TURING 图灵程序设计丛书 Web开发系列

MANNING

jQuery in Action

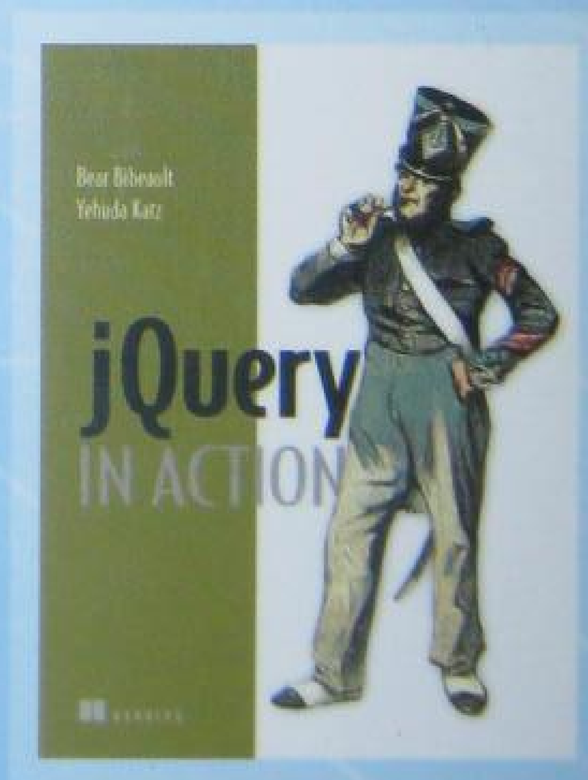
jQuery实战

[美] Bear Bibeault
Yehuda Katz 著

陈宁 等译

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



人民邮电出版社
POSTS & TELECOM PRESS

图灵程序设计丛书 Web开发系列

jQuery in Action

jQuery实战

[美] Bear Bibeault
Yehuda Katz 著

陈宁 等译

人民邮电出版社
北 京

图书在版编目（CIP）数据

jQuery 实战 /（美）比伯奥特（Bibeault, B.），（美）卡茨（Katz, Y.）著；陈宁等译 . —北京：人民邮电出版社，2009.2（2009.7 重印）
书名原文：jQuery in Action
ISBN 978-7-115-19599-9

Ⅰ. j… Ⅱ. ①比…②卡…③陈… Ⅲ. JAVA语言－程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2008）第212394号

内 容 提 要

jQuery 是目前最受欢迎的 JavaScript/Ajax 库之一，能用最少的代码实现最多的功能。本书全面介绍 jQuery 知识，展示如何遍历 HTML 文档、处理事件、执行动画以及给网页添加 Ajax。书中紧紧地围绕“用实际的示例来解释每一个新概念”这一宗旨，生动描述了 jQuery 如何与其他工具和框架交互以及如何生成 jQuery 插件。

本书适合各层次 Web 开发人员。

图灵程序设计丛书

jQuery 实战

◆ 著　[美] Bear Bibeault　Yehuda Katz
译　陈　宁 等
责任编辑　傅志红
执行编辑　杨　爽　谢灵芝

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

◆ 开本：800×1000　1/16
印张：17.5
字数：414 千字　2009 年 2 月第 1 版
印数：6 001 – 8 000 册　2009 年 7 月北京第 3 次印刷

著作权合同登记号　图字：01-2008-5178号

ISBN 978-7-115-19599-9/TP

定价：49.00元

读者服务热线：(010)88593802　印装质量热线：(010)67129223

反盗版热线：(010)67171154

版 权 声 明

Original English language edition, entitled *jQuery in Action* by Bear Bibeault, Yehuda Katz, published by Manning Publications Co., 209 Bruce Park Avenue, Greenwich, CT 06830. Copyright © 2008 by Manning Publications Co.

Simplified Chinese-language edition copyright © 2009 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由 Manning Publications Co.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本书内容。

版权所有，侵权必究。

对本书的赞誉

“这是经过作者深思熟虑写出的一部关于jQuery库的深入透彻的著作。”

——jQuery的创建者John Resig

“瞬间解决复杂的UI问题。这是一本必读之作。”

——Jonathan Bloomer，Soap Creative公司

“即使没有jQuery背景知识，你也能消化此书……读完这本书后，你就可以实现动画效果，创建你自己的插件，随心所欲地使用jQuery的实用工具函数。” ——David Hayden，微软MVP

“穿越jQuery的奇幻之旅。”

——John C. Tyler，UBS投资银行

“卓越JavaScript库之卓越向导。”

——Gregg Bolinger，VML公司

“本书就像jQuery本身——快速、实用、高效。” ——Eric Pascarello，《Ajax实战》作者之一

“事实证明，这是jQuery开发人员最喜欢的一本书。内容全面透彻而无冗词赘句。在展现jQuery优势的同时，也率真地坦露了其弱点。”

——slashdot.org

“它已经成为我最好的朋友。不愧为上乘之作。” ——Andrew Siemer，OTX Research公司

“JavaScript之风格要素。”

——Joshua Heyer，Trane公司

“无论你是初学者还是高级开发人员，都将从这本书中大受裨益。它因简洁而合理的结构形式提高了可读性。”

——dzone.com

“*jQuery in Action*是目前介绍jQuery最新版本1.2的最及时的图书。” ——javaranch.com

“在使用jQuery的过程中，你是否曾因不少困惑和难题而停滞不前？这本书将为你扫清全程障碍！”

——silverlightbooks.net

“这本书是众多技术书的典范：它要言不烦、言辞生动；每章之间环环相扣、彼此关联，有助于你高效地学习；作者对知识点和示例条分缕析，循序渐进地引导读者吸收jQuery妙道。你很快就会发现，读此书如饮甘饴。” ——Ernest Friedman-Hill, JavaRanch管理员

“虽然我读过不少图书评论，却很少为别人写评论。但读完*jQuery in Action*之后，我不得不写点评论，将它推荐给你，因为它实在太棒了，是一个完美的案头伴侣。” ——Amanzon读者评论

序

一切都是为了追求简单。当Web开发者想要编写几个简单交互的时候，为什么非要编写一段又一段冗长而复杂、像书一般沉甸甸的代码呢？事实上，复杂性从来就不是开发Web应用的必要条件。

在我开始着手创建jQuery时，就决定把重点放在小而简单的代码上，它们服务于Web开发者日复一日处理的所有实际应用。阅读本书之后，我非常高兴，因为书中出色地体现了jQuery库的这些原则。

本书特别注重以简洁的形式展示真实可用的代码，可作为想熟悉该库的人们的理想学习资源。

本书最让我满意的是Bear和Yehuda对库内部工作机制细节给予了极大的关注。他们不遗余力地对jQuery API进行调查研究和宣传推广。似乎过去每天我都收到来自他们的电子邮件或即时消息，请求解释、报告新发现的程序缺陷，或者建议该库可改进的地方。你可以放心，摆在你面前的这本书，是作者经过深思熟虑写出的一部关于jQuery库的深入透彻的著作。

本书内容让我感到惊奇的是，它还清晰地论述了jQuery插件以及插件开发背后的策略和理论。jQuery之所以能够如此简单，是因为它利用了插件体系结构。该体系结构提供许多有文档的扩展点，插件可在这之上添加功能。通常那些功能虽然有用却不够通用，才没有纳入jQuery，因此插件体系结构是十分必要的。本书讨论的几个插件，比如Forms（表单）、Dimension（尺寸）以及LiveQuery（实时查询）插件，已经被广泛采用，其理由显而易见：它们的构造、文档编写和维护都是专家级的。请务必对怎样利用和构造插件给予特别关注，因为插件的使用对于jQuery是非常重要的。

拥有本书这样的资源，jQuery项目一定会继续成长和成功。当你开始探索和利用jQuery时，本书将助你一臂之力。

John Resig
jQuery的创建者

前　言

本书的两位作者，一位是头发斑白的老手，他专注于编程的年代可追溯到FORTRAN当道之时，另一位是狂热的领域专家，他颖悟绝人，但如果离开了因特网，他就会对整个世界茫然不知所措。有着如此迥异的背景，两位作者如何走到一起参与合作项目？

答案显然是，jQuery。

我们出于对此极为有用的客户端工具的喜爱而走到一起，但采用的路线是如此不同，犹如白天和黑夜。

我（Bear）第一次听说jQuery是在写作*Ajax in Practice*[①]的时候。图书出版流程的尾声是雷厉风行的编辑阶段。文字编辑审阅全书以保证语句清晰和语法正确，技术编辑保证技术正确性。至少对我来说，这是写书过程中最令人抓狂的紧张时刻，我最不愿听到的就是“你真的应该添加一节全新的内容”。

我在*Ajax in Practice*中写的一章概述了几个支持Ajax的客户端库，其中一个库我已经相当熟悉（Prototype），对其他库（Dojo工具箱和DWR）我只好快速带过。

正当无数任务让我应接不暇的时候（保住一份工作，搞搞副业，处理家务琐事），技术编辑Valentin Crettaz不经意地爆一句：“你为什么不写一节关于jQuery的内容？”

我问：“J什么？”

他马上发表长篇大论，告诉我这个崭新的库有多么奇妙，真的应该与其他Ajax启用客户端库一起进行测评。我四处打听了一下：“有谁可曾听说*jQwerty*库？”

我得到很多积极回应，都很热情，并且一致赞同jQuery真的不同凡响。在一个下雨的星期天下午，我花了大约4个小时在jQuery网站阅读文档，并编写小测试程序，摸索一下jQuery的行事方式。然后匆匆写出了一节新的内容，发送给技术编辑，看看我是否达到了要求。

这节内容得到了热情的赞扬，然后我们继续工作直到最终完成*Ajax in Practice*一书（关于jQuery的这节内容最后还发表在*Dr. Dobb's Journal*的网站上）。

当尘埃落定，对jQuery的狂热已经在我的头脑深处根植了不屈的小种子。我喜欢此前在匆忙研究jQuery时所了解到的知识，于是着手更进一步学习，并开始在Web项目中利用jQuery。我还喜欢我所看到的效果。我开始替换前面项目中的老代码，看jQuery怎样简化页面。我实在是对它爱不释手。

---

① 中文版《Ajax实战：实例详解》，人民邮电出版社，2008。——编者注

我对这个新发现充满热忱，并且想与他人分享，我那时完全丧失了理智，竟然递交了出版本书的提议给Manning出版社。显然，我那时一定是蛮有说服力的。（作为惩罚，我让引起这一切麻烦的技术编辑Valentin，继续做本书的技术编辑。我打赌这一定会给他带来深刻的教训。）

就在那时，编辑Mike Stephens问道："这个项目你和Yehuda Katz合作，怎么样？"

"*Yehenta*是谁啊？"我问道……

Yehuda参与这个项目的方式和我截然不同。他早在jQuery连版本号也没有的时候就参与进去了。在他无意中发现Selectables插件之后，就对jQuery核心库产生了兴趣。他对于（那时）缺少在线文档多少有点失望，于是开始猛刷维基网页，并且建立了Visual jQuery网站（visualjquery.com）。

不久以后，他带头改善在线文档，向jQuery投稿，管理插件体系结构和生态系统，与此同时，他还积极向Ruby社区推广jQuery。

然后有一天他接到Manning出版社打来的电话（朋友把他的名字告知出版社），问他是否有兴趣和名叫Bear的人合作，出一本关于jQuery的书……

尽管背景不同，经历迥异，各有所长，走到一起参与项目的方式也不一样，但我们组成了很棒的团队，并且合作愉快。即便是地理上的距离（我在得克萨斯州的中心，而Yehuda在加利福尼亚州的海边），也没有形成障碍。感谢电子邮件和即时消息带来的便利！

我们想把我们二人的知识和才华凝聚成一本很棒的、能提供大量知识的好书。希望你阅读愉快，恰如我们创作时的愉快。

但愿你在阅读中保持清醒，不似我等神魂颠倒。

致　谢

你是否曾为电影结束时在屏幕上滚动的、仿佛没有尽头的致谢名单而感到惊讶甚至困惑？你是否曾怀疑真的需要那么多人去制作一部电影吗？

类似地，或许很多人会为出版一本书需要那么多人参与而感到惊讶。但事实就是如此，才华各异的贡献者各展其才，努力协作，才能完成你手中的书（或者你在屏幕上阅读的电子书）。

Manning出版社的全体职员孜孜不倦地和我们一起工作，确保本书达到预期的高质量，感谢他们的辛勤付出。没有他们，就没有本书。本书的致谢名单不仅包括出版人Marjan Bace、编辑Mike Stephens，也包括以下做出贡献的人员：Douglas Pudnick、Andrea Kaucher、Karen Tegtmayer、Katie Tenant、Megan Yockey、Dottie Marsico、Mary Piergies、Tiffany Taylor、Denis Dalinnik、Gabriel Dobrescu以及Ron Tomich。

对审稿人我们有说不尽的感谢话。他们帮忙确定本书的最终形式，从发现简单的排印错误，到更正术语和代码错误，甚至帮忙组织本书章目。每通过一个审稿流程，都极大地改善了最终产品的质量。感谢花时甚多审阅本书的以下各位：Jonathan Bloomer、Valentin Crettaz、Denis Kurilenko、Rama Krishna Vavilala、Philip Hallstrom、Jay Blanchard、Jeff Cunningham、Eric Pascarello、Josh Heyer、Gregg Bolinger、Andrew Siemer、John Tyler以及Ted Goddard。

特别感谢Valentin Crettaz。作为本书的技术编辑，他不但在多种环境下检查每段示例代码，还为本书的技术准确性做出了极大的贡献。

Bear Bibeault

这是我的第三本著作，要感谢的人物名单有长长的一大串，再次感谢javaranch.com网站的全体会员和职员。如果没有参加JavaRanch，我就没有机会开始写作，所以诚挚感谢Paul Wheaton 和Kathy Sierra，是他们启动了整件事情，同时感谢给予我鼓励和支持的会员，包括（但或许不局限于）Eric Pascarello、Ben Souther、Ernest Friedman Hill、Mark Herschberg以及Max Habbibi。

感谢Valentin Crettaz，不仅因为他是本书的技术编辑，还因为他给我介绍了jQuery。感谢我的同事Daniel Hedrick，他自愿提供了本书后半部分用到的PHP示例。

向我的爱侣Jay、宠物狗Little Bear（我们起初没能给它起名叫Bear，现在可以了吧？）和Cozmo致以一如既往的温暖谢意。她和它们忍受了我的影子方式的存在，写书的数月之内虽然同居一室，但我却很少把目光抽离MacBook Pro电脑键盘，抬头看上他们一眼。

最后感谢我的合著者Yehuda Katz，没有他，就不可能有这个项目。

Yehuda Katz

首先，感谢我的合著者Bear Bibeault，我受益于他丰富的写作经验。他那天才般的写作技巧和给人深刻印象的解决专业出版物中疑难问题的能力，是本书得以成功的极大保障。

说到本书得以完稿，有必要感谢我深爱的妻子Leah，她忍受了孤单的漫漫长夜和每个周末，向她要求这么长的写作时间，我感到很愧疚。本书得以完稿的功劳有她的一半，而且，她使我得以坚持走过本项目最艰难的部分。我爱你，Leah。

显然，如果没有jQuery库，就没有本书*jQuery in Action*。感谢jQuery的创建者John Resig，是他改变了客户端开发的面貌，减轻了全球Web开发者的负担（不管你相信与否，在中国、日本、法国等许多国家，我们拥有相当大的用户群）。我把他当成好朋友，他本人也是一位才华横溢的作者，帮助我完成了这个艰巨的任务。

jQuery的成长离不开令人钦佩的用户社区和核心团队成员，包括开发组的Brandon Aaron和Jörn Zaefferer，推广组的Rey Bango和Karl Swedberg，负责jQuery UI（用户界面）的Paul Bakaus，与我一道在插件组工作的Klaus Hartl和Mike Alsup。这个很棒的程序员团队把jQuery框架从紧凑、简单的核心操作基础库推进到世界级的JavaScript库，使它完备到对你提出的几乎任何需求，都有由用户贡献的（甚至是模块化的）支持。我可能会漏掉很多jQuery贡献者的名字，有许许多多这样的人。可以这样说，没有围绕这个库而兴起的独一无二的社区，我就不会在这里写作，所以再怎么感谢你们也不过分。

最后，我想感谢家人，自从我最近远离家乡，迁居到西海岸以来，就没能经常回去看望家人。在成长的岁月里，兄弟姐妹和我分享了亲密无间的手足情谊，并且家人对我的信任总是使我坚信能够完成任何事情。妈妈、Nikki、Abie，还有Yaakov：谢谢你们，我爱你们。

关于本书

以少成多。

平常和简单，是本书的宗旨：帮助你学会以较少的脚本在Web应用页面上做更多的事情。本书的两位作者，一位是jQuery贡献者和传道者，另一位是满怀希望和热情的用户，都确信jQuery正是能够帮你达到目的，是当今最好的库。

本书旨在帮助你利用jQuery快速地起步和有力地奔跑，并且希望你一路上享受到快乐。本书会讨论全部的核心jQuery API，以简单易懂的语法块形式呈现每个API方法，描述方法的参数和返回值。书中还加入了有效地使用API的小示例；并且对于“大概念”，提供了我们称为“实验室页面”的材料。这些综合而有趣的实验室页面有助于你理解jQuery方法在运行中的细微差别，你就不必亲自去写一大堆代码了。

所有的示例代码和实验室页面都可以从http://www.manning.com/bibeault下载[①]。

我们可以喋喋不休地说一些营销话语告诉你本书有多好，但你不想浪费时间去读，对吧？你真正想要的是伸臂探宝，不是吗？

是什么使你踌躇不前？请往下阅读！

读者

本书适合广大的开发人员，从初学者到高级Web开发者，如果他们想要在页面上自如地运用JavaScript，写出很棒的、交互式的富因特网应用（Rich Internet Application），而不必为了实现这种应用从零开始写出所有必需的客户端代码。

所有Web开发者都渴望借助jQuery带来的力量，开发出令用户满意的Web应用，他们将会从本书受益。

尽管Web开发新手也许会发现某些部分有点复杂，但不应因此而停止钻研本书的步伐。我们准备了有关JavaScript基本概念的附录，帮你充分发掘jQuery的潜力，只要理解了几个关键概念，读者将会发现jQuery库是利于新手上手的——这一切都不牺牲提供给高级Web开发者的力量。

不管是新手还是老手，客户端程序员如果学会把jQuery加入开发工具箱里，都将大受裨益。我们确信本书的内容将帮助你快速掌握这门学问。

---

① 也可以从图灵网站www.turingbook.com本书网页免费注册下载。——编者注

路线图

本书的组织有助于你以快速的、高效的方式消化吸收jQuery的知识。开头介绍jQuery的设计思想，然后快速前行到jQuery API的基本概念。随后领你通过不同区域，了解jQuery如何能帮助你写出极好的客户端代码，从事件处理一直到向服务器发起Ajax请求。最后，我们会概览最流行的jQuery扩展。

第1章介绍jQuery背后的原理，以及它如何遵循一些现代编码原则，例如，不唐突的JavaScript。我们探讨采用jQuery的理由，纵览一下它如何运作，同时了解一些主要概念，如文档就绪处理程序、实用工具函数、DOM（文档对象模型）元素创建，以及jQuery扩展的创建。

第2章介绍jQuery包装集（wrapped set）的概念——jQuery正是围绕这个核心概念运作的。我们探讨如何利用丰富强大的jQuery选择器，从页面文档中选择元素创建包装集（将被操作的DOM元素集合）。我们会看到这些选择器尽管功能强大，但还是会巧妙利用标准CSS表示法等我们已有的知识。

第3章研究如何利用jQuery包装集来操作页面DOM。内容涵盖改变元素的样式和特性、设置元素内容、往各处移动元素以及处理表单元素。

第4章展示如何利用jQuery大幅度地简化页面上的事件处理。毕竟，正是处理用户事件使得富因特网应用成为可能。那些不得不跨越不同浏览器实现而应对事件处理程序复杂性的人，一定会感激jQuery带来的简易性。

第5章的主题是动画与特效。这一章将探讨jQuery如何使得创建动画效果不仅没有痛苦，而且效率很高，充满乐趣。

第6章学习实用工具函数和jQuery提供的标志。这章内容不仅适合页面作者，也适合为jQuery编写扩展和插件的人。

第7章说明编写扩展和插件的过程。我们将学习jQuery如何使不懂得JavaScript或jQuery复杂知识的人也能格外轻松地编写扩展，同时了解为什么可以把任何可重用代码写成jQuery扩展。

第8章关注在富因特网应用开发中一个最重要的问题：发起Ajax请求。看jQuery如何使人们不用费脑筋就能在页面上利用Ajax，以及如何帮助我们避免在页面上使用Ajax带来的问题，与此同时大幅度地简化最常用的Ajax交互（比如返回JSON构造）。

第9章概括描述最流行、最强大的众多jQuery插件，确保我们知道如何查找更多有关插件的信息。同时这一章会研究能够处理表单和Ajax请求的、比核心jQuery更强大的插件，还有能够在页面上实现拖放的插件。

附录主要写给那些想要进修JavaScript概念的人。附录强调JavaScript的重要概念，如函数上下文、闭包等，理解这些概念对在页面上最有效地利用jQuery至关重要。

代码约定

所有在代码清单或正文中的代码采用像这样（`like this`）的字体，以便和普通文本相区分。正文中的方法和函数的名称及属性、XML元素以及特性也用同样的字体表示。

在某些情况下为了适应本书的版面宽度，重新排版了原始源代码。总地来说，在编写原始代码时我们留意了页宽限制，但有时你会发现代码清单中的代码和从网站下载的代码存在格式上的细微差别。在少数情况下，长代码行如果重新格式化就会改变其含义，此时书中的代码清单会采用续行记号（➥）。

为了突出重要概念，许多代码清单附加了代码注释。在很多情况下，使用带圈的编号与随后正文中的解释（也有带圈数字）一一对应。

代码下载

本书所有示例的源代码（还有一些在书中没有出现的附加代码）都可以从http://www.manning.com/jQueryinAction或http://www.manning.com/bibeault下载[①]。

每章的代码示例以易于在本机Web服务器上发布的形式来组织。把下载代码解压到你选定的文件夹，并且把此文件夹设置为Web应用文档根目录。启动页面是建立在Web应用根目录中的文件index.html。

除了第8章的示例和第9章的几个示例以外，大部分示例无需用到Web服务器，直接用浏览器打开就可运行。文件chapter8/tomcat.pdf将指导你轻松地设立Tomcat作为Web服务器，去运行第8、9章的示例。

所有示例在多种浏览器中经过测试，包括IE7、Firefox 2、Opera 9、Safari 2以及Camino 1.5。示例在IE6中一般也可以运行，但可能遇到一些页面布局问题。不过需要注意的是，所有的jQuery代码在IE6中运行都不会有问题，是示例中的CSS导致了页面布局异常。因为本书的目标读者是专业的Web开发者，所以我们假定所有读者都有一种或多种浏览器可以运行示例代码。

交流与反馈

本书英文原版论坛是http://www.manning.com/jQueryinAction或http://www.manning.com/bibeault。论坛提供了一个场合，让读者之间、读者与作者之间进行富有意义的对话。同时，本书的译者特意申请了电子邮箱jQueryin@126.com。读者在阅读本书的过程中发现的问题，或者有什么好的意见和建议都可以发送邮件到jQueryin@126.com及时与译者交流、探讨。读者还可以发送邮件到contact@turingbook.com及时与本书编辑取得联系。

关于书名

通过介绍、概览和“怎样去做”示例，实战（In Action）系列图书是为帮助学习和记忆而设计的。根据认知科学的研究，人们容易记住他们在自我推动的探索过程中所发现的事情。

虽然Manning出版社的工作人员没有谁是认知科学家，但我们确信，要使学习效果持久，就必须探索、动手实践，并且饶有兴致地复述所学内容。只有在积极探索新事物之后，人们才能理

① 也可以从图灵网站www.turingbook.com本书网页免费注册下载。——编者注

解和记住新事物，也就是掌握新事物。人类在实践中学习，因此，示例驱动是实战系列图书不可缺少的一部分。示例驱动鼓励读者尝试新事物、编写新代码以及探索新想法。

采用这个书名还有另一个更为现实的理由：我们的读者很忙。他们利用图书去完成工作任务或者解决问题。他们需要可以轻松地拿起和放下，想学的时候就能学到想学内容的书；他们需要在实际行动中帮得上忙的书。本系列的图书正是为这样的读者而设计的。

关于封面图片

本书封面上的人物肖像叫做“看守人”。这幅图片摘自法国的J. G. St. Saveur所著的《旅行百科》，1796年出版。那时为了乐趣而旅行是相对较新的现象，诸如此类的旅行指南很受欢迎，它们向旅游者和在家神游者介绍世界各地的人物风情，包括法国士兵、公务员、手工艺者、商人和农民的区域性服装样式和制服样式。

《旅行百科》中多姿多彩的图片生动地描绘了200年前世界各地的不同特色。在那个时代，即使两地仅相距几十英里，人们的服饰也可能是迥异的，服饰就标志着人们属于哪个地区。这本旅行指南活灵活现地反映出那个时代的隔离感和距离感，其实除了当今这个信息发达、无处不联系的时期以外，历史上的所有时期都存在这种隔离感和距离感。

服装样式从那时起发生了很大变化，原先各个地域多种多样的服饰正在逐渐消失。现在通常很难区分两个不同地区的人。如果从好的一方面看，我们也许是在用文化上和视觉上的多样性来换取更加多元化的个人生活，或者是多元化的、充满趣味的才智人生。

这本旅行指南中的图片重现了两个世纪前丰富多样的地域风情。Manning出版社用这样的图片作为封面来赞美计算机行业的独创性、开拓性和无穷乐趣。

目　录

第1章　引荐 jQuery……1
1.1　为什么是jQuery……1
1.2　不唐突的JavaScript……2
1.3　jQuery基本原理……4
1.3.1　jQuery包装器……4
1.3.2　实用工具函数……6
1.3.3　文档就绪处理程序……6
1.3.4　创建DOM元素……7
1.3.5　扩展jQuery……9
1.3.6　使用jQuery和其他库……10
1.4　小结……10
第2章　创建元素包装集……12
2.1　选择将被操作的元素……12
2.1.1　利用基本CSS选择器……14
2.1.2　利用子选择器、容器选择器和特性选择器……14
2.1.3　通过位置选择……18
2.1.4　利用自定义jQuery选择器……20
2.2　生成新HTML……22
2.3　管理包装元素集合……24
2.3.1　确定包装集的大小……25
2.3.2　从包装集获取元素……25
2.3.3　筛选元素包装集……27
2.3.4　利用关系获取包装集……32
2.3.5　还有更多使用包装集的途径……32
2.3.6　管理jQuery链……34
2.4　小结……35
第3章　用 jQuery 让页面生动起来……36
3.1　操作元素属性和特性……36
3.1.1　操作元素属性……38
3.1.2　获取特性值……38
3.1.3　设置特性值……40
3.1.4　删除特性……41
3.1.5　特性带来的快乐……42
3.2　修改元素样式……43
3.2.1　添加和删除类名称……43
3.2.2　获取和设置样式……45
3.2.3　样式相关的更有用的命令……50
3.3　设置元素内容……51
3.3.1　替换HTML或文本内容……51
3.3.2　移动和复制元素……52
3.3.3　包裹元素……56
3.3.4　删除元素……57
3.3.5　克隆元素……58
3.4　处理表单元素值……59
3.5　小结……61
第4章　事件……62
4.1　浏览器的事件模型……63
4.1.1　DOM第0级事件模型……64
4.1.2　DOM第2级事件模型……68
4.1.3　IE事件模型……73
4.2　jQuery事件模型……73
4.2.1　利用jQuery绑定事件处理程序……73
4.2.2　删除事件处理程序……77
4.2.3　Event实例……78
4.2.4　影响事件传播……79
4.2.5　触发事件处理程序……79
4.2.6　其他事件相关命令……81
4.3　让事件（以及更多）工作起来……84
4.4　小结……94
第5章　用动画和效果来装扮页面……95
5.1　使元素显示和隐藏……95
5.1.1　实现可折叠的列表……96
5.1.2　切换元素的显示状态……100
5.2　以动画方式使函数显示和隐藏……101
5.2.1　使元素逐渐地显示和隐藏……101
5.2.2　使元素淡入和淡出……105
5.2.3　使元素滑上和滑下……107

5.2.4 使动画停止……108
5.3 创建自定义的动画……109
5.3.1 一个自定义的放大动画……110
5.3.2 一个自定义的坠落动画……111
5.3.3 一个自定义的消散动画……112
5.4 小结……113
第6章 jQuery 实用工具函数……115
6.1 利用jQuery标志……115
6.1.1 检测用户代理……116
6.1.2 确定方框模型……121
6.1.3 检测要用的正确的浮动样式……122
6.2 使用jQuery和其他库……123
6.3 操作JavaScript对象和集合……126
6.3.1 修整字符串……126
6.3.2 对属性和集合进行迭代……127
6.3.3 对数组进行筛选……128
6.3.4 对数组进行转换……129
6.3.5 从JavaScript数组上找到更多乐趣……131
6.3.6 扩展对象……132
6.4 动态加载脚本……135
6.5 小结……138
第7章 用自定义插件来扩展 jQuery……139
7.1 为什么要扩展……139
7.2 jQuery插件创建准则……140
7.2.1 给文件和函数命名……140
7.2.2 小心$……141
7.2.3 简化复杂的参数列表……141
7.3 编写自定义实用工具函数……143
7.3.1 创建操作数据的实用工具函数……144
7.3.2 编写日期格式器……145
7.4 添加新的包装器方法……149
7.4.1 在包装器方法中应用多个操作……150
7.4.2 保留在包装器方法之内的状态……154
7.5 小结……161
第8章 利用 Ajax 与服务器交谈……163
8.1 温习Ajax……163
8.1.1 创建一个XHR实例……164
8.1.2 发起请求……165
8.1.3 跟踪进展……166
8.1.4 获得响应……167
8.2 加载内容到元素上……168
8.2.1 利用jQuery加载内容……169
8.2.2 加载动态的库存数据……171
8.3 发起GET和POST请求……175
8.3.1 利用jQuery获取数据……175
8.3.2 获取JSON数据……177
8.3.3 发起POST请求……186
8.4 完全控制Ajax请求……187
8.4.1 带着所有的修整发起Ajax请求……187
8.4.2 设置请求的默认值……189
8.4.3 全局函数……190
8.5 整合一切……194
8.5.1 实现工具提示行为……195
8.5.2 利用术语提示器……197
8.5.3 改进的空间……198
8.6 小结……200
第9章 卓越、强大和实用的插件……201
9.1 表单插件……201
9.1.1 获得表单控件的值……202
9.1.2 清除和复位表单控件……205
9.1.3 通过Ajax提交表单……207
9.1.4 上传文件……213
9.2 尺寸插件……213
9.2.1 宽度和高度的已扩展方法……213
9.2.2 获得滚动尺寸……215
9.2.3 关于偏移和位置……217
9.3 实时查询插件……219
9.3.1 建立主动事件处理程序……219
9.3.2 定义匹配和不匹配监听器……220
9.3.3 强制Live Query求值……221
9.3.4 使Live Query监听器过期……221
9.4 UI插件……225
9.4.1 鼠标交互……225
9.4.2 UI小部件和可视化效果……237
9.5 小结……237
9.6 尾声……238
附录 A JavaScript 必知必会……239
索引……253

