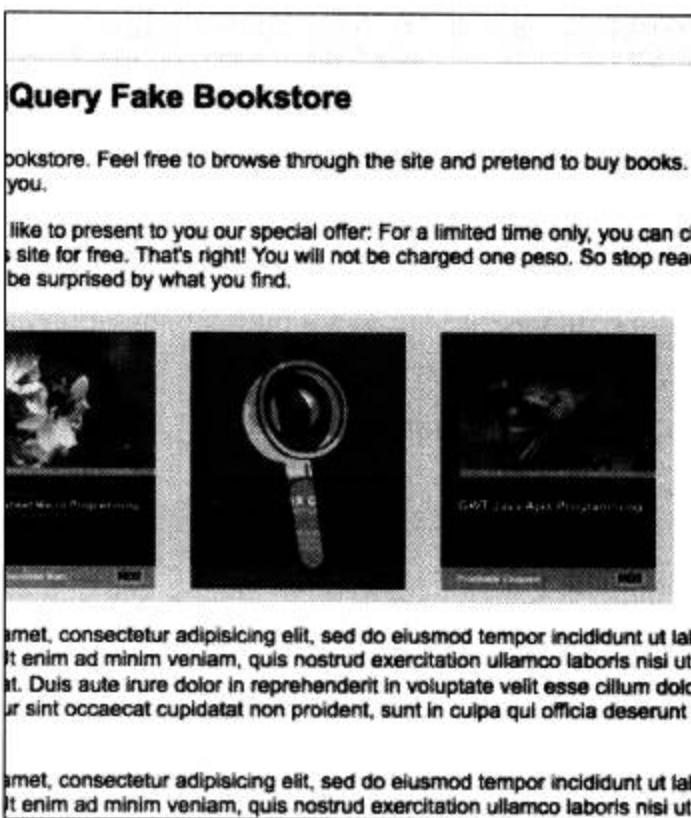


图 9-20

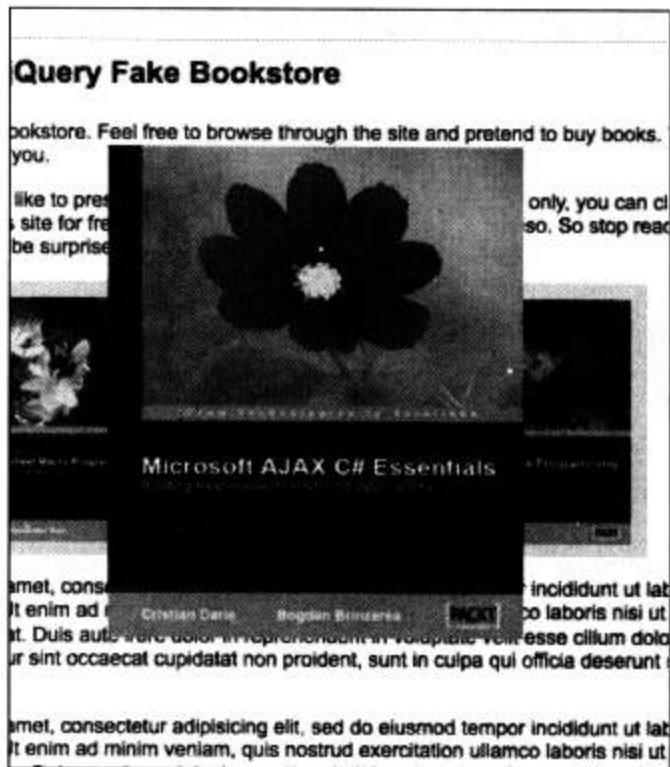


图 9-21

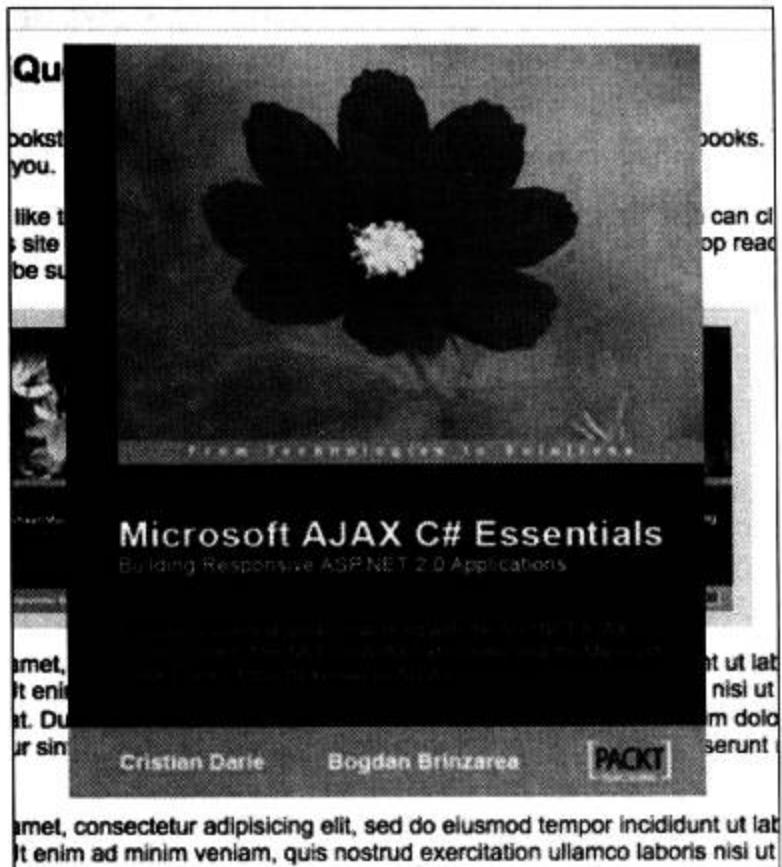


图 9-22

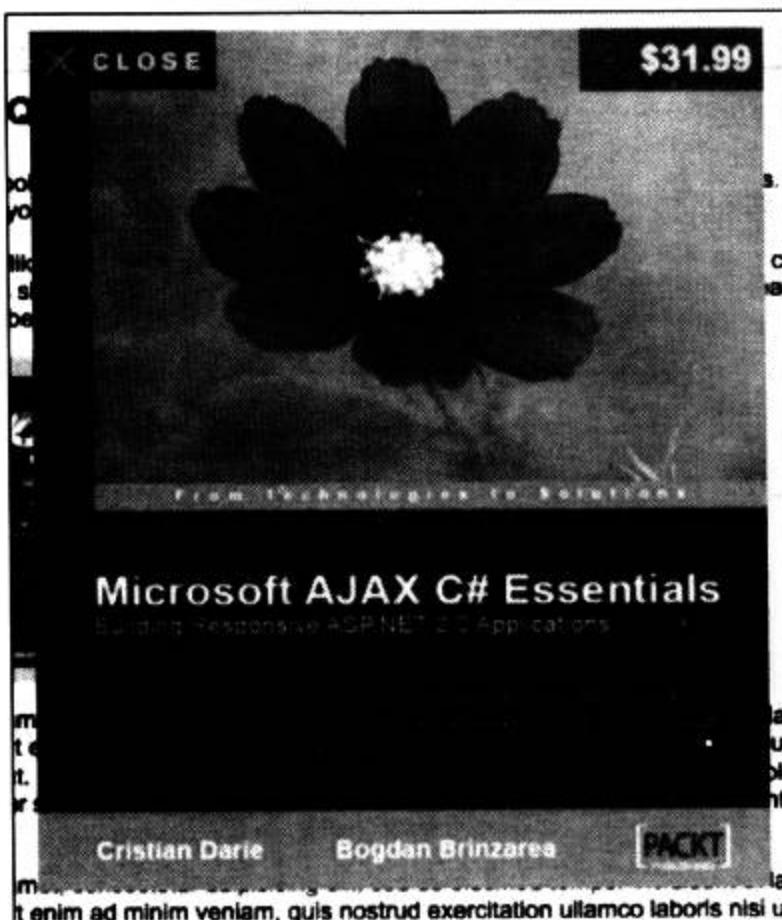


图 9-23

5. 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（X），或者仍然显示前一幅图像。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(startPos)
.show();
var performAnimation = function() {
    $enlargedCover.animate(endPos, 'normal', function() {
        $enlargedCover.one('click', function() {
            $closeButton.unbind('click').hide();
            $priceBadge.hide();
            $enlargedCover.fadeOut();
        });
        $closeButton
            .css({
                'left': endPos.left,
                'top' : endPos.top
            })
            .show()
            .click(function() {
                $enlargedCover.click();
            });
        $priceBadge
            .css({
                'right': endPos.left,
                'top' : endPos.top
            })
            .text(price)
            .show();
    });
};

if ($enlargedCover[0].complete) {
    performAnimation();
}
else {
    $enlargedCover.bind('load', performAnimation);
}
```

在此必须考虑两种情况：一种是图像可能会立即可用（比如使用了缓存），另一种是需要一定的加载时间。在第一种情况下，图像的complete属性值将为true，因此可以立即调用performAnimation()函数。在第二种情况下，则必须等到图像加载完成再调用PerformAnimation()。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。



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。

6. 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9-24所示。

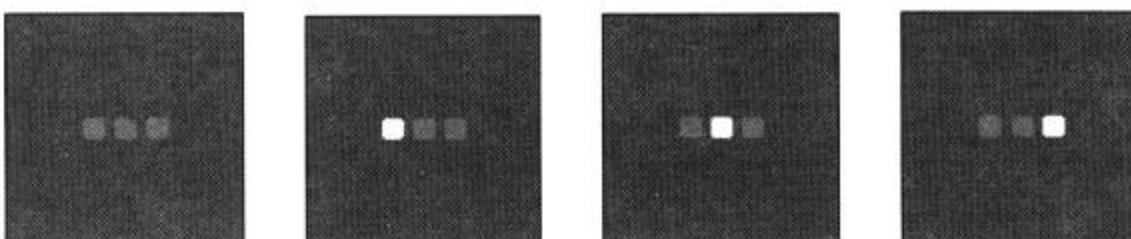


图 9-24

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9-25所示。



图 9-25

9.2.4 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，图像“传送带”的代码如下所示：

```
$(document).ready(function() {
    var spacing = 140;

    function createControl(src) {
        return $('')
            .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 = $('')
        .addClass('enlarged')
        .hide()
        .appendTo('body');
    var $closeButton = createControl('images/close.gif')
        .addClass('enlarged-control')
        .appendTo('body');
    var $priceBadge = $('')
        .addClass('enlarged-price')
        .css('opacity', 0.6)
        .css('display', 'none')
        .appendTo('body');
    var $waitThrobber = $('')
        .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 mouseenter mouseleave');
        // 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 startPos = $(this).offset();
    startPos.width = $(this).width();
    startPos.height = $(this).height();
    var endPos = {};

```

```
endPos.width = startPos.width * 3;
endPos.height = startPos.height * 3;
endPos.top = 100;
endPos.left = $('body').width() - endPos.width) / 2;
$enlargedCover.attr('src', $(this).attr('href'))
.css(startPos)
.show();
var performAnimation = function() {
$waitThrobber.hide();
$enlargedCover.animate(endPos, 'normal',
function() {
$enlargedCover.one('click', function() {
$ccloseButton.unbind('click').hide();
$priceBadge.hide();
$enlargedCover.fadeOut();
});
$ccloseButton
.css({
'left': endPos.left,
'top' : endPos.top
})
.show()
.click(function() {
$enlargedCover.click();
});
$priceBadge
.css({
'right': endPos.left,
'top' : endPos.top
})
.text(price)
.show();
});
);
if ($enlargedCover[0].complete) {
performAnimation();
}
else {
$enlargedCover.bind('load', performAnimation);
}
event.preventDefault();
})
.hover(function() {
$enlargeRollover.appendTo(this).show();
}, function() {
$enlargeRollover.hide();
});
```

```
};  
setUpCovers();  
});
```

9.3 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，同时还可以增强Web应用程序的可用性。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。

贯穿整本书，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。

jQuery社区不断发展，其创造的插件已经达到了数百个——小到选择器助手，大到全屏的用户界面部件。第7章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。本章介绍如何查找其他人开发的插件，以及如何在网页中使用它们。我们首先探索流行的Form插件和官方jQuery UI插件库，然后再罗列并简单介绍一些普遍受到欢迎的、“（本书）作者推荐”的插件。

10.1 查找插件和帮助

jQuery官方网站的插件库（地址为<http://plugins.jquery.com/>）囊括了大量插件，并给出了每个插件的用户评级、版本及bug报告。这个插件库也是查找文档的很好起点。库中列出的每个插件都提供ZIP文件下载，而且其中不少还有演示、示例代码及教程的链接。

此外，在通用代码库（如<http://github.com/>）及插件开发者的博客中，还可以找到更多插件。

假如在上述官方网站的插件库、作者网站及插件说明中，仍然找不到相关问题的答案，还可以访问jQuery Google Group，地址为<http://groups.google.com/group/jquery-en/>。很多插件作者都是这个讨论组的积极参与者，他们会愿意帮助解答新用户面临的难题。

10.2 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.plugin.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。一般来说，可以在自定义文件的`$(document).ready()`方法中添加一行代码，以调用某些操作：

```
$(document).ready(function() {
    $('#myID').somePlugin();
});
```

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，我们就可以按照需要自定义插件的操作，通常为方法传递一个映射作为参数。

```
$(document).ready(function() {
    $('#myID').somePlugin ({
        send: true,
        message: 'This plugin is great!'
    });
});
```

由此可见，jQuery插件的语法与jQuery核心方法的语法非常类似。知道了如何在网页中包含插件之后，下面我们就来看两个流行的插件。

10.3 Form 插件

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。这个插件的文件及其详细的文档，可以在<http://malsup.com/jquery/form/>查到。

这个插件的核心是`.ajaxForm`方法。利用该方法，基于常规的表单生成一次AJAX表单请求只需区区一行代码：

```
$(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（HTTP响应）。
- resetForm: 布尔值，默认为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，默认为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

一般来说，.ajaxForm()方法要比.ajaxSubmit()方法更方便，但会牺牲一点灵活性。在想让插件替我们处理所有事件绑定事宜，且在适当时候代替我们调用.ajaxSubmit()时，应该使用.ajaxForm()。在希望能够更精细地控制提交时事件绑定时，则使用.ajaxSubmit()更好。

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。而且，这样还可以获得另一个好处，即能够让表单自动具有渐进增强的能力，就算没有JavaScript，表单也仍然可以正常使用。

通常，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name和value属性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<input type="submit">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。

除非是通过表单提交上传文件，否则无论是.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：

```

$(document).ready(function() {
    $('#myForm').ajaxForm({
        timeout: 2000,
        error: function (xml, status, e) {
            alert(e.message);
        }
    });
});

```

此外，在不需要多少自定义的情况下，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：

```

$(document).ready(function() {
    $('#myForm').ajaxForm(function(responseText) {
        alert(responseText);
    });
});

```

10.4 jQuery UI 插件库

与Form插件只做一件事而且做好一件事相比，jQuery UI能够做的事则可谓包罗万象（而且，做得也都很好）。实际上，jQuery UI在某种意义上并不是插件，而是一个完整的插件库。

由Paul Bakaus领导的jQuery UI团队，创建了大量核心交互组件及成熟的部件（widget），使用它们可以创造出更加类似桌面应用程序的Web体验。交互式组件包括用于拖动、放置、排序和调整项目大小的方法。当前稳定的部件有折叠窗格、日期选择器、对话框、滑动条和标签页，还有另外一些也在积极开发中。此外，jQuery UI还为补充和增强jQuery的核心动画功能，提供了相当多的高级效果。

鉴于本章不可能面面俱到地介绍jQuery UI库，因此我们只能重点讨论UI效果、Sortable核心交互组件和Dialog部件。访问<http://jqueryui.com/>，可以下载所有jQuery UI模块，或者查看相应的文档及示例。

10.4.1 效果

jQuery UI中的效果（effect）模块由一个核心文件和一组独立的效果文件组成。其中，核心文件为创建颜色动画和基于类的动画提供了支持，同时也提供了高级的缓动函数。

1. 颜色动画

在文档中引用核心效果文件的情况下，扩展的`.animate()`方法可以接受另外一些样式属性，例如`borderTopColor`、`backgroundColor`和`color`。下面的代码实现的效果就是将白色背景上的黑色文本逐渐变为黑色背景上的白色文字：

```

$(document).ready(function() {
    $('#mydiv').animate({
        color: '#fff',
    });
});

```

```

    backgroundColor: '#000'
}, 'slow');
});