第1章 引荐jQuery

1

本章内容

- 为什么应当使用jQuery
- 不唐突的JavaScript意味着什么
- jQuery的基本原理和概念
- 结合其他JavaScript库使用jQuery

JavaScript曾经长期被严谨的Web开发者当作“玩具”语言，然而在过去数年间，随着人们对富因特网应用和Ajax技术重新燃起兴趣，JavaScript重获威望。这门语言不得不快速成长，因为客户端开发者已经抛弃剪切和粘贴JavaScript的方式，转而采用方便快捷、功能完备的JavaScript库。这些库一次性地彻底解决了跨浏览器的难题，并提供新颖的、改进了的Web开发方式。

作为JavaScript库世界的后来者，jQuery如暴风雪般横扫Web开发社区，很快赢得MSNBC等大网站，以及颇受关注的开源项目SourceForge、Trac和Drupal的支持。[①]

和其他着重关注JavaScript灵活技巧的工具包相比，jQuery力求改变Web开发者在创建页面的富功能时的思维方式。与其花时间杂耍JavaScript高级复杂的技巧，设计者不如充分利用自己现有的CSS（Cascading Style Sheet，层叠样式表）、XHTML（Extensible Hypertext Markup Language，可扩展超文本标记语言）及普通JavaScript的知识，去直接操作页面元素，实现更快的开发。

在本书中，我们将要深入考察jQuery。我们先来看看jQuery究竟给页面开发的盛宴带来了什么吧。

1.1 为什么是 jQuery

如果你曾花过时间试着给页面增添动态功能，就会发现经常要遵循这样一种模式：选择一个元素或一组元素，然后以某种方式对其进行操作。你可以隐藏或显示元素，给元素增加CSS类，使元素活动起来，或者修改元素的特性。

利用原始的JavaScript完成这些任务中的任何一个，都会需要数十行的代码。jQuery的创造者为了使这些常见任务变得简单而特意创造了该库。例如，设计者利用JavaScript给表格添加“斑马条纹”（利用形成对比的两种颜色使表格隔行突出显示）需要10多行代码，下面请看如何利用

---

① 此外，jQuery已确定成为ASP.NET MVC和Visual Studio未来版本中的正式组成部分。诺基亚手机平台Web Run-Time也将纳入jQuery。——编者注

jQuery来实现同样的功能：

```
$("table tr:nth-child(even)").addClass("striped");
```

如果现在对你来说，这些代码看起来有点神秘，也不用担心。稍后你就会明白它的工作原理，
2 并且简洁扼要却威力强大的jQuery语句会脱口而出，页面注入活力。下面简要探讨一下这个片段怎样工作。

我们标识页面中所有表格的所有偶数行（<tr>元素），并且将CSS类striped添加到每一个偶数行。借助于CSS规则中的类striped，把选定的背景色应用到偶数行，仅仅一行JavaScript就增进了整个页面的美感。

将其应用于示例表格，效果如图1-1所示。

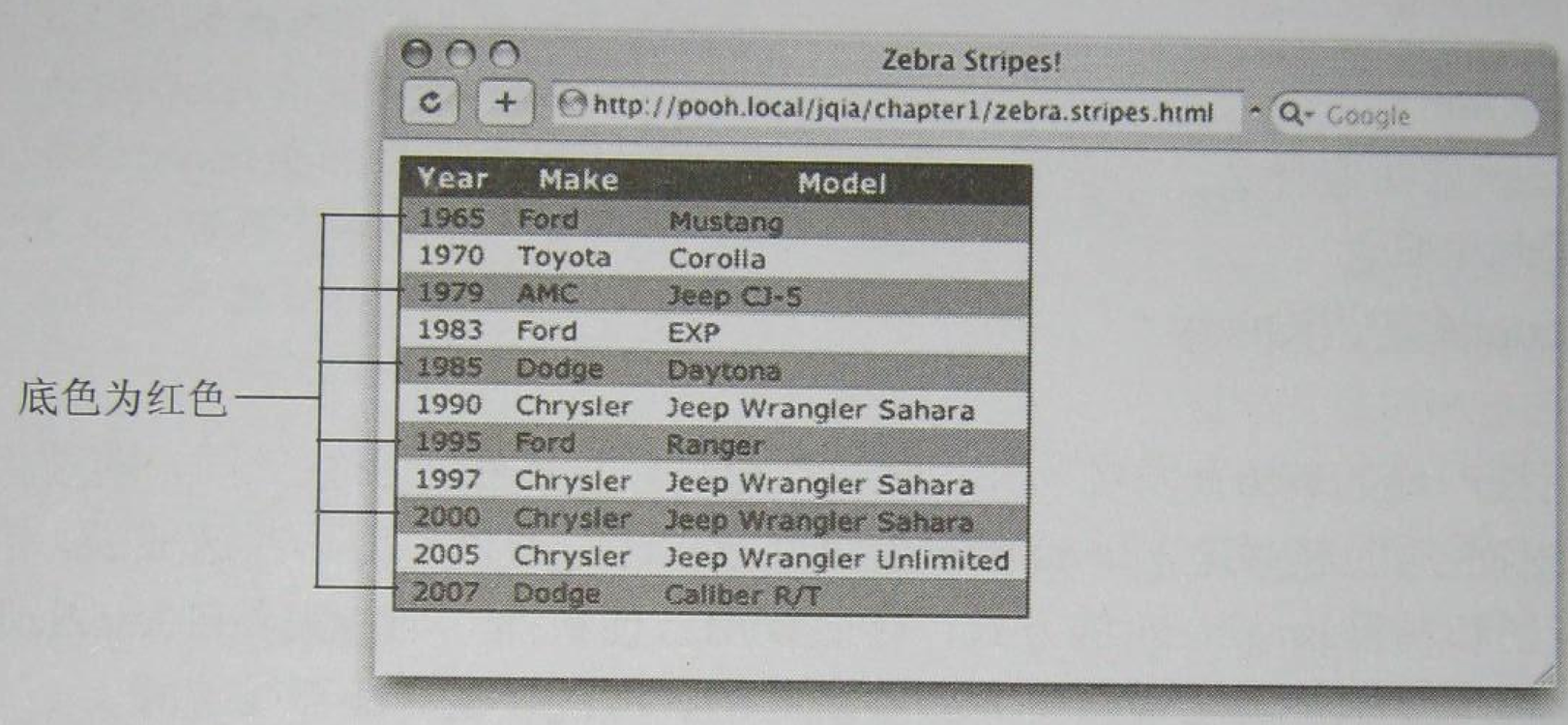


图1-1　给表格添加“斑马条纹”，用jQuery只需一个简单的语句就能实现

这条jQuery语句的真正威力来自选择器，也就是用于标识页面里的目标元素的表达式，我们可以借此轻而易举地标识和获取所需元素。本示例中，目标元素就是所有表格中所有偶数<tr>元素。在本书的可下载代码文件chapter1/zebra.stripes.html中，可以找到本页面的完整源代码。

我们下面将要学习如何轻松地创建选择器。不过首先，让我们了解jQuery的发明者对于在页面中最有效地利用JavaScript，持有什么样的想法。

1.2　不唐突的 JavaScript

记得在CSS出现之前糟糕的往日吗？那时候我们不得不在HTML页面中，把样式标记与文档结构标记混在一起使用。

从事页面设计的任何人，不管时间是长还是短，肯定都会记得，很可能还有些后怕。现在把CSS添加到Web开发工具包，就可以把样式信息从文档结构中分离，并且摒弃了<font>标签之类
3 的滑稽模仿[①]。把样式从结构分离出来，不仅文档易于处理，并且我们拥有了切换整个页面样式的灵活性，仅仅是改换一下样式表就成。

几乎没有人愿意回到把样式混入HTML元素的日子，然而下面这种写法还是太普遍：

```
<button
  type="button"
```

① 作者认为样式标签<font>是<body>等文档结构标签的拙劣模仿，在CSS被采用后应该摒弃。——译者注

```
  onclick="document.getElementById('xyz').style.color='red';">
    Click Me
</button>
```

不难看出，按钮元素的样式，包括标题的字体，并没有使用<font>标签和其他被否定的面向样式的标记，而是通过CSS规则在页面表现出来。虽然此声明并没有把样式标记混入结构里，但却把行为混入了结构里——作为按钮元素标记的一部分而嵌入的JavaScript，当按钮被点击时将被执行（在本示例中，就是按钮被点击时，名为xyz的DOM元素会变为红色）。

就像在HTML文档中应该把样式从结构中分离出来一样，出于完全相同的理由，把行为从结构中分离出来，会带来同样多甚至更多的好处。

行为与结构相分离被称为“不唐突的JavaScript”（Unobtrusive JavaScript），而jQuery的发明者努力让该库能帮助页面作者易于在页面中实现这种分离。不唐突的JavaScript，连同大量的jQuery实践经验一起，认为任何嵌入在HTML页面<body>里的JavaScript表达式或语句，不管是作为HTML元素的特性（比如 onclick），还是嵌入在页面<body>的脚本块里，都是不正确的。

你也许会问：“但要是没有onclick特性的话，该如何设置按钮的行为呢？”考虑下面按钮元素的变化：

```
<button type="button" id="testButton">Click Me</button>
```

简单多了！但现在你却发现，按钮什么也干不了。

不是把按钮的行为嵌入到按钮的标记里，而是将它转移到页面<head>节的脚本块里，从而处于页面<body>节的范围之外，如下所示：

```
<script type="text/javascript">
  window.onload = function() {
    document.getElementById('testButton').onclick = makeItRed;
  };

  function makeItRed() {
    document.getElementById('xyz').style.color = 'red';
  }
</script>
```

我们把脚本放在页面的onload处理程序中，给按钮元素的onclick特性指派makeItRed()函数。之所以将此脚本添加到onload处理程序中（而不是内联在结构标记中），是因为在试图操作按钮元素之前，必须先确认按钮元素的存在（在1.3.3节将看到jQuery如何提供更好的地方来存放此类代码）。

如果你觉得本示例的代码很怪异，别担心！附录A提供了一些JavaScript概念，为了有效地利用jQuery，你必须理解这些概念。在本章的剩余部分里，将会探讨jQuery如何使我们能以更加轻松、简短和通用的方式来编写前面的代码。

不唐突的JavaScript虽然是促进Web应用里明确分工的强大技巧，但应用起来却不无代价。也许你已经注意到，比起将行为放入按钮的标记来，不唐突的JavaScript需要多出好几行脚本去达到同样的目的。不唐突的JavaScript不仅可能增加需要编写的脚本量，还要求客户端脚本遵守某些规则，并且应用好的编码模式。

这些并不坏，说服我们像对待服务端代码那样小心谨慎地去写客户端代码，是件好事！但那要做很多额外的工作——如果没有jQuery的话。

如同前面提到的那样，jQuery团队特意使jQuery关注以下任务：使得在页面编程中使用不唐突的JavaScript技巧成为简单快乐的事儿，而无需耗费大量精力写大块代码。我们将发现有效地使用jQuery，可以编写更少的代码在页面上实现更多的功能。

下面探讨jQuery怎样使得给页面增添富功能如此容易，而没有预想中的痛苦。

1.3　jQuery基本原理

在其核心，jQuery重点放在从HTML页面里获取元素并对其进行操作。如果你熟悉CSS，就会很清楚选择器的威力，通过元素的特性或元素在文档中的位置去描述元素组。有了jQuery，你就能够利用现有知识去发挥选择器的威力，在很大程度上简化JavaScript代码。

jQuery把确保代码能跨越所有主要浏览器以一致的方式工作，摆在了高优先级的位置。许多更为困难的JavaScript问题，比如在执行页面操作之前必须等待页面加载完毕之类的问题，已经被悄悄地解决了。

5

我们一旦发现该库需要添加点什么功能，就有jQuery开发者已经内建了简单而有力的、用于扩展功能的方法。许多jQuery编程新手发现他们第一天就通过扩展jQuery而将多功能性付诸实践。

不过首先，让我们看看如何借用CSS知识去生成强大却又简洁的代码。

1.3.1　jQuery包装器

为了使设计和内容分离而把CSS引入Web技术的时候，需要以某种方式从外部样式表中引用页面元素组。开发出来的方法就是通过使用选择器——基于元素的特性或元素在HTML文档中的位置，简明地表现元素。

例如，选择器

```
p a
```

引用所有嵌套于<p>元素之内的链接（<a>元素）组。jQuery利用同样的选择器，不仅支持目前CSS中使用的常见选择器，还支持尚未被大多数浏览器完全实现的更强大的选择器。前面探讨的“斑马条纹”代码中的nth-child选择器，是在CSS3中定义的、更强大的选择器的例子。

收集一组元素，可以使用如下简单的语法：

```
$(selector)
```

或者

```
jQuery(selector)
```

也许刚开始你会觉得$()符号有点奇怪，但大部分jQuery用户很快就喜欢上它的简洁。

例如，为了获取嵌套在<p>元素内的一组链接，我们使用如下语句：

```
$("p a")
```

$()函数（jQuery()函数的别名）返回特别的JavaScript对象，它包含着与选择器相匹配的DOM元素的数组。该对象拥有大量预定义的有用方法，能够作用于该组元素。

6

用编程的话来说，这种构造称为包装器（wrapper），因为它用扩展功能来对匹配的元素进行包装。我们使用术语jQuery包装器或者包装集（wrapped set）[①]，来指能够在其上用jQuery定义的

① 书中还使用术语“匹配集”（matched set），所指的事物与“包装集”或“包装器”相同。——译者注

方法去操作的、匹配元素的集合。

假定我们想让带有CSS类notLongForThisWorld的所有 元素呈现淡出效果。jQuery语句如下所示：

```
$("div.notLongForThisWorld").fadeOut();
```

经常作为jQuery命令引用的许多这类方法的显著特征是，当完成了一个操作（比如淡出操作）时，它们返回相同的一组元素，提供给下一个操作。例如，假定除了让元素呈现淡出效果之外，还想添加新的CSS类removed到每个元素。编写如下代码：

```
$("div.notLongForThisWorld").fadeOut().addClass("removed");
```

这些jQuery链可以无限延续。任由jQuery链无限制地延续的话，你发现包含长达几十个命令的例子并不罕见。因为每个函数都作用在与最初的选择器相匹配的全部元素之上，所以没有必要循环遍历元素数组。所有这一切，jQuery已经在后台为我们完成！

即使选中的对象组被表现为非常复杂的JavaScript对象，如有必要也可以假定它是典型的元素数组。因此以下两个语句产生相同的结果：

```
$("#someElement").html("I have added some text to an element");
```

或者

```
$("#someElement")[0].innerHTML =
  "I have added some text to an element";
```

因为使用了ID选择器，所以只有一个元素与选择器相匹配。第一个示例使用jQuery的html()方法，将DOM元素的内容用某些HTML标记去替换。第二个示例使用jQuery获取元素数组，用数组下标0去选择第一个元素，然后使用通常的JavaScript方式来替换该元素的内容。

如果想用能匹配多个元素的选择器来获取同样的结果，那么以下两个片段产生一致的结果：

```
$("div.fillMeIn")
  .html("I have added some text to a group of nodes");
```

或者

```
var elements = $("div.fillMeIn");
for(i=0;i<elements.length;i++)
  elements[i].innerHTML =
    "I have added some text to a group of nodes";
```

随着事情变得愈发复杂，利用jQuery的可链接性，可以持续减少为获得你所求结果的必需代码的行数。除此之外，jQuery不仅支持你所认识和喜爱的选择器，还支持更高级的选择器——作为CSS规范的一部分而定义——甚至支持某些自定义选择器。

以下是几个示例：

```
$("p:even");
```

该选择器选择所有偶数的<p>元素。

```
$("tr:nth-child(1)");
```

该选择器选择每个表格的第一行。

```
$("body > div");
```

该选择器选择作为<body>直接子节点的 。

```
$("a[href$=pdf]");
```

该选择器选择指向PDF文件的链接。

```
$("body > div:has(a)")
```

该选择器选择作为<body>直接子节点的、包含链接（<a>）的 。

好一个威力强大的家伙！

你可以利用CSS的现有知识起步和快跑，然后学习jQuery支持的更高级的选择器。2.1节将详述jQuery选择器，并且在http://docs.jquery.com/Selectors可以查找选择器的完整列表。

选择DOM元素进行操作是页面中常见的需求，但还有些必须做的事情并不涉及DOM元素。下面概览一下除了元素操作之外jQuery提供的更多功能。

1.3.2　实用工具函数

包装元素以便于操作是jQuery的$()函数最常见的用途之一，但那并不是分派给它的唯一职责。它的另外一个职责是作为几个通用的实用工具函数的命名空间前缀。因为以选择器作为参数去调用$()而创建的jQuery包装器，已经赋予了页面作者如此强大的威力，所以对于大多数页面作者来说，很少需要实用工具函数提供的服务。为编写jQuery插件而做准备的第6章内容，将会仔细探讨大多数实用工具函数。因为在接下来的几节里，将会看到几个用到的实用工具函数，所以先在这里介绍其概念。

8

实用工具函数的表示法乍看也许有些怪异。例如，删除字符串前后空格的实用工具函数，其调用方法如下所示：

```
$.trim(someString);
```

如果你觉得前缀$.怪异，就记住$是JavaScript里的标识符，与其他标识符没有什么两样。利用标识符jQuery（而不是别名$）去调用同样的函数，看起来会熟悉一些：

```
jQuery.trim(someString);
```

到这里就变得清楚了，函数trim()不过是存在于别名为$的命名空间jQuery里罢了。

注意　尽管在jQuery文档中这些方法被称作实用工具函数，但确切无疑的是它们实际上是函数$()的方法。为了不与在线文档产生术语上的冲突，我们撇开技术上的区别而使用实用工具函数来描述这些方法。

1.3.5节将会探讨帮助扩展jQuery的一个实用工具函数，而1.3.6节将会探讨协助jQuery与其他客户端库和平共处的另一个实用工具函数。不过首先看jQuery的$函数履行的另外一个重要职责。

1.3.3　文档就绪处理程序

当我们欣然接受不唐突的JavaScript的时候，行为从结构里分离出来了。因此，执行对页面元素的操作是在创建页面元素的文档标记之外。为了达到这个目的，在执行操作之前要以某种方式

等待，直到页面的DOM元素完全加载。在“斑马条纹”示例中，必须加载整个表格之后才能添加“斑马条纹”。

传统上，window实例的onload处理程序被用于上述目的，就是在整个页面完全加载之后执行语句。该语法通常类似于：

```
window.onload = function() {
  $("table tr:nth-child(even)").addClass("even");
};
```

这样就能在文档完整地加载之后，执行“斑马条纹”代码。不幸的是，浏览器延迟执行onload代码，不仅是在构建DOM树之后，也是在所有图像和其他外部资源完整地加载并且页面在浏览器窗口显示完毕之后。结果是访问者会经历从看到页面时刻到onload脚本执行时刻之间的延迟。

更糟糕的是，如果图像或其他资源要花好一段时间去加载，访问者在页面丰富的行为变得可用之前，就不得不等待图像加载完毕。这就让整个“不唐突的JavaScript”运动，在很多真实生活的场景下不战而败[①]。

更好的解决办法就是，只要等到文档结构被完整地解析，同时浏览器已经把HTML转换成DOM树形式的时候，就立刻执行脚本使丰富的行为生效。以跨浏览器方式来实现这一点稍显困难，然而jQuery提供了简单的方式：一旦DOM树（而不是外部图像资源）加载完毕，就去激活代码的执行。定义这样的代码（利用“斑马条纹”示例）的正式语法如下所示：

```
$(document).ready(function() {
  $("table tr:nth-child(even)").addClass("even");
});
```

首先，用$()即jQuery()函数来包装文档实例，然后调用ready()方法，给其传递一个函数。当文档为操作准备就绪的时候，就去执行该函数。

称之为*正式语法*是有理由的，因为用得频繁得多的简写形式如下所示：

```
$(function() {
  $("table tr:nth-child(even)").addClass("even");
});
```

通过传递函数给$()，我们命令浏览器在执行代码前必须等待，直到DOM加载完毕（但仅仅是DOM）。更好的是，可以在同一个HTML文档中多次使用此技巧，并且浏览器按照函数在页面中定义的先后顺序，一一执行指定的函数。与此不同，window的onload机制只接受一个函数。如果我们利用的任何第三方代码已经出于其自身目的而采用了onload机制（并非最优方法），此限制就会导致难以查找的bug。

学习了$()函数的又一种用法之后，现在看看它还能做点别的什么。

1.3.4 创建 DOM 元素

到目前为止可以看出，jQuery的作者使得$()函数（你的记忆里它只是jQuery()函数的别名而已）足够通用以便履行许多职责，从而避免把一堆全局性的名称引入JavaScript命名空间。好，还有一个职责需要探讨。

---

① 指“不唐突的JavaScript”还没来得及执行，就被等得不耐烦的用户关闭了浏览器。——译者注

通过给$()函数传递包含 HTML 标记的字符串，可以即时创建相应的DOM元素。例如，新建段落元素如下所示：

```
$("<p>Hi there!</p>")
```

但创建无实体的DOM元素（或元素层次结构）不是那么有用。通常，通过这种调用而创建元素层次结构之后，接着利用jQuery的一个DOM操作函数对其进行操作。

请看代码清单1-1的示例。

代码清单1-1　即时创建HTML元素

```
<html>
  <head>
    <title>Follow me!</title>
    <script type="text/javascript" src="../scripts/jquery-1.2.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $("<p>Hi there!</p>").insertAfter("#followMe");
      });
    </script>
  </head>

  <body>
    <p id="followMe">Follow me!</p>
  </body>
</html>
```

❶ 用于创建HTML元素的文档就绪处理程序

❷ 将被追加元素的现有元素

此代码在文档<body>中建立名为followMe❷的现有HTML段落元素。在<head>节的脚本块里，建立文档就绪处理程序❶，利用如下语句把新建的段落元素插入DOM树的现有元素之后：

```
$("<p>Hi there!</p>").insertAfter("#followMe");
```

11

结果如图1-2所示。

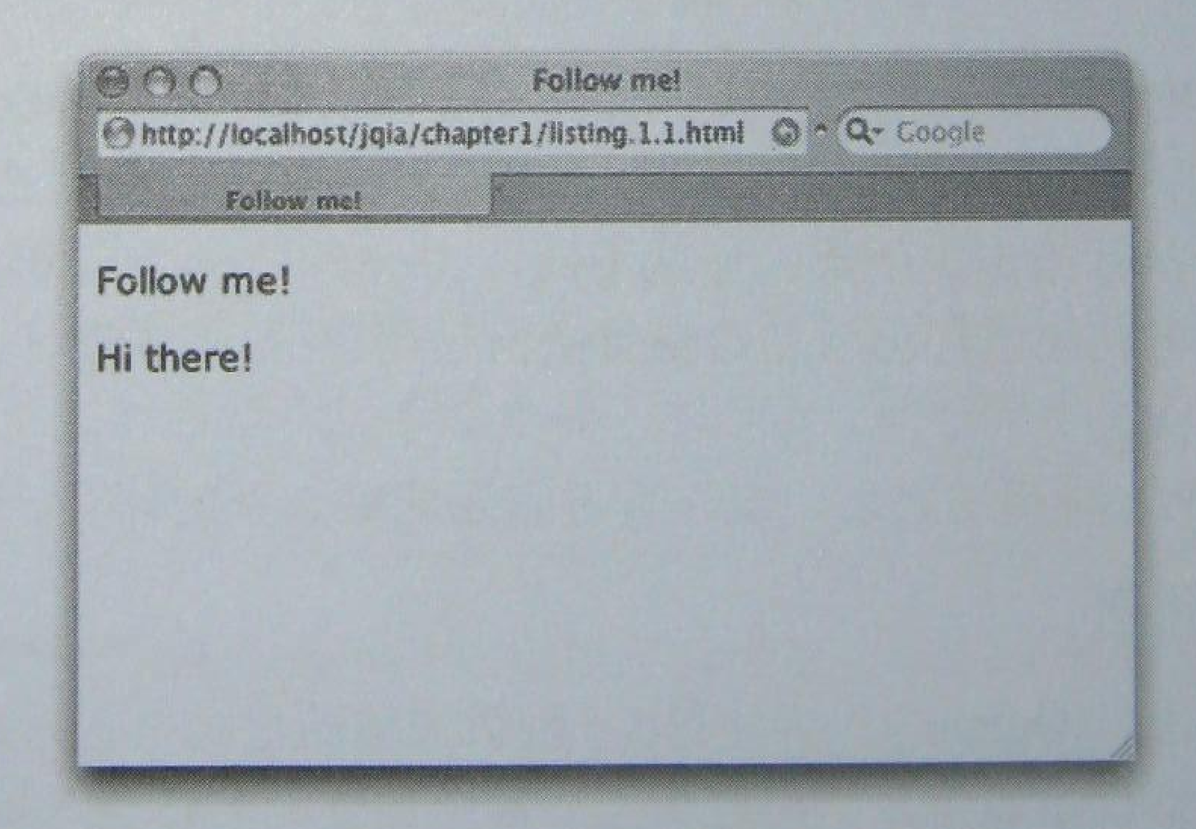


图1-2　动态创建和插入元素

在第2章里将研究整套DOM操作函数，你将看到jQuery提供许多方式去操作DOM，以便获取我们想要的任何结构。

既然你已经了解jQuery的基本语法，下面请看该库最强大的一个特征。

1.3.5 扩展 jQuery

在页面中我们将发现自己一次又一次使用jQuery包装器所提供的大量有用函数。但没有哪个库能够预料所有人的需求。甚至可以这样说，没有哪个库应该设法将每个人需要的东西都预先准备好。这样做可能导致一堆大而笨重的、包含很少用到的功能的代码，这只会把事情搞砸。

jQuery库的作者认识到这一点，因此努力标识大多数页面作者需要的功能，在核心库里仅仅纳入了那些需求。与此同时，jQuery库的作者认识到页面作者拥有各自不同的需求，因此从设计上就特意使得jQuery易于用附加功能来进行扩展。

但为什么扩展jQuery而不是编写独立函数去填补所有空白呢？

因为扩展jQuery更简单！通过扩展jQuery，可以利用它提供的强大功能，特别是在元素选择方面。

下面请看特别示例：jQuery并不提供预定义的、用于禁用一组表单元素的函数。如果我们在Web应用中大量使用表单，就会发现利用以下语法非常便利：

```
$("form#myForm input.special").disable();
```

12

幸运的是，通过扩展在调用$()时返回的包装器，jQuery从设计上就便于扩展函数集。下面请看扩展jQuery的基本惯用语法：

```
$.fn.disable = function() {
  return this.each(function() {
    if (typeof this.disabled != "undefined") this.disabled = true;
  });
}
```

这里引入了许多新语法，但不用太担心。你只要继续学习以下几章，到那时这种语法就会变成老招式了。你将一遍又一遍地使用这种惯用语法。

首先，$.fn.disable表示我们用名为disable的函数来扩展$包装器。在函数内部，this是将被操作的包装DOM元素的集合。

然后，调用包装器的each()方法遍历包装集里每个元素。第2章将详细地探讨与此类似的方法。在传递给each()的迭代器函数的内部，this是指向当前迭代的具体DOM元素的指针。不要被眼前代码搞糊涂了，在这两个嵌套的函数中，两个this分别指向不同的对象。多写几个扩展函数，自然就会记住了。

对每个元素，检查该元素是否拥有disabled特性，如果拥有，则把disabled的值设为true。因为我们返回each()方法的结果（即包装器），所以全新的disable()方法才支持jQuery命令链，就像许多jQuery原生方法那样。可以这样编写：

```
$("form#myForm input.special").disable().addClass("moreSpecial");
```

从页面代码的角度来看，我们的新方法disable()仿佛内建在库里一样！此技巧格外强大，以至于大部分jQuery新用户发现自己几乎一开始使用jQuery库，就给它建立了小的扩展。

此外，富有进取心的jQuery用户已经用一套又一套称为插件的有用函数扩展了jQuery。第9章将更多地讨论以这种方式来扩展jQuery，同时介绍可自由获取的官方插件。

13 在潜心研究利用jQuery使页面动起来之前，你也许想知道：我们将来是否可以和其他库（比如Prototype）一起使用jQuery？这些库也使用$简写符号。下一节公布对这个问题的答复。

1.3.6 使用jQuery和其他库

即使jQuery提供了一套强大的工具，从而满足了页面作者的大部分需求，但有时候，页面也要求采用多个JavaScript库。这种情形很可能发生，因为我们处在把应用从前面采用的库过渡到jQuery的过程中，或者我们想在页面上同时使用jQuery和另一个库。

jQuery团队明确表示他们把重点放在满足社区用户的需求上，而无意于排挤其他库，并且做好了让jQuery在页面上与其他库和平共处的准备。

首先，他们遵循最佳实践准则，避免因为使用大量的标识符而污染全局命名空间——可能不仅妨碍到其他库，也妨碍到你想在页面上采用的名称。jQuery标识符和它的$别名防止jQuery对全局命名空间的侵入。1.3.2节提到的作为jQuery命名空间的一部分而定义的实用工具函数，就是在这点上采取谨慎态度的好榜样。

虽然其他库不太可能找个好理由将全局标识符取名为jQuery，但在特殊场合下，便利的$别名却惹来麻烦。其他JavaScript库，最为著名的是颇受欢迎的Prototype库，出于自身目的而使用了$名称。因为在该库中使用$名称是其操作的关键，所以会导致严重的冲突。

深思熟虑的jQuery作者凭借名副其实的实用工具函数noConflict()，提供了消除冲突的办法。导致冲突的库被加载之后的任何时刻，都可以调用：

```
jQuery.noConflict();
```

把$还原到非jQuery库所定义的含义。

7.2节将进一步说明使用此实用工具函数的细节。

1.4 小结

在此旋风式的jQuery介绍中，我们囊括了大量的材料，准备深入研究利用jQuery快速轻松地启用富因特网应用开发。

jQuery通常对于任何需要执行操作的页面都有用，但它不是平庸的JavaScript操作，而是高度关注使页面作者能够在页面中采用“不唐突的JavaScript”概念。采用jQuery的解决方案，把行为
14 从结构中分离出来，就像CSS把样式从结构中分离出来的方式那样，从而实现更好的页面组织、增强代码的多功能性。

尽管事实上jQuery在JavaScript命名空间中只引入了两个新名称——自命名的jQuery函数和它的$别名——通过使该函数高度多功能化，或基于函数的不同参数来调整所执行的操作，该库还是提供了大量的功能。

正如我们所见，jQuery()函数可用于执行以下任务：

- 选择和包装DOM元素以便操作；
- 充当全局实用工具函数的命名空间；
- 根据HTML标记来创建DOM元素；
- 建立代码，当DOM为操作准备好时立刻执行。

jQuery的举止像是页面上的好公民，不仅自己尽量少地占用JavaScript全局命名空间，还通过提供官方办法去尽可能少地减少当名称冲突依然发生时（比如另一个库，如Prototype，要求使用$名称的时候）对命名空间的侵占。对用户那样友好，夫复何求？

你可以从jQuery网站http://jquery.com/获得jQuery最新版本。曾被用于测试本书配套代码的jQuery版本（1.2.1版）也包含在可下载的代码里。

下一章将探索jQuery给富因特网应用的页面作者提供的一切。下面开始下一章的学习之旅，届时将通过DOM操作给页面带来活力。

第2章 创建元素包装集

2

本章内容

- 利用选择器，选择将被jQuery包装的元素
- 创建并放置新的HTML元素到DOM中
- 操作元素包装集

在上一章中，我们讨论了jQuery的$()函数的多种调用方式。它的功能从选择DOM元素，到定义DOM加载之后所执行的函数，不胜枚举。

在本章中，我们仔细检查如何标识将被操作的DOM元素，通过着眼于$()最强大的、使用最频繁的两大功能：利用选择器来选择DOM元素，创建新的DOM元素。

通过操作组成页面的DOM元素，能够实现富因特网应用所要求的大量功能。但在操作DOM元素之前，必须先标识和选择DOM元素。让我们开始研究一下jQuery指定被操作的目标元素的多种方式。

2.1 选择将被操作的元素

几乎使用任何jQuery方法（经常作为jQuery命令而被引用）的时候，我们必须做的第一件事，就是选择将被操作的页面元素。有时，想要选择的元素集合是易于描述的，比如“页面上的所有段落元素”。有时，需要比较复杂的描述，比如“所有拥有CSS类listElement并且包含链接的列表元素”。

幸好jQuery提供了健壮的选择器语法，让我们能轻松、优雅和简明地指定几乎任何元素集合。你很可能已经知道大部分语法——jQuery采用你已经知道和喜爱的CSS语法，并且扩展了一些用于选择元素的自定义方法，以便帮助你执行复杂而又常见的任务。

为了帮助你学习元素选择，我们编配了选择器实验室页面。在本书可下载代码示例中包含了这个实验室页面。如果你还没有下载示例代码，现在就是下载的好时机。请看本书的前面部分，了解如何查找并下载代码。

选择器实验室页面允许你输入jQuery选择器字符串，然后（实时！）显示哪些DOM元素被选中。选择器实验室页面可在chapter2/lab.selectors.html的示例代码里找到。

选择器实验室页面的显示如图2-1所示（如果窗格显得排列不齐，就得放大浏览器窗口）。

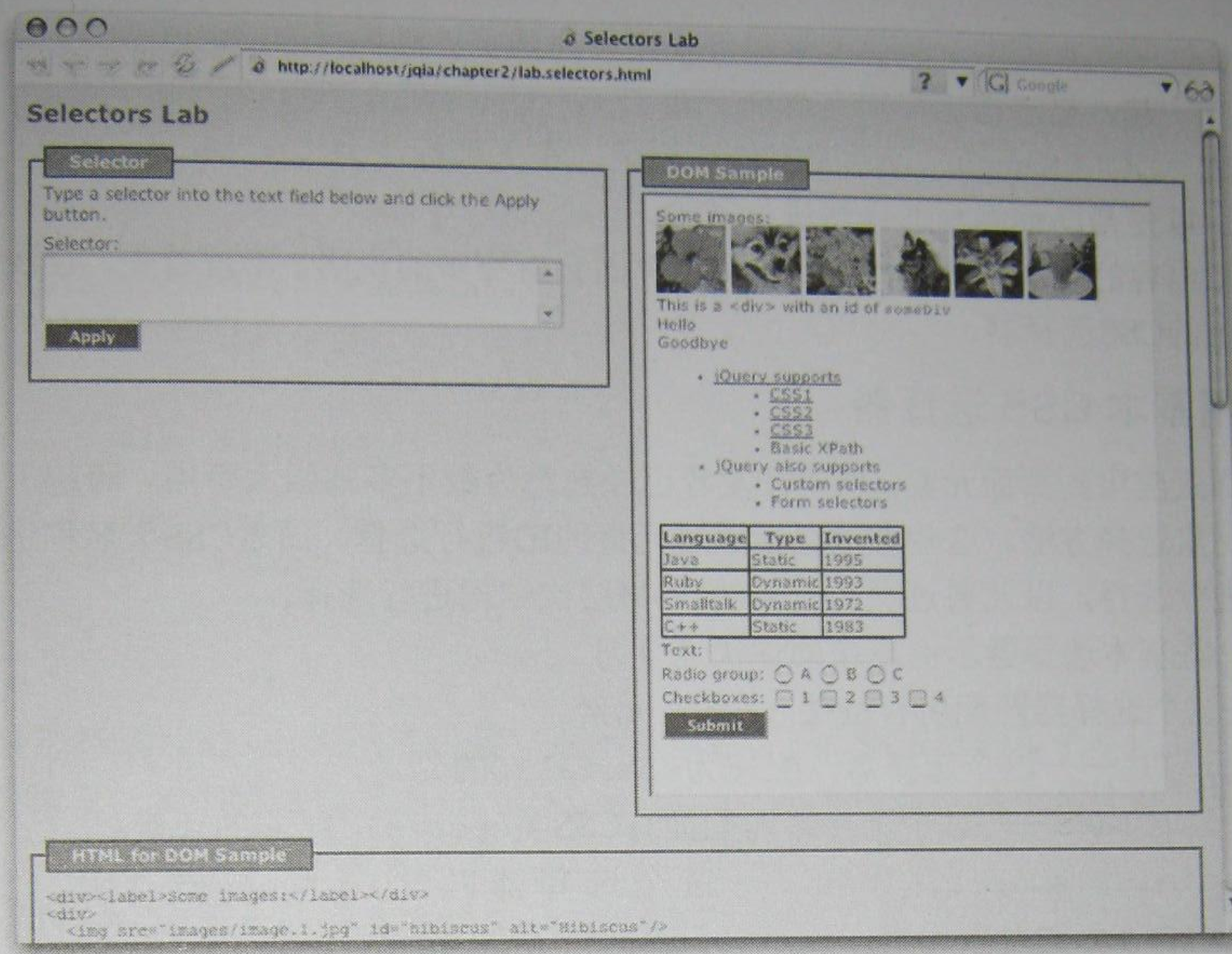


图2-1 选择器实验室页面允许任意输入选择器并实时观察其行为

Selector窗格在左上部，包含了文本框和按钮。为了进行试验，请在文本框里输入选择器，然后点击按钮Apply。例如在文本框里输入字符串`li`，然后点击按钮Apply。

你输入的选择器（此时是`li`）被应用到HTML页面，即加载到右上部DOM Sample窗格`<iframe>`节的页面。示例页面的jQuery代码，导致所有匹配元素以红色边框来突出显示。点击按钮Apply之后，你会看到如图2-2所示的显示结果，即页面中的所有`<li>`元素以红色边框突出显示。

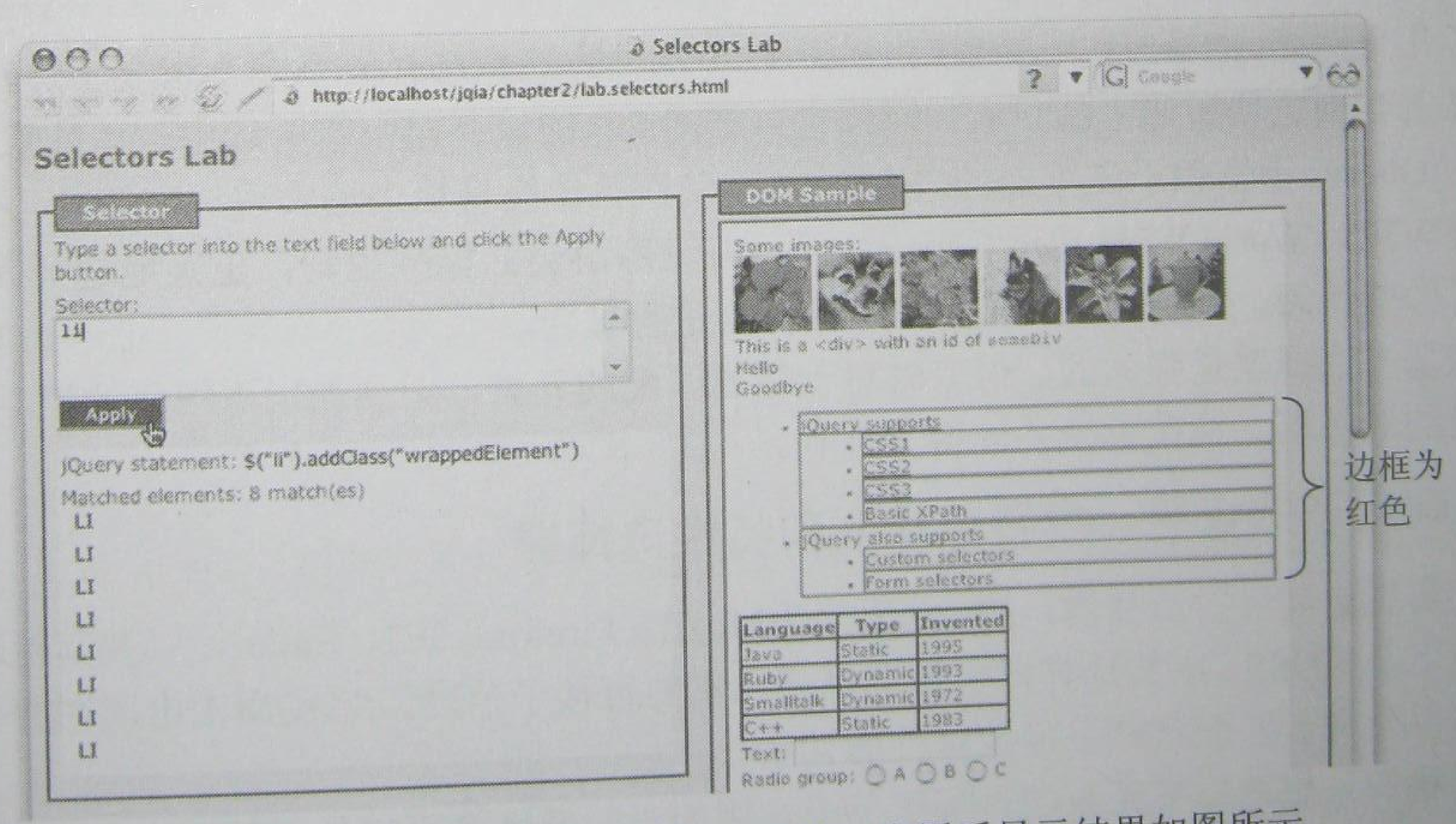


图2-2 值为`li`的选择器匹配所有`<li>`元素，应用后显示结果如图所示

请注意，在示例页面中，<li>元素已经描出红色边框；已执行的jQuery语句，连同已选中元素的标签名称一起，显示在选择器文本框下面。

用来生成“DOM Sample”页面的HTML标记，在下部标注为“HTML for DOM Sample”的窗格中显示，以便帮助你编写选择器进行试验。

随着本章内容的推进，关于利用本实验室页面会有更多的说明。不过首先，让我们涉足熟悉的领域：传统的CSS选择器。

18

2.1.1　利用基本 CSS 选择器

对于把样式应用到页面元素，Web开发者已经熟悉为数不多却强大有用、跨越所有浏览器而工作的一组元素选择方法。这些方法包括通过元素的ID进行选择，通过CSS类名称进行选择，通过标签名称进行选择，以及通过页面元素的DOM层次结构进行选择。

以下是一些基本选择器示例，帮助你快速复习。

- a——这个选择器匹配所有链接（<a>）元素。
- #specialID——这个选择器匹配id为specialID的元素。
- .specialClass——这个选择器匹配拥有CSS类specialClass的元素。
- a#specialID.specialClass——这个选择器匹配id为specialID、拥有CSS类specialClass的链接元素。
- p a.specialClass——这个选择器匹配拥有CSS类specialClass、在<p>元素内声明的链接元素。

可以混合匹配多种基本选择器类型，以便选择相当细化的元素集合。实际上，最别出心裁和富有创造性的网站，都会利用基本选择器的组合去创造令人目眩的页面显示。

19

我们可以突破思维定势，在jQuery里利用已经用惯的CSS选择器。为了利用jQuery来选择元素，请把选择器包装在$()里，如下所示：

```
$("p a.specialClass")
```

除了几个例外，jQuery完全兼容CSS3，所以用这种方式来选择元素毫不奇怪——在样式表里，能被标准兼容浏览器所选中的元素，同样也可被jQuery的选择器引擎所选中。请注意，在浏览器内运行的jQuery，不依赖于浏览器的CSS实现。即使浏览器没有恰当地实现标准CSS选择器，jQuery依然遵守W3C（World Wide Web Consortium，万维网协会）标准规则，正确地选择元素。

为了做些练习，请输入各种基本CSS选择器进行试验。

这些基本选择器是强大的，但有时对于想要匹配的元素，必须加以更细化的控制。jQuery用更加高级的选择器，迎接挑战并取得了胜利。

2.1.2　利用子选择器、容器选择器和特性选择器

为了实现更高级的选择器，jQuery采用为Mozilla Firefox、IE7、Safari，以及其他现代浏览器所支持的下一代CSS。高级选择器包括选择某元素的直接子节点，在DOM中出现在其他元素之后的元素，以及拥有匹配某些条件的特性的元素。

有时我们只想选择某元素的直接子节点。例如，想要选择直接处于某列表之下的列表元素，

而不是属于子列表的列表元素。考虑以下摘自选择器实验室页面的示例DOM的HTML片段：

```
<ul class="myList">
  <li><a href="http://jquery.com">jQuery supports</a>
    <ul>
      <li><a href="css1">CSS1</a></li>
      <li><a href="css2">CSS2</a></li>
      <li><a href="css3">CSS3</a></li>
      <li>Basic XPath</li>
    </ul>
  </li>
  <li>jQuery also supports
    <ul>
      <li>Custom selectors</li>
      <li>Form selectors</li>
    </ul>
  </li>
</ul>
```

假定想要选择远端jQuery网站的链接，而不是描述不同CSS规范的多个本地页面的链接。利用基本CSS选择器，我们尝试选择器`ul.myList li a`。不幸的是，这个选择器会抓取所有的链接，因为所有这些链接（`<a>`）都是列表（`<li>`）元素的后代节点。

只要在选择器实验室页面输入选择器`ul.myList li a`，然后点击按钮Apply，就可以验证这一点。结果如图2-3所示。

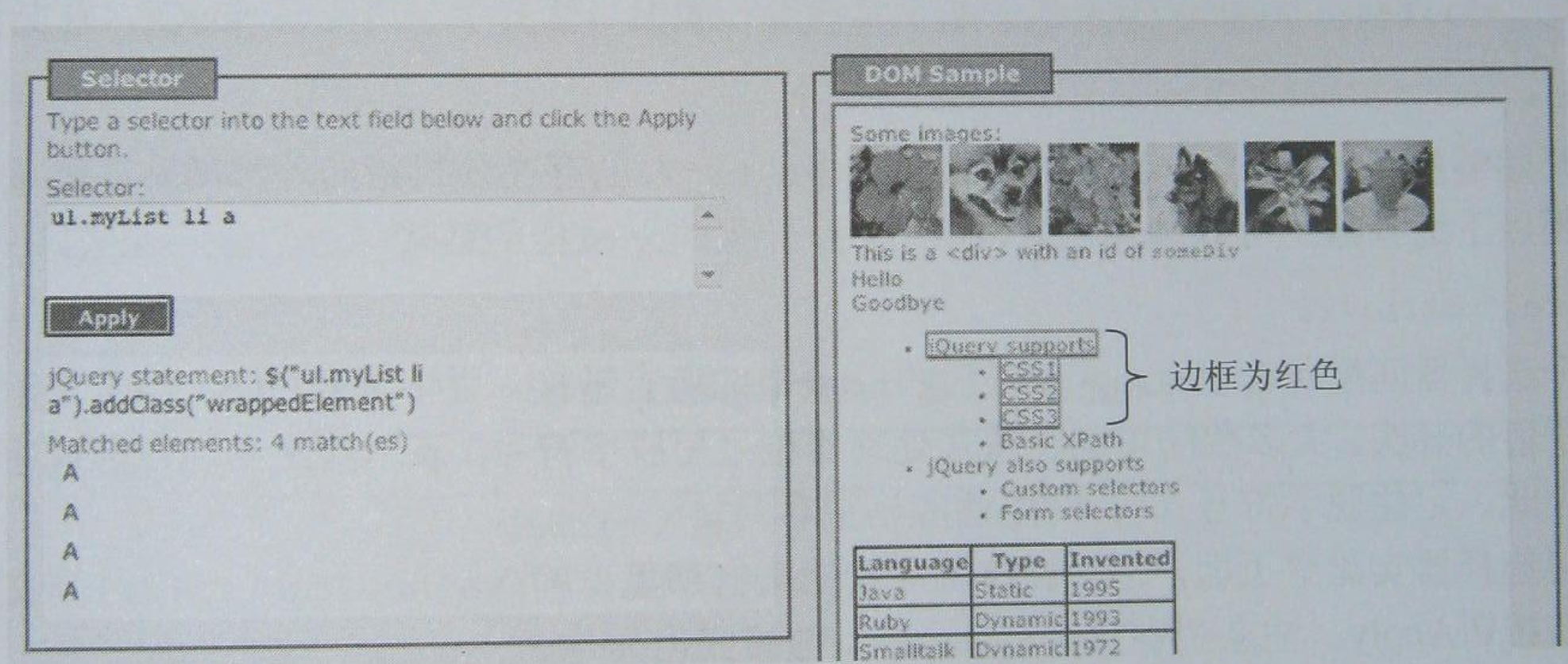


图2-3 所有`<a>`元素，不管所处层次深浅，只要是`<li>`元素的后代节点，都会被`ul.myList li a`选中

更高级的解决办法是利用子选择器：父节点与直接子节点以右尖括号（>）隔开，如下所示：

```
p > a
```

这个选择器只匹配作为`<p>`元素的直接子节点的链接。如果链接被嵌套在更深一层，比如处在`<p>`之内的`<span>`之内，则链接不会被选中。

回到示例上来，考虑如下选择器：

```
ul.myList > li > a
```

这个选择器仅选择作为列表元素的直接子节点的链接，依此类推，列表元素又是拥有CSS类

myList的<ul>元素的直接子节点。排除包含在子列表中的链接，因为充当子列表<li>元素的父节点的<ul>元素，并不拥有CSS类myList。结果如图2-4所示。

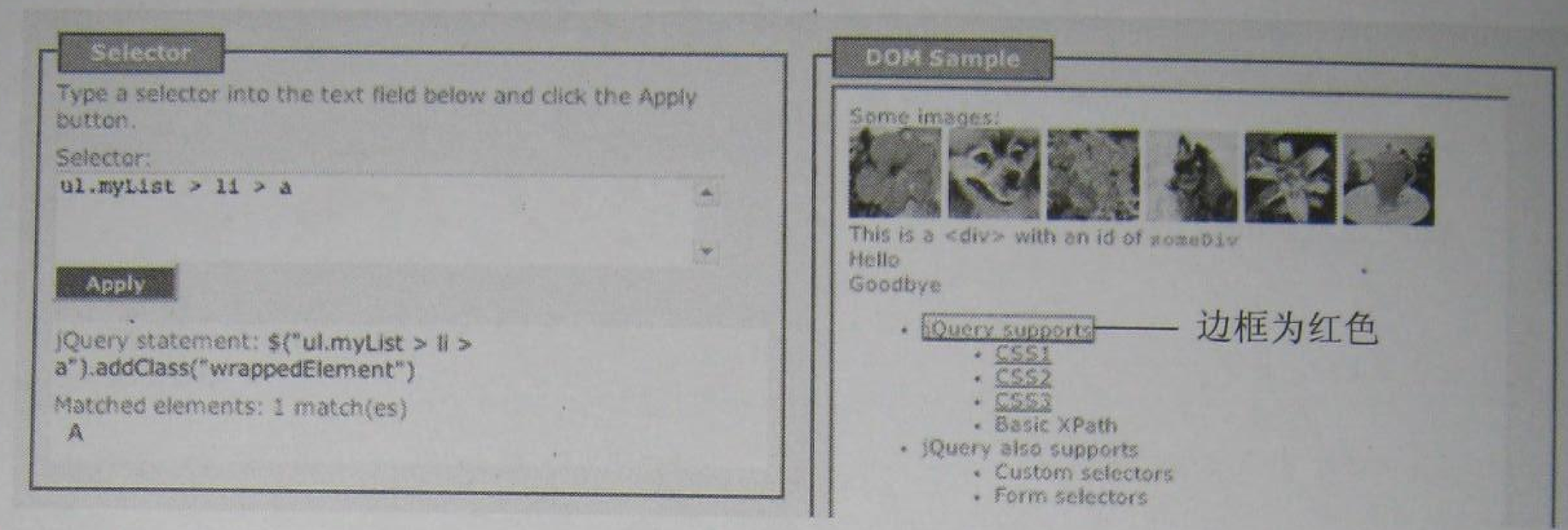


图2-4　利用选择器ul.myList>li>a，只选择父节点的直接子节点来匹配

特性选择器也极为强大。假设我们只想绑定特殊行为到指向本网站之外地址的链接。再次观察前面检查过的实验室示例的部分代码：

```
<li><a href="http://jquery.com">jQuery supports</a>
    <ul>
      <li><a href="css1">CSS1</a></li>
      <li><a href="css2">CSS2</a></li>
      <li><a href="css3">CSS3</a></li>
      <li>Basic XPath</li>
    </ul>
</li>
```

在开始链接（<a>）href特性值时，字符串http://的存在使得指向外部网站的链接与众不同。利用以下选择器，可以选择包含以http://开头的href值的链接：

```
a[href^=http://]
```

这个选择器匹配包含以http://开头的href值的所有链接。脱字符号[1]（^）用于表示匹配出现在某个值的开头。大多数的正则表达式处理器也采用脱字符号，表示匹配出现在候选字符串的开头。因此，记住脱字符号（^）的用法应该很容易。

访问选择器实验室页面，示例HTML片段就摘自那里，输入a[href^=http://]到文本框，然后点击按钮Apply。注意观察只有jQuery网站链接突出显示。

特性选择器还有其他使用方式。如果要匹配拥有指定特性的元素，而不管特性的值是什么，可用：

22

```
form[method]
```

这个选择器匹配拥有显式method特性的任何<form>元素。

为了匹配具体的特性值，可使用如下语法形式：

```
input[type=text]
```

这个选择器匹配type特性值为text的所有<input>元素（即单行的文本字段）。

---

① 脱字符号（^），在文章里插字使用。——译者注

我们已经看到“匹配特性值的开头部分”选择器的实际效果。再看另一个：

```
div[title^=my]
```

这个选择器匹配title特性值以my开头的所有 元素。

那么，“特性值以……结尾”的选择器呢？正要讨论：

```
a[href$=.pdf]
```

如果想要找到引用PDF文件的所有链接，这就是有用的选择器。

还有这样的选择器，用来找到特性值的任何部位中包含某个字符串的元素：

```
a[href*=jquery.com]
```

正如我们所料，这个选择器匹配引用jQuery网站的所有<a>元素。

除特性以外，有时需要选择元素，当且仅当这个元素包含某个其他元素时才选它。在前面的列表示例中，假设想要在包含链接的列表元素中应用某个行为。利用容器选择器，jQuery支持这种选择：

```
li:has(a)
```

这个选择器匹配包含<a>元素的所有<li>元素。请注意，这与li a选择器是不相同的，li a匹配<li>元素里的所有<a>元素。请利用选择器实验室页面，验证这两种形式选择器之间的差异。

表2-1显示了在jQuery里可用的CSS选择器。

请注意，jQuery只支持一层嵌套。比如，支持如下的一层嵌套语句：

```
foo:not(bar:has(baz))
```
①

但是，如果再嵌套一层，比如：

```
foo:not(bar:has(baz:eq(2)))
```

就不支持了②。

表2-1 jQuery所支持的基本CSS选择器

选择器	描　述
*	匹配任何元素
E	匹配标签名称为E的所有元素
E F	匹配标签名称为F、作为E的后代节点的所有元素
E>F	匹配标签名称为F、作为E的直接子节点的所有元素
E+F	匹配前面是邻近兄弟节点E的所有元素F（E和F紧挨着）
E~F	匹配前面是任何兄弟节点E的所有元素F（E和F可以不紧挨着）
E:has(F)	匹配标签名称为E、至少有一个标签名称为F的后代节点的所有元素
E.C	匹配带有类名C的所有元素E。.C等效于*.C
E#I	匹配id特性值为I的元素E。#I等效于*#I③
E[A]	匹配带有特性A的所有元素E（不管特性A的值是什么）

① 例如，div:not(li:has(a))匹配在其后代节点里，不存在包含<a>的<li>的 元素。——译者注

② 从形式上看，是不支持三重括号(…(…(…)…) …)形式的选择器。——译者注

③ 每个元素的id值，应在一个页面内唯一（各不相同）。——译者注

（续）

选择器	描　述
E[A=V]	匹配所有元素E，其特性A的值正好是V
E[A^=V]	匹配所有元素E，其特性A的值以V开头
E[A$=V]	匹配所有元素E，其特性A的值以V结尾
E[A*=V]	匹配所有元素E，其特性A的值包含V

有这些知识在手，现在可以回到选择器实验室页面了。请多花点时间利用表2-1中各种类型的选择器进行试验。尝试进行有目标的选择，比如包含文本Hello或Goodbye的元素（提示：你必须利用选择器的组合才能完成任务[1]）。

到目前为止，似乎已讨论的选择器力量还不足够，还有更多选项赋予我们更强的、筛选页面元素的能力。

2.1.3　通过位置选择

有时候，必须根据元素在页面上的位置或者与其他元素的关系去选择元素。也许你想要选择页面上的第一个链接、奇数或偶数的段落，或者每个列表的最后一项。jQuery支持实现这些具体选择的机制。

24

例如，思考：

```
a:first
```

此格式的选择器匹配页面上的第一个<a>元素。

怎样选择序数为偶数（或奇数）的元素呢？

```
p:odd
```

这个选择器匹配所有序数为奇数的段落元素。当然，也可以选择序数为偶数的元素：

```
p:even
```

另一种形式：

```
li:last-child
```

选择父元素的最后子节点。在本示例中，选择每个<ul>元素的最后<li>子节点。

有很多这样的选择器，有时对于困难的问题，它们能够提供简明得令人惊讶的解决办法。表2-2是这些位置选择器的列表。

表2-2　jQuery支持更高级的位置选择器：根据在DOM里的位置来选择元素

选　择　器	描　述
:first	页面的最先的匹配。li a:first返回最先的、并且在列表（ ）项下的链接
:last	页面的最后的匹配。li a:last返回最后的、并且在列表（ ）项下的链接
:first-child	最先的子元素。li:first-child返回每个列表的最先的项
:last-child	最后的子元素。li:last-child返回每个列表的最后的项

① 例如div[title=myTitle1] span或div[title=myTitle2] span。——译者注

（续）

选择器	描述
:only-child	返回没有兄弟节点的所有元素
:nth-child(n)	第*n*个子节点[1]（*n*从1开始）。li:nth-child(2)返回每个列表的第2个 项
:nth-child(even|odd)	偶数或奇数的子节点。li:nth-child(even)返回每个序列的偶数子节点
:nth-child(Xn+Y)	根据传入的公式计算的第n个子节点。如果Y为0，则忽略Y。*n*从0开始，且X不等于0。li:nth-child(3n)返回3的倍数的项，而li:nth-child(5n+1)返回5的倍数的项的下一项[2]
:even 或 :odd	页面范围内偶数或奇数的匹配元素。li:even返回全部偶数 项
:eq(*n*)	第*n*个匹配元素（*n*从0开始）
:gt(*n*)	第*n*个匹配元素（不包括）之后的元素（*n*从0开始）
:lt(*n*)	第*n*个匹配元素（不包括）之前的元素（*n*从0开始）