```

在动画效果进行了差不多一半时的<div>效果如图10-1所示。

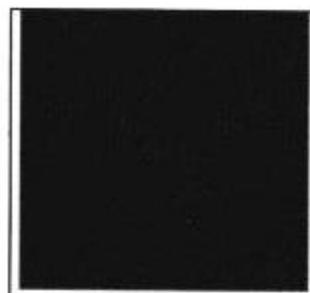


图 10-1

这是元素变换过程中的一的画面，文本正由黑变白，而背景则越来越黑。

2. 基于类的动画

前几章介绍过3个操作类的方法：`.addClass()`、`.removeClass()`和`.toggleClass()`。现在，这3个方法都可以接受第二个可选的参数，用于控制动画时长。换句话说，我们可以这样使用它们：`.addClass('highlight', 'fast')`、`.removeClass('highlight', 'slow')`或`toggleClass('highlight', 1000)`。

3. 高级缓动函数

高级缓动函数用于在动画过程中的各个时点上改变变换的速度和距离。例如，`easeInQuart`函数让动画以其开始速度的4倍结束。在任何核心jQuery动画方法或jQuery UI效果方法中，都可以指定自定义的缓动函数。具体指定方式根据使用的语法不同，可能是添加一个参数，也可能是为选项映射中添加一个选项。以前面的颜色动画为例，可以通过为`.animate()`方法多传递一个参数来指定`easeInQuart`函数：

```

$(document).ready(function() {
  $('#mydiv').animate({
    color: '#fff',
    backgroundColor: '#000'
  }, 'slow', 'easeInQuart');
});

```

也可以通过在另一个选项映射中添加选项来实现：

```

$(document).ready(function() {
  $('#mydiv').animate({
    color: '#fff',
    backgroundColor: '#000'
  }, {
    duration: 'slow',
    easing: 'easeInQuart'
  });
});

```

要了解所有缓动函数的演示效果, 请读者访问<http://gsgd.co.uk/sandbox/jquery/easing/>。

4. 其他效果

效果模块的独立效果文件中包含了非常多的变换, 而这些变换都可以通过`.effect()`方法实现, 其中一些变换则是jQuery的`.show()`、`.hide()`和`.toggle()`方法的扩展。例如, 爆炸(`explode`)效果会把元素“炸”成指定的块数, 然后隐藏元素, 而使用`.effect()`方法可以实现相同效果:

```
$(document).ready(function() {
    $('#explode').effect('explode', {pieces: 16}, 800);
});
```

另外, 使用`.hide()`方法也可以实现相同效果:

```
$(document).ready(function() {
    $('#explode').hide('explode', {pieces: 16}, 800);
});
```

无论采用上述哪种方式, 隐藏盒子的效果最初都如图10-2所示。

在动画变换期间, 则会看到如图10-3所示的效果。



图 10-2

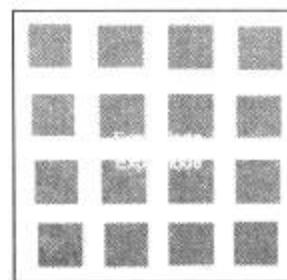


图 10-3

动画结束后, 盒子就完全不见了。

10.4.2 交互组件

在jQuery UI的交互式组件中, Sortable是比较有特色的一种。Sortable可以将任何元素组转换为拖放风格的列表。图10-4是一个无序列表, 我们为其中每个项应用了一些CSS样式。

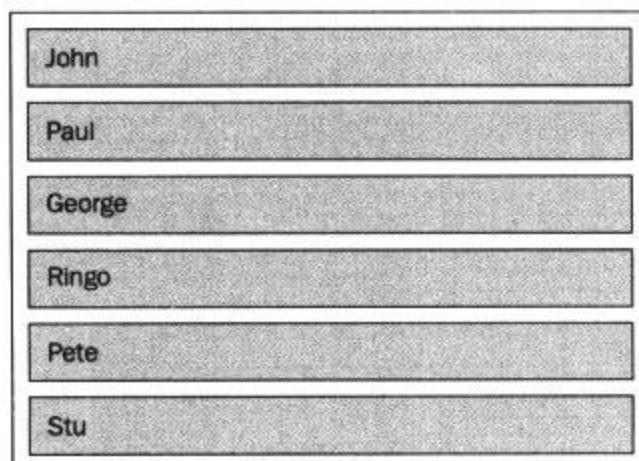


图 10-4

HTML代码非常简单：

```
<ul id="sort-container">
  <li>John</li>
  <li>Paul</li>
  <li>George</li>
  <li>Ringo</li>
  <li>Pete</li>
  <li>Stu</li>
</ul>
```

现在，要想让这个列表可以通过拖放排序，只需下列几行代码即可：

```
$(document).ready(function() {
  $('#sort-container').sortable();
});
```

`$(document).ready()` 中的这一行代码，就足以让用户可以拖动每个项，然后再把该项放在列表中的其他位置上（如图10-5所示）。

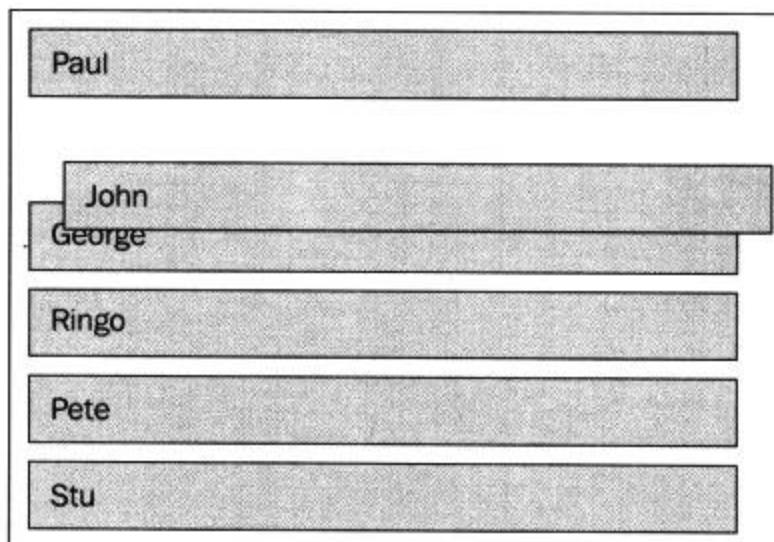


图 10-5

通过向`.sortable()`方法中添加选项，可以创建更丰富的用户交互。这个方法支持30多种选项，但我们在此只演示其中几个：

```
$(document).ready(function() {
  $('#sort-container').sortable({
    opacity: .5,
    cursor: 'move',
    axis: 'y'
  });
});
```

前两个选项`opacity`和`cursor`的作用一目了然。第三个选项`axis`的作用，是把排序过程中元素的移动限定在特定的轴向上（在此是y轴），如图10-6所示。

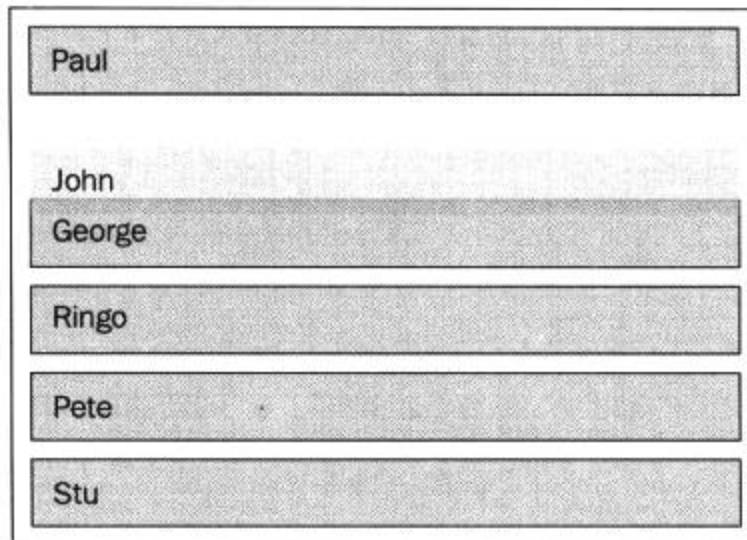


图 10-6

通过被排序元素的浅颜色背景很容易看出，我们在此也利用了为其自动添加的一个类（`ui-sortable-helper`）^①，在样式表中为它应用了样式。

要了解与jQuery UI核心交互组件相关的更多信息，请访问<http://docs.jquery.com/UI#Interaction>。

10.4.3 部件

除了交互式组件之外，jQuery UI库中还提供了一批稳定的用户界面部件。无论从外观还是功能上看，这些“开箱即用”的部件都非常类似我们熟悉的桌面应用程序中的相应元素。例如，Dialog部件利用可拖动和可调整大小的组件生成了一个对话框，从而免去了我们自己编写的麻烦。

与其他UI部件一样，Dialog部件可以接受很多选项。这个部件名副其实的`.dialog()`方法能够接受相应的字符串参数，用以改变对话框的行为。在最基本的情况下，`.dialog()`方法可以将现有元素转换为一个对话框并在其中将元素内容显示出来。例如，可以从下面这个简单的`<div>`结构开始。

```
<div id="dlg">My Dialog</div>
```

毫不奇怪，这个`<div>`元素看起来非常普通——就是一个简单的文本块（如图10-7所示）。

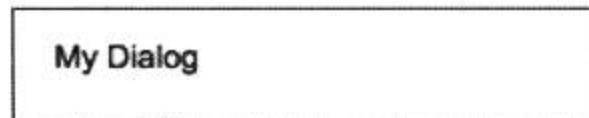


图 10-7

只要DOM就绪，就可以在JavaScript文件中调用`.dialog()`方法并生成基本的对话框。

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

此时，原来的文本被包装在了对话框中（如图10-8所示）。

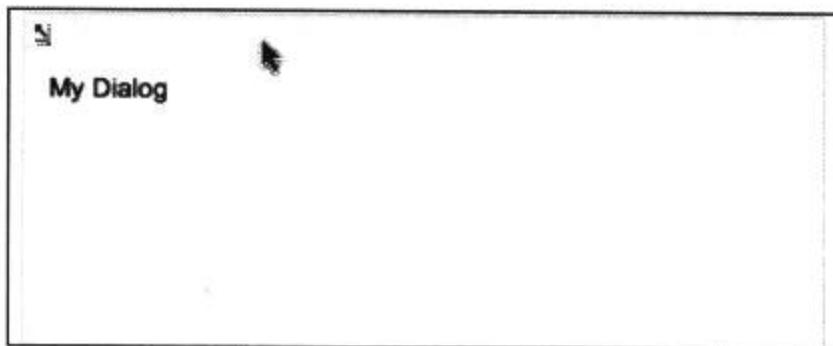


图 10-8

单击并拖动对话框的边框，可以调整对话框的大小。单击对话框上边框之下的顶部区域可以移动对话框。而单击其右上角的X按钮，则可以关闭它。

不过，通过为对话框应用样式，应该还可以得到更好的结果。虽然jQuery UI为确保部件的运作，也提供了最低限度的样式，但它还是把对部件外观的决定权留给了我们。图10-9是应用默认主题后的对话框。

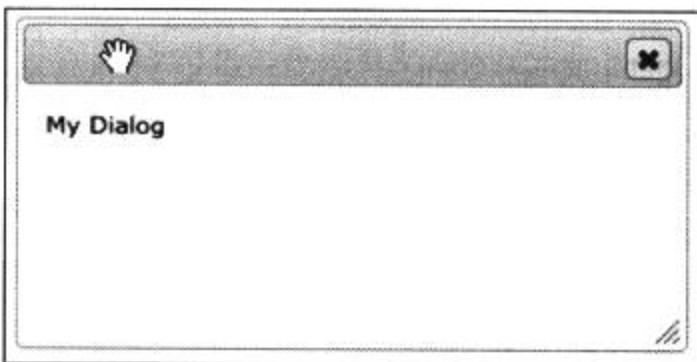


图 10-9

现在，对话框的各个区域得到了非常清晰的表现，而鼠标指针的变化，更是为对话框的某一部分可以拖动和可以用来调整大小提供了明确的视觉反馈。

与jQuery UI中的其他方法一样，`.dialog()`方法也有很多选项。其中一些选项用于指定对话框的外观，其他一些选项则会影响事件的触发。下面给出了部分选项的示例：

```
$(document).ready(function() {
    var $dlg = $('#dlg');
    var dlgText = $dlg.text();
    $dlg.dialog({
        autoOpen: false,
        title: dlgText,
        open: function() {
            $dlg.empty();
        },
        buttons: {
            'add message': function() {
                $dlg.append('<p>Inserted message</p>');
            }
        }
    });
});
```

```

},
'erase messages': function() {
    $('p', $dlg).remove();
}
});
$('#do-dialog').click(function() {
    $dlg.dialog('open');
});
});
});

```

在此，我们将对话框设置为初始隐藏状态，待用户单击id="do-dialog"的按钮时再打开。将对话框中原来的文本移动到了其标题区，同时添加了两个按钮：一个用于向对话框中添加文本，另一个用于清除对话框中的文本。这两个按钮分别有一个响应单击事件的关联函数，用于添加和移除文本。在单击3下add message按钮后，包含以上选项的对话框如图10-10所示。

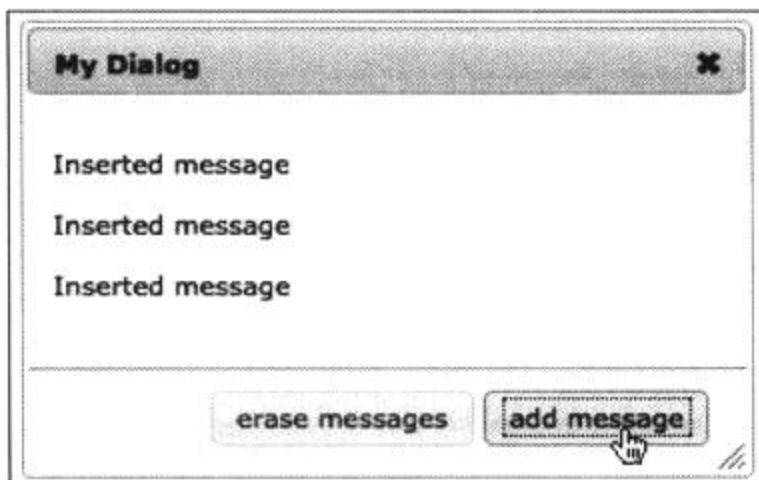


图 10-10

访问[http://jqueryui.com/demos/dialog/#options^①](http://jqueryui.com/demos/dialog/#options)，可以找到配置对话框外观和行为的很多其他选项。

10.4.4 jQuery UI ThemeRoller

jQuery UI库最近增添的一项名为ThemeRoller（主题卷轴）的功能，这是一个面向UI部件的基于Web的交互式主题引擎。有了ThemeRoller，就可以在瞬间创建出高度自定义、专业化的元素。如前例所示，我们为刚才创建的对话框应用了默认主题；在没有应用自定义设置的情况下，这个主题可以通过ThemeRoller输出（如图10-11所示）。

访问<http://jqueryui.com/themeroller/>，根据需要修改不同选项，然后单击Download theme按钮，即可得到完全自定义的样式。下载后的样式表文件及图像被打包在一个.zip文件中，将该文件解压缩至适当文件夹后，即可使用它们。例如，通过选择不同的颜色和纹理，就可以在几分钟内把前面的对话框外观修改成如图10-12所示。

^① 原文链接已经无效。——译者注

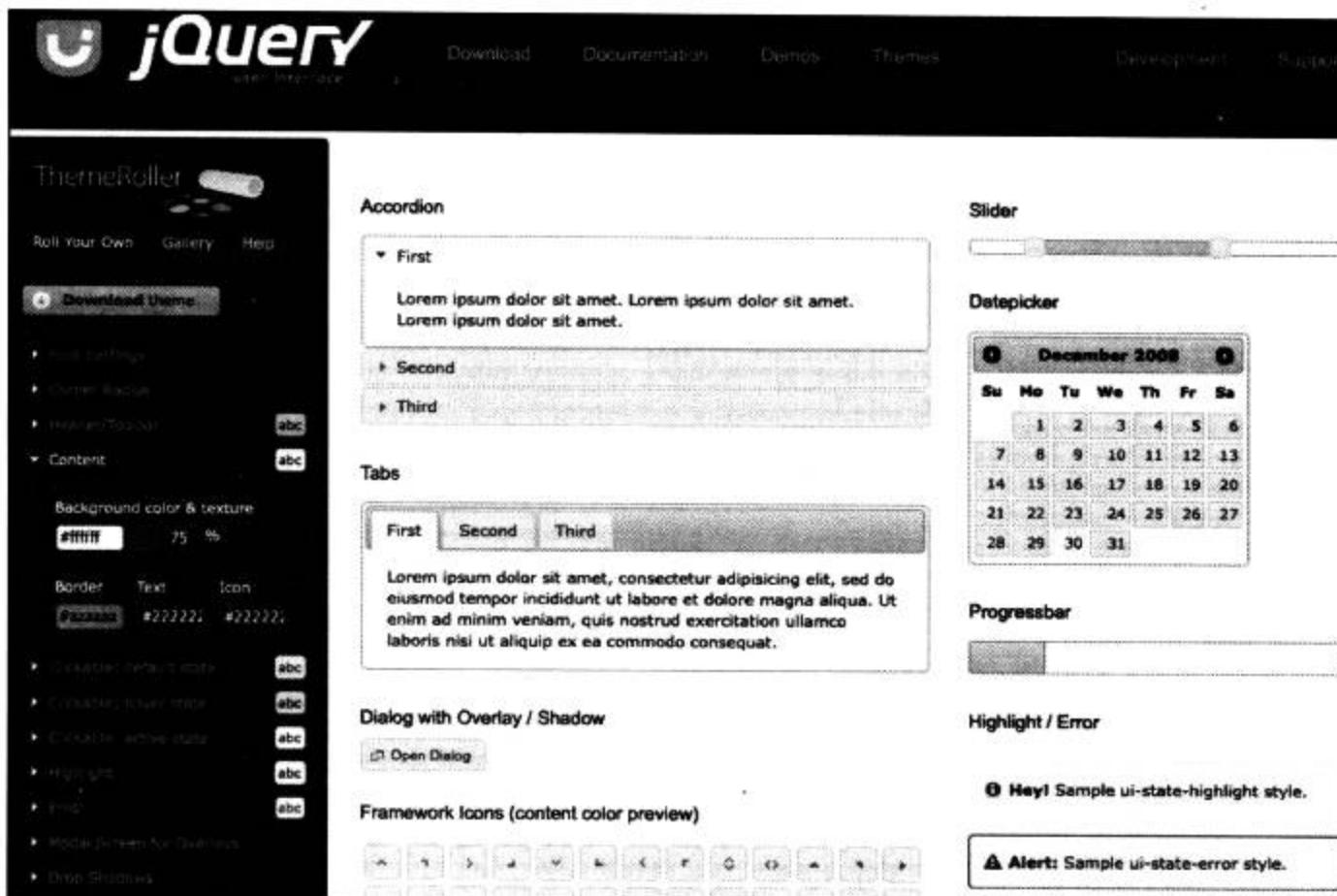


图 10-11



图 10-12

10.5 其他插件

除了本章及本书其他章节提到的插件之外，下列流行且代码可靠的插件也是本书作者推荐的。

10.5.1 表单类

第8章讨论了各种操作表单的方式，而使用下列插件则可以帮助我们轻松地完成类似任务。

1. Autocomplete

<http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/>

<http://plugins.jquery.com/project/autocomplete>

Autocomplete插件由jQuery核心开发人员Jörn Zaefferer编写，该插件能够在用户填写文本输入字段时显示可能的匹配项列表，如图10-13所示。

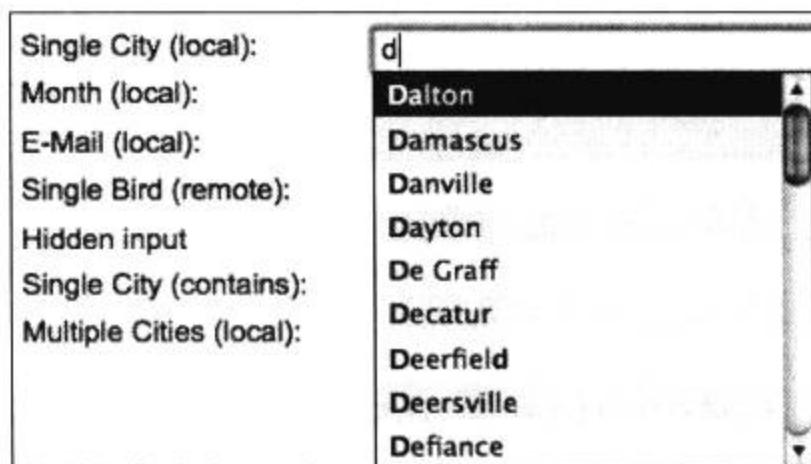


图 10-13

2. Validation

<http://bassistance.de/jquery-plugins/jquery-plugin-validation/>

<http://plugins.jquery.com/project/validate>

Validation是Jörn Zaefferer开发的另一个插件，作为一个极为灵活的工具，该插件可以基于各种标准来验证表单输入字段，如图10-14所示。

The screenshot shows a validation form titled "Validating a complete form". It contains six input fields: Firstname, Lastname, Username, Password, Confirm password, and Email. Each field has a corresponding error message to its right: "Please enter your firstname", "Please enter your lastname", "Please enter a username", "Please provide a password", "Please provide a password", and "Please enter a valid email address". Below the inputs is a checkbox labeled "Please agree to" with the options "Please accept our policy" and "our policy".

图 10-14

3. Jeditable

<http://www.appelsiini.net/projects/jeditable>

<http://plugins.jquery.com/project/jeditable>

Jeditable插件可以在响应用户的某些操作（如单击或双击）时，将非表单元素转换为可以编辑的输入字段。修改后的內容可以自动发送回服务器保存起来，如图10-15所示。

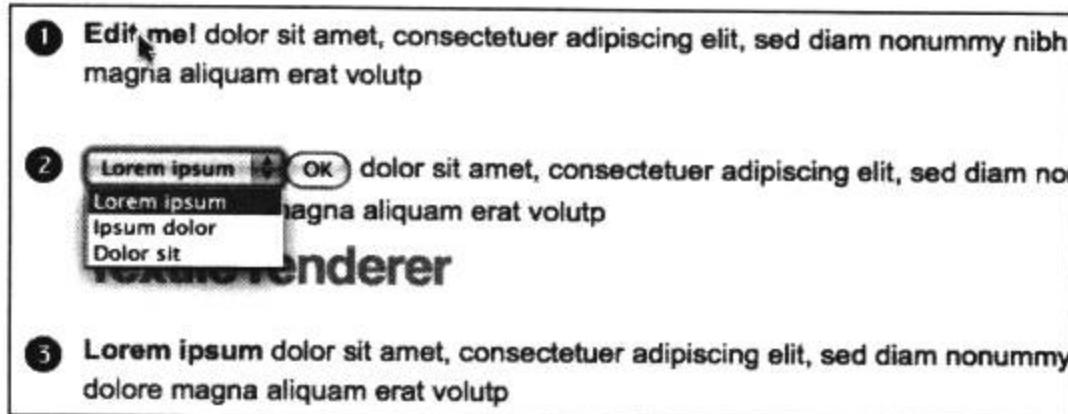


图 10-15

4. Masked input

<http://digitalbush.com/projects/masked-input-plugin/>

<http://plugins.jquery.com/project/maskedinput>

Masked Input输入插件为用户输入既定格式的数据（如日期、电话号码、社会保障号码）提供了一种更简便的方式。该插件可以自动在字段中放入某些字符（例如日期中的斜杠），同时也允许用户输入插件选项中指定的另外一组字符，如图10-16所示。

The following example is a demonstration from the usage tab.

Date	11/04/_____	99/99/9999
Phone	_____	(999) 999-9999
Tax ID	_____	99-99999999
SSN	_____	999-99-9999
Product Key	_____	a*-999-a999
Eye Script	_____	~9.99 ~9.99 999

图 10-16

10.5.2 表格类

第7章曾讨论过排列、美化及增强表格数据的技术。事实上，许多插件开发者已经针对类似任务发布了很多插件。

1. Tablesorter

<http://tablesorter.com/>

<http://plugins.jquery.com/project/tablesorter>

Tablesorter插件可以把任何带有`<thead>`和`<tbody>`元素的表格，转换为无需刷新页面即可排序的表格。该插件的特色功能包括多列排序、针对不同格式（例如日期、时间、货币、URL等）排序的解析器、次要“隐藏”排序，另外还可以通过一个部件系统对其进行扩展，如图10-17所示。

First Name	Last Name	Age	Total	Discount	Difference	Date
Bruce	Evans	22	\$13.19	11%	-100.9	Jan 18, 2007 9:12 AM
Bruce	Aimothy	45	\$153.19	44.7%	+77	Jan 18, 2001 9:12 AM
Clark	Kent	18	\$15.89	44%	-26	Jan 12, 2003 11:14 AM
John	Hood	33	\$19.99	25%	+12	Dec 10, 2002 5:14 AM
Peter	Parker	28	\$9.99	20.9%	+12.1	Jul 6, 2006 8:14 AM

图 10-17

2. jqGrid

<http://www.trirand.com/blog/>

<http://plugins.jquery.com/project/jqGrids>

jqGrid是一个AJAX驱动的JavaScript控件，支持在Web页面中动态展示和操作表格数据。该插件提供的选项包括行内编辑、单元格编辑、分页导航、多项选择、子网格以及树状网格等，如图10-18所示。这个插件还配有完备的使用文档，参见<http://www.secondpersonplural.ca/jqgriddocs/>。

图 10-18

3. Flexigrid

<http://code.google.com/p/flexigrid/>

<http://plugins.jquery.com/project/flexigrid>

Flexigrid插件与jqGrid类似，也是一个功能齐全的网格插件。该插件支持JSON数据、分页显示、快速搜索、表列的显示隐藏和调整大小，还支持行排序，如图10-19所示。

图 10-19

10.5.3 图像类

图像操作通常是一些需要较多依赖于服务器端处理的任务。然而，许多插件作者以jQuery为媒介，开发了很多通过JavaScript来简化图像处理的插件。

1. Jcrop

<http://deepliquid.com/content/Jcrop.html>

<http://plugins.jquery.com/project/Jcrop>

Jcrop插件为Web应用程序增添了一种裁切图像的快捷方式。该插件的功能包括锁定纵横比、限定最大和最小尺寸、键盘微调、交互式挂勾（hook）和自定义样式，如图10-20所示。

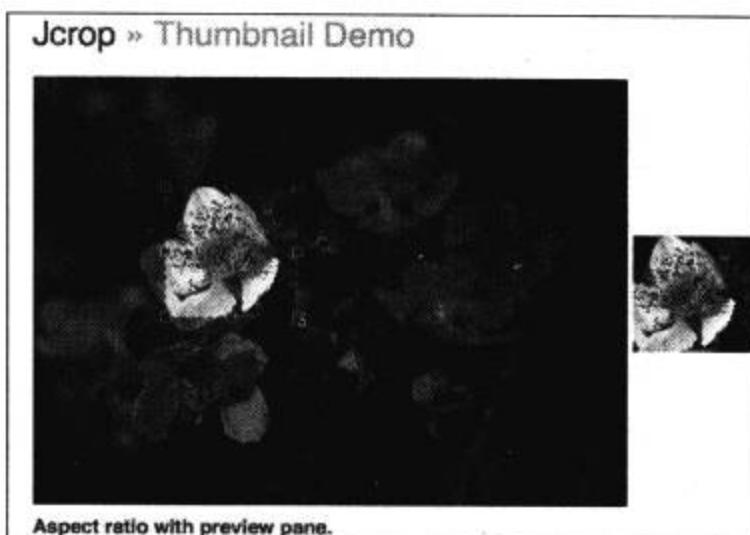


图 10-20

2. Magnify

<http://www.jnathanson.com/index.cfm?page=pages/jquery/magnify/magnify>

<http://plugins.jquery.com/project/magnify>

在拥有一大一小两张比例相当、内容完全相同的图片时，可以使用Magnify生成一个“放大镜”，特别适合查看产品细节或模仿特写镜头，如图10-21所示。

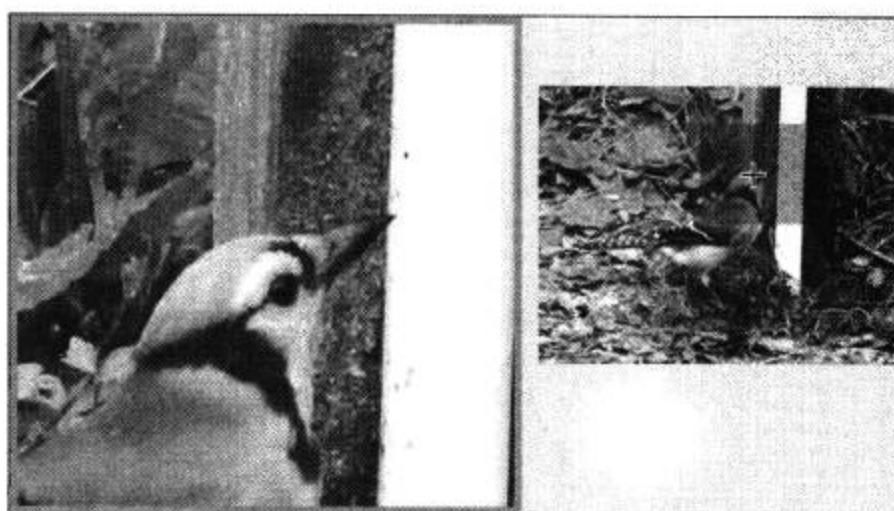


图 10-21

10.5.4 亮盒及模态对话框

第9章中有一个例子，展示了无需使用弹出式窗口即可在页面上覆盖显示详细信息的技术，这种技术被称为亮盒效果（lightbox）。下列插件可以辅助创建类似的覆盖式亮盒效果。

1. FancyBox

<http://fancy.klade.lv/>

这个亮盒插件以样式见长，具有类似Mac的外观和优雅的投影效果。除了显示自动缩放的图像之外，FancyBox插件还可以展示行内或者`<iframe>`内容，如图10-22所示。

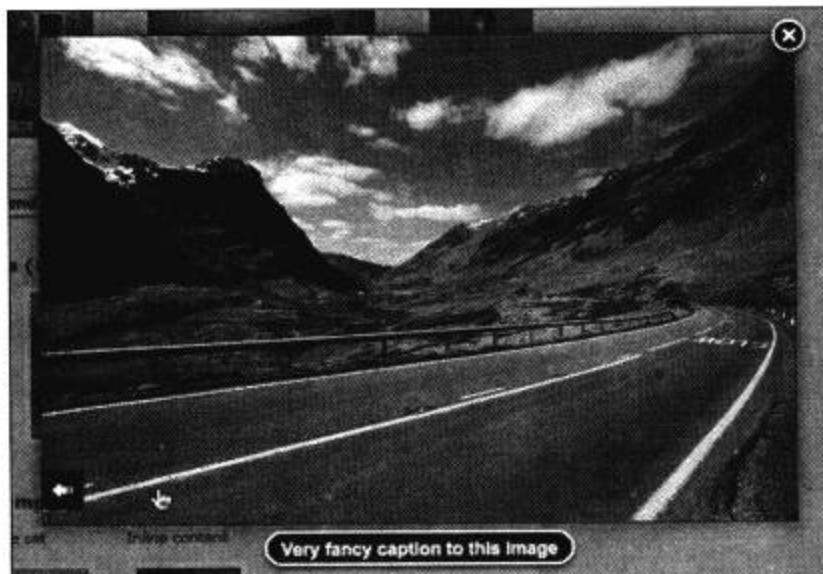


图 10-22

2. Thickbox

<http://jquery.com/demo/thickbox/>

Thickbox是一个功能全面的亮盒插件，它可以通过一个混合的模态对话框，展示一幅或多幅图像、行内内容、`<iframe>`内容，甚至通过AJAX提供的内容，如图10-22所示。



图 10-23

3. BlockUI

<http://malsup.com/jquery/block/>

<http://plugins.jquery.com/project/blockUI>

BlockUI插件模仿了同步行为，但不会锁定浏览器。在被激活时，它会阻止用户与页面（或页面某一部分）交互，直至被禁用，如图10-24所示。

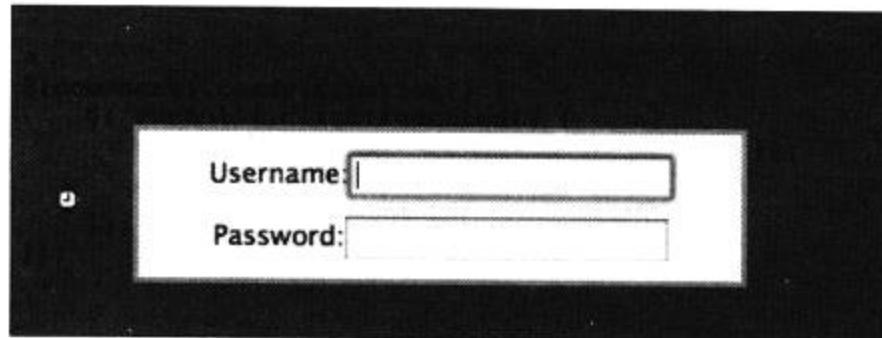


图 10-24

4. jqModal

<http://dev.iceburg.net/jquery/jqModal/>

<http://plugins.jquery.com/project/jqModal>

jqModal插件是一个轻量级的模态对话框解决方案，但也十分强大和灵活。这个插件以可扩展性为特色，支持使用它的Web开发人员扩展其交互能力和外观主题，如图10-25所示。