选择器nth-child从1开始计数，而其他选择器从0开始计数。为了与CSS兼容，nth-child从1开始计数，但是jQuery自定义的选择器遵循更为常见的从0开始的编程惯例。随着使用次数的增多，记住哪个是哪个渐渐成为第二天性（习惯），但刚开始的时候会有点糊涂。

让我们进一步刨根究底。

思考如下代码，它包含着几种编程语言及其基本信息。

```
<table id="languages">
  <thead>
    <tr>
      <th>Language</th>
      <th>Type</th>
      <th>Invented</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Java</td>
      <td>Static</td>
      <td>1995</td>
    </tr>
    <tr>
      <td>Ruby</td>
      <td>Dynamic</td>
      <td>1993</td>
    </tr>
    <tr>
      <td>Smalltalk</td>
      <td>Dynamic</td>
      <td>1972</td>
    </tr>
    <tr>
```

① 如果*n*等于0，就会导致选择所有元素的意外（异常）结果。——译者注

② 根据Xn+Y计算得到的结果数列中，不包括0。——译者注

```
        <td>C++</td>
        <td>Static</td>
        <td>1983</td>
      </tr>
    </tbody>
  </table>
```

假定想获取包含编程语言名称的所有表格单元格。因为它们都是所在行的第一个单元格，所以可以使用：

```
table#languages tbody td:first-child
```

也可以轻松地使用：

```
table#languages tbody td:nth-child(1)
```

但第一种语法直截了当并且更加简明。

为了获取语言类型的单元格，把选择器改成利用:nth-child(2)，并且利用:nth-child(3)或:last-child获取语言发明的年份。如果确实想要最后的表格单元格（包含文本1983的单元格），就利用选择器td:last。除此之外，选择器td:eq(2)返回包含文本1995的单元格，而td:nth-child(2)返回显示编程语言类型的所有单元格。再次请你记住，:eq从0开始计数，而:nth-child从1开始计数。

在继续前进之前，回到选择器实验室页面，尝试从列表选择第2项和第4项[①]。然后，利用3种不同方式，选择包含文本1972的表格单元格[②]。除此之外，尝试感受nth-child选择器和绝对位置选择器（:eq(n)）之间的差异。

虽然到目前为止我们研究的CSS选择器已经强大得令人不可思议，但我们还要继续讨论怎样从jQuery选择器挤出更大的力量。

2.1.4　利用自定义 jQuery 选择器

CSS选择器赋予我们强大的力量和灵活性去匹配想要的DOM元素，但有时我们想根据CSS规范没有预料到的特征去选择元素。

例如，当我们想要选择用户已选中的所有复选框时。因为试图通过特性匹配的话，将只检查在HTML标记中指定的控件初始状态，所以jQuery提供自定义选择器:checked——从匹配元素集合把处于选中状态的元素筛选出来。例如，尽管选择器input选择所有<input>元素，但是，选择器input:checked只会把搜索范围限定为处于选中状态的<input>元素。自定义选择器:checked运行起来像CSS特性选择器（比如[foo=bar]），因为两者都根据某种标准筛选正在匹配的元素集。这些自定义选择器组合起来以后会很强大，比如选择器:radio:checked和:checkbox:checked。

27

前面讨论过，jQuery支持所有CSS筛选选择器，同时支持几个由jQuery定义的自定义选择器。如表2-3所描述。

---

① 可用选择器li:nth-child(2n)。——译者注

② 例如，选择器td:eq(8)、td:nth-child(3n):eq(2)和td:gt(7):lt(1)。——译者注

表 2-3 jQuery 自定义的筛选选择器赋予标识目标元素的无穷力量

选择器	描述
:animated	选择当前处于动态控制之下的元素。第 5 章讲述动画和特效
:button	选择任何按钮（input[type=submit]、 input[type=reset]、input[type=button]或 button）
:checkbox	只选择复选框元素（input[type=checkbox]）
:checked	只选择已选中的复选框或单选按钮（为 CSS 所支持）
:contains(foo)	只选择包含文本 foo 的元素
:disabled	只选择在界面上已经禁用的表单元素（为 CSS 所支持）
:enabled	只选择在界面上已经启用的表单元素（为 CSS 所支持）
:file	选择所有文件元素（input[type=file]）
:header	只选择标题元素（<hn>，n 代表数字 1 到 6），例如：<h1>、<h2>…<h6>
:hidden	只选择隐藏元素
:image	选择表单图像元素（input[type=image]）
:input	只选择表单元素（<input>、<select>、<textarea>、<button>）
:not(filter)	根据指定的筛选器进行求反
:parent	只选择拥有后代节点（包括文本）的元素，而排除空元素
:password	只选择口令元素（input[type=password]）
:radio	只选择单选按钮元素（input[type=radio]）
:reset	选择复位按钮元素（input[type=reset]或 button[type=reset]）
:selected	选择已选中的选项元素
:submit	选择提交按钮元素（button[type=submit]或 input[type=submit]）
:text	只选择文本字段元素（input[type=text]）
:visible	只选择可见元素

许多自定义jQuery选择器是和表单相关的，允许简明地指定具体的元素类型或状态。也可以组合筛选选择器。例如，假定只想选择已启用的已选中复选框，可以使用

```
:checkbox:checked:enabled
```

利用选择器实验室页面，试验尽可能多的筛选选择器，直到你觉得很好地掌握了它们的操作。

对于可供利用的选择器集合，筛选选择器是极为有用的补充。除此之外，如果反转这些筛选选择器，又怎么样呢？

利用:not筛选器

如果想要对筛选器[1]求反，假定我们匹配任何不是复选框的<input>元素，就采用:not筛选器。它为CSS筛选器所支持，并与自定义的jQuery筛选选择器协同工作。

为了选择非复选框<input>元素，使用

```
input:not(:checkbox)
```

认清筛选选择器和查找选择器之间的差异是非常重要的。筛选选择器，通过对元素应用更高的选择标准（就像前面的示例那样），缩小正在匹配的元素的集合；查找选择器，比如后代选择

① 筛选器，即筛选选择器的简称。下同。——译者注

器（空格符）、子节点选择器（>）以及兄弟节点选择器（+），则查找与已选择元素具有某种关系的其他元素，而不是通过把标准应用于已匹配元素来限制匹配范围。

可以把:not筛选器应用到筛选选择器，但不能应用到查找选择器。

```
div p:not(:hidden)
```

29 这是完全正确的选择器，但div :not(p:hidden)是不正确的。

在第一种情形中，选择作为 元素的后代节点的、处于非隐藏状态的所有<p>元素。第二个选择器是非法的，因为它试图把:not应用到不是筛选器的选择器（在p:hidden中的p不是筛选器）。

为了使事情变得简单，筛选选择器被简单地标识，因为它们都以冒号（:）或者左方括号（[）开头。除筛选选择器以外，任何其他选择器都不能在:not()筛选器里使用。

正如我们所见，jQuery提供庞大的工具集，从页面上选择现有元素，再通过jQuery方法进行操作（第3章会讨论）。在查看操作方法之前，先看如何用$()函数来创建新HTML元素，并包含到匹配集里。

“等一等！”他们说，“还有更多！”

我们强调过，并将继续强调，jQuery的强大之处在于它允许通过插件轻松地进行扩展。如果你能熟练利用XPath（XML Path，XML路径）语言去选择XML文档内元素，你就是幸运儿。一个jQuery插件提供基本的XPath支持，以便和jQuery卓越的CSS选择器和自定义选择器协同使用。你可在http://jquery.com/plugins/project/xpath找到插件。

请记住，虽然对XPath的支持是基本的，但是，应该足以实现强大的选择（与jQuery其他功能相结合）。

首先，插件支持典型的/和//选择器。例如，/html//form/fieldset选择直接处于<form>元素之下的页面上所有<fieldset>元素。

也可利用*选择器来表示任何元素。例如/html//form/*/input，它选择直接处于某一元素之下的所有<input>元素，而该元素又正好处于<form>元素之下。

XPath插件也支持父选择器..（两个圆点），用于选择前面元素选择器[1]的父节点。例如，//div/..匹配作为 元素的直接父节点的所有元素。

XPath插件还支持XPath特性选择器（//div[@foo=bar]），同时支持容器选择器（//div[@p]，选择至少包含一个<p>元素的 元素）。该插件借助于前面所描述的jQuery位置选择器，从而也支持position()。例如position()=0相当于:first，而position()>n
30 相当于:gt(n)。

2.2　生成新HTML

有时候，我们想生成新HTML片段插入到页面里。利用jQuery，这就是小事一桩。因为正如在第1章所见，$函数不仅能够选择现有页面元素，还能够生成HTML。思考：

① “前面元素选择器”，即..符号前面的那一段选择器。例如对//div/..来说，就是//div/。——译者注

```
$("<div>Hello</div>")
```

这个表达式新建 元素以便添加到页面里。在现有元素的包装集上可执行的任何jQuery命令，在新建的HTML片段上同样可以执行。看第一眼也许印象不深，但是当我们把事件处理程序、Ajax以及特效揉合在一起（后面几章就那样做），会看到它如何派上用场。

注意，如果想要创建空 元素，可走捷径：

```
$("<div>")    ← 等同于$("<div></div>")和 $("<div/>")
```

就像对待一生中的许多事情那样，这里有个小警告：利用这个技巧无法可靠地创建<script>元素。不过有许多技巧足以处理这种情况，通常这种情况一开始就使我们想要创建<script>元素。

先体验一下以后将要学会的事情（如果某些地方你目前还不太理解，也不用担心），请看以下代码：

```
$("<div class='foo'>I have foo!</div><div>I don't</div>")
  .filter(".foo").click(function() {
    alert("I'm foo!");
  }).end().appendTo("#someParentDiv");
```

在这个片段中，首先创建了两个 元素，一个带有类foo而另一个没有。然后收窄选择范围，只选择带有类foo的 元素，并给它绑定事件处理程序。一旦点击，就会触发警告对话框。最后，使用end()函数（请参见2.3.6节）还原到包含两个 元素的完整集合，并把这两个元素追加到id为someParentDiv的元素之后，从而把它们绑定到DOM树。

区区一个语句就完成了那么多任务，这的确展示了怎样不写很多脚本而完成很多任务。

在可下载代码chapter2/new.divs.html里，提供了运行本示例的HTML页面。利用浏览器加载文件，显示结果如图2-5所示。

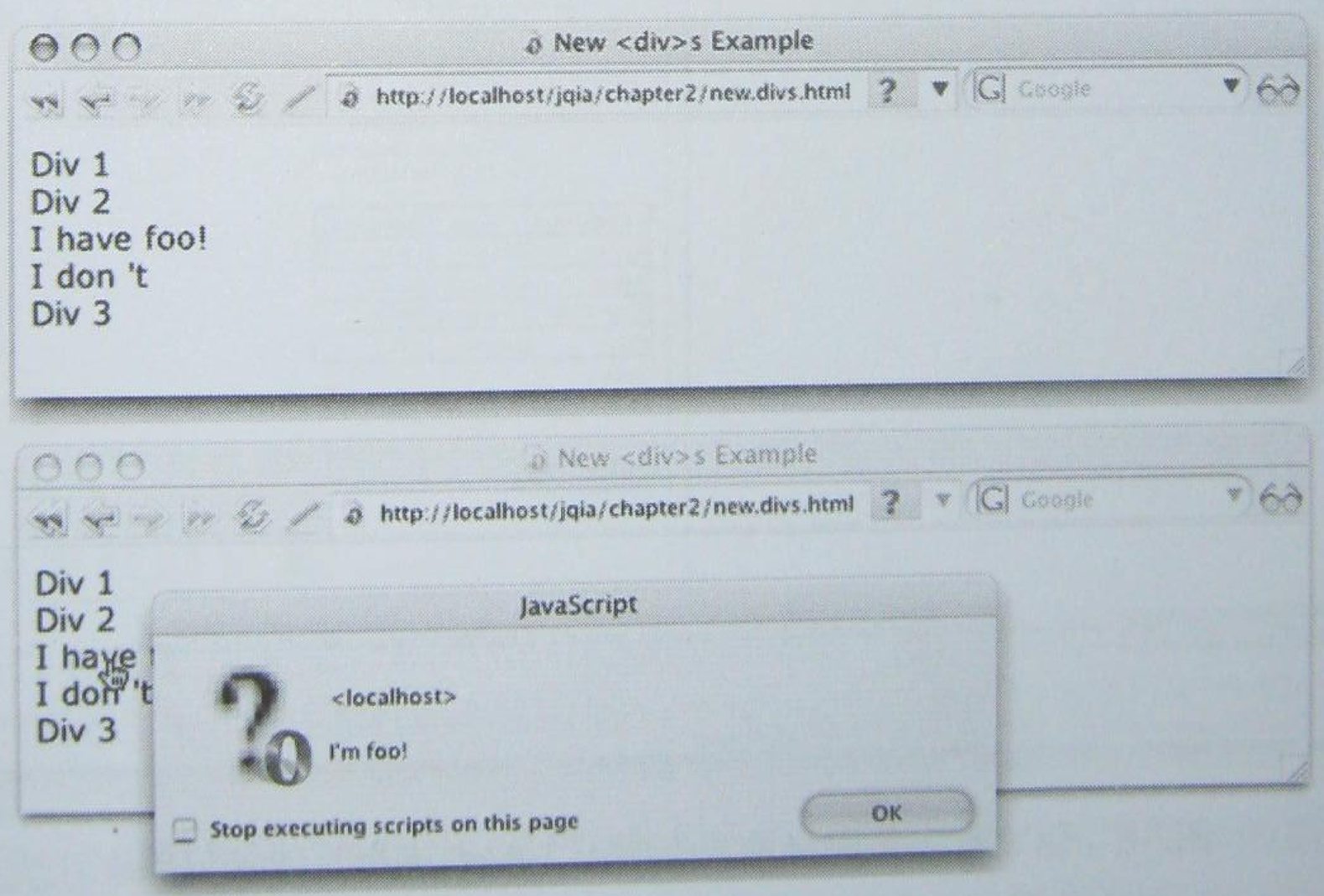


图2-5 仅用一个jQuery语句，就能在脚本控制之下新建HTML元素，并添加高级特性，如事件处理程序

31

在初始化加载时，如图2-5的上半部分所示，新建的 元素被添加到DOM树（因为在页面的就绪处理程序里放置了如上示例代码），正好位于包含文本Div 2的元素（其id为someParentDiv）之后；下半部分则显示，当点击第一个新建的 元素时，就触发了预定义的事件处理程序。

不用太担心，因为还没有讲到完全理解以上示例所需的许多知识点——但很快就会讲到。事实上，我们先要切入“操作包装集”的主题，包括在本示例中用到的filter()命令。

2.3　管理包装元素集合

不管是利用选择器来匹配现有DOM元素，还是利用HTML片段新建元素（或者两者结合），只要获得元素包装集，就可以利用强大的jQuery命令集来操作元素了。下一章开始研究操作元素的命令。但还要准备好什么？如果想进一步精简由jQuery函数包装而得到的元素集呢？

32

在本节里，我们将探索对元素包装集进行精简、扩展或取子集的多种途径，以便进一步操作。

为了尽力帮助你直观地理解知识，我们建立另一个实验室页面，它在本章可下载示例代码里，即包装集实验室页面，可在chapter2/lab.wrapped.set.html找到。该页面看起来很像在本章前半部分看到的选择器实验室页面，如图2-6所示。

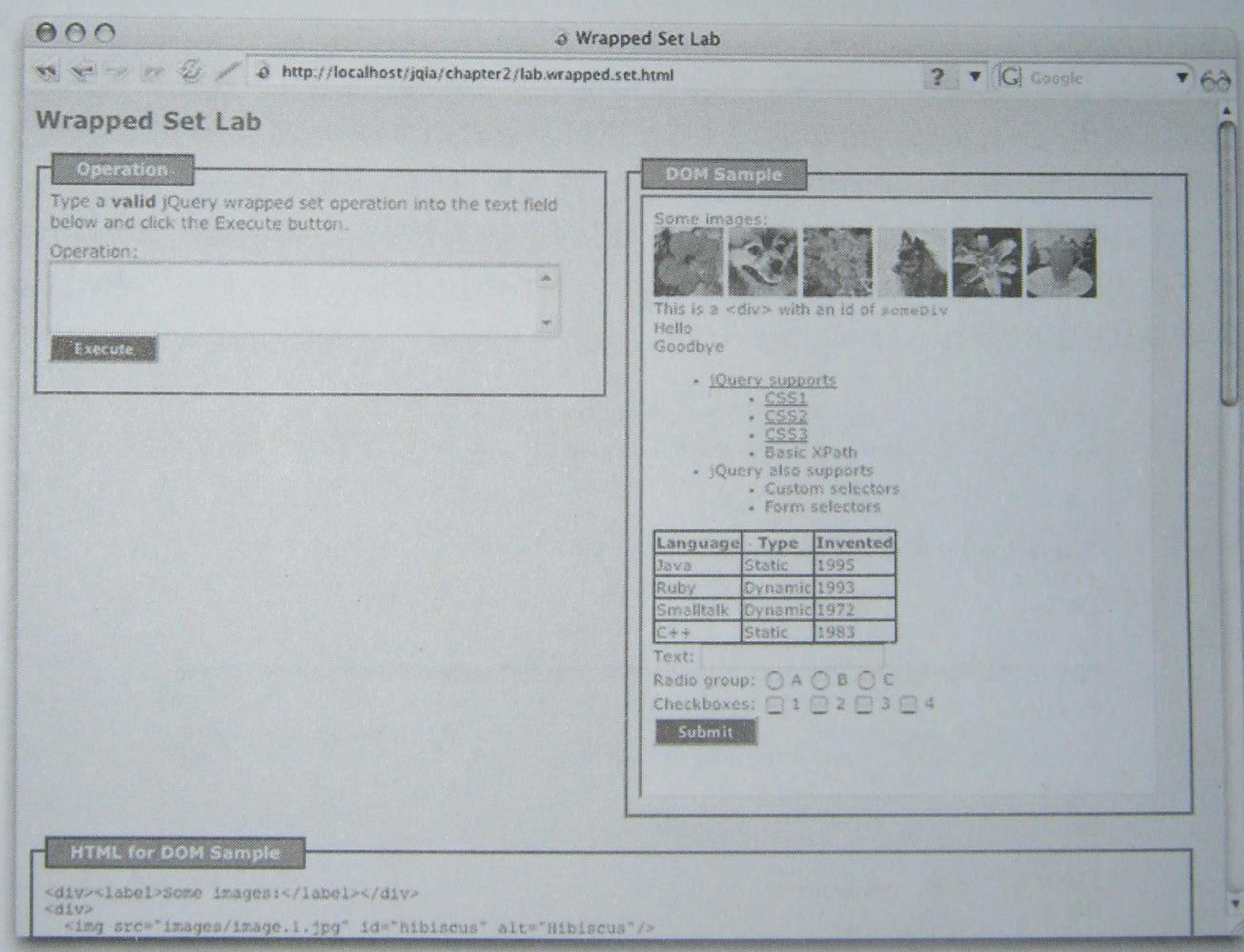


图2-6　包装集实验室协助我们学习怎样创建和管理包装集

包装集实验室页面不仅看起来像选择器实验室页面，而且操作方式也很类似。不同的是，在

包装集实验室页面，不是输入选择器，而是输入完整的jQuery包装集操作代码。操作被应用到DOM Sample，并且像选择器实验室页面那样显示操作结果。

从这种意义上来说，包装集实验室是选择器实验室的通用版。后者只允许输入单个的选择器，而包装集实验室允许输入任何产生jQuery包装集的表达式。多亏jQuery链工作方式，表达式也可包含多个命令，这一切造就了研究jQuery操作的强大实验室。请注意，必须输入语法正确、产生jQuery包装集的表达式。否则，你将面临一堆无用的JavaScript错误。

33

阅读以下各节，看包装集实验室真正发挥的作用。

2.3.1 确定包装集的大小

以前曾提到过，jQuery元素包装集运行起来与数组非常相似：像JavaScript数组那样拥有length属性。而这个属性的值，就是包装元素的个数。

如果想要利用方法而不是length属性，可以利用jQuery定义的size()方法，它同样返回包装元素的个数。

考虑以下语句：

```
$('#someDiv')
  .html('There are '+$('a').size()+' link(s) on this page.');
```

里面的$('a')包装器，匹配所有<a>类型的元素，接着调用size()方法，返回匹配元素的个数。把个数用于拼凑字符串，然后调用html()方法（请参见3.3.1节），把字符串设置成id为someDiv的元素的内容。

size()命令的正式语法如下所示：

命令语法：size

size()

返回包装集里元素的个数。

参数

无

返回

元素的个数

好了，现在知道拥有多少元素了。如果想要直接存取元素，又该怎么做呢？

2.3.2 从包装集获取元素

通常，只要拥有元素包装集，就可使用jQuery命令对它们执行某种操作。例如，调用hide()方法把它们全部隐藏起来。但有时候需要获取一个或一组元素的直接引用，以便对它们执行原始的JavaScript操作。

jQuery允许把包装集当成JavaScript数组进行处理，因此，可以利用简单的数组下标，即通过位置来获取在包装序列里的任何元素。例如，从页面上带有alt特性的所有<img>元素的集合中获取第一个元素，可以这样写：

34

```
$('img[alt]')[0]
```

如果更喜欢使用方法而不是数组下标，可以利用jQuery定义的get()方法来达到目的。

命令语法：get

get(index)

获取包装集里的一个或所有匹配元素。如果不指定参数，包装集里的所有元素就以JavaScript数组形式返回；如果指定了下标参数，就返回下标所对应的元素。

参数

index　(数值)，下标用于指定将被返回的单个元素。如果省略，整个包装集以数组形式返回。

返回

一个DOM元素或DOM元素数组。

如下片段

```
$('img[alt]').get(0)
```

等效于前面利用数组下标的示例，即$('img[alt]')[0]。

get()方法也可用于把元素包装集转化为普通的JavaScript数组，思考：

```
var allLabeledButtons = $('label+button').get();
```

这个语句把页面上前面是近邻<label>元素的所有<button>元素包装到jQuery包装器里，然后创建由那些元素所组成的JavaScript数组，赋值给变量allLabeledButtons。

可以利用逆运算，获取包装集里特定元素的下标。假定因为某个理由，想要知道在页面上整个图像集里id为findMe的图像的顺序下标，可用以下语句获取下标：

35

```
var n = $('img').index($('img#findMe')[0]);
```

index()命令的语法如下所示：

命令语法：index

index(element)

在包装集里查找传入的元素，并返回该元素在包装集里的顺序下标；如果该元素不在包装集里，则返回-1。

参数

element　元素的引用，用于确定元素的下标。

返回

已传入元素在包装集里的顺序下标。如果找不到该元素，则返回-1。

现在，不只是获取元素，怎么着手调整元素包装集呢？

2.3.3 筛选元素包装集

拥有元素包装集之后，你可能想添加元素扩大集合，或者把原始匹配的元素包装集缩小为它的子集。jQuery提供了一整套管理元素包装集的方法。首先，看看如何添加元素到包装集。

1. 添加更多元素到包装集

我们经常需要添加更多元素到现有包装集。某个命令应用到原始包装集之后，想要添加更多元素时，这种能力尤为有用。记住，jQuery链使其能够在单个语句里执行大量任务。

不过首先，研究一个简单的情形。假定想要匹配带有alt或title特性的所有<img>元素。强大的jQuery选择器允许在单个选择器里实现这个功能，比如

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

但为了说明add()方法的操作过程，采用如下语句匹配相同的元素集合：

```
$('img[alt]').add('img[title]')
```

以这种方式来调用add()方法，允许把多个选择器链在一起形成“或”关系。因此，示例代码创建两个包装集的并集，每个包装集分别由各自选择器匹配得来。像add()这样的方法，也可用于替代选择器。例如end()方法（请参见2.3.6节）可用于删除通过add()方法所添加的元素。

36

命令语法：add

add(expression)

把表达式参数所指定的元素添加到包装集。表达式可以是选择器、HTML片段、DOM元素或DOM元素数组。

参数

expression （字符串|元素|数组）指定添加到包装集的元素。参数如果是jQuery选择器，则全部匹配元素都被添加到集合；如果是HTML片段，则创建适当的元素并添加到集合；如果是DOM元素或DOM元素数组，则直接添加到集合。

返回

包装集

用浏览器打开包装集实验室页面，输入如上示例（要和书上的代码一样），然后点击按钮Execute。页面执行jQuery操作，选中带有alt或title特性的所有图像。

观察DOM Sample的HTML源代码，可见所有花朵图像都拥有alt特性，小狗图像都拥有title特性，而咖啡壶图像则哪个特性都没有。因此，应该料到除咖啡壶以外，所有图像都将成为包装集的一部分。图2-7展示了页面结果相关部分的屏幕截图。

37

可以看到，6个图像中的5个（除咖啡壶以外）添加到包装集了（在本书的印刷版本里，由于是灰度图像，红色边框可能有点难以辨认）。

现在，观察add()方法的更为现实的用途。假定想要把粗边框应用到带有alt特性的所有<img>元素，然后把一定级别的透明度应用到带有alt或title特性的所有<img>元素。因为我们想要把操作应用到包装集，然后添加更多元素到包装集，所以这回CSS选择器的逗号运算符（,）

帮不上忙。利用多个语句可以轻松完成任务，但利用jQuery链的力量在单个语句中完成任务，更加高效和简单，比如

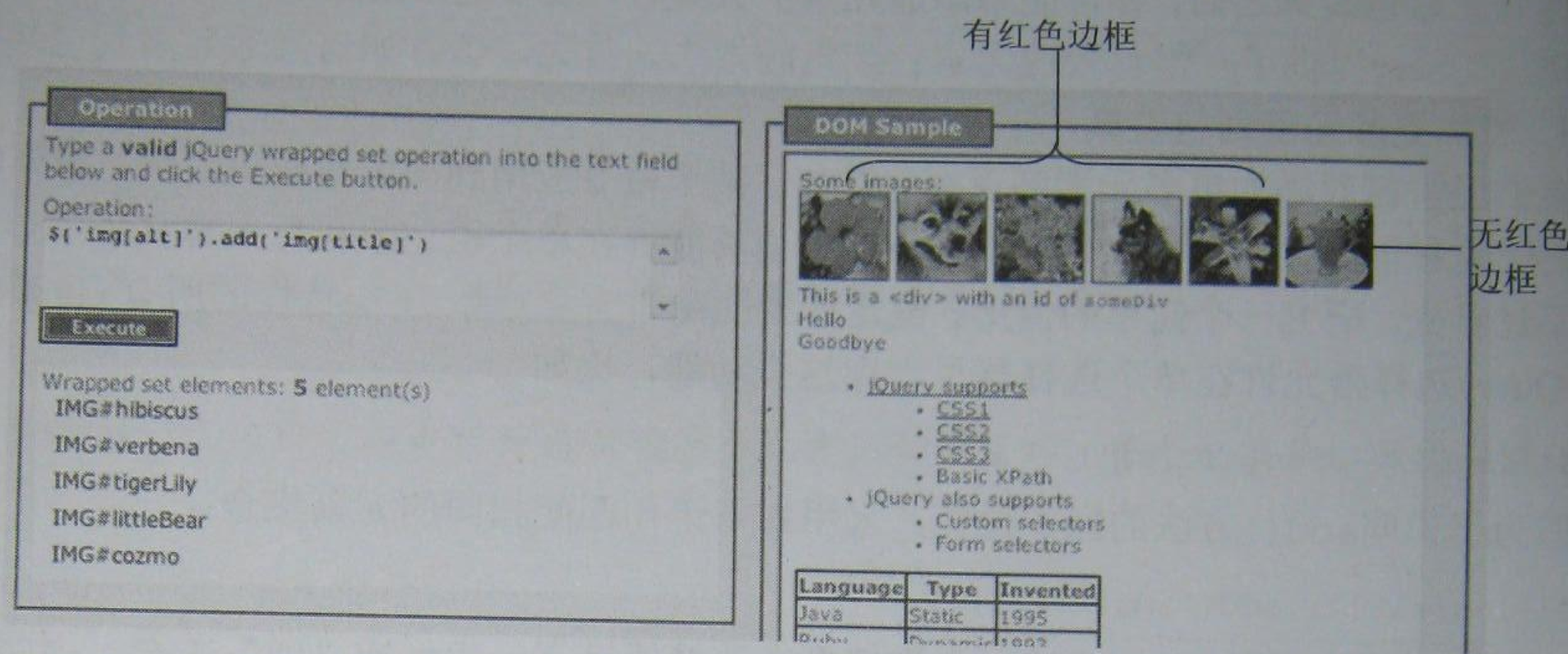


图2-7　不出所料，jQuery表达式匹配带有alt或title特性的图像元素

```
$('img[alt]').addClass('thickBorder').add('img[title]')
➡ .addClass('seeThrough')
```

这个语句首先创建带有alt特性的所有<img>元素的包装集，接着应用预定义的类thickBorder实现粗边框，然后添加带有title特性的所有<img>元素，最后应用类seeThrough到新扩大的包装集，以便实现透明效果。