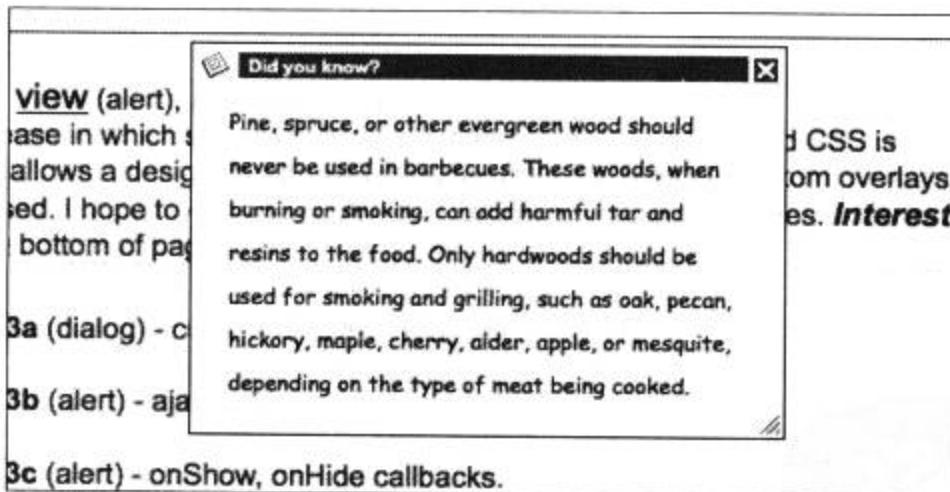


图 10-25

10.5.5 图表类

与图像操作类似，图表在过去也主要依赖于服务器端处理。善于发明创造的程序员们，已经开发出了很多在浏览器中创建图表的手段，下列就是以jQuery插件形式发布的几种有代表性的实现技术。

1. Flot

<http://code.google.com/p/flot/>

<http://plugins.jquery.com/project/plot>

Flot插件使用`<canvas>`元素来生成数据集的平面线图（plot），而且可以根据设置让用户来修改线图。由于集成了Excanvas变换脚本，Flot也可以在IE中渲染图形，实际上是将Canvas指令转换成了IE可以识别的VML格式，如图10-26所示。

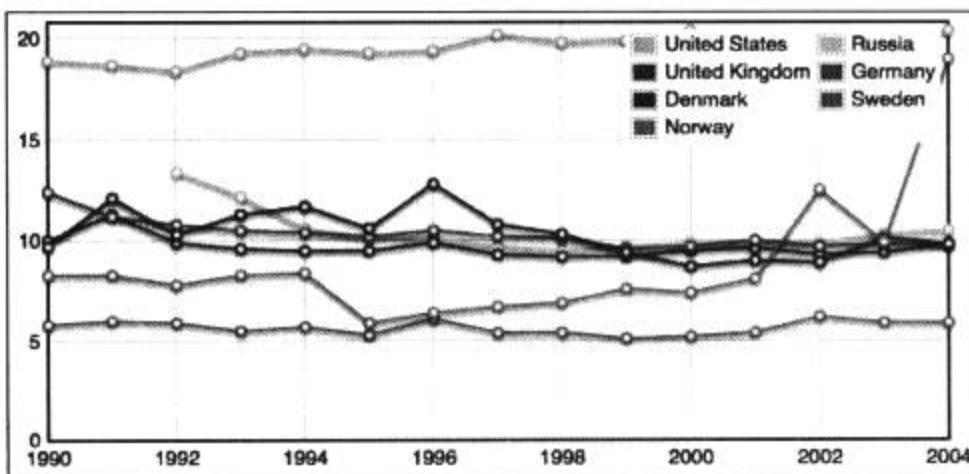


图 10-26

2. Sparklines

<http://omnipotent.net/jquery.sparkline/>

<http://plugins.jquery.com/project/sparklines>

这个以数据可视化专家Edward Tufte发明的概念Sparklines命名的插件，可以生成简单小巧的行内图表。与Flot相似的地方是，Sparklines也使用`<canvas>`元素渲染图表。不过，针对IE的转换则是由插件本身完成的，没有使用Excanvas，如图10-27所示。

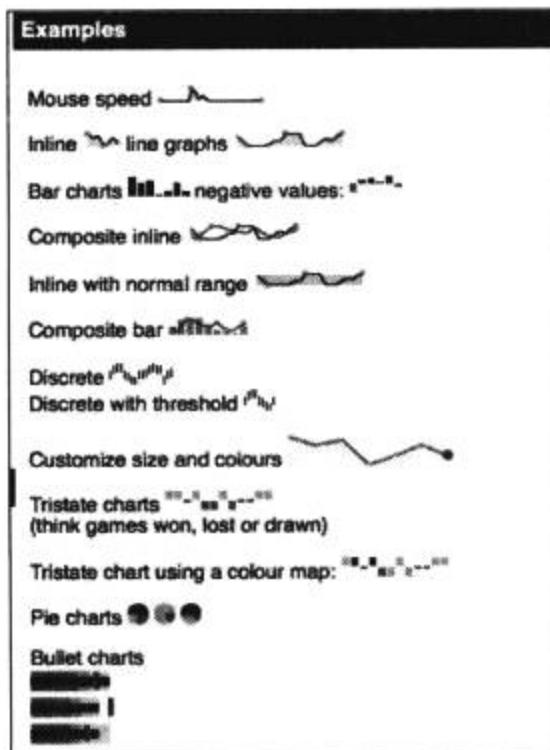


图 10-27

10.5.6 事件类

本书前面曾多次涉及事件，jQuery也为截取并响应用户事件（如鼠标单击或敲击键盘）提供了很多工具。虽然核心库包含的方法不少，但要实现的高级技术则是没有穷尽的。下列插件为实现一些较不常见的任务提供了帮助。

1. hoverIntent

<http://cherne.net/brian/resources/jquery.hoverIntent.html>

<http://plugins.jquery.com/project/hoverIntent>

hoverIntent插件提供了一个代替`.hover()`的方法，虽然同样根据用户鼠标指针移入或移出元素作出反应，但它可以避免用户意外触发不期望的动画。这个插件通过监控鼠标指针移动的速度来判断用户的意图（intent），特别适合用于下拉菜单动画。

2. Live query

<http://github.com/brandonaaron/livequery/>

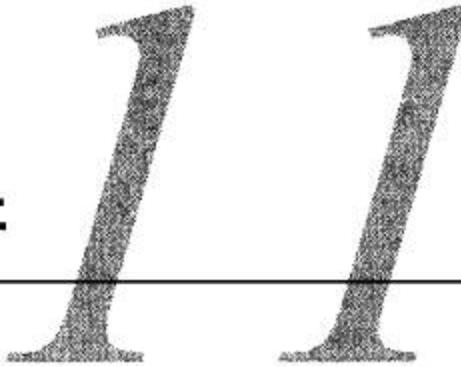
<http://plugins.jquery.com/project/livequery>

与jQuery内置的`.live()`方法类似，Live Query插件能够为DOM中的元素动态添加和维护事件绑定，无论相应元素是什么时候创建的。这个插件提供的实现在某些情形下可能是更合适的选择。

10.6 小结

本章介绍了在网页中整合第三方插件的各种方式。其中，着重讨论了Form插件和jQuery UI库。此外，也向读者展示了其他一些插件。下一章，我们探讨如何利用jQuery的插件架构，开发不同类型的自定义插件。

开发插件



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，这个过程与编写使用插件的代码相比，不会复杂多少。

本章，我们介绍几种不同插件的创建方法，有简单的也有复杂的。首先从添加新的全局函数的插件开始，然后再逐步讨论各种形式的jQuery对象方法。此外，本章还将探讨如何以新表达式来扩展jQuery的选择符引擎。最后，再介绍怎样把插件共享给其他开发人员。

11.1 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.globalFunction = 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.globalFunction();
```

而且，也可以使用\$别名并写成：

```
$().globalFunction();
```

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。

11.1.1 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.functionOne = function() {
    alert('This is a test. This is only a test.');
```

```

};

jQuery.functionTwo = function(param) {
    alert('The parameter 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：

```

$.functionOne();
$.functionTwo('test');

```

还可以使用`$.extend()`函数来使函数的定义更加清晰：

```

jQuery.extend({
    functionOne: function() {
        alert('This is a test. This is only a test.');
    },
    functionTwo: function(param) {
        alert('The parameter is "' + param + '".');
    }
});

```

以上代码也会得到同样的结果。

不过，这样也会面临不同命名空间冲突的风险。即使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 = {
    functionOne: function() {
        alert('This is a test. This is only a test.');
    },
    functionTwo: function(param) {
        alert('The parameter is "' + param + '".');
    }
};

```

这样其实是为全局函数创建了另一个命名空间——`jQuery.myPlugin`。尽管某种程度仍然可以称它们为“全局函数”，但它们实际上已经变成了`myPlugin`对象的方法，而`myPlugin`对象则是全局`jQuery`对象的一个属性。因此，在调用这些函数时必须包含插件的名字：

```

$.myPlugin.functionOne();
$.myPlugin.functionTwo('test');

```

通过使用这一技术（及一个足够特别的插件名字），完全可以避免在全局函数中发生命名空间的冲突。

11.1.2 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.myplugin.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，把函数库编写为`jQuery`扩展还有另一个关键的好处：由于我们知道页面中一定会包含`jQuery`，

因此这些函数可以直接调用jQuery对象。



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。读者应该还记得放弃这个简写方式的\$.noConflict()方法。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\$。

11.1.3 创建实用方法

核心jQuery库提供的许多全局函数都是实用方法。换句话说，这些方法为频繁执行的任务提供了快捷方式。数组处理函数\$.each()、\$.map()和\$.grep()都是这方面的例子。下面我们就通过添加一个新的\$.sum()函数，来示范如何创建这种实用方法。

这个新方法接受一个数组参数，并把数组中的值加起来，然后返回结果。实现这个插件的代码很简单：

```
jQuery.sum = function(array) {
    var total = 0;
    jQuery.each(array, function(index, value) {
        total += value;
    });
    return total;
};
```

注意，这里使用\$.each()方法迭代遍历了数组的值。当然，在此使用简单的for()循环也没有问题，但我们完全可以假定在加载插件之前已经加载了jQuery库，所以可以在此使用更方便的语法。

为了测试这个方法，下面编写一个简单的页面用于显示它的输入和输出：

```
<body>
    <p>Array contents:</p>
    <ul id="array-contents"></ul>
    <p>Array sum:</p>
    <div id="array-sum"></div>
</body>
```

接着，再编写一小段脚本，把数组的值及这些值的和添加到上面创建的占位符标记中。

```
$(document).ready(function() {
    var myArray = [52, 97, 0.5, -22];
    $.each(myArray, function(index, value) {
        $('#array-contents').append('<li>' + value + '</li>');
    });
    $('#array-sum').append($.sum(myArray));
});
```

通过HTML页面呈现的结果（图11-1），可知这个插件方法是正确的。



图 11-1

以上我们介绍了命名空间保护的问题，也知道了可以在假定核心jQuery库有效的情况下开发插件。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。

11.2 添加jQuery对象方法

jQuery中大多数内置的功能都是通过其对象的方法提供的，而且这些方法也是插件之所以诱人关键。当函数需要操作DOM元素时，就是将函数创建为jQuery对象方法的好机会。

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.myMethod = function() {
  alert('Nothing happens.');
}
```



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').myMethod();
```

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。

11.2.1 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 selected ' + this.length + ' elements.');
}
```

为了确定可以怎样利用对象环境，下面我们将编写一个小插件，用以操作匹配元素的类。这个新方法接受两个类名，它能够基于每次调用更换应用于每个元素的类。

```
jQuery.fn.swapClass = function(class1, class2) {
    if (this.hasClass(class1)) {
        this.removeClass(class1).addClass(class2);
    }
    else if (this.hasClass(class2)) {
        this.removeClass(class2).addClass(class1);
    }
};
```

首先，测试每个匹配的元素是否应用了class1，如果是则将该类替换成class2。然后，再测试class2并在必要时替换成class1。如果两个类都不存在，则什么也不做。

为了测试这个方法是否有效，需要以下HTML：

```
<ul>
    <li>Lorem ipsum dolor sit amet</li>
    <li class="this">Consectetur adipisicing elit</li>
    <li>Sed do eiusmod tempor incididunt ut labore</li>
    <li class="that">Magnna aliqua</li>
    <li class="this">Ut enim ad minim veniam</li>
    <li>Quis nostrud exercitation ullamco</li>
    <li>Laboris nisi ut aliquip ex ea commodo</li>
    <li class="that">Duis aute irure dolor</li>
</ul>
<input type="button" value="Swap classes" id="swap" />
```

我们为this类定义的样式是粗体文本，为that类定义的样式是斜体文本，如图11-2所示。

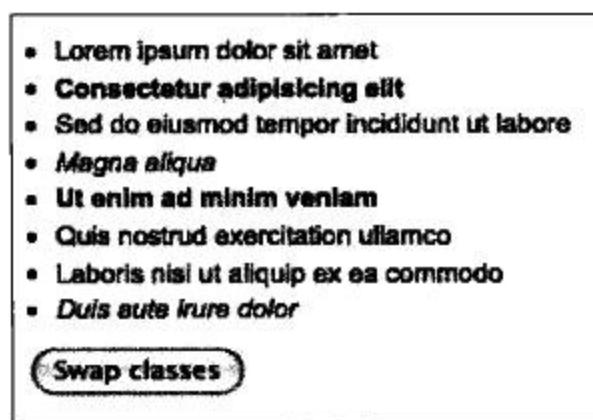


图 11-2

接下来，就可以在按钮被单击时调用新定义的方法了：

```
$(document).ready(function() {
    $('#swap').click(function() {
        $('li').swapClass('this', 'that');
        return false;
    });
});
```

但是，好像是有问题——单击按钮后，每一行都应用了that类（图11-3）。

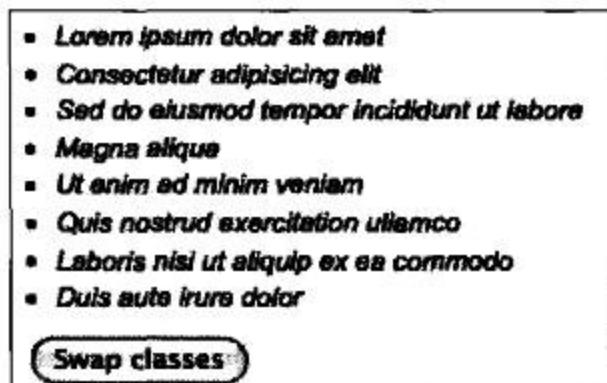


图 11-3

读者大概没有忘记，jQuery的选择符表达式可能会匹配零、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。然而，我们在此调用的`.hasClass()`只会检查最先匹配的元素。换句话说，我们应该独立检查和操作每一个元素。

要在无论匹配多个元素的情况下都保证行为正确，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，因此可以调整代码为每个匹配的元素应用类。

```
jQuery.fn.swapClass = function(class1, class2) {
    this.each(function() {
        var $element = jQuery(this);
        if ($element.hasClass(class1)) {
            $element.removeClass(class1).addClass(class2);
        }
        else if ($element.hasClass(class2)) {
            $element.removeClass(class2).addClass(class1);
        }
    });
};
```



注意！在对象方法体内，关键字`this`引用的是一个jQuery对象，但在每次调用的`.each()`方法中，`this`引用的则是一个DOM元素。

这样，再单击按钮，切换类的操作就不会影响到不带有任何类的元素了，如图11-4所示。

11.2.2 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：

```

jQuery.fn.swapClass = function(class1, class2) {
    return this.each(function() {
        var $element = jQuery(this);
        if ($element.hasClass(class1)) {
            $element.removeClass(class1).addClass(class2);
        }
        else if ($element.hasClass(class2)) {
            $element.removeClass(class2).addClass(class1);
        }
    });
};

```

- [Lorem ipsum dolor sit amet](#)
- [Consectetur adipisicing elit](#)
- [Sed do eiusmod tempor incididunt ut labore](#)
- [Magnna aliqua](#)
- [Ut enim ad minim veniam](#)
- [Quis nostrud exercitation ullamco](#)
- [Laboris nisi ut aliquip ex ea commodo](#)
- [Duis aute irure dolor](#)

[Swap classes](#)

图 11-4

前面，在调用了`.swapClass()`之后，如果想对元素再执行其他操作，必须通过一条新语句重新取得元素。而在添加`return`之后，就可以在我们的插件方法上面连缀内置的方法了（结果如图11-5所示）：

```

$(document).ready(function() {
    $('#swap').click(function() {
        $('li')
            .swapClass('this', 'that')
            .css('text-decoration', 'underline');
        return false;
    });
});

```

- [Lorem ipsum dolor sit amet](#)
- [Consectetur adipisicing elit](#)
- [Sed do eiusmod tempor incididunt ut labore](#)
- [Magnna aliqua](#)
- [Ut enim ad minim veniam](#)
- [Quis nostrud exercitation ullamco](#)
- [Laboris nisi ut aliquip ex ea commodo](#)
- [Duis aute irure dolor](#)

[Swap classes](#)

图 11-5

11.3 DOM 遍历方法

在某些情况下，我们定义的插件方法可能会改变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 = [];
    this.each(function() {
        grandparents.push(this.parentNode.parentNode);
    });
    grandparents = jQuery.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，就是使用标准的.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内置的.setArray方法把这组匹配的元素转换成新数组。现在，仅通过调用一个方法就可以查找并操作某个元素的祖父级元素了。

为了测试这个方法，可以定义一个深度嵌套的

结构：

```
<div>Deserunt mollit anim id est laborum</div>
<div>Ut enim ad minim veniam
    <div>Quis nostrud exercitation
        <div>Ullamco laboris nisi
            <div>Ut aliquip ex ea</div>
            <div class="target">Commodo consequat
                <div>Lorem ipsum dolor sit amet</div>
            </div>
        </div>
    </div>
    <div>Duis aute irure dolor</div>
    <div>In reprehenderit
        <div>In voluptate</div>
        <div>Velit esse
            <div>Cillum dolore</div>
            <div class="target">Fugiat nulla pariatur</div>
        </div>
        <div>Excepteur sint occaecat cupidatat</div>
    </div>
    </div>
    <div>Non proident</div>
</div>
<div>Sunt in culpa qui officia</div>
```

我们用粗体来标识了其中的目标元素（**<div class="target">**），此时的页面如图11-6所示。

接着，使用我们定义的新方法，就可以取得目标元素的祖父级元素了：

```
$(document).ready(function() {
    $('.target').grandparent().addClass('highlight');
});
```

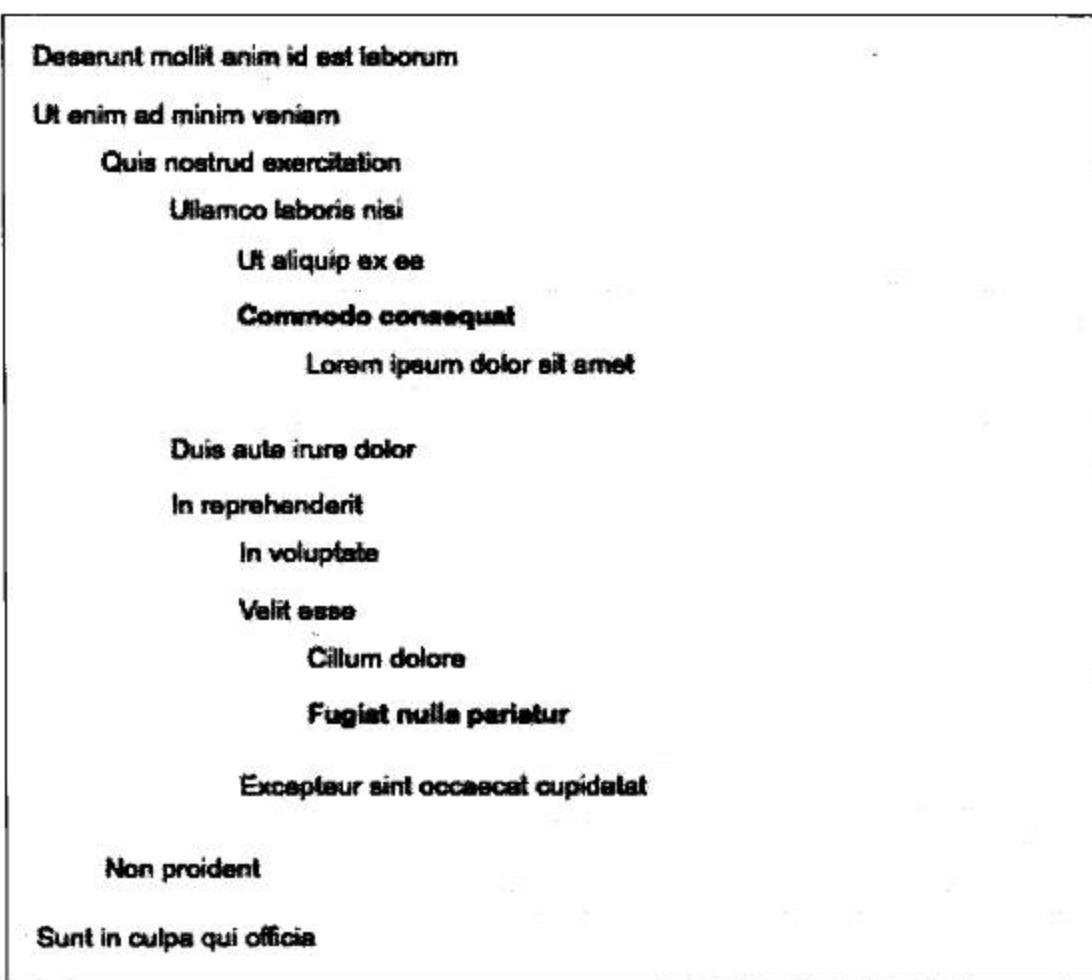


图 11-6

添加highlight类导致了页面中的两组祖父级元素的内容都显示为斜体，如图11-7所示。

然而，这个方法是破坏性的。换句话说，实际的jQuery对象也会因此被修改——如果把jQuery对象保存一个变量中，这个副作用就会变得很明显。

```
$(document).ready(function() {
    var $target = $('.target');
    $target.grandparent().addClass('highlight');
    $target.hide();
});
```

这些代码应该突出显示祖父级元素，然后再隐藏目标元素。但实际的效果反而是隐藏了祖父级元素，如图11-8所示。

保存在\$target中的jQuery对象已经改为引用祖父级元素了。为了避免这种情况，需要让方法变成非破坏性的。而借助jQuery在内部为每个对象维护的栈，可以做到这一点。

```
jQuery.fn.grandparent = function() {
```

```

var grandparents = [];
this.each(function() {
    grandparents.push(this.parentNode.parentNode);
});
grandparents = jQuery.unique(grandparents);
return this.pushStack(grandparents);
};

```

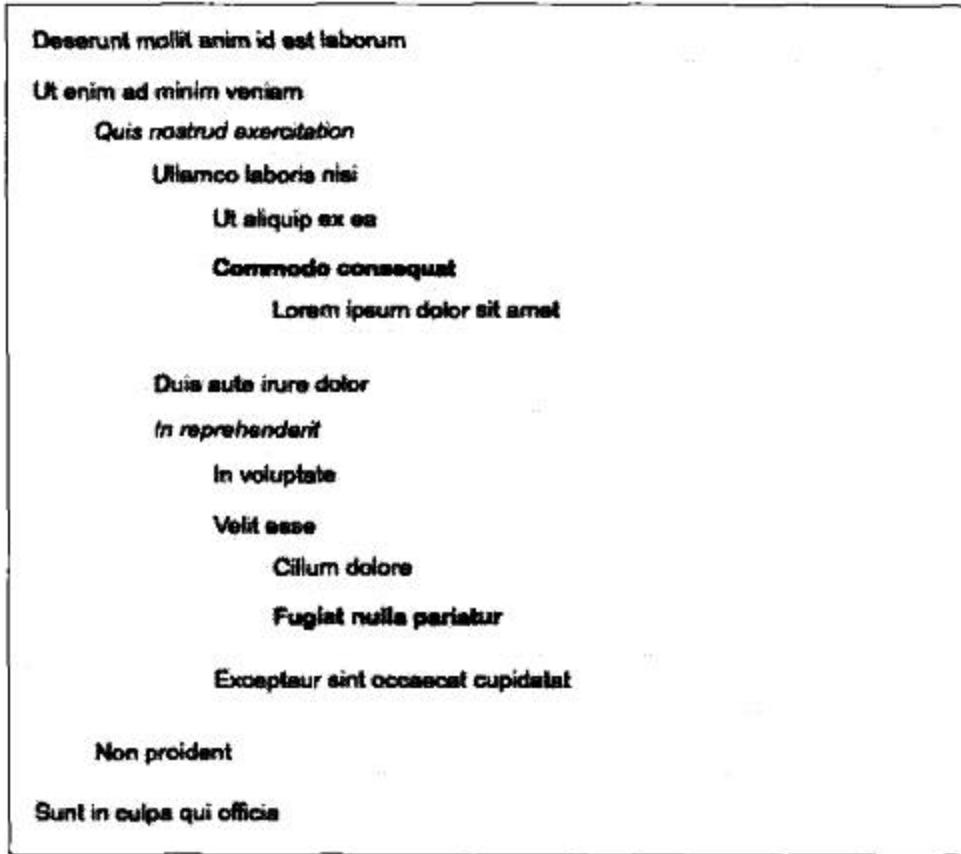


图 11-7

```

Dolor sit amet
Ut enim ad minim veniam
Non proident
Sunt in culpa qui officia

```

图 11-8

调用`.pushStack()`而不是`.setArray()`，会创建一个新jQuery对象，而不是修改原有对象。这样，`$target`就没有被修改，最初的目标对象将被代码隐藏起来，如图11-9所示。

这样还会带来一个额外的好处，即`.pushStack()`也支持`.end()`和`.andSelf()`方法与插件方法共同使用，因此可以把这些方法连缀起来（如图11-10所示）：

```

$(document).ready(function() {
    $('.target').grandparent().andSelf().addClass('highlight');
});

```

```

Deserunt mollit anim id est laborum
Ut enim ad minim veniam
    Quis nostrud exercitation
        Utamco laboris nisi
            Ut aliquip ex ea
                Duis aute irure dolor
                    In reprehenderit
                        In voluptate
                            Velit esse
                                Cillum dolore
                                    Excepteur sint occaecat cupidatat
                                        Non proident
                                            Sunt in culpa qui officia

```

图 11-9

```

Deserunt mollit anim id est laborum
Ut enim ad minim veniam
    Quis nostrud exercitation
        Utamco laboris nisi
            Ut aliquip ex ea
                Commodo consequat
                    Lorem ipsum dolor sit amet
                        Duis aute irure dolor
                            In reprehenderit
                                In voluptate
                                    Velit esse
                                        Cillum dolore
                                            Fugiat nulla pariatur
                                                Excepteur sint occaecat cupidatat
                                                    Non proident
                                                        Sunt in culpa qui officia

```

图 11-10



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。

11.4 添加新的简写方法

jQuery中的很多方法都是另外一些底层方法的简写方法。例如，大多数事件方法都是调

用`.bind()`或`.trigger()`的简写方法，而许多AJAX方法也会在内部调用`$.ajax()`。这些简写方法提供了很大的便利，使开发人员无需考虑各种复杂的选项。

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。例如，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3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'
    });
};

jQuery.fn.slideFadeToggle = function() {
    return this.animate({
        height: 'toggle',
        opacity: 'toggle'
    });
};
```

现在，我们就可以在需要时调用`.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);
};

jQuery.fn.slideFadeToggle = function(speed, callback) {
    return this.animate({
        height: 'toggle',
        opacity: 'toggle'
    }, 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。为了演示这个方法，需要一个简单的HTML页面：

```

<body>
    <p>Lorem 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.</p>
    <div class="controls">
        <input type="button" value="Slide and fade out"
            id="out" />
        <input type="button" value="Slide and fade in" id="in" />
        <input type="button" value="Toggle" id="toggle" />
    </div>
</body>

```

在按钮被单击时，脚本就会调用我们定义的新方法：

```

$(document).ready(function() {
    $('#out').click(function() {
        $('p').slideFadeOut('slow');
        return false;
    });
    $('#in').click(function() {
        $('p').slideFadeIn('slow');
        return false;
    });
    $('#toggle').click(function() {
        $('p').slideFadeToggle('slow');
        return false;
    });
});

```

而动画效果，也跟我们想象的一样（如图11-11所示）。

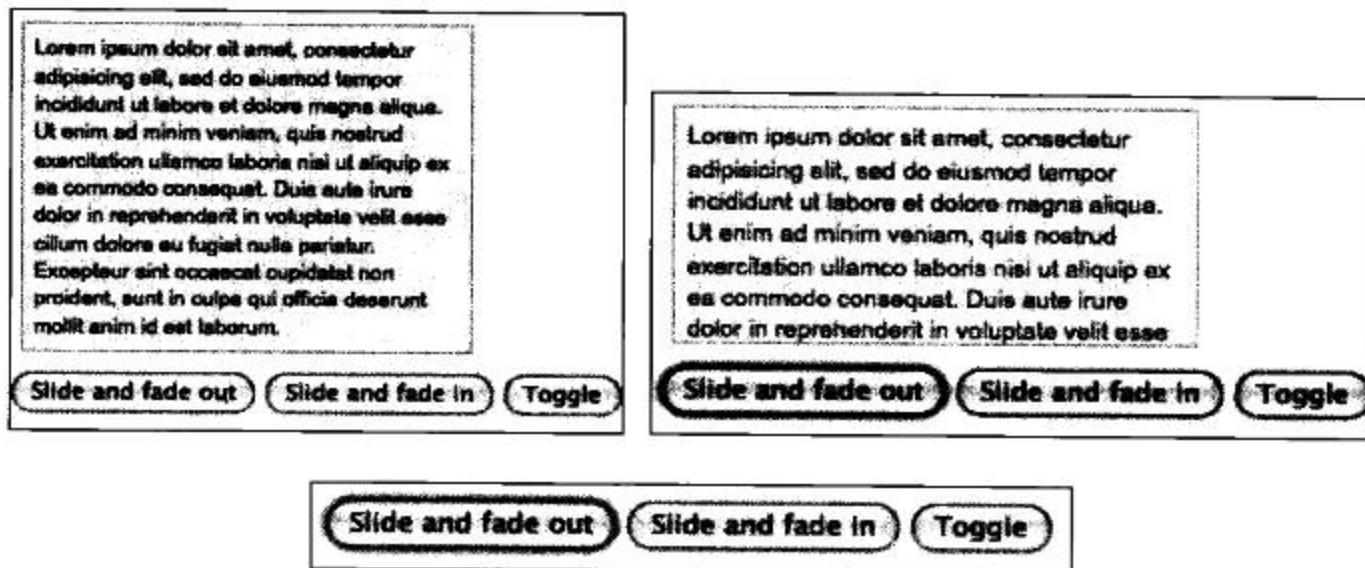


图 11-11

11.5 方法的参数

前面我们看到了一些插件方法的例子。其中一些方法显式地接受参数，另一些则不然。如前所述，关键字this始终是方法的执行环境，但除此之外我们还可以提供影响方法执行的其他信息。虽然到现在为止我们接触的参数很少，但参数列表实际上可以很长。下面就介绍几种管理方法参数的技巧，以便令使用插件的人能够更方便。

下面，我们就以一个为文本块加投影的插件方法为例来说明这一点。例子中使用的技术与第9章实现新闻标题翻转效果时用到的技术类似，即定义一系列部分透明的元素，然后把它们相继排列在页面的不同位置上。

```
jQuery.fn.shadow = function() {
    return this.each(function() {
        var $originalElement = jQuery(this);
        for (var i = 0; i < 5; i++) {
            $originalElement
                .clone()
                .css({
                    position: 'absolute',
                    left: $originalElement.offset().left + i,
                    top: $originalElement.offset().top + i,
                    margin: 0,
                    zIndex: -1,
                    opacity: 0.1
                })
                .appendTo('body');
        }
    });
};
```

对于每个调用此方法的元素，都要复制该元素一定数量的副本，调整每个副本的不透明度。然后，再通过绝对定位方式，以该元素为基准按照不同的偏移量定位这些副本。

同样，我们仍然可以通过简单的HTML（效果如图11-12所示）来测试这个插件方法：

```
<body>
  <h1>The quick brown fox jumps over the lazy dog.</h1>
</body>
```

The quick brown fox jumps over the lazy dog.

图 11-12

由于没有定义参数，因此调用插件方法自然比较简单（效果如图11-13所示）：

```
$(document).ready(function() {
  $('h1').shadow();
});
```

The quick brown fox jumps over the lazy dog.