输入如上语句到包装集实验室页面（已经预定义指定的CSS类），可看到如图2-8所示结果。

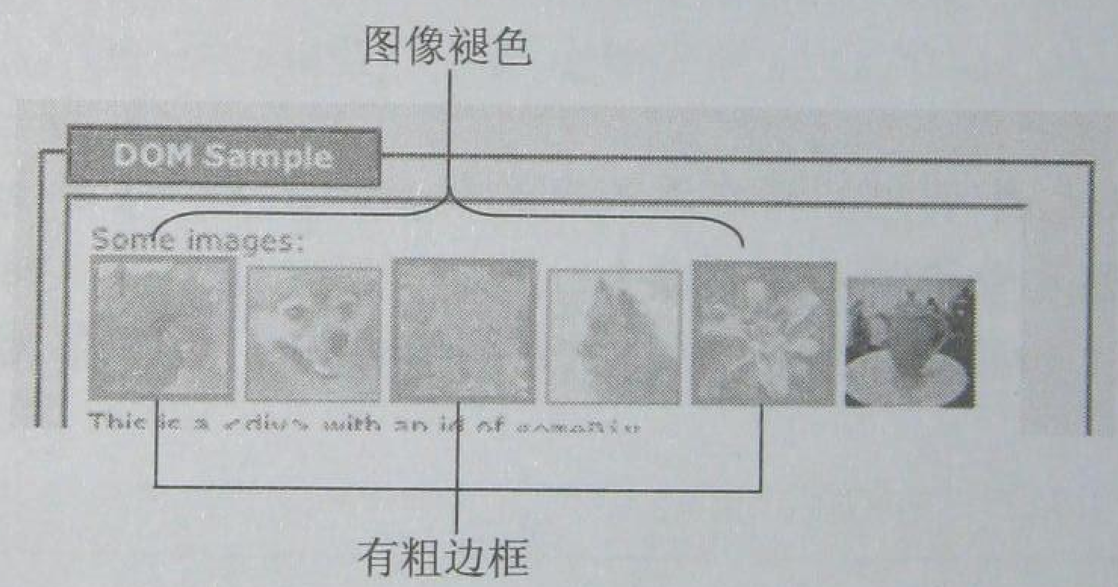


图2-8　通过jQuery链在单个语句里执行复杂的操作的结果

可以看到结果是，花朵图像（带有alt）拥有粗边框，而除咖啡壶（既没有alt又没有title的唯一图像）以外，所有的图像都褪色了，这是由于应用了CSS的不透明度规则。

如果传入元素的引用，add()方法也可用于添加元素到现有包装集。只要把一个元素引用或元素引用的数组传递给add()方法，就可以添加元素到包装集。假定有一个元素引用，命名为someElement变量，则可以把元素添加到带有alt属性的所有图像集中：


```
$('img[alt]').add(someElement)
```

那似乎不够灵活，add()方法不仅允许把现有元素添加到包装集，还能通过接受传入的包含HTML标记的字符串而添加新元素。考虑以下语句：

```
$('p').add('<div>Hi there!</div>')
```

这个语句首先创建文档里所有<p>元素的包装集，然后新建 元素并添加到包装集。请注意，这样做只是添加新元素到包装集，而在语句里并没有调用添加新元素到DOM的方法。这时候，可以调用jQuery的append()方法（请保持耐心，很快就讲到这样的方法，也可参见3.3.2节）添加所选择的元素或新建的HTML到DOM的某个部分。

调用add()方法来扩大包装集，简单而有力。下面观察从包装集删除元素的jQuery方法。

2. 整理包装集的内容

正如我们所见，在jQuery里调用add()方法，把多个选择器链在一起形成“或”关系[1]，从而创建包装集，是简单的事情。除此之外，通过调用not()方法把选择器链在一起形成“与非”关系，也是可以的。这与前面讨论过的:not选择器有相似之处。然而，调用方式却与add()方法相似，即在jQuery命令链的任何一环，从包装集里删除元素。

假定想要选择页面里带有title特性的所有<img>元素，除title特性值包含文本puppy的元素以外。可以想到能够表达这个条件的一个选择器，即img[title]:not([title*=puppy])，但为了举例说明，假定忘记了:not筛选器。通过调用not()方法，从包装集里删除任何与传入选择器表达式相匹配的元素，从而表达“与非”关系类型。为了进行上述匹配，可以编写：

```
$('img[title]').not('[title*=puppy]')
```

在包装器实验室页面输入这个表达式并执行。你将会看到只选中棕褐色的小狗图像。原始包装集包含带有title特性的黑色小狗图像，调用not()方法后被删除了，因为其title包含文本puppy。

请注意，传递给not()方法的选择器仅限于删除任何元素引用的筛选表达式（允许它暗示所有元素类型）。如果错误地传递了更加显式的选择器img[title*=puppy]给not()方法，就无法得到预期结果，因为not()方法不支持元素选择器[2]。

命令语法：not

not(expression)

根据表达式参数的值，从包装集里删除元素。如果参数是jQuery筛选选择器，则从包装集里删除任何匹配表达式的元素；如果参数是元素引用，则从包装集里删除该元素。

参数

expression （字符串|元素|数组）jQuery筛选器表达式、元素引用或元素引用的数组，定义从包装集里删除的元素。

返回

包装集

---

① “或”关系是两个包装集的并集，而“与非”关系是两个包装集的差集。——译者注

② 元素选择器又叫查找选择器，特征是包含具体元素类型的名称。比如，img[title*=puppy]的img；筛选选择器的特征以[（左方括号）或:（冒号）开头，请参见2.1.4节。——译者注

和add()方法一样，如果传入元素引用或元素引用的数组，not()方法也可用于从包装集里删除个别元素。传入元素引用的数组，显得好玩而又强大，因为任何jQuery包装集都能够当成元素引用的数组来使用（请记住这一点）。

有时候，筛选包装集的方式难以或不可能用选择器表达式表达出来。在这种情况下，必须求助于编程方式去筛选包装集里的元素。迭代包装集的所有元素，调用not(element)方法删除不符合选择标准的元素。但是，jQuery团队不想让用户被迫亲自做这一切工作，所以定义了filter()方法。

filter()方法，当传给它一个函数，它会为每个包装集元素反复调用那个函数，如果哪个元素的函数调用返回false值，就删除哪个元素。利用筛选函数体的函数上下文（即this），每次调用都能存取当前包装集元素。

例如，假定出于某种理由，想要创建包含数字值的所有<td>元素的包装集。即使和jQuery的选择器表达式一样强大，也不可能利用选择器表达式表达上述要求。在这种情况下，filter()方法就派上用场了，如下所示：

```
$('td').filter(function(){return this.innerHTML.match(/^\d+$/)})
```

这个jQuery表达式首先创建所有<td>元素的包装集，然后为每一个包装集元素各调用一次传递给filter()方法的函数，并且将当前包装集元素作为当次调用的this值。函数利用正则表达式确定元素内容是否匹配已描述的模式（一个或多个数字所组成的序列），如果不匹配，则返回false。筛选函数调用返回false的任何元素，都会从包装集里删除。

40

命令语法：filter

filter(expression)

利用传入的选择器表达式或筛选函数，从包装集里筛选元素。

参数

expression　（字符串|函数）如果参数是字符串，则指定jQuery选择器，用于从包装集里删除所有与选择器不匹配的元素，也就是说，留下与选择器匹配的元素；如果参数是一个函数，则用于确定筛选条件。为包装集里的每一个元素各调用一次该函数，以当前元素作为当次调用的函数上下文（this）。函数调用返回值为false的任何元素都会从包装集里删除。

返回

包装集

再次打开包装集实验室页面，输入如上表达式并执行。你将会看到只有Invented那一列表格单元格，是最终选中的<td>元素。

也可以在调用filter()方法时，传入选择器表达式，但是，选择器表达式必须遵守前面已说明的和not()方法一样的约束，也就是说，必须是带有隐式元素类型的筛选选择器。以这种方式调用时，filter()方法和相应的not()方法的操作方式刚好相反——删除和传入选择器不匹配的任何元素（not()方法则是删除和传入选择器匹配的任何元素）。这不是强大的方法，因为如

果最初采用更富限制性的选择器，通常会容易些。然而，在jQuery的命令链里，filter()方法派上了用场。例如，考虑

```
$('img').addClass('seeThrough').filter('[title*=dog]')
➡ .addClass('thickBorder')
```

这个命令链语句选择所有图像，并对所有图像应用seeThrough类，然后缩小集合，只留下title特性包含文本dog的图像元素，接着对留下的元素应用另一个名为thickBorder的类。最终结果是，所有图像变得半透明，却只有棕褐色小狗的图像被粗边框围了起来。

not()和filter()方法给予我们强大的手段，以便根据包装元素各个方面的任何标准，即时调整元素包装集。还可以根据元素在集合里的位置，获取包装集的子集。下面看哪个方法有此能耐。

3. 获取包装集的子集

有时，希望根据元素在包装集里的位置，获取包装集的子集。为此，jQuery提供名为slice()的方法，这个命令创建并返回来自原始包装集的任何连续部分的新包装集，或把它叫做“原始包装集的切片（slice）”。此命令的语法如下所示。

命令语法：slice

slice(begin,end)

创建并返回新包装集，新包装集包含原始包装集的连续的一部分。

参数

begin　（数字）将被包含在返回切片中的第一个元素的下标（从0开始）。

end　（数字，可选）不被包含在返回切片中的第一个元素的下标（从0开始）；或一个数字，该数字超出了原始包装集的最后一个元素的下标。如果省略，则切片延伸到原始包装集的末尾。

返回

新建的包装集

如果想根据原始包装集里的元素地址，从一个包装集获取包含单个元素的另一个包装集，就可以调用slice()方法，将包装集里该元素的下标（从0开始）传递给它。例如，如果想获取第3个元素，可用：

```
$('*').slice(2,3);
```

这个语句选择页面上的所有元素，然后生成包含原始包装集的第3个元素的新包装集。

注意这与$('*').get(2)不同，后者返回包装集里的第3个元素（请参见2.3.2节），而不是包含元素的包装集。

因此，下面这样的语句：

```
$('*').slice(0,4);
```

选择页面上所有元素，然后创建包含前4个元素的新包装集。

为了从包装集的后半部获取元素，如下语句：

42

```
$('*').slice(4);
```

匹配页面上所有元素，然后返回包含（除前4个元素以外）所有元素的新包装集。

还没完呢！利用jQuery我们还可以根据元素与DOM中其他元素的关系来获取包装集子集。请往下阅读。

2.3.4　利用关系获取包装集

jQuery允许根据HTML DOM中包装元素与其他元素的层次关系，从现有包装集里获取新包装集。请注意，2.3节前面大部分的方法会修改调用它们的包装集。而本小节里的方法稍微不同。就像slice()方法一样，本节里的方法返回新包装集，而原始包装集保持不变。

表2-4显示这些方法及其描述。其中每个方法都接受可选的选择器表达式，任何被选择元素必须与其匹配。如果没有传入选择器参数，就选择所有合格元素。

根据和其他DOM元素的关系，从DOM选择元素。在这一点上，这些方法给予我们很大程度上的自由，但事情还没完。下面看jQuery如何进一步处理包装集。

表2-4　根据关系来获取新包装集的方法

方　法	描　述
children()	返回原始包装集元素的所有唯一[1]子元素所组成的包装集
contents()	返回原始包装集元素的内容的包装集，这些元素可能包含文本节点（这个方法经常用于获取<iframe>元素的内容）
next()	返回原始包装集元素的所有唯一的下一个兄弟元素所组成的包装集
nextAll()	返回包含原始包装集元素的所有后续兄弟元素的包装集
parent()	返回原始包装集所有元素的唯一直接父元素所组成的包装集
parents()	返回原始包装集所有元素的唯一祖先元素所组成的包装集，包括直接父元素和一直向上追溯的祖先元素，但不包括文档根元素
prev()	返回原始包装集元素的所有唯一的上一个兄弟元素所组成的包装集
prevAll()	返回包含原始包装集元素的所有前面兄弟元素的包装集
siblings()	返回原始包装集元素的所有唯一兄弟元素所组成的包装集

43

除contents()以外，表2-4里的所有方法都接受包含字符串的参数，用于对结果进行筛选。

2.3.5　还有更多使用包装集的途径

似乎所有这一切还不够，jQuery还有几个窍门让我们定义包装对象的集合。

find()方法可以把现有包装集仔细搜索一遍，并返回包含所有匹配传入选择器表达式的元素的新包装集。例如，假设有包装集变量wrappedSet，我们可用如下语句，获取段落内所有引文（<cite>元素）的新包装集：

```
wrappedSet.find('p cite')
```

请注意，如果这是发生在单个语句里的一切，也可通过给jQuery选择器传入上下文参数来完成：

① 在这个表中的“唯一”，是指在集合中“各不相同”，而不是指“一个”。下同。——译者注

```
$('p cite',wrappedSet)
```

像许多别的jQuery包装集方法那样，当在jQuery操作链内调用时，find()方法的力量才爆发出来。

命令语法：find

find(selector)

返回新包装集，包含原始包装集里与传入选择器表达式相匹配的所有元素。注意：原始包装集里的元素的后代，会因为与传入的选择器表达式相匹配而被包含在新包装集里。

参数

selector （字符串）一个jQuery选择器，元素必须匹配这个选择器，才能成为新包装集的一部分。

返回

新建的包装集

除在包装集里查找匹配选择器的元素以外，jQuery也提供方法查找包含指定字符串的元素。contains()方法[1]返回新包装集，由元素体的内容中包含传入字符串的所有元素所组成。思考

```
$('p').contains('Lorem ipsum')
```

这个表达式产生包装集，包含所有包含文本Lorem ipsum的段落。请注意，字符串测试应用到元素体的内容的各个方面，包括标记和后代元素的特性值，但是，不匹配受测原始元素的标记或特性值。 44

命令语法：contains

contains(text)

返回新包装集，由包含text参数所传入的文本字符串的元素所组成。

参数

text （字符串）添加到新包装集里的元素必须包含的文本。

返回

新建的包装集

本节最后一个方法允许对包装集进行测试，看是否至少包含一个匹配给定选择器的元素。如果至少有一个元素匹配选择器，is()方法就返回true，否则返回false。例如：

```
var hasImage = $('*').is('img');
```

如果当前页面包含图像元素，这个语句就把变量hasImage的值设为true。

---

① contains()方法在jQuery 1.2中已被移除。以下语句将添加与前版本的contains()方法功能相似的方法：jQuery.fn.contains=function(text){return$(this).filter(":contains("+text+")");}。

命令语法：is

is(selector)

确定包装集里是否有元素匹配传入的选择器表达式。

参数

selector　（字符串）选择器表达式，用于测试包装集的元素。

返回

如果至少有一个元素匹配传入的选择器，则返回true，否则返回false。

2.3.6　管理 jQuery 链

对于在单个语句里把jQuery包装器方法链起来完成大量任务的能力，我们一直极为重视（并将继续重视，因为它太重要了）。这种链能力不但允许以简洁的方式写出强大的操作，而且提高了效率，因为它能够把多个命令应用到包装集，而不必重新计算包装集。

利用命令链里调用的方法，可能生成多个包装集。例如，调用clone()方法（3.3.5节将会详述）生成新包装集，即创建原始包装集元素的副本。一旦生成新包装集，就无法引用原始包装集，因此，我们就被剥夺了构造多功能jQuery命令链的能力。

45

思考以下语句：

```
$('img').clone().appendTo('#somewhere');
```

这个语句内产生两个包装集：一个是页面上所有<img>元素的原始包装集，另一个是由原始包装集元素的副本所组成的新包装集。clone()方法返回新包装集作为结果，而appendTo()命令正是在这个新包装集上进行操作的。

那么，在克隆原始包装集后，如果我们想接着应用一个命令，比如添加CSS类名到原始包装集，该怎么办呢？不可以添加到现有命令链的末端，因为那样做的话，将会影响克隆得到的新包装集元素，而不是原始包装集的图像元素。

为了满足这个需求，jQuery提供了end()命令。在jQuery链内，一调用这个方法，方法就会作为返回值回退到前一个包装集，因此，后续操作将应用到前一个包装集。

考虑以下语句：

```
$('img').clone().appendTo('#somewhere').end().addClass('beenCloned');
```

appendTo()方法返回克隆得到的新包装集，只要调用end()就回退到前一个包装集（原始图像），并在其上执行addClass()命令。如果不插入end()命令，addClass()就操作克隆得到的新包装集。

命令语法：end

end()

在jQuery命令链内调用，以便回退到前一个包装集。

参数
无

返回
前一个包装集

也许这让人联想到，jQuery命令链在执行期间所产生的包装集保存在栈内。当调用end()时，最顶端的（最近产生的）包装集出栈，公开前一个包装集，让后续命令进行操作。

另一个修改包装集栈的便利的jQuery方法是andSelf()。这个方法把栈内最顶端的两个包装集合并为一个包装集。

命令语法：andSelf

andSelf()

合并命令链内最近产生的两个包装集[1]。

参数
无

返回
合并后的包装集

2.4 小结

本章重点介绍通过jQuery所提供的多种方法，标识HTML页面上的元素，用于创建和调整元素的集合（从本章往后，作为包装集引用）。

jQuery提供多功能的、强大的、模仿CSS选择器的一组选择器，用于在页面文档内以简洁强大的语法标识元素。这些选择器不仅包括当前大多数浏览器所支持的CSS2语法，还包括CSS3语法。jQuery同时提供几个自定义选择器、插件，甚至一些基本XPath选择器。

jQuery也允许利用HTML片段即时新建元素，用于创建或扩大包装集。这些孤立元素可以连同包装集里的任何其他元素一起进行操作，并最终附加到页面文档的几个部分。

jQuery提供一套调整包装集的健壮的方法，去筛选包装集的内容，要么在创建包装集后立即进行，要么在执行一组链接命令的中途进行。将筛选标准应用到现有包装集，也可以轻松地创建新包装集。

总而言之，jQuery提供大量的工具，确保能够轻松而精确地标识我们想要操作的页面元素。

本章涵盖大量的基础知识点，但不涉及对页面DOM元素的操作。知道如何选择想要操作的元素，也就可以利用jQuery命令的力量来给页面增添活力了。

① 即把当前包装集与前一个包装集，合并为一个包装集。——译者注

第3章 用jQuery让页面生动起来

本章内容

- 获取和设置元素特性
- 操作元素类名称
- 设置元素内容
- 处理表单元素值
- 修改DOM树

记得那些日子吗（不久以前）？那时初出茅庐的页面作者试图给页面增加引人注目的东西却招来反感，比如滚动字幕、闪烁文本、影响页面文本可读性的过分花哨的背景花样、恼人的动态GIF图像，也许最糟糕的是，页面加载后不请自播的背景声音。（用来测试用户关闭浏览器的速度有多快？）

从那儿以后我们走了漫长的路。

如今聪明的Web开发者和设计者有了更深的认识，并利用DHTML（Dynamic HyperText Markup Language，动态超文本标记语言）的力量来改善用户Web体验，而不是显摆恼人的花招。

不管是逐渐地显示内容、创建优于HTML所提供的基本控件集的输入控件，还是给予用户随其所好地调整页面的能力，DHTML或DOM操作已经让许多Web开发者为用户带来了惊喜（而不是烦扰）。

几乎每天我们都会不期而遇地看到某些网页，其表现让我们惊叹："哇！我还不知道页面能做那些事情！"作为称职的专业人士（更因为对这些事情的无限好奇心），我们立即开始研究源代码来查明那是怎样实现的。

但并不是非得亲自编写所有脚本，我们发现jQuery提供了一套健壮的工具用于操作DOM，只用极少量的代码就使那种令人赞叹的页面成为现实。前面一章介绍了jQuery提供的多种方式用于选择DOM元素来形成包装集，而本章则发挥jQuery的力量在包装集上执行操作，给页面带来动感与活力，并注入令人惊叹的因素。

3.1 操作元素属性和特性

提到DOM元素，我们可以操作的最基本部分是指派到元素的属性和特性。这些属性和特性作为解析页面HTML标记的结果，初始化时被指派到DOM元素，并且可以在脚本控制下动态地修

改。

为了确保我们正确地理解术语和概念，考虑以下图像元素的HTML标记。

```
<img id="myImage" src="image.gif" alt="An image" class="someClass"
➡ title="This is an image"/>
```

在这个元素标记里，标签名称为img，而标记id、src、alt、class和title表示元素的特性，每个特性均由名称和值所组成。浏览器读取并解释这个元素标记，创建在DOM里表示这个元素的JavaScript对象。除存储特性以外，这个对象还拥有多个属性，包括一些表示标记特性值的属性（甚至有一个属性用于保持特性列表）。图3-1显示了这个过程的简化概览。

49

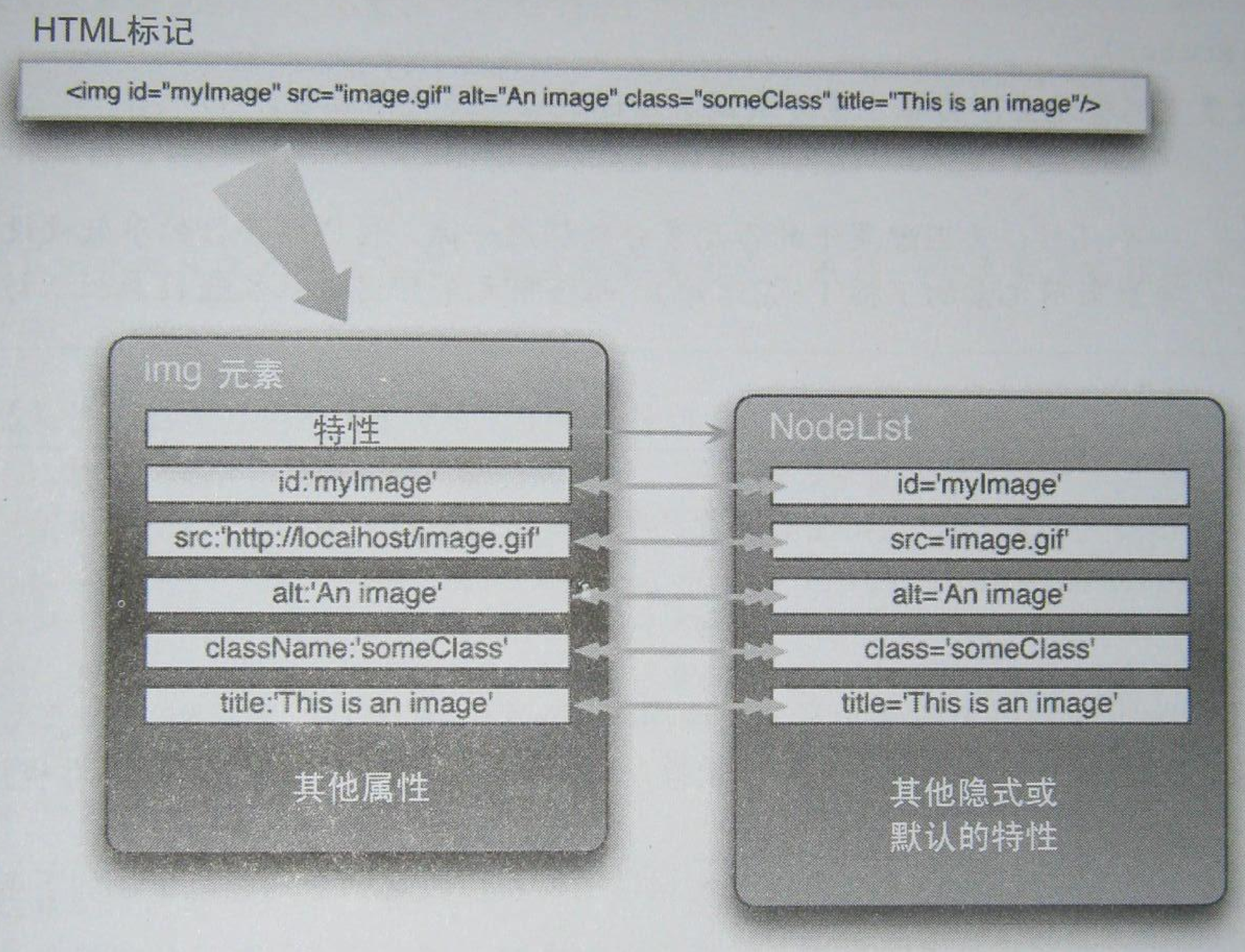


图3-1 HTML标记被解释为DOM元素，包含<img>标签里的特性和根据特性创建的属性

HTML标记由浏览器解释成为表示图像的DOM元素，创建节点列表对象NodeList（DOM定义的容器类型之一），并且指派给名为attributes的元素属性。特性和它的对应属性（我们将称它为特性属性[1]）之间存在动态关联。修改一个特性会导致对应的特性属性发生变化，反之亦然。虽说这样，值却并不总是一致。例如，设置图像元素的src特性为image.gif，将导致src属性被设置为图像的绝对URL（Uniform Resource Locator，统一资源定位符）。

50

从大体上来说，JavaScript特性属性的名称与对应的特性一一匹配，但在某些情况下并不一致。例如，本示例中的class特性被表示为className特性属性。

通过jQuery可以轻松地操作元素特性和存取元素，所以也能够修改元素属性。操作特性还是操作属性，取决于想做什么以及想如何去实现。

① 特性属性，是指DOM元素中能够和HTML元素中某个特性对应得上的属性。——译者注

先看如何获取和设置元素属性。

3.1.1　操作元素属性

jQuery没有用于获取或修改元素属性的具体命令。然而可以利用原生JavaScript表示法，存取属性和属性值。诀窍在于首先获取元素引用。

查看或修改包装集组成元素的最简便办法是利用each()命令。这个命令语法如下。

命令语法：each

each(iterator)

遍历包装集里所有元素，为各元素分别调用传递进来的迭代器函数。

参数

iterator　一个函数，为匹配集中的各元素分别调用一次。传递到函数的参数被设置为包装集里当前元素的下标（从0开始），而当前元素可通过函数this属性来访问。

返回

包装集

通过这个命令可以轻松地把属性值设置到匹配集里的所有元素上。例如

```
$('img').each(function(n){
  this.alt='This is image['+n+'] with an id of '+this.id;
});
```

这个语句为页面上每个图像元素分别调用内联函数，利用元素下标和id值修改元素alt属性。请注意，因为这是特性属性，而特性属性与同名特性动态关联，所以alt特性也被间接地更新。

51

类似地，可以利用each()方法来将特定属性的所有值收集到一个数组里，如下所示：

```
var allAlts = new Array();
$('img').each(function(){
  allAlts.push(this.alt);
});
```

如果只想获取单个元素的属性值（请记住匹配集可以当成JavaScript数组来处理），就可以通过如下语句获取属性：

```
var altValue = $('#myImage')[0].alt;
```

与处理属性相比，用JavaScript处理特性不是那么直接，所以jQuery提供处理特性的辅助手段。请阅读下一节。

3.1.2　获取特性值

就像许多其他jQuery命令那样，attr()命令也可以用来进行读操作或写操作。相同的jQuery命令能够执行这样迥异的操作，是因为传给命令的参数的个数和类型决定了采用命令的哪个重载

变体[1]。

attr()命令可以用于从匹配集里第一个元素获取特性值，或将特性值设置到所有已匹配元素。

进行读操作的attr()命令的重载变体语法如下。

命令语法：attr

attr(name)

获取指派到包装集里第一个元素指定特性的值。

参数

name （字符串）特性的名称。该特性的值将被获取。

返回

包装集里第一个元素指定特性的值。如果包装集为空，或第一个元素没有该特性，就返回undefined。

即使通常认为特性是由HTML预定义的，但是对于通过JavaScript或HTML标记设置的自定义特性，也可以利用attr()方法。为了说明这一点，我们修改前面示例的<img>元素，增加自定义特性标记（用粗体突出显示）：


```
<img id="myImage" src="image.gif" alt="An image" class="someClass"
➥ title="This is an image" custom="some value"/>
```

请注意，我们已经添加自定义特性，并缺乏想象力地命名为custom。如下语句可以获取custom特性值，就像任何标准特性那样：

```
$("#myImage").attr("custom")
```

警告 使用非标准的特性名称（如custom）虽然是常见手法，却会导致标记被认为是非法的——它会导致有效性测试失败。这可能影响可存取性，还可能影响程序解析，因为那些程序指望你的网站是用有效的HTML或XHTML编写的。

在HTML里，特性名称是不区分大小写的。不管特性（如title）在标记里怎样声明，都可以存取或设置（下一节会讲到），通过利用任何大小写变体——Title、TITLE、TiTlE，或任何其他等效组合来完成。即使在XHTML标记里特性名称必须小写，也依然可利用任何大小写变体来获取特性值。

这时你也许会问："存取属性那么简便（参见3.1.1节），为什么处理特性根本就不是那样呢？"

这个问题的答案是jQuery的attr()命令不仅仅是JavaScript的getAttribute()和setAttribute()方法的包装。除允许存取元素特性集以外，jQuery也允许存取常用属性。由于

① 这实质上就是"重载"。重载，是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。"重载变体"来源于这个概念。——译者注

传统上处处依赖于浏览器，所以对页面作者来说存取属性是件痛苦的事情。

这组规范化存取名称如表3-1所示。

表3-1　jQuery attr()的规范化存取名称

规范化名称	源名称
class	className
cssFloat	IE用styleFloat，其他浏览器用cssFloat（当和.css一起使用时）
float	IE用styleFloat，其他浏览器用cssFloat（当和.css一起使用时）
for	htmlFor
maxlength	maxLength
readonly	readOnly
styleFloat	IE用styleFloat，其他浏览器用cssFloat（当和.css一起使用时）

53

除了这些有用的快捷方式以外，用于写操作的attr()重载变体还提供了便利的功能。请往下阅读。

3.1.3　设置特性值

用jQuery把特性值设置到包装集元素上，有两种办法。先讲极为直接的办法，允许一次设置一个特性（对包装集里的所有元素）。语法如下所示。

命令语法：attr

attr(name,value)

为包装集里所有元素的name特性设置传递进来的值。

参数

name　（字符串）将被设置的特性的名称。

value　（字符串|对象|函数）指定特性的值。可以是有返回值的任何JavaScript表达式，或是一个函数。请看以下关于如何处理这个参数的讨论。

返回

包装集

这个attr()重载变体，乍看起来也许觉得简单，其实它的操作是相当复杂的。

attr的最基本形式是，当value参数是任何有返回值（可以是数组）的JavaScript表达式时，把表达式的已计算值设置为特性值。

当value参数是一个函数引用时，事情变得更加有趣。在这种情况下，为包装集各元素分别调用函数，把函数返回值作为特性值。每次调用函数时传递一个参数，表示元素在包装集里从0开始的下标。另外建立当前元素作为this变量，用于函数调用以便函数为各个特定元素调整处理方式——这就是以这种方式来调用函数的主要力量所在。