图 11-13

11.5.1 简单参数

现在，我们来介绍稍微复杂一些的插件方法。本节方法的执行有赖于一些用户可能会修改的数字值。可以把这些值定义为参数，以便用户根据需要来修改。

```
jQuery.fn.shadow = function(slices, opacity, zIndex) {
  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < slices; i++) {
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left + i,
          top: $originalElement.offset().top + i,
          margin: 0,
          zIndex: zIndex,
          opacity: opacity
        })
        .appendTo('body');
    }
  });
};
```

然后，在调用这个方法时，必须提供3个参数值（效果如图11-14所示）。

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

```
$('h1').shadow(10, 0.1, -1);
});
```

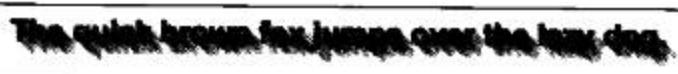


图 11-14

这些参数都起到了应有的作用——投影比较长，“切片”数是前面示例中的两倍——但方法的接口还不够理想。这3个参数非常容易搞混，它们的次序也没有任何规则可循。可见，标记参数的方式亟待改进，结果应该既方便编写方法调用的人，也能够让将来希望看懂同样代码的人对参数的作用一目了然。

11.5.2 参数映射

在介绍jQuery API时，我们曾看到过很多将映射作为方法参数的例子。作为一种向插件用户公开选项的方式，映射要比刚刚使用的参数列表更加友好。映射会为每个参数提供一个有意义的标签，同时也会让参数次序变得无关紧要。而且，只要有可能通过插件来模仿jQuery API，就应该使用映射来提高一致性和易用性。

```
jQuery.fn.shadow = function(opts) {
    return this.each(function() {
        var $originalElement = jQuery(this);
        for (var i = 0; i < opts.slices; i++) {
            $originalElement
                .clone()
                .css({
                    position: 'absolute',
                    left: $originalElement.offset().left + i,
                    top: $originalElement.offset().top + i,
                    margin: 0,
                    zIndex: opts.zIndex,
                    opacity: opts.opacity
                })
                .appendTo('body');
        }
    });
};
```

在这个新的接口中，不一样的地方就是引用每个参数的方式：不再引用个别的变量名，而是通过函数的opts参数的属性来访问每个值。

调用这个方法则需要传递一个值的映射，而不是独立的参数了（效果如图11-15所示）：

```
$(document).ready(function() {
    $('h1').shadow({
        slices: 5,
        opacity: 0.25,
        zIndex: -1
    });
});
```

The quick brown fox jumps over the lazy dog.

图 11-15

这样，只要扫一眼调用方法的代码，就能明确知道每个参数的作用。

11.5.3 默认参数值

随着方法的参数逐渐增多，始终指定每个参数并不是必须的。此时，一组合理的默认值可以增强插件接口的易用性。所幸的是，以映射作为参数可以帮我们很好地达成这一目标，它可以为用户未指定的参数自动传入默认值。

```
jQuery.fn.shadow = function(options) {
    var defaults = {
        slices: 5,
        opacity: 0.1,
        zIndex: -1
    };
    var opts = jQuery.extend(defaults, options);
    return this.each(function() {
        var $originalElement = jQuery(this);
        for (var i = 0; i < opts.slices; i++) {
            $originalElement
                .clone()
                .css({
                    position: 'absolute',
                    left: $originalElement.offset().left + i,
                    top: $originalElement.offset().top + i,
                    margin: 0,
                    zIndex: opts.zIndex,
                    opacity: opts.opacity
                })
                .appendTo('body');
        }
    });
};
```

在这个方法的定义中，我们定义了一个新映射，名为`defaults`。实用函数`$.extend()`可以用接受的选项映射参数覆盖`defaults`中的项，并保持选项映射中未指定的默认项不变。

接下来，我们仍然以映射调用同一个方法，但这次只指定一个有别于默认值的不同参数（效果如图11-16所示）：

```
$(document).ready(function() {
    $('h1').shadow({
        opacity: 0.05
    });
});
```

The quick brown fox jumps over the lazy dog.

图 11-16

未指定的参数使用预先定义的默认值。`$.extend()`方法甚至可以接受`null`值，在用户可以接受所有默认参数时，我们的方法可以直接执行而不会出错：

```
$(document).ready(function() {
    $('h1').shadow();
});
```

11.5.4 回调函数

当然，方法参数也可能不是一个简单的数字值，可能会更复杂。在各种jQuery API中经常可以看到另一种参数类型，即回调函数。回调函数可以极大地增加插件的灵活性，但却用不着在创建插件时多编写多少代码。

要在方法中使用回调函数，需要接受一个函数对象作为参数，然后在方法中适当的位置上调用该函数。例如，可以扩展前面定义的文本投影方法，让用户能够自定义投影相对于文本的位置。

```
jQuery.fn.shadow = function(options) {
    var defaults = {
        slices: 5,
        opacity: 0.1,
        zIndex: -1,
        sliceOffset: function(i) {
            return {x: i, y: i};
        }
    };
    var opts = jQuery.extend(defaults, options);
    return this.each(function() {
        var $originalElement = jQuery(this);
        for (var i = 0; i < opts.slices; i++) {
            var offset = opts.sliceOffset(i);
            $originalElement
                .clone()
                .css({
                    position: 'absolute',
                    left: $originalElement.offset().left
                        + offset.x,
                    top: $originalElement.offset().top
                        + offset.y,
                    margin: 0,
                    zIndex: opts.zIndex,
                    opacity: opts.opacity
                })
                .appendTo('body');
        }
    });
};
```

投影的每个“切片”相对于原始文本都有不同的偏移量。此前，这个偏移量简单地等于切片

的索引值。现在，偏移量都根据sliceOffset()函数来计算，而这个函数是用户可以覆盖的参数。例如，用户可以在两个方向上指定负值偏移量：

```
$ (document).ready(function() {
  $('h1').shadow({
    sliceOffset: function(i) {
      return {x: -i, y: -2*i};
    }
  });
});
```

这样会导致投影的叠加起来向左上方（不是向右下方）延伸（如图11-17所示）。

图 11-17

回调函数可以像这样简单地修改投影方向，也可以根据插件用户的定义，对投影位置作出更复杂的调整。如果未指定回调函数，则会使用默认行为。

11.5.5 可定制的默认值

我们在前面已经看到了，通过为方法参数设定合理的默认值，能够显著改善用户使用插件的体验。但是，到底什么默认值合理有时候也很难说。如果有脚本会多次调用我们的插件，每次调用都要传递一组不同于默认值的参数，那么通过定制默认值就可以减少很多需要编写的代码量。

要支持默认值的可定制，需要把它们从方法定义中移出，然后放到外部代码可以访问的地方：

```
jQuery.fn.shadow = function(options) {
  var opts = jQuery.extend({}, jQuery.fn.shadow.defaults, options);
  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < opts.slices; i++) {
      var offset = opts.sliceOffset(i);
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left + offset.x,
          top: $originalElement.offset().top + offset.y,
          margin: 0,
          zIndex: opts.zIndex,
          opacity: opts.opacity
        })
        .appendTo('body');
    }
  });
}
```

```

    });
};

jQuery.fn.shadow.defaults = {
  slices: 5,
  opacity: 0.1,
  zIndex: -1,
  sliceOffset: function(i) {
    return {x: i, y: i};
  }
};

```

默认值被放在了投影插件的命名空间里，可以通过`$.fn.shadow.defaults`直接引用。而对`$.extend()`的调用也必须修改，以适应这种变化。由于现在所有对`.shadow()`的调用都要重用`defaults`映射，因此不能让`$.extend()`修改它。我们就在此将一个空映射`({})`作为`$.extend()`的第一个参数，让这个新对象成为被修改的目标。

于是，使用我们插件的代码就可以修改默认值了，修改之后的值可以被所有后续对`.shadow()`的调用共享。而且，在调用方法时仍然可以传递选项。

```

$(document).ready(function() {
  $.fn.shadow.defaults.slices = 10;

  $('h1').shadow({
    sliceOffset: function(i) {
      return {x: -i, y: i};
    }
  });
});

```

因为在此提供了新的默认值，以上脚本会创建带10个切片的投影。而由于在调用方法时提供了`sliceOffset`回调函数，所以投影也将朝向左下方（如图11-18所示）。



图 11-18

11.6 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。

最简单的选择符表达式是伪类，即以冒号开头的表达式，如`:checked`或`:nth-child()`。为演示创建选择符表达式的过程，下面我们来构建一个名为`:css()`伪类。这个选择符允许我们基于CSS属性的数字值查找元素。

在使用选择符表达式查找元素时，jQuery会在内部的一个名叫`expr`的映射中查找匹配指令`(instruction)`。该映射中包含基于元素执行的JavaScript代码，如果对代码求值的结果为`true`，则

会将相应元素包含在结果集中。可以使用`$.extend()`函数把新表达式添加到这个映射中。

```
jQuery.extend(jQuery.expr[':'], {
  'css': function(element, index, matches, set) {
    var parts = /(([\w-]+)\s*([<>=]+\s*)\s*(\d+)/
      .exec(matches[3]);
    var value = parseFloat(jQuery(element).css(parts[1]));
    switch (parts[2]) {
      case '<':
        return value < parseInt(parts[3]);
      case '<=':
        return value <= parseInt(parts[3]);
      case '=':
      case '==':
        return value == parseInt(parts[3]);
      case '>=':
        return value >= parseInt(parts[3]);
      case '>':
        return value > parseInt(parts[3]);
    }
  }
});
```

以上代码告诉jQuery，`css`是一个可以在选择符表达式中前置冒号的有效字符串，当遇到这个字符串时，应该调用给定的函数以确定当前元素是否应该包含在结果集中。

需要在此求值的函数会接受以下4个参数。

- `element`: 当前的DOM元素。大多数选择符都需要这个参数。
- `index`: DOM元素在结果集中的索引。这个参数对`:eq()`和`:lt()`等选择符比较有用。
- `matches`: 包含解析当前选择符的正则表达式结果的数组。通常，`matches[3]`是这个数组中唯一有用的项；对于`:a(b)`形式的选择符而言，`matches[3]`项中包含着`b`，即圆括号中的文本。
- `set`: 到目前为止匹配的整个DOM元素集合。这个参数极少用到。

伪类选择符需要使用包含在这4个参数中的信息，以便决定相关元素是否应该包含在结果集中。在此，`element`和`matches`对我们而言都是必需的。

在选择符函数中，我们首先通过正则表达式把选择符分割成有用的部分。对于`:css(width < 200)`这个选择符，我们希望返回宽度小于200的所有元素。因此，需要从圆括号的文本中提取出属性名（`name`）、比较运算符（`<`）和要比较的值（`200`）。正则表达式`/(([\w-]+)\s*([<>=]+\s*)\s*(\d+)/`用于执行这一搜索，把字符串中的上述3个部分放在`parts`数组中以备后用。

然后，需要取得属性的当前值。在此使用了jQuery的`.css()`方法，返回选择符中指定属性的值。由于返回的属性值是字符串，因此要使用`parseFloat()`将其转换为数字值。

最后，执行比较。`switch`语句会根据选择符的内容来确定比较类型，返回比较的结果（`true`或`false`）。

这样我们就有了一个可以在jQuery代码中使用的新选择符表达式。可以通过下面这个简单的HTML文档来演示它的用途（如图11-19所示）：

```
<body>
  <div>Deserunt mollit anim id est laborum</div>
  <div>Ullamco</div>
  <div>Ut enim ad minim veniam laboris</div>
  <div>Quis nostrud exercitation consequat nisi</div>
  <div>Ut aliquip</div>
  <div>Commodo</div>
  <div>Lorem ipsum dolor sit amet ex ea</div>
</body>
```



图 11-19

使用新选择符表达式，突出显示列表中文本较短的项就成了小菜一碟（效果如图11-20所示）：

```
$(document).ready(function() {
  $('div:css(width < 100)').addClass('highlight');
});
```



图 11-20

11.7 共享插件

在开发完成一个插件之后，接下来应该考虑发布它，以便为其他人提供便利，当然，也可能得到别人的改进。可以在官方的jQuery插件库（<http://plugins.jquery.com/>）发布插件：先登录（如有必要先注册），然后为插件添加描述，再上传代码的.zip文档。但是，在共享之前，还应该确保自己的插件已经没有什么错误，而且经过了适当地处理，最后才能发布。

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。

11.7.1 命名约定

所有插件文件都应该命名为`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。

11.7.2 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。

11.7.3 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。

如果方法接受一些配置选项，最好使用映射作为参数。这样，每个选项都会有一个标签，而且用户也无需按顺序传递参数。应该为映射定义默认值，且默认值可以在必要时被覆盖。

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。

11.7.4 文档格式

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/>。

11.8 小结

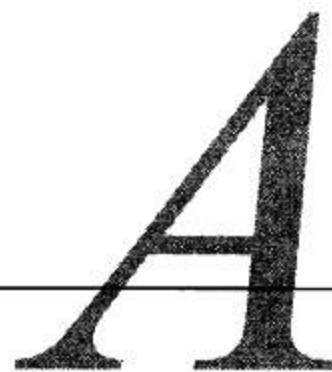
在这最后的一章中，我们看到jQuery核心提供的功能并没有限制这个库的能力。众多可用的插件充分地扩展了jQuery的功能，而且，我们也可以方便地创建自己的插件，进一步拓展这个库的能力。

本章，我们学习了各种插件的创建，既有利用jQuery库的全局函数，也有操作DOM元素的jQuery对象方法，还有可以轻松自定义的扩展方法，以及用于查找DOM元素的新选择符表达式。

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



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

以下资源提供了有关jQuery库本身的参考和详细信息。

A.1.1 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/>

A.1.2 jQuery API

除了jquery.com上的官方文档之外，还可以在如下地址找到jQuery API：

<http://remysharp.com/jquery-api/>

A.1.3 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-1.2/>

A.1.4 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/>

A.1.5 Adobe AIR jQueryAPI 查看器

Remy Sharp已经把其整理的API打包成了Adobe AIR应用程序，可供离线浏览：

<http://remysharp.com/downloads/jquery-api-browser.air.zip>

A.2 JavaScript 参考

以下站点中包含JavaScript语言的参考和教程、辅助性工具及其他有用链接。

A.2.1 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/>

A.2.2 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/>

A.2.3 MSDN JScript 参考

微软开发者网络中的JScript参考囊括了全部函数、对象及其他特性的介绍，对于理解IE对ECMAScript的实现，具有无可替代的参考价值：

<http://msdn.microsoft.com/zh-cn/library/x85xxsf4.aspx>

A.2.4 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/>

A.2.5 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

在对站点进行最后优化时，通常需要考虑压缩JavaScript代码，从而减少站点用户下载这些文件的时间。

A.3.1 YUI Compressor

jQuery压缩代码时，使用的就是Yahoo! YUI库提供的JavaScript压缩器。这是一个基于Java的命令行工具，可以免费下载。压缩之后能够显著减少代码大小，从而提高下载速度；而且，在必要时还可以应用Gzip压缩选项，进一步减少代码大小。

<http://developer.yahoo.com/yui/compressor/>

A.3.2 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>

A.3.3 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

格式及语义正确的HTML和XHTML文档，能够最大程度地发挥jQuery库的作用。下面列出的资源中包含对两种标记语言的编写规范。

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

我们在本书中展示的特效和动画全都有赖于层叠样式表的威力。如果想在自己的站点中加入漂亮的视觉变换效果，有时候就要求助于以下CSS资源。

A.5.1 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/>

A.5.2 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/>

A.5.3 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 有用的博客

现有的任何技术，都有可能发展出或引入新技术及新特性。要让自己与时俱进，最好的方法就是经常光顾下列站点，浏览其中有关Web开发的新闻。

A.6.1 jQuery 官方博客

John Resig及其他贡献者会在这个博客中发布有关jQuery新版本及其他项目团队的活动通知，偶尔也会发表一些教程或评述文章。

<http://jquery.com/blog/>

A.6.2 Learning jQuery

Karl Swedberg维护的一个包括jQuery教程、例子及公告的博客。受邀作者包含jQuery团队成员Mike Alsup和Brandon Aaron：

<http://www.learningjquery.com/>

A.6.3 Ajaxian

由Dion Almaer和Ben Galbraith创建的一个更新频繁的博客站点，其中包含大量新闻和专题，间或会发布一些JavaScript教程：

<http://ajaxian.com/>

A.6.4 John Resig

jQuery创建人John Resig的个人博客，经常讨论一些有关JavaScript的高级主题：

<http://ejohn.org/>

A.6.5 JavaScript ant

其中包括大量有关JavaScript及其在现代Web浏览器中用法的文章，还提供了经过分类的JavaScript在线资源：

<http://javascriptant.com/>

A.6.6 Robert's talk

Robert Nyman撰写的面向因特网，特别是客户端脚本编程的文章：

<http://www.robertnyman.com/>

A.6.7 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/>

A.6.8 Snook

Jonathan Snook的全面的编程/Web开发博客：

<http://snook.ca/>

A.6.9 Matt Snider 的 JavaScript 资源

Matt Snider专注于JavaScript及相关框架的研究：

<http://mattsnider.com/>

A.6.10 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/>

A.6.11 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/>

A.6.12 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.6.13 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：