54

考虑以下语句：

```
$('*').attr('title',function(index) {
  return 'I am element ' + index + ' and my name is ' +
         (this.id ? this.id : 'unset');
});
```

这个命令将分别作用于页面上的所有元素，设置各元素的title特性为一个字符串，即由DOM中元素下标和各个特定元素id特性值所组成的字符串。

我们会采用这个办法来指定特性值，只要那个值依赖于元素其他特性值，而不依赖于无关值。

用于写操作的attr()第二个重载变体允许一次性指定多个特性。

命令语法：attr

attr(attributes)

把已传递对象指定的特性和值设置到包装集的所有元素上。

参数

attributes　一个对象。对象属性被作为特性而复制到包装集里所有元素上。

返回

包装集

这种格式是设置多个特性到包装集里所有元素的快速简便的方式。传入的参数可以是任何对象引用，但通常是对象字面量[1]，其属性指定将要设置的特性的名称和值。考虑：

```
$('input').attr(
  { value: '', title: 'Please enter a value' }
);
```

这个语句把所有<input>元素的value设置为空字符串，同时把title设置为字符串Please enter a value。

请注意，如果传递进来的对象（作为value参数）的属性值是一个函数引用，则它的操作方式类似于前面的attr()格式——为匹配集里各元素分别调用函数。

警告　IE浏览器不允许<input>元素的name特性被修改。如果你想修改IE浏览器里<input>元素的name特性，就必须用拥有目标名称的新元素取代旧元素。

现在我们懂得如何获取和设置特性了。但怎样删除特性呢？

3.1.4 删除特性

为了从DOM元素中删除特性，jQuery提供了removeAttr()命令。语法如下所示。

① 对象字面量，可以理解为字面上的对象，格式为{key1:value1, key2:value2,...}。——译者注

命令语法：removeAttr

removeAttr(name)

从每个已匹配元素删除指定的特性。

参数

name　（字符串）将要删除的特性的名称。

返回

包装集

请注意，删除一个特性不会从JavaScript DOM元素删除任何对应的属性，尽管可能导致属性值的改变。例如，从一个元素删除readonly特性，会导致元素readOnly属性值从true变为false，但属性本身不会从元素中删除。

现在看一些示例，学习如何在页面上运用这些知识。

3.1.5　特性带来的快乐

假定我们想要在新窗口里打开网站上指向外部域名的所有链接。如果由我们全盘掌握整个标记的话，这个任务就相当简单，如下所示：

56

```
<a href="http://external.com" target="_blank">Some External Site</a>
```

那当然好，但如果我们运行CMS（Content Management System，内容管理系统）或维基（wiki）网站，在那儿终端用户能够添加内容，但不能添加target="_blank"到所有外部链接，不是吗？首先，尝试确定我们想要什么——我们想要在新窗口里打开所有href特性以http://开头的链接（我们确定通过设置target特性为_blank就可实现）。

可以利用本节学到的技巧简洁地完成任务，如下所示：

```
$("a[href^=http://]").attr("target","_blank");
```

首先选择href特性以http://开头（这表明指向外部域名）的所有链接。然后设置其target特性为_blank。仅用一行jQuery代码就完成了任务！

另一个jQuery特性操作功能的卓越用途是帮助解决Web应用（富因特网应用等）长期存在的问题：可恶的双重提交问题。这是Web应用的常见问题，由于表单提交的延迟，有时几秒钟或更长，让用户有机会多次点击提交按钮，从而导致服务器端代码的种种麻烦。

为了解决问题，我们绑定处理程序到表单的提交事件上，在提交按钮第一次点击之后禁止提交按钮。通过这种方式，用户就不会有机会多次点击提交按钮，并且看到可见的指示“表单正在提交中”（假定已禁止的按钮在浏览器里表现为那样）。不要为以下示例的事件处理细节而担心（第5章会学到大量这方面的知识），现在专注于attr()命令的使用：

```
$("form").submit(function() {
  $(":submit",this).attr("disabled", "disabled");
});
```

在事件处理程序体内，用:submit选择器来获取表单内的所有提交按钮，并将disabled特性

值修改为"disabled"（W3C官方推荐的特性设置）。请注意，建立匹配集的时候我们提供this的上下文值（第二个参数）。第5章研究事件处理时你将会发现，在事件处理程序中进行操作的时候，this指针总是引用已绑定事件的页面元素。

57

警告 以这种方式禁止提交按钮，不会免除服务器端代码的责任——预防双重提交或任何其他类型的验证问题。添加这种功能到客户端代码，能够提高界面对终端用户的友好程度，并帮助防止正常情况下的双重提交问题。但它防止不了攻击或其他黑客企图，所以服务器端代码必须继续保持警惕。

在本节前面部分我们曾提到className属性，它作为在某种情况下属性名称不同于标记特性名称的示例。但事实上，类名称在其他方面也有点特殊，因此jQuery对类名称进行特殊处理。3.2节将说明处理类名称的更好办法，而不是直接存取className属性或利用attr()命令。

3.2 修改元素样式

修改元素样式的方法有两种。可以添加或删除CSS类，从而导致现有样式表根据新的CSS类而重塑元素的样式。也可以操作DOM元素，直接应用新样式。

看jQuery怎样使得修改元素样式类变得简单。

3.2.1 添加和删除类名称

类名称特性和DOM元素的类名称属性在格式和语义上与众不同，并且对富用户界面的创建来说也很重要。给元素添加类名称或者从元素中删除类名称，是动态修改元素样式呈现的基本手段之一。

使得元素类名称与众不同的方面之一（也是必须应对的一个挑战），就是每个元素可被指派任意多个类名称。在HTML里，class特性被用来提供以空格分隔、由多个类名称所组成的字符串。例如：

```
<div class="someClass anotherClass yetAnotherClass"></div>
```

不幸的是，在DOM元素相应的className属性里，这些类名称不是表示为名称数组，而是空格分隔的字符串。多么令人失望，多么麻烦！这意味着每当我们想给（或从）已拥有类名称的元素添加（或删除）类名称，就必须解析字符串，以便在读取时确定单独的名称，而在写入时确保还原为空格分隔的有效格式。

58

尽管编写代码处理这一切并不是艰巨任务，但是，隐藏这些机械操作的细节而抽象出简洁的API，终归是个好主意。幸好jQuery已经为我们做好了。

添加类名称到匹配集的所有元素，如果利用以下addClass()命令进行操作就很简便。

命令语法：addClass

addClass(names)

添加指定的一个或多个类名称到包装集的所有元素。

参数

names　一个字符串，包含将要添加的一个类名称，或者包含空格分隔的、将要添加的多个类名称。

返回

包装集

利用如下removeClass()命令来删除类名称显得直接干脆。

命令语法：removeClass

removeClass(names)

从包装集各元素里删除指定的一个或多个类名称。

参数

names　一个字符串，包含将要删除的一个类名称，或者包含空格分隔的、将要删除的多个类名称。

返回

包装集

我们经常想来回切换一组样式，也许是为了表现两个状态之间的变化，或出于在界面上有意义的任何其他理由。利用toggleClass()命令，jQuery使得这个操作非常简便。

59

命令语法：toggleClass

toggleClass(name)

如果在元素中指定类名称不存在，则添加指定类名称；如果元素已拥有指定类名称，则从元素中删除指定类名称。请注意，每个元素单独测试，所以一些元素被添加类名称，而其他则被删除类名称。

参数

name　一个字符串，包含用于切换的类名称。

返回

包装集

当我们想要快速简便地在元素之间切换视觉呈现时，toggleClass()命令最有用。记得图1-1的“斑马条纹”示例吗？当某些事件发生时，如果有合理的理由要交换奇数行和偶数行的背景颜色（也许再反过来），该怎么做呢？用toggleClass()命令来隔行添加类名称，同时从剩下行删除类名称，实现起来几乎是小菜一碟。

试试看。在文件chapter3/zebra.stripes.html中，你会发现除一些细微变化以外，其他几乎和第1章的“斑马条纹”页面一模一样。我们添加了以下函数到页眉的<script>元素：

```
function swap() {
  $('tr').toggleClass('striped');
}
```

这个函数利用toggleClass()命令为所有<tr>元素切换名为striped的类。我们还给表格的onmouseover和onmouseout特性添加函数调用：

```
<table onmouseover="swap();" onmouseout="swap();">
```

结果就是每当鼠标光标进入或离开表格时，拥有striped类的所有<tr>元素将会删除类，而没有striped类的所有<tr>元素将会添加类。这个（相当烦人的）活动在图3-2中分两部分显示。

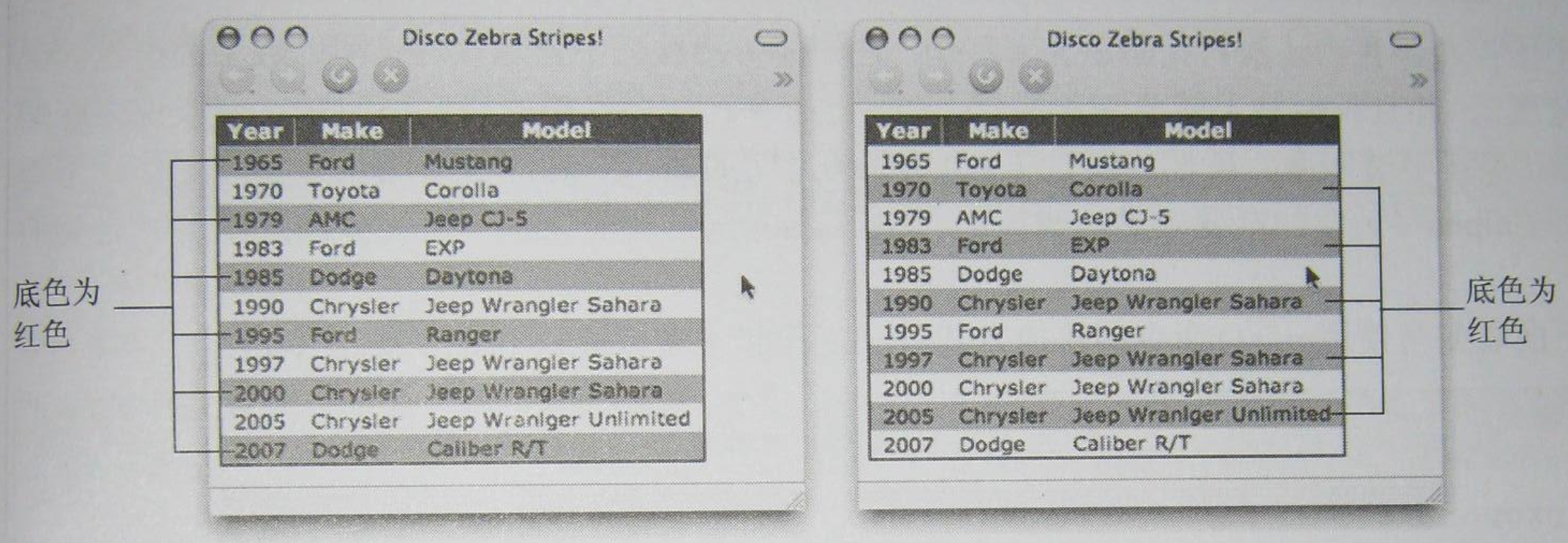


图3-2 每当鼠标光标进入或离开表格时，切换striped类的“有”与“无”

CSS类名称是操作元素样式呈现的强大工具。但有时我们考虑直接操作样式实质内容，比如直接在元素上声明样式。看jQuery为我们提供了什么。

3.2.2 获取和设置样式

虽然修改元素class特性允许从样式表已定义样式中选择一组预定样式加以应用，但有时我们却想完全覆盖样式表。直接在元素上应用样式将自动覆盖样式表，使我们能够对单独元素及其样式施加更细化的控制。

css()方法工作起来类似attr()方法，允许通过指定名称和值来设置单独的CSS属性，或通过传入一个对象来设置一系列CSS属性。首先看怎样指定名称和值。

命令语法：css

css(name,value)

设置指定的值到每个已匹配元素的指定的CSS样式属性。

参数

name （字符串）将要设置的CSS属性名称

value （字符串|数字|函数）一个包含属性值的字符串、数字或函数。如果传递函数作为参数，则为包装集各元素分别调用函数，以函数返回值作为CSS样式属性值。每次函数调用

的this属性，设置为当前正在计算的元素。

返回

61 包装集

如上所述，value参数也像attr()命令那样接受一个函数。这意味着（例如），可以使包装集里所有元素的宽度都扩大20像素，如下所示：

```
$("div.expandable").css("width",function() {
   return $(this).width() + 20 + "px";
});
```

不用担心，我们还没讨论width()命令。它如你所愿丝毫不差地执行（即以数字类型返回元素的宽度），并且我们马上就要深入探讨width()命令。一个有趣的补充说明是，一般说来麻烦的opacity（不透明度）属性，通过传入0.0到1.0之间的值，居然能够跨浏览器完美工作，不用再惹IE的alpha（透明的渐变效果）滤镜、-moz-opacity（用于定义一个元素的透明度）之类的麻烦！

下面看如何使用css()命令的快捷方式——工作起来酷似attr()的快捷版本。

命令语法：css

css(properties)

为所有已匹配元素设置已传递对象里多个键所指定的CSS属性为相关的值（value）[1]。

参数

properties　（对象）指定一个对象。对象的属性被复制为包装集里所有元素的CSS属性。

返回

包装集

就像attr()命令的快捷版本（请参见3.1.3节）那样，可以使用函数作为properties参数对象里的任何CSS属性值。jQuery为包装集各元素分别调用函数，确定将要应用的属性值。

最后我们可以为css()传入名称，以便获取与该名称相关联的属性的已计算样式。当我们说已计算样式时，意思是指所有链接、嵌入和内联的CSS[2]都已应用之后的样式。给人深刻印象的是，这个命令能够跨浏览器完美工作。甚至传入opacity（不透明度）也能返回一个字符串，表示0.0到1.0之间的一个数字。

62

命令语法：css

css(name)

获取包装集里第一个元素name所指定CSS属性的已计算样式值。

---

① 即把一个“名称/值对”对象的属性设置为所有已匹配元素的样式属性。——译者注

② “链接”指外部样式表文件，“嵌入”指在<style> </style>之间定义的样式块，“内联”指在元素标记内使用style特性定义的样式。一般推荐采用第一种方式。——译者注

参数

name （字符串）指定一个CSS属性名称，返回它的已计算样式值。

返回

已计算样式值

切记css()命令的这个重载变体总是返回字符串，所以如果你需要数字或其他类型，就必须对返回的字符串进行解析。

对于经常存取的一组CSS值，jQuery体贴地提供了方便的命令，可以轻松存取这些值并转换为最常用的类型。具体而言，我们可以通过width()和height()命令获取（或设置）元素的宽度和高度的数值。为了设置宽度或高度，请看如下命令语法。

命令语法：width和height

width(value)

height(value)

对匹配集里所有元素设置指定的宽度或高度。

参数

value （数值）将要设置的值，以像素为单位。

返回

包装集

切记这两个函数是更为详细的css()函数的快捷方式，所以

```
$("div.myElements").width(500)
```

等效于

```
$("div.myElements").css("width","500px")
```


为了获取宽度或高度，请看如下命令语法。

命令语法：width和height

width

height

获取包装集的第一个元素的宽度或高度。

参数

无

返回

已计算的宽度或高度，以数字类型返回

函数以数字类型返回宽度和高度值的事实，并不是这些命令提供的唯一便利之处。如果你曾试图通过读取style.width或style.height属性查找元素的宽度或高度，你就会面临令人遗憾的事实——这些属性只被那个元素相应的style特性所设置。想要通过这些属性查找元素的大小，首先必须设置这些属性[1]。这不完全是有用的典范！

另一方面，width()和height()命令计算并返回元素的大小。虽然通常没有必要知道简单页面上元素的准确大小，以便让元素恰当地布局，但在富因特网应用里，知道准确的大小却至关紧要——才能够恰当地放置活动元素，如上下文菜单、自定义工具提示、已扩展控件，以及其他动态组件。

下面让这两个函数工作起来。图3-3显示用两个主要元素来设置的示例：一个是测试对象 ，它包含一段文本（以边框和背景颜色突出显示）；另一个是 ，用于显示测试对象的宽度和高度。

64

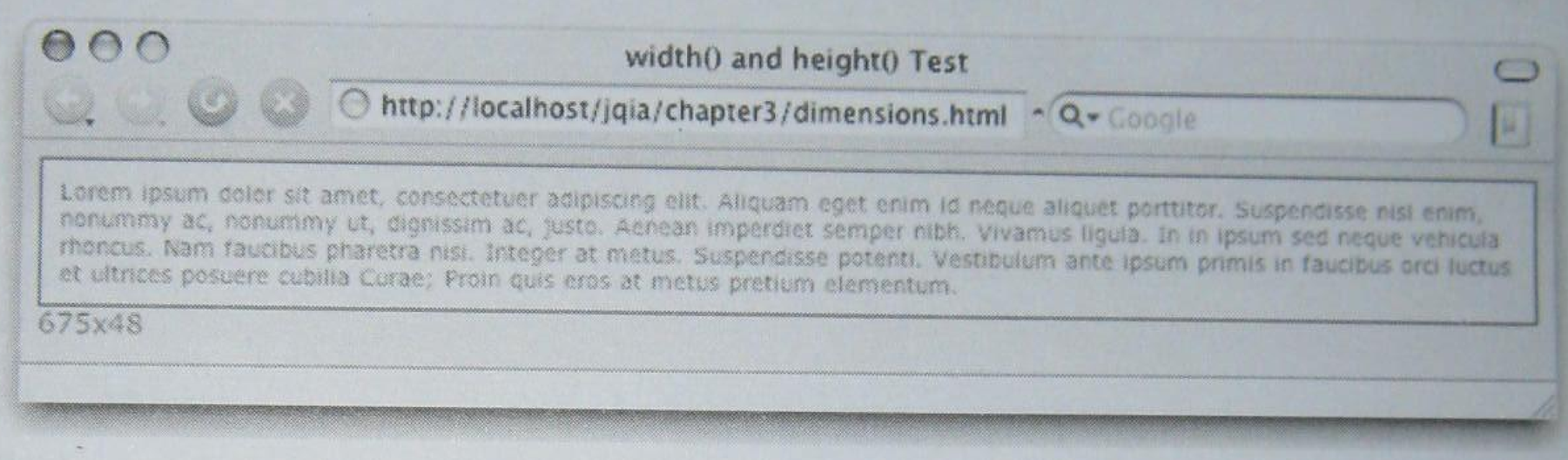


图3-3　受测元素的宽和高不是固定的，而是取决于浏览器窗口的宽度

因为没有应用指定宽和高的样式规则，所以测试对象的宽和高预先并不知道。元素宽度由浏览器窗口的宽度决定，元素高度取决于需要多少空间来显示所包含的文本。改变浏览器窗口的宽度会引起元素的宽度和高度同时发生变化。

在页面中定义用width()和height()命令来获取测试对象 （命名为testSubject）宽和高的函数，并在第二个 （命名为display）中显示结果值。

```
function report() {
  $('#display').html(
    $('#testSubject').width()+'x'+$('#testSubject').height()
  );
}
```

调用页面就绪处理程序里的这个函数，将会导致显示测试对象具体大小值675和48，如图3-3所示。

在<body>元素onresize特性里，我们也添加函数调用：

```
<body onresize="report();">
```

改变浏览器宽度导致的显示结果如图3-4所示。

确定元素在任何时刻的已计算大小，此能力对于在页面上精确布局动态元素来说至关紧要。这个页面的完整代码如代码清单3-1所示，并可在文件chapter3/dimensions.html找到。

65

① 或设置图像的style特性。例如。——译者注

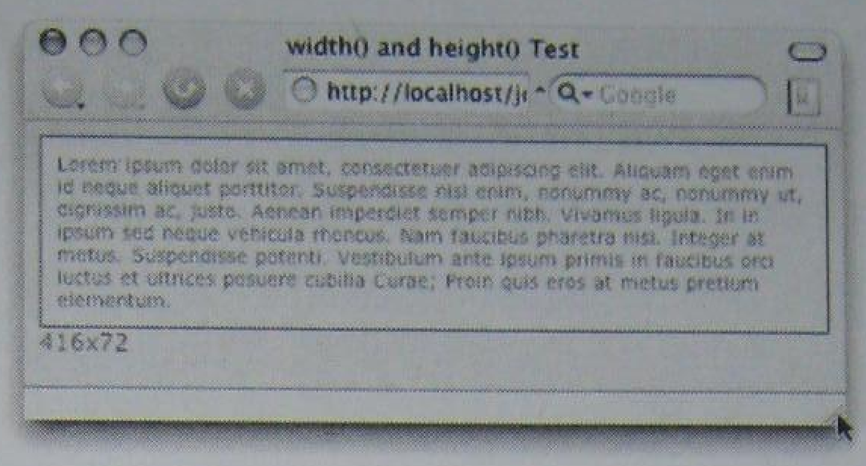


图3-4 改变浏览器宽度导致测试对象改变大小，这个变化反映于已计算值

代码清单3-1 动态跟踪一个元素的宽和高

```
<html>
  <head>
    <title>width() and height() Test</title>
    <link rel="stylesheet" type="text/css" href="../common.css">
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js"></script>
    <script type="text/javascript">
      $(function(){
        report();          ◁ 当页面就绪时调用 report()函数
      });

      function report() {  ◁ 显示测试对象的宽和高
        $('#display').html(
          $('#testSubject').width()+'x'+$('#testSubject').height()
        );
      }
    </script>
    <style>                ◁ 把样式应用到测试对象
      #testSubject {
        background-color: plum;
        border: 1px solid darkmagenta;
        padding: 8px;
        font-size: .85em;
      }
    </style>
  </head>

  <body onresize="report();">   ◁ 当窗口大小改变时报告测试对象的宽和高
    <div id="testSubject">      ◁ 用假文本声明测试对象
      Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
      Aliquam eget enim id neque aliquet porttitor. Suspendisse
      nisl enim, nonummy ac, nonummy ut, dignissim ac, justo.
      Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum
      sed neque vehicula rhoncus. Nam faucibus pharetra nisi.
      Integer at metus. Suspendisse potenti. Vestibulum ante
      ipsum primis in faucibus orci luctus et ultrices posuere
      cubilia Curae; Proin quis eros at metus pretium elementum.
    </div>
    <div id="display"></div>   ◁ 在这个区域里显示宽和高
  </body>
</html>
```

66

可能你已经发现这个事实，示例HTML标记中嵌入了行为，从而违反了“不唐突的JavaScript”规则。现在暂时允许这样，但下一章我们将学习绑定事件处理程序的更好方式。既然已经探讨了怎样操作包装集元素样式，下面就看两个相关的面向样式的活动，你也许想实现这些活动并想知道如何实现。

3.2.3　样式相关的更有用的命令

确定元素是否拥有特定类，是极为常见的需求。可以调用jQuery的hasClass()函数来实现：

```
$("p:first").hasClass("surpriseMe")
```

如果匹配集里某个元素拥有指定类，这个语句就返回true。这个命令的语法如下所示。

命令语法：hasClass

hasClass(name)

确定匹配集里是否有元素拥有已传递name参数所指定的类名称。

参数

name　（字符串）将要检查的类名称。

返回

如果包装集里某个元素拥有已传递的类名称，就返回true；否则返回false。

回忆第2章讲到的is()命令（请参见2.3.5节），用如下语句也可以完成同样的事情：

```
$("p:first").is(".surpriseMe")
```

事实上，jQuery内部工作正是这样实现hasClass()函数的！但可以论证的是，hasClass()命令有助于编写可读性更高的代码。

另一个常用的功能是，以数组形式来获取指定元素的已定义类列表，取代麻烦的以空格分隔的列表。可以试着编写如下语句来实现：

```
$("p:first").attr("class").split(" ");
```

回忆一下，如果查找的特性不存在，attr()命令将返回undefined。因此如果<p>元素没有任何类名称，这个语句将抛出错误。通过检查特性是否存在，可以解决这个问题。如果想以可重用的jQuery扩展包装整件事情，可以编写：

67

```
$.fn.getClassNames = function() {
  if (name = this.attr("className")) {
    return name.split(" ");
  }
  else {
    return [];
  }
};
```

不用为jQuery扩展的语法细节而担心，第7章将详细探讨这个主题。重要的是可以在脚本的任何地方利用getClassNames()来获取类名称数组或空数组（如果元素没有CSS类）。棒极了！

既然我们已经学会如何获取和设置元素样式，下面就讨论修改元素内容的不同办法。

3.3 设置元素内容

说到修改元素内容，有一场关于哪种技术更好的正在进行的争议：利用DOM API方法，还是修改元素内部HTML。在大多数情况下，修改元素的HTML更加轻松有效，所以jQuery有多种修改元素HTML的方法。

3.3.1 替换 HTML 或文本内容

首先是简单的`html()`命令。如果不带参数进行调用，就返回元素的HTML内容；如果带着参数进行调用，就像其他jQuery函数那样，设置元素的HTML内容。

下面说明如何获取元素的HTML内容。

命令语法：html

html()

获取匹配集里第一个元素的HTML内容。

参数

无

返回

第一个已匹配元素的HTML内容。返回值与访问那个元素的`innerHTML`属性所获得的内容一致。

下面说明如何设置所有已匹配元素的HTML内容。

命令语法：html

html(text)

把传入的HTML片段设置为所有匹配元素的内容。

参数

`text` （字符串）将被设置为元素内容的HTML片段。

返回

包装集

也可以只获取或设置元素的文本内容。不带参数调用`text()`命令，返回所有文本连接而成的字符串。例如，假定有如下HTML片段：

```
<ul id="theList">
  <li>One</li>
  <li>Two</li>
  <li>Three</li>
  <li>Four</li>
</ul>
```

如下语句

```
var text = $('#theList').text();
```

导致变量text赋值为OneTwoThreeFour。

命令语法：text

text()

把包装集里元素的所有文本内容连接起来，并返回字符串作为命令的结果

参数

无

返回

连接而成的字符串

69

也可以利用text命令来设置包装集元素的文本内容。这种格式的语法如下所示。

命令语法：text

text(content)

把所有已包装元素的文本内容设置为已传递值。如果已传递文本包含尖括号（<和>），则这些字符被替换为等价的HTML实体[①]。

参数

content　（字符串）将要设置到已包装元素里的文本内容。任何尖括号字符将被转义为HTML实体。

返回

包装集

请注意，用这些命令来设置元素的内部HTML或文本，将替换元素里的原先内容，所以使用这些命令必须小心谨慎。如果你不想覆盖元素原先的所有内容，可以用其他几个方法按原样保留元素的内容，并不修改元素的内容或周围的元素。下面介绍这几个方法。

3.3.2　移动和复制元素

如果想把内容添加到现有内容的末尾，可以利用append()命令。

命令语法：append

append(content)

把传入的HTML片段或元素追加到所有已匹配元素的内容之后。

① <的HTML实体是<，而>的HTML实体是>。——译者注

参数

content （字符串|元素|对象）将被追加到包装集各元素的一个字符串、元素或包装集。请看下文的详细说明。

返回

包装集

这个函数接受包含HTML片段的字符串、现有或新建DOM元素的引用，或jQuery元素包装集。考虑如下简单情况：

```
$('p').append('<b>some text<b>');
```


此语句将从传入字符串而创建的HTML片段，追加到页面上所有<p>元素的现有内容的末尾。这个命令的更加复杂的用法是标识DOM现有元素作为追加项。考虑如下代码：

```
$("p.appendToMe").append($("a.appendMe"))
```

这个语句把带有类appendMe的所有链接，追加到带有类appendToMe的<p>元素。对原始元素的布置取决于作为追加目标的元素的数量。如果是单一的目标，则元素从原始位置删除——执行把原始元素移动到新父元素的操作。如果有多个目标，则原始元素留在原处，而原始元素的副本被追加到各个目标元素——复制操作。

如果不想追加整个包装集，也可以引用一个特定DOM元素，如下所示：

```
var toAppend = $("a.appendMe")[0];
$("p.appendToMe").append(toAppend);
```

标识为toAppend的元素是被移动还是被复制，取决于标识为$("p.appendToMe")的元素的数量：如果匹配一个元素则进行移动操作，如果匹配多个元素则进行复制操作。

如果我们想从一个地方移动或复制元素到另一个地方，更简单的解决办法是利用appendTo()命令[1]，该命令允许获取元素并移动到DOM里另外的地方。

命令语法：appendTo

appendTo(target)

把包装集里所有元素移动到指定目标的内容的末尾。

参数

target （字符串|元素）一个包含jQuery选择器的字符串，或一个DOM元素。包装集各元素将追加到target所指定的那个位置。如果多个元素与一个选择器字符串匹配，则包装集各元素将被复制并追加到与选择器匹配的每个元素。

返回

包装集

本节大多数函数的共同语义是，如果目的地只标识一个目标元素，则元素被移动；如果目的

① A.append(B)表示把B追加到A，A是目标；A.appendTo(B)表示把A追加到B，B是目标。——译者注

地标识多个目标元素，则源元素留在原始位置，且被复制到每个目标元素。在讲述工作方式类似的其他命令之前，先看一个示例，确保弄清楚这个重要概念。我们建立实验室页面，上面有一些元素充当appendTo()操作的源元素，另一些元素充当目标元素。“移动和复制实验室页面”的初始显示如图3-5所示。