^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

<http://www.alistapart.com/>

A.7 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。以下是采用jQuery的部分框架的列表。

- Digitalus Site Manager: <http://code.google.com/p/digitalus-site-manager/>
- Drupal: <http://drupal.org/>
- DutchPIPE: <http://dutchpipe.org/>
- Hpricot: <http://code.whyluckyisstiff.net/hpricot/>
- JobberBase: <http://www.jobberbase.com/>
- Laconica: <http://laconi.ca/>
- Piwik: <http://piwik.org/>
- Pommo: <http://pommo.org/>
- symfony: <http://www.symfony-project.org/>
- SPIP: <http://www.spip.net/>
- Textpattern: <http://www.textpattern.com/>
- Trac: <http://trac.edgewall.org/>
- WordPress: <http://wordpress.org/>
- Z-Blog: <http://www.rainbowsoft.org/zblog>

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

开发工具



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。

B.1.1 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。

B.1.2 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://chrispederick.com/work/web-developer/>

B.1.3 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/>

B.1.4 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。

B.2.1 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>

B.2.2 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/>

B.2.3 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/>

B.2.4 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。

B.3.1 “开发”菜单

自Safari 3.1开始，就可以在“偏好设置”菜单的“高级”选项卡中设置显示“开发”菜单。这个菜单中包含了Web检查器和JavaScript控制台工具。

B.3.2 Web Inspector

Safari 3提供了检查个别页面元素和收集应用到每个元素的CSS规则信息的能力。

<http://trac.webkit.org/projects/webkit/wiki/Web%20Inspector>

WebKit的当前构建中包含能从根本上增强Web检查器的工具，包含了许多优秀的Firefox特性，如集成的JavaScript调试器Drosera。

<http://trac.webkit.org/ wiki/Drosera>

B.4 针对 Opera 的工具

作为一个桌面浏览器，Opera的市场份额虽然有限，但它在嵌入系统和移动设备领域则是一个有力的竞争者。而且，现代Web开发也必须考虑到它的潜力。

Dragonfly

虽然还处于早期开发阶段，但Dragonfly的目标则是成为计算机和移动设备中面向Opera浏览器的调试环境。Dragonfly的功能与Firebug类似，包括JavaScript调试和CSS及DOM检查、编辑等。

B.5 其他工具

与前面推荐的针对浏览器的工具不同，以下工具的适用范围更广一些。

B.5.1 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>

B.5.2 NitobiBug

与Firebug Lite类似，NitobiBug也是一个跨浏览器的工具，但它包含的功能与更可靠、更完善的Firebug接近。NitobiBug的长处在于DOM和对象检查，同时它还具有功能强大的控制台。而且，通过包含Nitobi JavaScript文件并调用`nitobi.Debug.log()`，可以调用控制台和检查器。

<http://www.nitobibug.com/>

B.5.3 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://github.com/kswedberg/jquery-tmbundle/>

B.5.4 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/>

B.5.5 Fiddler

Fiddler也是一个HTTP调试代理，其功能与Charles相似。根据官方站点描述，Fiddler“包含一个强大的基于事件的脚本子系统，可以使用.NET语言扩展”：

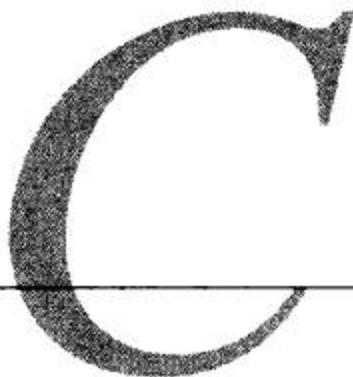
<http://www.fiddlertool.com/fiddler/>

B.5.6 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，并拥有基于Firebug的JavaScript调试器。

<http://www.aptana.com/>

JavaScript闭包



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）的基于函数的构造，这就是闭包。

在我们讨论中，将会展示很多小段代码示例，这些代码示例也会输出一些的消息。在输出消息时，我们不依赖于浏览器的日志机制（如Firefox的`console.log()`），也不会生成一系列`alert()`对话框，而是会使用下面定义的jQuery插件方法：

```
jQuery.fn.print = function(message) {
    return this.each(function() {
        $('

在定义了这个方法后，调用$('#example').print('hello')，就可以在<div id="example">中添加消息“hello”。



## 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 outerFn() {
 function innerFn() {
 }
}
```



innerFn()就是一个被包含在outerFn()作用域中的内部函数。这意味着，在outerFn()内


```

部调用innerFn()是有效的，而在outerFn()外部调用innerFn()则是无效的。下列代码会导致一个JavaScript错误：

```
function outerFn() {
    $('#example-2').print('Outer function');
    function innerFn() {
        $('#example-1').print('Inner Function');
    }
}
$('#example-1').print('innerFn():');
innerFn();
```

不过，通过在outerFn()内部调用innerFn()，则可以成功地运行：

```
function outerFn() {
    $('#example-2').print('Outer function');
    function innerFn() {
        $('#example-2').print('Inner function');
    }
    innerFn();
}
$('#example-2').print('outerFn():');
outerFn();
```

结果会产生如下输出：

```
outerFn():
Outer function
Inner function
```

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.1.1 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，只允许通过内部函数实现代码隐藏，而且这些函数因此也会永远被埋没在它们的父函数中。然而，JavaScript则允许开发人员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 globalVar;

function outerFn() {
    $('#example-3').print('Outer function');
    function innerFn() {
        $('#example-3').print('Inner function');
    }
    globalVar = innerFn;
}
```

```
$('#example-3').print('outerFn():');
outerFn();
$('#example-3').print('globalVar():');
globalVar();
```

在函数定义之后调用outerFn()会修改全局变量globalVar，此时它引用的是innerFn()。这意味着，后面调用globalVar()的操作就如同调用innerFn()一样，也会执行输出消息的语句：

```
outerFn():
Outer function
globalVar():
Inner function
```

注意，此时在outerFn()外部直接调用innerFn()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outerFn()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 outerFn() {
  $('#example-4').print('Outer function');
  function innerFn() {
    $('#example-4').print('Inner function');
  }
  return innerFn;
}
$('#example-4').print('var fnRef = outerFn():');
var fnRef = outerFn();
$('#example-4').print(fnRef());
fnRef();
```

这里，并没有在outerFn()内部修改全局变量，而是从outerFn()中返回了一个对innerFn()的引用。通过调用outerFn()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，从而触发消息输出：

```
var fnRef = outerFn();
Outer function
fnRef():
Inner function
```

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.1.2 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 outerFn() {
  function innerFn() {
    var innerVar = 0;
```

```

    innerVar++;
    $('#example-5').print('innerVar = ' + innerVar);
}
return innerFn;
}
var fnRef = outerFn();
fnRef();
fnRef();
var fnRef2 = outerFn();
fnRef2();
fnRef2();

```

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：

```

innerVar = 1
innerVar = 1
innerVar = 1
innerVar = 1

```

内部函数可以像其他函数一样引用全局变量：

```

var globalVar = 0;
function outerFn() {
    function innerFn() {
        globalVar++;
        $('#example-6').print('globalVar = ' + globalVar);
    }
    return innerFn;
}
var fnRef = outerFn();
fnRef();
fnRef();
var fnRef2 = outerFn();
fnRef2();
fnRef2();

```

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：

```

globalVar = 1
globalVar = 2
globalVar = 3
globalVar = 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 outerFn() {
    var outerVar = 0;
    function innerFn() {
        outerVar++;
        $('#example-7').print('outerVar = ' + outerVar);
    }
    return innerFn;
}
var fnRef = outerFn();

```

```

fnRef();
fnRef();
var fnRef2 = outerFn();
fnRef2();
fnRef2();

```

这一次，对内部函数的调用会产生有意思的行为：

```

outerVar = 1
outerVar = 2
outerVar = 1
outerVar = 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innerFn()都会独立地递增outerVar。也就是说，第二次调用outerFn()没有继续沿用outerVar的值，而是在第二次函数调用的作用域中创建并绑定了一个新的outerVar的实例。结果，就造成了在上面的调用之后继续调用fnRef()会导致输出3，而再次调用fnRef2()也会导致输出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，我们称既不是内部函数局部变量，也不是其参数的变量为自由变量，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.2 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 outerFn() {
  var outerVar = 0;
  function innerFn1() {
    outerVar++;
    $('#example-8').print('(1) outerVar = ' + outerVar);
  }
  function innerFn2() {
    outerVar += 2;
    $('#example-8').print('(2) outerVar = ' + outerVar);
  }
  return {'fn1': innerFn1, 'fn2': innerFn2};
}
var fnRef = outerFn();
fnRef.fn1();
fnRef.fn2();
fnRef.fn1();
var fnRef2 = outerFn();
fnRef2.fn1();
fnRef2.fn2();
fnRef2.fn1();

```

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：

```
(1) outerVar = 1
(2) outerVar = 3
(1) outerVar = 4
(1) outerVar = 1
(2) outerVar = 3
(1) outerVar = 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innerFn1()为outerVar递增1时，就为调用innerFn2()设置了outerVar的新的起点值，反之亦然。同样，我们也看到对outerFn()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.3 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收一个函数作为参数。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.3.1 \$(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 innerFn() {
        readyVar++;
        $('#example-9').print('readyVar = ' + readyVar);
    }
    innerFn();
    innerFn();
});
```

这看上去同前面的很多例子都差不多，只不过外部函数是传入到\$(document).ready()中的一个回调函数。由于innerFn()定义在这个回调函数中，而且引用了位于回调函数作用域中的

`readyVar`, 因此 `innerFn()` 及其环境就创建了一个闭包。我们两次调用这个内部函数, 通过观察两次输出之间保持的 `readyVar` 的值, 就可以证明这一点:

```
readyVar = 1
readyVar = 2
```

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.3.2 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, 于是它们也会创建闭包。以一个简单的单击处理程序为例:

```
$(document).ready(function() {
    var counter = 0;
    $('#example-10 a.add').click(function() {
        counter++;
        $('#example-10').print('counter = ' + counter);
        return false;
    });
});
```

由于变量 `counter` 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()` 应用程序会递增并显示该变量的值。由于创建了闭包, 每次单击按钮都会引用 `counter` 的同一个实例。也就是说, 消息会持续显示一组递增的值, 而不是每次都显示1。

```
counter = 1
counter = 2
counter = 3
```

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 counter = 0;
    $('#example-11 a.add').click(function() {
        counter++;
        $('#example-11').print('counter = ' + counter);
        return false;
    });
    $('#example-11 a.subtract').click(function() {
        counter--;
        $('#example-11').print('counter = ' + counter);
        return false;
    });
});
```

因为这两个函数引用的是同一个变量counter，所以两个链接的递增和递减操作会影响同一个值，而不是各自独立的值。

```
counter = 1
counter = 2
counter = 1
counter = 0
```

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创建。换句话说，无论命名函数还是匿名函数，都可以用来创建闭包。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() {
  $('#example-12 a').each(function(index) {
    $(this).click(function() {
      $('#example-12').print('index = ' + index);
      return false;
    });
  });
});
```

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链接一样多的函数。这些函数分别作为一个单击处理程序被添加给了相应的链接。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() {
  $('#example-13 a').each(function(index) {
    function clickHandler() {
      $('#example-13').print('index = ' + index);
      return false;
    }
    $(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() {
    $('#example-14').print('index = ' + index);
    return false;
  }
  $('#example-14 a').each(function(index) {
    $(this).click(clickHandler);
  });
});
```

这个版本会导致无论单击哪个链接都会触发一个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.4 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。以图 C-1 中的指针链接为例：

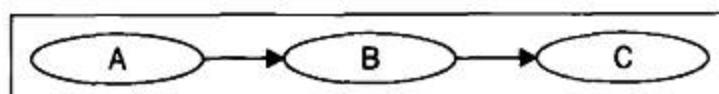


图 C-1

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（如图 C-2 所示）时，处理起来也会更加困难。

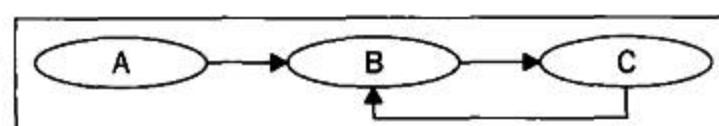


图 C-2

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.4.1 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 outerFn() {
    var outerVar = {};
    function innerFn() {
        alert(outerVar);
    }
    outerVar.fn = innerFn;
}

```

```

    return innerFn;
};


```

这里创建了一个名为outerVar的对象，该对象在内部函数innerFn()中被引用。然后，为outerVar创建了一个指向innerFn()的属性，之后返回了innerFn()。这样就在innerFn()上创建了一个引用outerVar的闭包，而outerVar又引用了innerFn()。但是，也可能会出现比这种情况更隐蔽的引用循环：

```

function outerFn() {
    var outerVar = {};
    function innerFn() {
        alert('hello');
    }
    outerVar.fn = innerFn;
    return innerFn;
};

```

这里我们修改了innerFn()，使它不再引用outerVar。但是，这样做仍然没有断开循环。即使innerFn()不再引用outerVar，outerVar也仍然位于innerFn()的封闭环境中。由于闭包的原因，位于outerFn()中的所有变量都隐含地被innerFn()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.4.2 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——onclick属性自身——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.4.3 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。

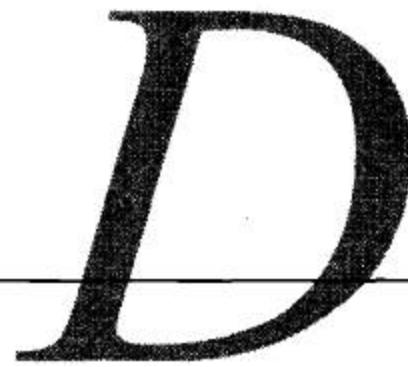
但是，这并不意味着我们完全脱离了险境。当对DOM元素进行其他操作时，仍然要处处留心。只要是将JavaScript对象指定给DOM元素，就可能在IE中导致内存泄漏。jQuery只是有助于减少发生这种情况的可能性。

有鉴于此，jQuery为我们提供了另一个避免这种泄漏的工具。在第7章中我们曾看到过，使用`.data()`方法可以像使用`expando`属性一样，将信息附加到DOM元素。由于这里的数据并非直接保存在`expando`中（jQuery使用一个内部映射并通过它创建的ID来保存这里所说的数据），因此永远也不会构成引用循环，从而有效回避了内存泄漏问题。无论什么时候，当我们觉得`expando`好像是一种方便的数据存储机制时，都应该首选`.data()`这种更安全可靠的替代方案。

C.5 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。

快 速 参 考



本 本附录提供了jQuery API的快速参考，包括选择符表达式和方法。有关jQuery API参考的更详细内容，请读者参考本书的姊妹篇*jQuery Reference Guide*和jQuery文档站点(<http://docs.jquery.com>)。

D.1 选择符表达式

jQuery的工厂函数`$()`用于在页面中查找要操作的元素。这个函数接收一个按照类似CSS语法构成的字符串作为参数，这个字符串参数就叫选择符表达式。本书第2章详细讨论了选择符表达式(见表D-1)。

表 D-1

选 择 符	匹 配
<code>*</code>	所有元素
<code>#id</code>	带有给定ID的元素
<code>element</code>	给定类型的所有元素
<code>.class</code>	带有给定类的所有元素
<code>a, b</code>	与a或b匹配的元素
<code>a b</code>	作为a后代的b匹配的元素
<code>a > b</code>	作为a子元素的b匹配的元素
<code>a + b</code>	作为a直接子元素的b匹配的元素
<code>a ~ b</code>	作为a同辈的b匹配的元素
<code>:first</code>	结果集中的第一个元素
<code>:last</code>	结果集中的最后一个元素
<code>:not(a)</code>	结果集中与a不匹配的所有元素
<code>:even</code>	结果集中的偶数元素(从0开始计数)
<code>:odd</code>	结果集中的奇数元素(从0开始计数)
<code>:eq(index)</code>	结果集中索引为index的元素(从0开始计数)
<code>:gt(index)</code>	结果集中所有位于给定索引之后(大于该索引)的元素(从0开始计数)
<code>:lt(index)</code>	结果集中所有位于给定索引之前(小于该索引)的元素(从0开始计数)