71

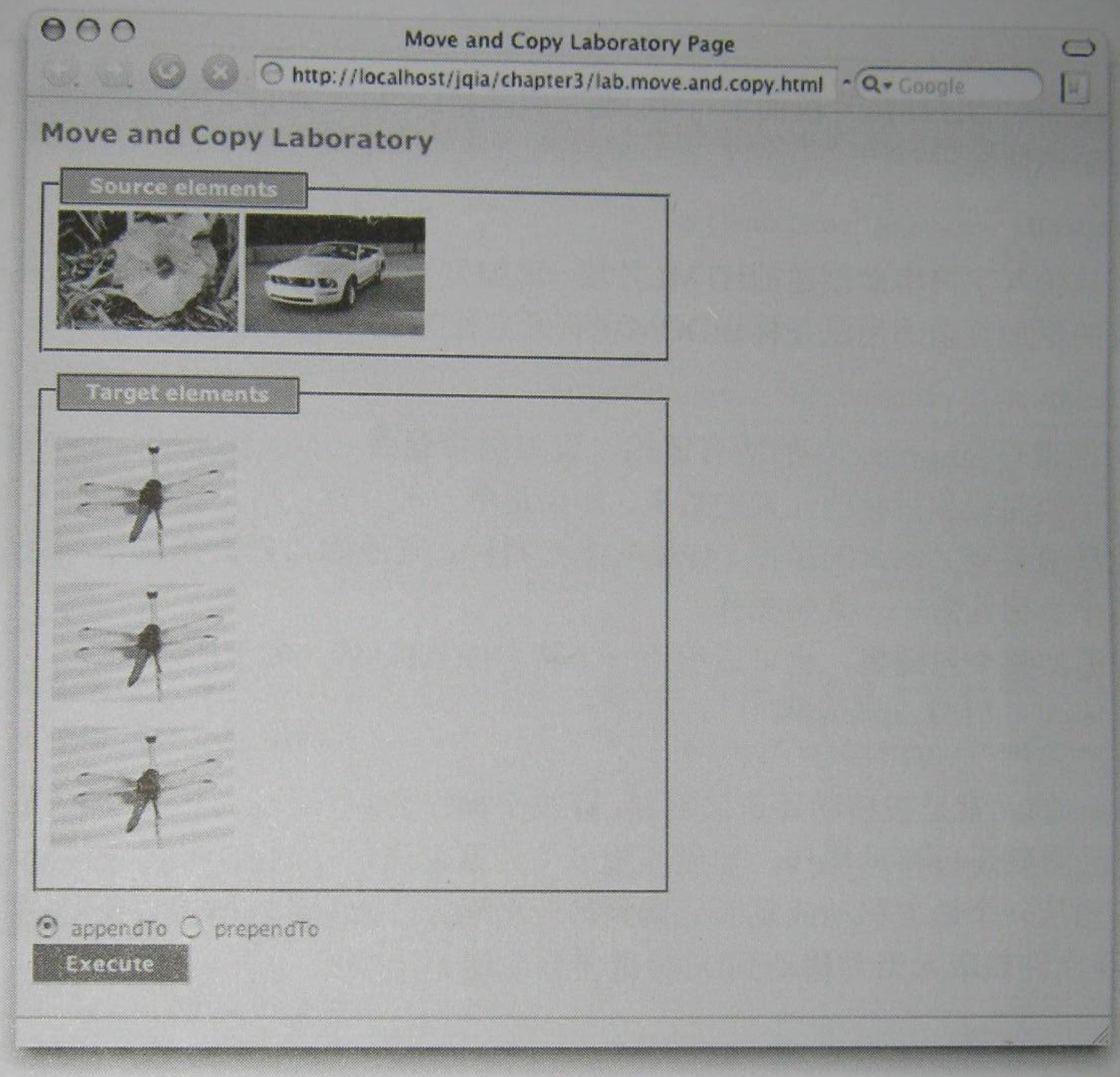


图3-5　设置“移动和复制实验室页面”是为了说明appendTo和prependTo命令的操作

测试候选对象在字段集[1]里的HTML标记如下所示：

```
<fieldset id="source">
  <legend>Source elements</legend>
  <img id="flower" src="flower.png"/>
  <img id="car" src="car.png"/>
</fieldset>
<fieldset id="targets">
  <legend>Target elements</legend>
  <p><img src="dragonfly.png"/></p>
  <p><img src="dragonfly.png"/></p>
  <p><img src="dragonfly.png"/></p>
</fieldset>
```

72

① 字段集一般表现为带标题的框。——译者注

源字段集包含两个图像：一个id为flower，另一个id为car。这两个图像元素将用作命令的源。目标字段集包含3个<p>元素，各自包含一个图像。这3个段落元素将作为命令的目标。

请用浏览器打开这个页面。页面文件是chapter3/lab.move.and.copy.html。选中单选按钮appendTo，点击按钮Execute，就会执行相当于如下的代码：

```
$('#flower').appendTo('#targets p')
$('#car').appendTo('#targets p:first')
```

第一个语句在花朵图像上执行appendTo()命令，指定3个段落元素作为目标。因为目标元素多于一个，所以预计花朵图像会被复制。第二个语句在小汽车图像上执行相同命令，但仅指定第一个段落元素作为目标。因为只有一个目标元素，因此预计小汽车图像会被移动。

图3-6是点击按钮Execute之后的页面截图，显示以上预料是正确的。

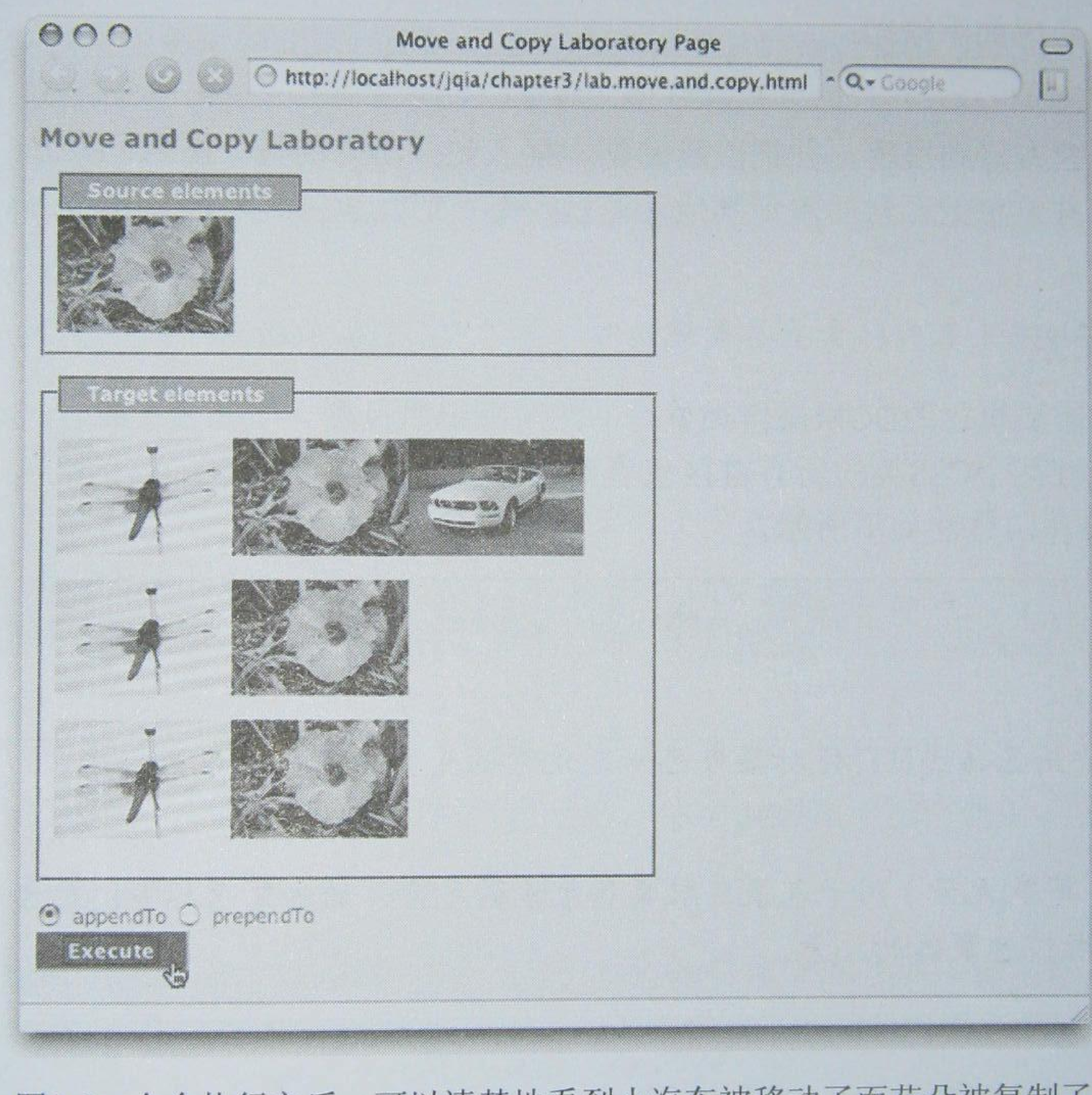


图3-6 命令执行之后，可以清楚地看到小汽车被移动了而花朵被复制了

从这些结果可以清楚地看到，当有多个目标时，源元素被复制；当只有一个目标时，源元素被移动。

有几个相关命令的工作方式类似于append()和appendTo()。

- prepend()和prependTo()——像append()和appendTo()那样执行，不过是在目标元素的内容之前插入源元素，而不是之后。这两个命令也可以在“移动和复制实验室页面”里进行演示，只要先选中单选按钮PrependTo，再点击按钮Execute。

- before()和insertBefore()——在目标元素之前插入元素，而不是在目标元素的第一个子元素之前。
- after()和insertAfter()——在目标元素之后插入元素，而不是在目标元素的最后一个子元素之后。

因为这些命令的语法与append()和appendTo()命令是如此相似，因此不必再浪费空间去显示各命令的语法说明。为了确认这些命令的语法格式，请查阅上文append()和appendTo()的语法块。

73

继续前进之前还有一件事情……

还记得前一章讲过如何用jQuery的$()函数来创建新的HTML片段吗？好，当这个技巧和appendTo()、prependTo()、insertBefore()以及insertAfter()配对使用的时候，就成为十分有用的诀窍。考虑如下语句：

```
$('<p>Hi there!</p>').insertAfter('p img');
```

74

这个语句创建友好的段落，并把段落的副本插入到段落内的每个图像元素之后。

但有时我们并不想把元素插入到其他元素里，而是把元素包裹起来。看jQuery为那种功能的实现提供了什么。

3.3.3 包裹元素

另一种经常需要执行的DOM操作类型是，在某标记里包裹一个元素（或者一系列元素）。例如，我们想把带有某个CSS类的所有链接包裹到 里面。可以用jQuery的wrap()命令来完成这个DOM修改操作。语法如下所示。

命令语法：wrap

wrap(wrapper)

把匹配集各元素用已传递HTML标签或已传递元素的克隆副本分别包裹起来。

参数

wrapper （字符串|元素）用于包裹匹配集各元素的元素开始和结束标签；或者一个将被克隆且用作包裹器的元素。

返回

包装集

为了把带有surprise类的各链接分别包裹在带有类hello的 里，我们编写

```
$("a.surprise").wrap("<div class='hello'></div>")
```

如果想用页面上第一个 元素的克隆副本分别包裹各链接：

```
$("a.surprise").wrap($("div:first")[0]);
```

如果多个元素被收集于匹配集，则wrap()方法分别操作其中的每个元素。如果宁可把集合里所有元素包裹起来作为一个单元，则可以改用wrapAll()方法。

命令语法：wrapAll

wrapAll(wrapper)

用已传递元素的克隆副本或已传递HTML标签，把匹配集的元素作为一个单元包裹起来[1]。

参数

wrapper （字符串|元素）用于包裹匹配集所有元素的元素开始和结束标签，或一个将被克隆且用作包裹器的元素。

返回

包装集

有时不想包裹匹配集里的元素，而想包裹元素的内容。在这种情况下可以用wrapInner()函数。

命令语法：wrapInner

wrapInner(wrapper)

用已传递元素的克隆副本或已传递HTML标签，把匹配集各元素的内容（包括文本节点）分别包裹起来。

参数

wrapper （字符串|元素）用于包裹匹配集各元素的元素开始和结束标签；或者一个将被克隆且用作包裹器的元素。

返回

包装集

既然懂得了如何创建、包裹、复制和移动元素，你可能想知道如何删除元素。

3.3.4 删除元素

如果我们想清空或删除一组元素，则可以用remove()命令来完成。语法如下所示。

命令语法：remove

remove()

从页面DOM里删除包装集里所有元素。

参数

无

---

① 包裹完成后整个单元放置在匹配集第一个元素的位置，匹配集其余元素放置在匹配集第一个元素之后。
——译者注

返回
包装集

请注意，就像许多其他jQuery命令，这个命令也以包装集作为返回结果。从DOM里删除的元素仍然被该包装集引用着（因此不符合垃圾回收条件），并可用其他jQuery命令来进一步操作，包括appendTo()、prependTo()、insertBefore()、insertAfter()以及任何其他类似的命令。

76

为了清空DOM元素的内容，可以使用empty()命令。语法如下所示。

命令语法：empty

empty()

清空匹配集里所有DOM元素的内容。

参数
无

返回
包装集

常见的习惯用法是利用remove()和after()命令来实现替换操作。考虑下面代码：

```
$("div.elementToReplace").after("<p>I am replacing the div</p>").remove();
```

因为after()函数返回包含 的原始包装集，然后删除原始包装集，所以结果是新建的<p>元素替换了原始 元素。

如果将来你发现自己一次又一次地利用这个习惯用法，就请记住，总是可以把有用的语句封装起来形成jQuery扩展，如下所示：

```
$.fn.replaceWith = function(html) {
  return this.after(html).remove();
};
```

利用这个jQuery扩展，执行像前面示例那样的操作，如下所示：

```
$("div.elementToReplace").replaceWith("<p>I am replacing the div</p>");
```

这里我们看到创建jQuery扩展的又一个示例。重要的是，在创建jQuery扩展时应当返回包装集，以便在调用jQuery扩展之后命令链能够继续下去（除非jQuery扩展的目的就是返回其他数据）。你应该逐渐摸清扩展jQuery的门道，但目前不会的话也不用担心，因为本书后面（确切地说是第7章）会详细说明。

77

有时候，我们不想移动元素，而想复制元素……

3.3.5 克隆元素

还有一个操作DOM的途径是复制元素，把复制得到的副本添加到DOM树的其他部分。jQuery提供了便利的包装器方法clone()命令来实现。

命令语法：clone

clone(copyHandlers)

创建包装集里元素的副本，并返回包含这些副本的新包装集。元素以及任何后代元素都被复制。事件处理程序是否被复制，取决于参数copyHandlers的设置。

参数

copyHandlers （布尔型）如果为true，复制事件处理程序；如果为false或省略，则不复制事件处理程序。

返回

新建的包装集

如果不对进行处理的话，利用clone()获得现有元素的副本没有什么用处。一般说来，只要生成了克隆的包装集，就可用另一个jQuery命令把它附加到DOM中的某个地方。例如：

```
$('img').clone().appendTo('fieldset.photo');
```

这个语句获取所有图像元素的副本并追加到所有带有类名photo的<fieldset>元素里。一个稍为复杂的示例如下所示：

```
$('ul').clone().insertBefore('#here');
```

这个命令链除克隆操作的目标元素不同之外，其他操作都很类似。克隆所有的<ul>元素，包括它们的后代元素（一般情况下任何<ul>元素都会拥有多个<li>后代元素）。

最后再举一个示例：

```
$('ul').clone().insertBefore('#here').end().hide();
```

这个语句执行和前面示例相同的操作，但是在插入克隆副本之后，利用end()命令来选择原始包装集（即原始目标）并隐藏它。这个示例强调克隆操作在新包装集中创建了一组新元素。 78

既然已经讨论了怎样处理普通DOM元素，下面简要探讨如何处理特殊的元素类型——表单元素。

3.4 处理表单元素值

因为表单元素拥有特别的属性，所以jQuery核心包含多个便利的函数，用于获取和设置表单元素的值，对表单元素进行序列化，根据表单属性选择元素等操作。这几个函数适用于一般情况，而表单插件——由jQuery核心团队成员开发并被官方认可的插件——提供健壮得多的功能集。第9章将讨论表单插件。

注意 当我们使用术语“表单元素”时，是指在表单中出现的、拥有name和value特性的元素。在表单提交时，名称和值特性将作为请求参数提交到服务器。通过手工编写脚本处理表单元素是需要技巧的，不仅因为元素可以被禁用，还因为W3C为控件定义了不成功状态。该状态决定在表单提交时应该忽略哪些元素，这增加了一点复杂度。

下面看在表单元素上执行的最常见操作：存取表单元素值。jQuery的val()命令考虑了大多数的常见情况，返回包装集里第一个表单元素的value特性。语法如下所示。

命令语法：val

val()

返回匹配集里第一个元素的value特性。如果是多选元素，则返回所有选中项的数组。

参数

无

返回

已获取的值或值数组

这个命令虽然很有用，但必须注意几个限制之处。如果包装集里第一个元素不是表单元素，就会抛出JavaScript错误。这个命令也不区分复选框和单选按钮的选中或非选中状态，而是返回复选框或单选按钮的value特性值，不管其选中与否。

79

对于单选按钮，jQuery选择器与val()方法相结合的力量能够反败为胜。考虑包含名为radioGroup的单选按钮组（名称相同的一组单选按钮）的表单，以及如下表达式：

```
$('[name=radioGroup]:checked').val()
```

这个表达式返回一个已选中单选按钮的值（如果全部未选中则返回undefined）。这比循环一组单选按钮查找已选中元素轻松多了，不是吗？既然val()只考虑包装集里第一个元素，对于多个控件可被选中的复选框组，就不适用了。

如果想通过表单提交来获取控件的值，利用serialize()命令（请参见第8章）或官方的表单插件，则情况更好。

另一个我们将会执行的常见操作是设置表单元素的值。给val()命令传递值也就是用于这个目的。语法如下所示。

命令语法：val

val(value)

把传入的值设置为所有已匹配表单元素的值。

参数

value　一个字符串，用于设置包装集里各表单元素的value属性值。

返回

包装集

就像用于读操作的val()的重载变体那样，这个函数也有局限性。例如，不能把多个值设置到多选列表。这就是在表单插件中提供更为健壮的功能的理由。除迄今为止列举的局限性以外，这个函数能够执行以下操作：获取复选框组的值的数组、对包装集里的元素进行序列化、清空字

段，甚至把DOM表单转换为适用于Ajax的格式。

val()方法的另一个用途是使复选框或单选按钮变为选中状态，或选择<select>元素内的选项。这个val()重载变体的语法如下所示。

命令语法：val

val(values)

导致包装集里任何复选框、单选按钮或<select>元素的选项变为已选中（checked）或已选择（selected）状态，只要它们的值和已传递值数组的任何一个值相匹配。

参数

values　一个值数组，用于确定哪些元素将被选中或选择。

返回

包装集

考虑如下语句：

```
$('input,select').val(['one','two','three']);
```

这个语句将搜索页面上所有<input>和<select>元素，只要它们的值和输入字符串one、two或three中的任何一个相匹配。搜索到的任何匹配复选框或单选按钮将变为已选中状态，而任何匹配的选项将变为已选择状态。

这使得val()的用途远远超出文本元素的范围。

3.5 小结

在本章中，我们已经超越了选择元素的技术，并开始操作元素。利用迄今为止所学的技术，我们能够通过强大的标准来选择元素，然后如外科手术般地把它们移动到页面的任何部分。

我们可以有选择地复制或移动元素，甚或从无到有创建全新的元素；可以追加、前置或包裹页面上任何元素或元素集。同时我们学会如何以一致的方式来处理单个元素或一组元素，从而导向强大且严密的逻辑。

有了这些知识储备，我们准备深入探讨更为高级的概念，就从典型的难以对付的页面事件处理开始吧。

第4章 事　件

本章内容

- 浏览器所实现的事件模型
- 利用jQuery把事件处理程序绑定到元素上
- Event对象实例
- 在脚本控制下触发事件处理程序

熟悉百老汇戏剧Cabaret（歌厅），或其后的同名好莱坞电影的任何人，很可能记得*Money Makes the World Go Around*（有钱能使鬼推磨）这首歌。这个愤世嫉俗的观点也许适用于物质世界，而在万维网（World Wide Web）的虚拟王国里，是事件使一切运转！

和许多其他GUI管理系统那样，通过HTML网页所呈现的界面是异步的和事件驱动的（即使用于把网页传送到浏览器的HTTP协议，本质上是完全同步的）。不管GUI是作为桌面程序利用Java Swing、X11、.NET框架去实现，还是作为Web应用里的页面利用HTML和JavaScript去实现，过程几乎相同：

(1) 建立用户界面；

(2) 等待有趣的事情发生；

(3) 做出相应的反应；

(4) 重复。

步骤（1）建立用户界面的显示，其他步骤定义用户界面的行为。在网页里，浏览器响应Web服务器发送给它的标记（HTML和CSS），负责建立用户界面的显示。网页里包含的脚本则定义用户界面的行为。

这些脚本表现为事件处理程序的形式，也称为监听器，对显示页面时发生的各种事件做出反应。这些事件可以由系统产生（比如计时器或异步请求的完成），但在大多数情况下是用户操作的结果（比如移动或点击鼠标，或通过键盘输入文本）。假如没有这些事件反应能力，万维网的最大用途可能局限于显示小猫们的图像。

尽管HTML确实定义了少许几个无需我们编写脚本的内建语义操作（比如点击<a>标签导致重新加载页面，或点击一个提交按钮提交表单），但是想要页面展示任何其他行为，我们就不得不处理用户和页面交互时发生的各种事件。

在本章里，我们探讨浏览器公开事件的各种方式、如何建立处理程序用来控制事件触发时做

出的反应，以及我们面临的浏览器事件模型的诸多差异所带来的挑战。然后看jQuery如何穿透浏览器导致的这层迷雾，把我们从这些负担中解放出来。

下面首先探讨浏览器是怎样实现事件模型的。

83

你需要了解的JavaScript

jQuery带给Web应用的最大好处之一，就是无需动手编写一大堆脚本就能实现大量的脚本启用行为。jQuery处理了具体细节，所以我们能够集中关注Web应用真正需要做的事情！

直到目前为止，我们几乎是免费乘车[1]。你只需掌握基本的JavaScript技能，就能理解和编写前面几章介绍的jQuery示例。在本章和后面各章，你必须理解几个JavaScript重要概念，才能有效地利用jQuery库。

根据你的学习背景，可能你已经熟悉这些概念，但有些页面作者即使没有牢固地掌握底层到底在干些什么，也能够编写大量JavaScript代码——JavaScript的灵活性使得这种情况有可能发生。在往下探讨之前，应该确保你已经把这些核心概念装进大脑。

如果你已经熟悉JavaScript对象和函数类的工作方式，并很好地领会了函数上下文和闭包等概念，就可以阅读本章和后面各章。如果对这些概念感到生疏或模糊，那么强烈推荐你先看附录A，帮助你加速理解这些必备的概念。

4.1 浏览器的事件模型

远在某人考虑如何使浏览器的事件处理标准化之前，网景通信公司（Netscape Communications Corporation）就在其Netscape Navigator（网景航海家）浏览器中引入了事件处理模型。所有现代浏览器依然支持这个模型，这大概也是为广大页面作者所理解得最深、采用得最多的事件处理模型。

这个模型有好几个名称。你可能听说它叫做网景事件模型（Netscape Event Model）、基本事件模型（Basic Event Model），甚或相当含糊的浏览器事件模型（Browser Event Model）。但大多数人称之为DOM第0级事件模型（DOM Level 0 Event Model）。

注意 术语DOM Level用于表示W3C DOM规范的实现达到什么需求级别。不存在DOM Level 0，但该术语用于非正式地说明DOM Level 1之前所实现的那些规范。

84

W3C直到DOM 第2级才创建标准化的事件处理模型，并在2000年11月引进。这个模型赢得所有标准兼容的现代浏览器比如Firefox、Camino（连同其他Mozilla浏览器）、Safari以及Opera的支持。IE浏览器继续走自己的路，并支持DOM第2级事件模型的功能子集，尽管采用的是专有接口。

在了解jQuery如何使得这个令人愤慨的事实变为不成问题的问题之前，让我们花点时间了解事件模型如何操作。

① 比喻前面几章学习jQuery不费什么力气和脑筋。——译者注

4.1.1　DOM 第 0 级事件模型

DOM第0级事件模型大概是大多数Web开发者在页面上所采用的事件模型。除在一定程度上独立于浏览器以外，使用起来还相当简便。

在这个事件模型下，事件处理程序是通过把函数实例的引用指派到DOM元素的属性而声明的。定义这些属性用来处理特定类型的事件。例如，指派函数到onclick属性用来处理点击事件、指派函数到onmouseover属性用来处理mouseover事件，而元素支持这些事件类型。

浏览器允许指定事件处理程序的函数体，作为DOM元素的HTML里的特性值，从而提供创建事件处理程序的简便写法。定义这样的处理程序的示例如代码清单4-1所示。这个页面可在本书的可下载代码文件chapter4/dom.0.events.html找到。

代码清单4-1　声明DOM第0级事件处理程序

```
<html>
  <head>
    <title>DOM Level 0 Events Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $('#vstar')[0].onmouseover = function(event) {
          say('Whee!');
        }
      });

      function say(text) {
        $('#console').append('<div>'+new Date()+' '+text+'</div>');
      }
    </script>
  </head>
  <body>
    <img id="vstar" src="vstar.jpg"
         onclick="say('Vroom vroom!');"/>
    <div id="console"></div>
  </body>
</html>
```

❶ 在就绪处理程序里定义mouseover事件处理程序

❷ 实用工具函数输出文本到控制台

❸ 对<img>元素进行设置

❹ 元素充当控制台

85

这个示例采用两种风格声明事件处理函数：声明在脚本控制下和声明在标记特性里。

页面首先声明就绪处理程序❶，从其中（利用jQuery）获取id为vstar的图像元素的引用，并把元素的onmouseover属性设置为以内联形式声明的函数实例。当mouseover事件在元素上触发时，这个函数就成为元素的事件处理程序。请注意，这个函数要求传入一个参数。我们很快就会学到关于这个参数的更多知识。

我们也声明实用工具函数say()❷，用于输出文本消息到页面上的 元素❹。这将给我们免除警告的麻烦，因为如果利用alert()函数而不是这个实用工具函数的话，每当事件在页面上发生时，就会弹出警告框指示此刻有事件发生。

在页面<body>节（包含console元素），我们定义图像元素❸，并在元素上定义事件处理程

序。我们已经看到如何在就绪处理程序❶里定义在脚本控制下的事件处理程序，但这里利用<img>元素的onclick特性来定义点击事件的处理程序。

用浏览器打开这个页面（请找到文件chapter4/dom.0.events.html），在图像上晃动几下鼠标指针[①]然后点击图像，就会得到类似于图4-1所显示的结果。

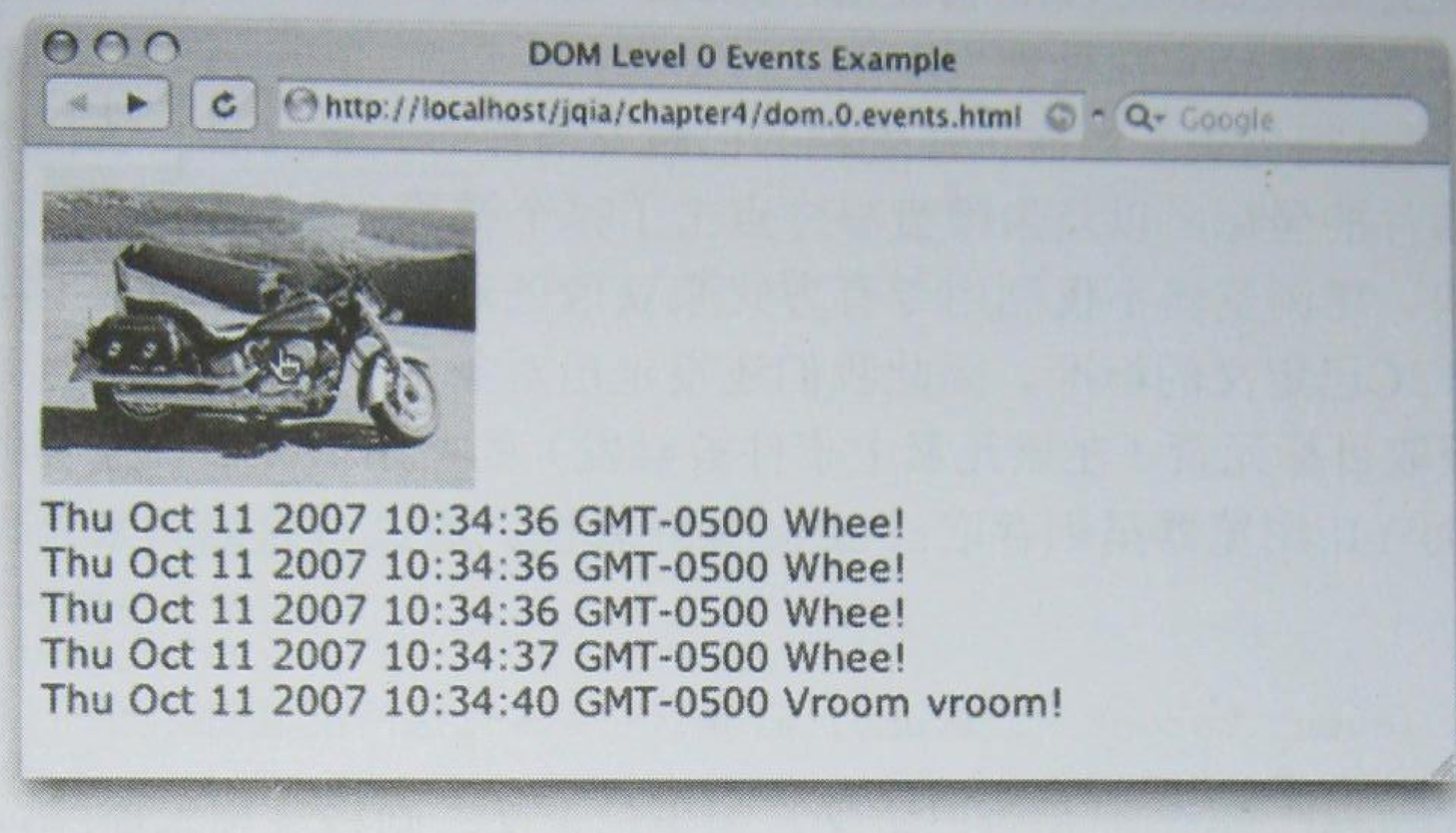


图4-1 在图像上晃动鼠标并点击图像，触发事件处理程序——输出消息到控制台

我们利用onclick特性，在<img>元素标记里声明点击事件处理程序：

```
onclick="say('Vroom vroom!');"
```

这可能让人误以为say()函数成为元素的点击事件处理程序，但实际上并不是那样。如果通过特性标记来声明处理程序，匿名函数就会利用特性的值作为函数体而自动创建。通过特性标记声明处理程序，操作执行结果相当于（假设imageElement是图像元素的引用）如下代码：

```
imageElement.onclick = function(event) {
 say('Vroom vroom!');
}
```