(续)

选 择 符	匹 配
:header	标题元素（如<h1>、<h2>）
:animated	其动画正在播放的元素
:contains(text)	包含给定文本text的元素
:empty	不包含子节点的元素
:has(a)	后代元素中至少有一个与a匹配的元素
:parent	包含子节点的元素
:hidden	隐藏的元素，包括通过CSS隐藏的元素及<input type="hidden" />
:visible	与:hidden匹配的元素相反
[attr]	带有属性attr的元素
[attr=value]	attr属性的值为value的元素
[attr!=value]	attr属性的值不为value的元素
[attr^=value]	attr属性的值以value开头的元素
[attr\$=value]	attr属性的值以value结尾的元素
[attr*=value]	attr属性的值包含子字符串value的元素
:nth-child(index)	作为其父元素第index个子元素的元素（从1开始计数）
:nth-child(even)	作为其父元素第偶数个子元素的元素（从1开始计数）
:nth-child(odd)	作为其父元素第奇数个子元素的元素（从1开始计数）
:nth-child(formula)	作为其父元素第n个子元素的元素（从1开始计数）。formula（公式）的格式为an+b，a、b为整数
:first-child	作为其父元素第一个子元素的元素
:last-child	作为其父元素最后一个子元素的元素
:only-child	作为其父元素唯一一个子元素的元素
:input	所有<input>、<select>、<textarea>和<button>元素
:text	type="text"的<input>元素
:password	type="password"的<input>元素
:radio	type="radio"的<input>元素
:checkbox	type="checkbox"的<input>元素
:submit	type="submit"的<input>元素
:image	type="image"的<input>元素
:reset	type="reset"的<input>元素
:button	type="button"的<input>元素及<button>元素
:file	type="file"的<input>元素
:enabled	启用的表单元素
:disabled	禁用的表单元素
:checked	选中的复选框和单选按钮元素
:selected	选中的<option>元素

D.2 DOM 遍历方法

在使用`$()`创建了jQuery对象之后，通过调用下列DOM遍历方法，可以修改其中匹配的元素，以便将来操作。本书第2章讨论了DOM遍历方法（见表D-2）。

表 D-2

遍历方法	返回的jQuery对象包含
<code>.filter(selector)</code>	与给定的选择符 <code>selector</code> 匹配的选中元素
<code>.filter(callback)</code>	回调函数 <code>callback</code> 返回 <code>true</code> 的选中元素
<code>.eq(index)</code>	从0开始计数的第 <code>index</code> 个选中元素
<code>.slice(start, [end])</code>	从0开始计数的给定范围内的选中元素
<code>.not(selector)</code>	与给定的选择符 <code>selector</code> 不匹配的选中元素
<code>.add(selector)</code>	选中元素再加上与给定选择符 <code>selector</code> 匹配的元素
<code>.find(selector)</code>	与给定选择符 <code>selector</code> 匹配的后代元素
<code>.contents()</code>	子节点（包括文本节点）
<code>.children([selector])</code>	子节点，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.next([selector])</code>	每个选中元素紧邻的下一个同辈元素，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.nextAll([selector])</code>	每个选中元素之后的所有同辈元素，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.prev([selector])</code>	每个选中元素紧邻的上一个同辈元素，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.prevAll([selector])</code>	每个选中元素之前的所有同辈元素，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.siblings([selector])</code>	所有同辈元素，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.parent([selector])</code>	每个选中元素的父元素，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.parents([selector])</code>	所有祖先元素，可以传入可选的选择符 <code>selector</code> 进一步筛选
<code>.closest selector</code>	与选择符 <code>selector</code> 匹配的第一个元素，遍历路径从选中元素开始，沿DOM树向上在其祖先节点中的查找
<code>.offsetParent()</code>	第一个选中元素被定位的父元素（如，通过 <code>relative</code> 或 <code>absolute</code> 定位）
<code>.andSelf()</code>	选中元素再加上内部jQuery栈中之前选中的元素
<code>.end()</code>	内部jQuery栈中之前选中的元素
<code>.map(callback)</code>	对每个选中元素调用回调函数 <code>callback</code> 之后的结果

D.3 事件方法

为了对用户的行为作出反应，需要使用下面给出的事件方法来注册处理程序。注意，许多DOM元素仅适用于特定的元素类型，本附录没有给出相关的细节。本书第3章详细讨论了事件方法（见表D-3）。

表 D-3

事件方法	说 明
<code>.ready(handler)</code>	绑定在DOM和CSS完全加载后调用的处理程序 <code>handler</code>

(续)

事件方法	说 明
.bind(type, [data], handler)	绑定在给定类型的事件type发送到元素时调用的处理程序handler
.one(type, [data], handler)	绑定在给定类型的事件type发送到元素时调用的处理程序handler，并在handler被调用后立即解除绑定
.unbind([type], [handler])	解除元素上绑定的处理程序（可以指定事件类型或处理程序，不指定则解除所有绑定）
.live(type, handler)	绑定当给定事件发送到元素后调用的处理程序，使用事件委托
.die(type, [handler])	移除前面通过live()绑定到元素上的处理程序
.blur(handler)	绑定当元素失去键盘焦点时调用的处理程序
.change(handler)	绑定当元素的值改变时调用的处理程序
.click(handler)	绑定当元素被单击时调用的处理程序
.dblclick(handler)	绑定当元素被双击时调用的处理程序
.error(handler)	绑定当元素接收到错误事件（取决于浏览器）时调用的处理程序
.focus(handler)	绑定当元素获得键盘焦点时调用的处理程序
.keydown(handler)	绑定当元素拥有键盘焦点且有键被按下时调用的处理程序
.keypress(handler)	绑定当元素拥有键盘焦点且有按键事件发生时调用的处理程序
.keyup(handler)	绑定当元素拥有键盘焦点且有键被释放时调用的处理程序
.load(handler)	绑定当元素加载完成时调用的处理程序
.mousedown(handler)	绑定当在元素中按下鼠标键时调用的处理程序
.mouseenter(handler)	绑定当鼠标指针进入元素时调用的处理程序。不受事件冒泡影响
.mouseleave(handler)	绑定当鼠标指针离开元素时调用的处理程序。不受事件冒泡影响
.mousemove(handler)	绑定当在元素中移动鼠标指针时调用的处理程序
.mouseout(handler)	绑定当鼠标指针离开元素时调用的处理程序
.mouseover(handler)	绑定当鼠标指针进入元素时调用的处理程序
.mouseup(handler)	绑定当在元素中释放鼠标键时调用的处理程序
.resize(handler)	绑定当调整元素大小时调用的处理程序
.scroll(handler)	绑定当元素的滚动位置改变时调用的处理程序
.select(handler)	绑定当元素中的文本被选中时调用的处理程序
.submit(handler)	绑定当表单元素被提交后调用的处理程序
.unload(handler)	绑定当元素从内存中被卸载后调用的处理程序
.hover(enter, leave)	绑定当鼠标指针进入元素时调用的enter及当鼠标指针离开元素时调用的leave处理程序
.toggle(handler1, handler2, ...)	绑定当元素被单击时调用的handler1，当元素再次被单击时调用的handler2，……依次绑定在后续单击时调用的一系列处理程序
.trigger(type, [data])	触发元素上的事件并执行该事件的默认操作
.triggerHandler(type, [data])	触发元素上的事件，但不执行任何默认操作
.blur()	触发blur事件
.change()	触发change事件

(续)

事件方法	说 明
.click()	触发click事件
.dblclick()	触发dblclick事件
.error()	触发error事件
.focus()	触发focus事件
.keydown()	触发keydown事件
.keypress()	触发keypress事件
.keyup()	触发keyup事件
.select()	触发select事件
.submit()	触发submit事件

D.4 效果方法

可以使用效果方法为DOM元素应用动画。第4章详细讨论了效果方法（见表D-4）。

表D-4

效果方法	说 明
.show()	显示匹配的元素
.hide()	隐藏匹配的元素
.show([speed], [callback])	通过高度、宽度及透明度动画显示匹配的元素
.hide([speed], [callback])	通过高度、宽度及透明度动画隐藏匹配的元素
.toggle([speed], [callback])	显示或隐藏匹配的元素
.slideDown([speed], [callback])	以滑入方式显示匹配的元素
.slideUp([speed], [callback])	以滑出方式隐藏匹配的元素
.slideToggle([speed], [callback])	以滑动方式显示或隐藏匹配的元素
.fadeIn([speed], [callback])	以淡入方式显示匹配的元素
.fadeOut([speed], [callback])	以淡出方式隐藏匹配的元素
.fadeTo(speed, opacity, [callback])	调整匹配元素的不透明度
.animate(attributes, [speed], [easing], [callback])	针对指定的CSS属性执行自定义动画
.animate(attributes, options)	.animate()的底层接口，支持对动画队列的控制
.stop([clearQueue], [jumpToEnd])	停止当前播放的动画，然后启动排列的动画（如果有）
.queue()	取得第一个匹配元素上的动画队列
.queue(callback)	在动画队列的最后添加回调函数
.queue(newQueue)	以新队列替换原队列
.dequeue()	执行队列中的下一个动画

D.5 DOM操作方法

第5章详细了DOM操作方法（见表D-5）。

表 D-5

方 法	说 明
.attr(key)	取得属性key的值
.attr(key, value)	设置属性key的值为value
.attr(key, fn)	设置属性key的值为fn（基于每个匹配的元素单独调用）返回的结果
.attr(map)	根据传入的键-值对参数设置属性的值
.removeAttr(key)	移除属性key
.addClass(class)	为每个匹配的元素添加传入的类
.removeClass(class)	从每个匹配的元素中移除传入的类
.toggleClass(class)	（针对每个匹配的元素）如果传入的类存在则移除该类，否则添加该类
.hasClass(class)	如果匹配的元素中至少有一个包含传入的类，则返回true
.html()	取得第一个匹配元素的HTML内容
.html(value)	将每个匹配元素的HTML内容设置为传入的value
.text()	取得所有匹配元素的文本内容，返回一个字符串
.text(value)	设置每个匹配元素的文本内容为传入的value
.val()	取得第一个匹配元素的value属性的值
.val(value)	设置每个匹配元素的value属性的值为传入的value
.css(key)	取得CSS属性key的值
.css(key, value)	设置CSS属性key的值为传入的value
.css(map)	根据传入的键-值对参数设置CSS属性的值
.offset()	取得第一个匹配元素相对于视口的上、左坐标值（单位：像素）
.position()	取得第一个匹配元素相对于.offsetParent()返回元素的上、左坐标值（单位：像素）
.scrollTop()	取得第一个匹配元素的垂直滚动位置
.scrollTop(value)	设置每个匹配元素的垂直滚动位置为传入的value
.scrollLeft()	取得第一个匹配元素的水平滚动位置
.scrollLeft(value)	设置每个匹配元素的水平滚动位置为传入的value
.height()	取得第一个匹配元素的高度
.height(value)	设置每个匹配元素的高度为传入的value
.width()	取得第一个匹配元素的宽度
.width(value)	设置每个匹配元素的宽度为传入的value
.innerHeight()	取得第一个匹配元素的包含内边距但不包含边框的高度
.innerWidth()	取得第一个匹配元素的包含内边距但不包含边框的宽度
.outerHeight(includeMargin)	取得第一个匹配元素的包含内边距、边框及可选的外边距的高度
.outerWidth(includeMargin)	取得第一个匹配元素的包含内边距、边框及可选的外边距的宽度
.append(content)	在每个匹配元素内部的末尾插入content
.appendTo(selector)	将匹配的元素插入到selector选择符匹配的元素内部的末尾
.prepend(content)	在每个匹配元素内部的开头插入content
.prependTo(selector)	将匹配的元素插入到selector选择符匹配的元素内部的开头
.after(content)	在每个匹配元素的后面插入content

(续)

方 法	说 明
<code>.insertAfter(selector)</code>	将匹配的元素插入到selector选择符匹配的元素的后面
<code>.before(content)</code>	在每个匹配元素的前面插入content
<code>.insertBefore(selector)</code>	将匹配的元素插入到selector选择符匹配的元素的前面
<code>.wrap(content)</code>	将匹配的每个元素包装在content中
<code>.wrapAll(content)</code>	将匹配的每个元素作为一个单元包装在content中
<code>.wrapInner(content)</code>	将匹配的每个元素内部的内容包装在content中
<code>.replaceWith(content)</code>	将匹配的元素替换为content
<code>.replaceAll(selector)</code>	将selector选择符匹配的元素替换为匹配的元素
<code>.empty()</code>	移除每个匹配元素的子节点
<code>.remove([selector])</code>	从DOM中移除匹配的节点，也可以通过selector选择符筛选
<code>.clone([withHandlers])</code>	返回所有匹配元素的副本，也可以复制事件处理程序
<code>.data(key)</code>	取得与第一个匹配元素关联的key键的数据项
<code>.data(key, value)</code>	设置与每个匹配元素关联的key键的数据项为value
<code>.removeData(key)</code>	移除与每个匹配元素关联的key键的数据项

D.6 AJAX方法

使用AJAX方法可以不刷新页面就从服务器取得信息。第6章详细讨论了AJAX方法（见表D-6）。

表 D-6

AJAX方法	说 明
<code>\$.ajax(options)</code>	使用传入的options生成一次AJAX请求。这是一个通常由其他便捷方法调用的底层方法
<code>.load(url, [data], [callback])</code>	向传入的url生成一次AJAX请求，然后将响应放在匹配的元素中
<code>\$.get(url, [data], [callback], [returnType])</code>	使用GET方法向传入的url生成一次AJAX请求
<code>\$.getJSON(url, [data], [callback])</code>	向传入的url生成一次AJAX请求，并且将响应作为JSON数据结构解析
<code>\$.getScript(url, [callback])</code>	向传入的url生成一次AJAX请求，并且将响应作为JavaScript脚本执行
<code>\$.post(url, [data], [callback], [returnType])</code>	使用POST方法向传入的url生成一次AJAX请求
<code>.ajaxComplete(handler)</code>	绑定当任意AJAX事务完成后调用的处理程序
<code>.ajaxError(handler)</code>	绑定当任意AJAX事务发生错误时调用的处理程序
<code>.ajaxSend(handler)</code>	绑定当任意AJAX事务开始时调用的处理程序
<code>.ajaxStart(handler)</code>	绑定当任意AJAX事务开始但没有其他AJAX事务活动时调用的处理程序
<code>.ajaxStop(handler)</code>	绑定当任意AJAX事务结束但没有其他AJAX事务还在活动时调用的处理程序
<code>.ajaxSuccess(handler)</code>	绑定当任意AJAX事务成功完成时调用的处理程序
<code>\$.ajaxSetup(options)</code>	为后续的AJAX事务设置默认选项

(续)

AJAX方法	说 明
.serialize()	将一组表单控件的值编码为一个查询字符串
.serializeArray()	将一组表单控件的值编码为一个JSON数据结构
\$.param(map)	将任意值的映射编码为一个查询字符串

D.7 其他方法

表D-7中的实用方法不能归入前面的几类中，但在使用jQuery编写脚本时仍然是非常有用的。

表 D-7

方法或属性	说 明
\$.support	返回一个属性的映射，表示浏览器是否支持各种特性和标准
\$.each(collection, callback)	迭代遍历集合，针对集合中的每一项执行回调函数
\$.extend(target, addition, ...)	扩展target对象，即将后面传入对象的属性添加到这个对象中
\$.grep(array, callback, [invert])	通过使用回调函数测试来筛选数组
\$.makeArray(object)	将对象转换为一个数组
\$.map(array, callback)	针对数组中每一项执行回调函数，将返回的结果组织成一个新数组返回
\$.inArray(value, array)	确定数组array中是否包含值value
\$.merge(array1, array2)	合并数组array1和array2
\$.unique(array)	从数组中移除重复的DOM元素
\$.isFunction(object)	确定对象是不是一个函数
\$.trim(string)	从字符串末尾移除空白符
\$.noConflict([extreme])	向其他库让渡\$标识符使用权，恢复使用jQuery标识符
.hasClass(className)	确定匹配元素是否包含给定的类
.is(selector)	确定是否有匹配的元素与给定的选择符表达式匹配
.each(callback)	迭代遍历匹配的元素，针对每个元素执行回调函数
.length	取得匹配元素的个数
.get()	取得与匹配元素对应的DOM节点的数组
.get(index)	取得匹配元素中与传入的索引值对应的DOM节点
.index(element)	取得给定DOM节点在匹配元素集合中的索引值