请注意特性的值如何被用作已生成函数的函数体，并注意匿名函数自动创建，因此在已生成函数中可以利用event参数。

在继续探讨关于event参数的所有事情之前，我们应当注意到，利用定义DOM第0级事件处理程序的特性机制，违反了在1.2节探讨的“不唐突的JavaScript”原则。在页面里使用jQuery时，应该遵守“不唐突的JavaScript”原则，避免把显示标记与JavaScript所定义的行为混合在一起。在本章结束之前，将看到jQuery提供的比这两个方式更好的声明事件处理程序的方式。

首先，让我们探讨关于event参数的一切事情。

1. Event实例

在大多数浏览器里，当触发事件处理程序时，名为Event的类实例作为第一个参数传递到处理程序。一直占据主流地位的IE浏览器，却以专门的方式处理事情，它把Event实例指派到window对象的名为event的属性。

① 即鼠标的光标。下同。——译者注

为了处理这个矛盾，我们将经常看到事件处理程序的第一个语句如下所示：

```
if (!event) event = window.event;
```

87

这个语句消除了矛盾。它首先进行对象检测，检查event参数是否为undefined（或null）。如果是那样的话就把window的event属性值指派到event参数。在这个语句之后可以引用event参数，而不用再关心怎样使它在处理程序中变得可用。

Event实例的属性提供关于当前正在处理的已触发事件的大量信息，包括触发的是哪个元素上的事件、鼠标事件的坐标，以及为键盘事件点击了哪个键等。

但没有那么快。IE浏览器不仅利用专有方式来获取处理程序的Event实例，还采用Event类的专有定义取代W3C已定义的标准。因此我们还没走出对象检测的森林[①]。

例如，为了获取目标元素（在该元素上事件被触发）的引用，在标准兼容的浏览器里我们存取target属性，而在IE浏览器里则存取srcElement属性。通过对象检测来处理这个矛盾，语句如下所示：

```
var target = (event.target) ? event.target : event.srcElement;
```

这个语句测试event.target是否已定义，如果已定义，则指派到本地target变量。否则，指派event.srcElement到本地target变量。对于其他Event属性，我们采取类似的步骤。

迄今为止，似乎事件处理程序只和充当事件触发器的那个元素相关——例如代码清单4-1的图像元素。但是事件传播是贯穿整棵DOM树的，下面探讨这个主题。

2. 事件冒泡

当触发DOM树里元素上的事件时，浏览器的事件处理机制会检查在那个元素上是否已经建立特定的事件处理程序。如果是，就调用处理程序。但到这儿事情还远远没有结束呢。

在目标元素获得机会处理事件之后，事件模型检查目标元素的父元素，看是否为同类型事件建立了处理程序。如果是，则也调用父元素的处理程序。在这之后，再检查其父元素，然后父元素，然后父元素……持续不停直到DOM树的顶部。因为事件处理向上传播就像香槟酒杯里冒起的气泡（假定DOM树的根在顶部），所以把这个过程称为事件冒泡。

88

下面修改代码清单4-1的示例，以便我们能够看到这个过程如何进行。请考虑代码清单4-2。

代码清单4-2 事件传播从起点到DOM树的顶部

```
<html id="greatgreatgrandpa">
  <head>
    <title>DOM Level 0 Events Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $('*').each(function(){
          var current = this;
```

❶ 选择页面上的每个元素

① 比喻下面还必须进行对象检测。——译者注

```
      this. onclick = function(event) {
        if (!event) event = window.event;
        var target = (event.target) ?
                     event.target : event.srcElement;
        say('For ' + current.tagName + '#'+ current.id +
            ' target is ' + target.id);
      }
    });
  });

  function say(text) {
    $('#console').append('<div>'+text+'</div>');
  }
  </script>
 </head>

 <body id="greatgrandpa">
  <div id="grandpa">
   <div id="pops">
    <img id="vstar" src="vstar.jpg"/>
   </div>
  </div>
  <div id="console"></div>
 </body>
</html>
```

❷ 应用onclick处理程序到每个已选择元素

我们在页面上的多处地方进行有趣的改动。首先删除任何mouseover处理代码，以便集中关注点击事件。其次把作为事件实验目标的图像元素嵌入两个嵌套的 元素里，以便把图像元素放在DOM层次结构的深处。最后给页面里的几乎每个元素以特定和唯一的id——甚至包括<body>和<html>标签在内！

保留控制台和say()实用工具函数，目的和前面示例一样，用于报告消息。

下面看更为有趣的改动。

在页面的就绪处理程序里，我们利用jQuery选择页面上的所有元素，并利用each()方法❶迭代每个元素。对于每个已匹配的元素，在本地变量current记录它的实例，并建立onclick处理程序❷。这个处理程序首先采用上节讨论的依赖于浏览器的技巧，找到Event实例并标识事件目标，然后输出消息到控制台。消息是这个示例最为有趣的部分。

这个示例运用闭包（如果闭包这个主题让你感到心口灼热的话，请参见附录A的A.2.4节）显示当前元素的标签名称和id，后跟目标元素的id。通过这样做，输出到控制台的每个消息都显示冒泡过程中关于当前元素以及启动整个事情的目标元素的信息。

用浏览器打开页面（找到文件chapter4/dom.0.propagation.html），点击图像，显示结果如图4-2所示。

这清楚地说明事件触发的时候，事件首先传递到目标元素，然后依次传递到各祖先元素，直到<html>元素为止。

这是一种强大的能力，因为允许把处理程序建立在任何一级的元素上，从而处理在后代元素上发生的事件。<form>元素上的处理程序对后代元素上的任何change事件做出反应，从而实现基

于元素新值的显示的动态变化。

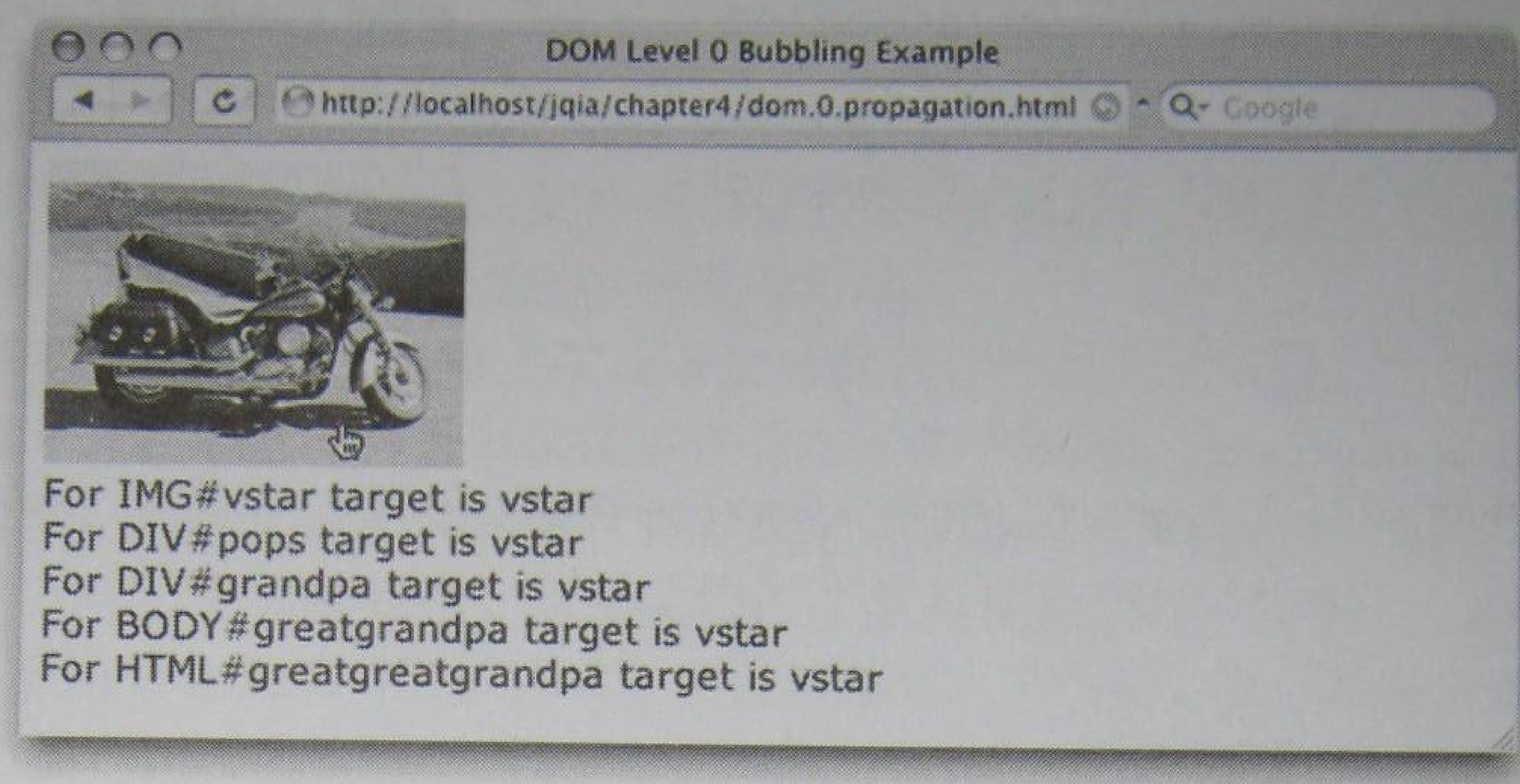


图4-2 控制台消息清楚地显示事件传播过程——从目标元素沿着DOM树冒泡而上直到树根为止

90

但是，如果不想让事件传播出去呢？我们能够停止传播吗？

3. 影响事件传播和语义

可能有些场合我们想阻止事件沿着DOM树向上传播；可能因为我们认真严谨，并知道已经完成响对事件所必需的任何处理；也可能想要预防可能发生在链高层的不需要的处理。

不管什么理由，可以通过在Event实例上已提供的机制来防止事件向上传播。对于标准兼容的浏览器，调用Event实例的stopPropagation()方法可以中止事件往祖先层次传播；对于IE浏览器，则将Event实例中名为cancelBubble的属性设置为true。有趣的是，许多标准兼容的现代浏览器也支持cancelBubble机制，即便那不是任何W3C标准的一部分。

某些事件具有关联的默认语义。例如，<a>元素上的点击事件导致浏览器导航到元素的href特性值，而<form>元素上的提交事件导致表单提交。假如希望取消事件的这些语义——有时称为默认操作——就应该将处理程序的返回值设置为false。

在表单验证领域需要频繁地利用这种操作是。在表单的提交事件的处理程序里，可以在表单的<input>元素上进行有效性检查，如果检测到任何数据输入项有问题，就返回false。

我们也可能看见在<form>元素上如下形式的代码：

```
<form name="myForm" onsubmit="return false;" ...
```

这能有效地防止表单在任何情况下被提交，除非是在脚本控制下提交（通过form.submit()，这不会触发提交事件）。这是在大量Ajax应用里被利用的常见技巧，在那里异步请求代替了表单提交。

在DOM第0级事件模型下，几乎我们在事件处理程序里采取的每个步骤都包括使用浏览器特定检测，以便确定采取什么操作。多么令人头痛！但仍然不要把阿司匹林丢掉——当我们考虑更为高级的事件模型时，并不会变得更加容易。

4.1.2 DOM 第 2 级事件模型

DOM第0级事件模型的一个严重缺点是，因为属性被用于存储作为事件处理程序的函数的引

用，所以每个元素对于任何特定的事件类型，每次只能注册一个事件处理程序。当元素被点击时，假设有两件想做的事情，则如下语句无法达到目的：

91

```
someElement.onclick = doFirstThing;
someElement.onclick = doSecondThing;
```

因为第2个赋值语句替换onclick属性的原先值，所以当触发事件时只调用doSecondThing。当然，可以在第3个函数中包含这两个函数，也就是让第3个函数调用这两个函数。但是，随着页面变得更加复杂，越来越难以保持对这种事情的跟踪，在富因特网应用中极有可能就是这样。此外，如果在页面里采用多个可重用的组件或库，它们可能完全不知道其他组件的事件处理要求。

我们可以采取其他解决方案：实现观察者模式，即建立处理程序的发布/订阅方案，或者利用闭包的技巧。但页面本来就够复杂了，而这些还将增加复杂性。

除建立标准事件模型以外，DOM第2级事件模型被设计来解决这些类型的问题。在DOM第2级事件模型下，让我们看如何在DOM元素上建立事件处理程序甚或多个事件处理程序。

1. 建立事件

并非把函数引用指派到元素属性，DOM第2级事件处理程序（也称为监听器）是通过一个元素方法而建立的。每个DOM元素都定义名为addEventListener()的方法，用于把事件处理程序（监听器）附加到元素上。这个方法的格式如下所示：

```
addEventListener(eventType,listener,useCapture)
```

参数eventType是一个字符串，用于标识将要处理的事件类型。一般说来，这个字符串和DOM第0级事件模型里使用的、不带on前缀的事件名称是一致的。例如click、mouseover、keydown等。

参数listener是函数的引用（或内联函数），用于在元素上建立指定事件类型的处理程序。就像在基本事件模型里那样，Event实例作为第一个参数传递到这个函数。

最后的参数useCapture是布尔类型的，稍后我们讨论第2级模型里事件传播的时候，再对其操作进行研究。目前把它设置为false。

92

为了利用DOM第2级事件模型，我们再次修改代码清单4-1的示例，并且只关注click事件类型。这次我们在图像元素上建立3个点击事件处理程序。新的示例代码可在文件chapter4/dom.2.events.html找到，内容如代码清单4-3所示。

代码清单4-3 利用DOM第2级模型建立事件处理程序

```
<html>
  <head>
    <title>DOM Level 2 Events Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js">
    </script>
    <script type="text/javascript">
      $(function(){
        var element = $('#vstar')[0];
        element.addEventListener('click',function(event) {
          say('Whee once!');
```

❶ 建立3个事件处理程序

```
        },false);
        element.addEventListener('click',function(event) {
          say('Whee twice!');
        },false);
        element.addEventListener('click',function(event) {
          say('Whee three times!');
        },false);
      });

      function say(text) {
        $('#console').append('<div>'+text+'</div>');
      }
    </script>
  </head>

  <body>
    <img id="vstar" src="vstar.jpg"/>
    <div id="console"></div>
  </body>
</html>
```

这段代码简单明白地说明我们能在同一个元素上为同一个事件类型建立多个事件处理程序。这是用基本事件模型无法轻松做到的事情。在页面的就绪处理程序❶中获取图像元素的引用，然后为click事件建立3个事件处理程序。

93

用标准兼容的浏览器（非IE）加载这个页面，并点击图像，显示结果如图4-3所示。

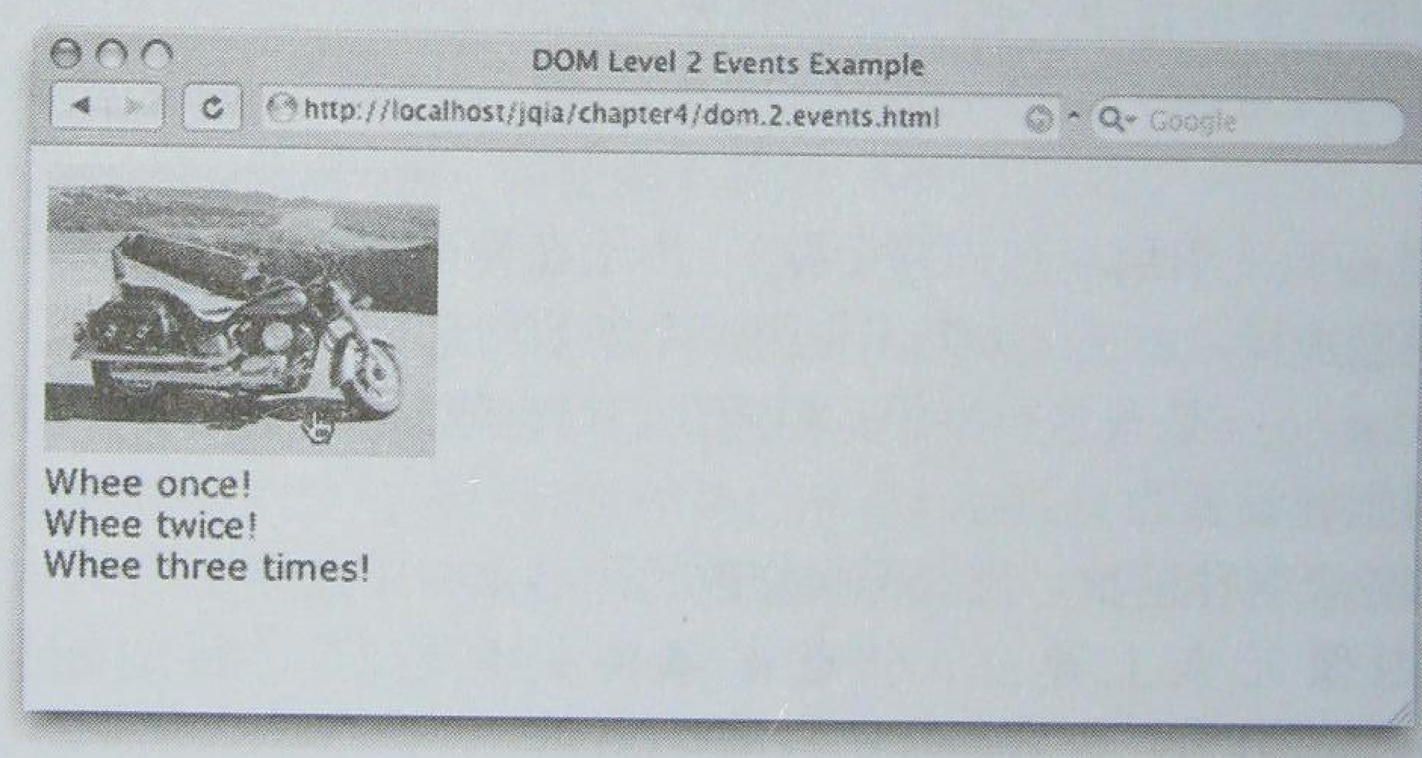


图4-3 点击一次图像，显示结果是已建立的所有3个click事件处理程序都被触发

请注意，即使这3个处理程序按建立时的顺序触发，这种顺序也不为标准所保证！代码的测试者没有观察到不同于建立时的顺序，但编写依赖于这种顺序的代码将是愚蠢的。请始终保持清醒的认识：在元素上建立的多个处理程序可能以随机顺序触发。

现在，那个useCapture参数怎么样？

2. 事件传播

正如前面所见，利用基本事件模型，一旦触发元素上的事件，事件就从目标元素沿着DOM树向上传播到所有祖先元素。高级的第2级模型不但提供冒泡阶段，而且提供附加阶段：捕获阶段。

在DOM第2级事件模型下，一旦事件被触发，事件首先从DOM树的根[1]向下传播到目标元素，然后再次从目标元素向上传播到DOM树的根。前者（根到目标）称为捕获阶段，而后者（目标到根）称为冒泡阶段。

当把函数建立为事件处理程序时，可以标志为捕获型处理程序，这种情况下事件在捕获阶段中触发；或者标志为冒泡型处理程序，事件在冒泡阶段中触发。正如此刻你所猜想的那样，函数addEventListener()的参数useCapture用来标识建立哪个类型的处理程序。参数值为false时，建立冒泡型处理程序；反之参数值为true时，建立捕获型处理程序。

回想一下代码清单4-2的示例，在那儿探索了基本事件模型通过DOM层次结构的事件传播。在那个示例中，我们将图像元素嵌入到两层 元素里。在这样的层次结构中，把<img>元素作为目标的点击事件，通过DOM树的传播路径如图4-4所示。

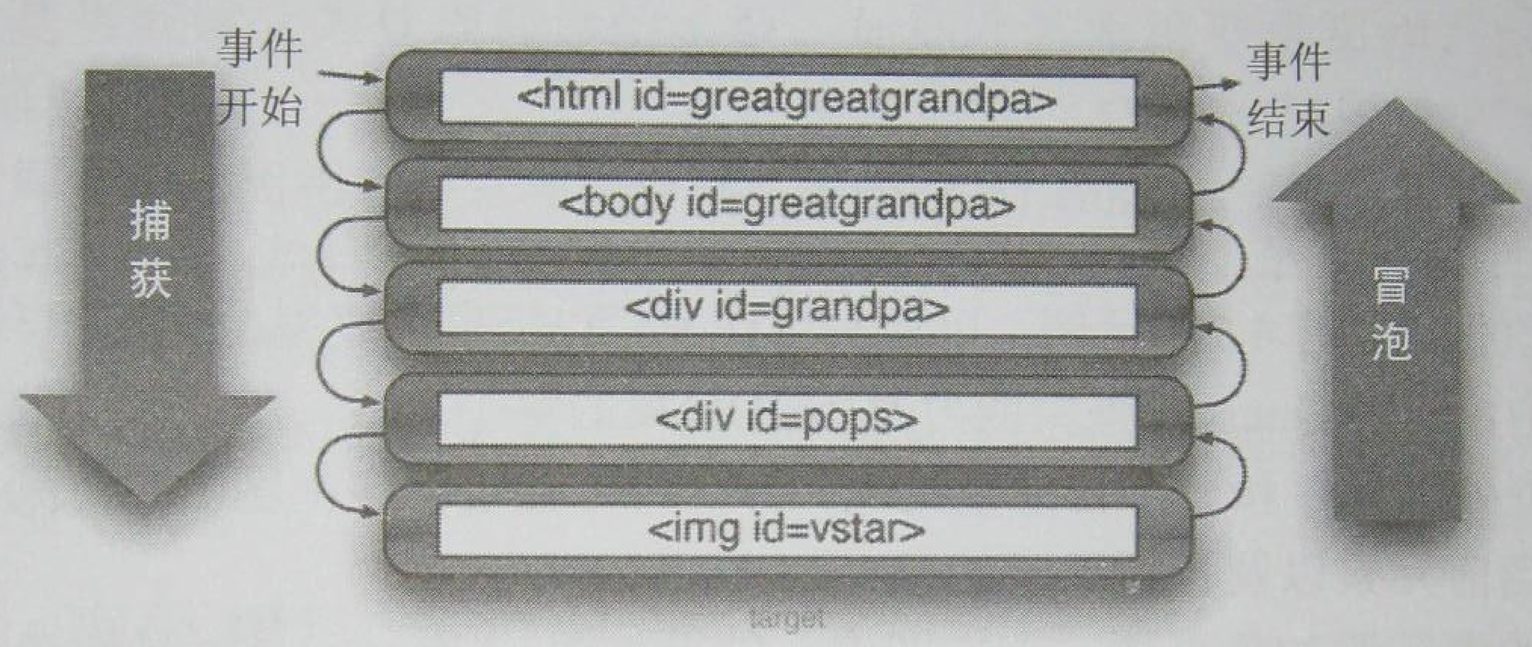


图4-4 在DOM第2级事件模型里，事件传播遍历DOM层次结构两次：一次在捕获阶段从顶到目标，另一次在冒泡阶段从目标到顶

动手测试一下吧，代码清单4-4是页面的代码。这个页面包含图4-4（chapter4/dom.2.propagation.html）的元素层次结构。

代码清单4-4 跟踪冒泡型和捕获型处理程序的事件传播

```
<html id="greatgreatgrandpa">
  <head>
    <title>DOM Level 2 Propagation Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $('*').each(function(){          ❶ 在所有元素上建立监听器
          var current = this;
          this.addEventListener('click',function(event) {
            say('Capture for ' + current.tagName + '#'+ current.id +
                ' target is ' + event.target.id);
            },true);
          this.addEventListener('click',function(event) {
            say('Bubble for ' + current.tagName + '#'+ current.id +
```

① 假定我们看到的DOM树根在上部。——译者注

```
                ' target is ' + event.target.id);
            },false);
        });
    });
    function say(text) {
      $('#console').append('<div>'+text+'</div>');
    }
  </script>
</head>

<body id="greatgrandpa">
  <div id="grandpa">
    <div id="pops">
      <img id="vstar" src="vstar.jpg"/>
    </div>
  </div>
  <div id="console"></div>
</body>
</html>
```

95

这段代码修改代码清单4-2的示例，以便利用DOM第2级事件模型API建立事件处理程序。在就绪处理程序❶中，利用jQuery的强大能力遍历DOM树的每个元素。在每个元素上建立两个处理程序：一个捕获型处理程序和一个冒泡型处理程序。每个处理程序都输出消息到控制台，标识处理程序的类型、当前元素以及目标元素的id。

用标准兼容的浏览器加载页面，点击图像，结果如图4-5所示，显示事件在不同处理阶段通过DOM树的传播情况。

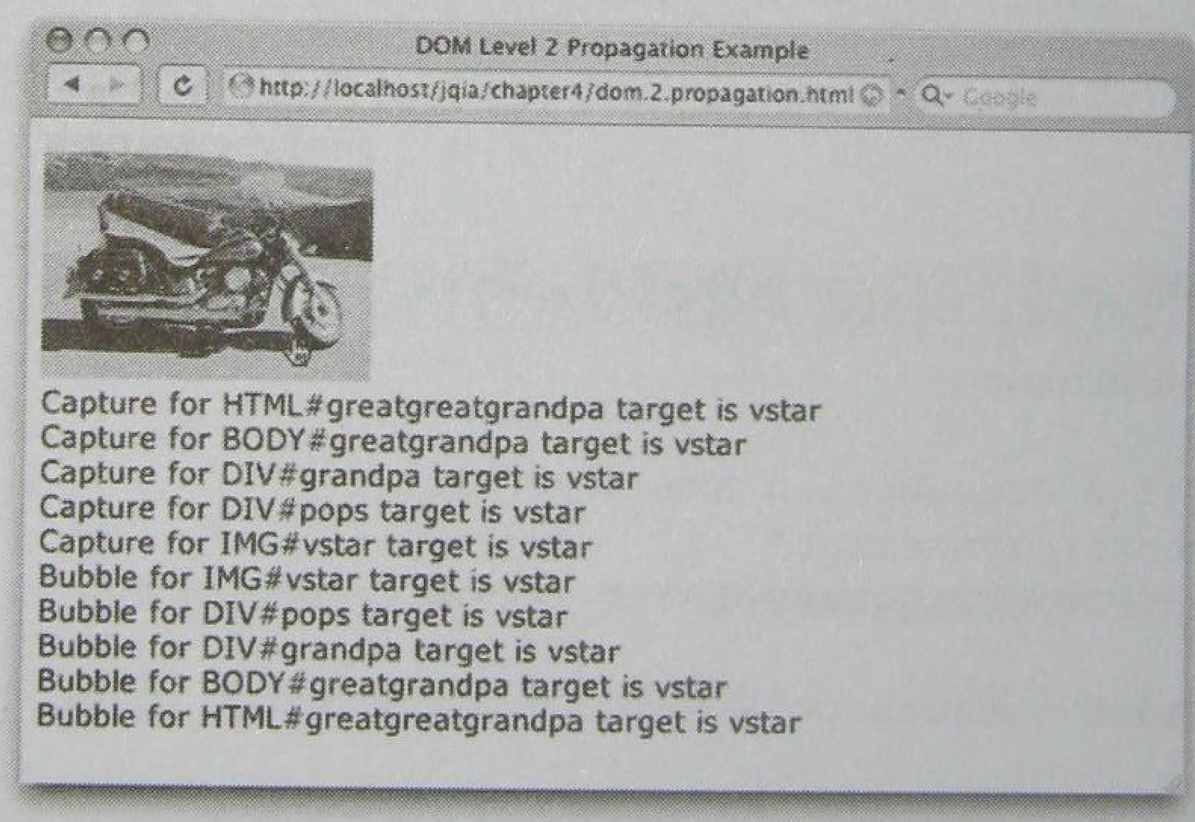


图4-5　点击图像，结果是每个处理程序都输出一条标识事件路径的消息到控制台

请注意，因为我们为目标元素建立了捕获型和冒泡型处理程序，所以对于目标元素和所有祖先节点，两个处理程序都已执行。

好，既然我们经历了所有的麻烦理解了以上知识，就应该认识到捕获型处理程序几乎从不在网页里使用。理由很简单：IE浏览器（依然莫名其妙地占据主导地位）不支持DOM第2级事件模型。虽然IE浏览器确实拥有与第2级标准的冒泡阶段相对应的专有模型，但它根本不支持捕获阶段。

在看jQuery如何帮忙收拾这个烂摊子之前，先简要地介绍IE模型。

96

4.1.3 IE 事件模型

IE浏览器（IE6以及最令人失望的IE7）对DOM第2级事件模型不提供支持。但微软浏览器的这两个版本提供与标准模型的冒泡阶段非常类似的专有接口。

IE模型为每个DOM元素定义名为`attachEvent()`的方法，而不是`addEventListener()`。这个方法接受类似于标准模型那样的两个参数，如下所示：

```
attachEvent(eventName,handler)
```

第1个参数是字符串，它命名将要附加的事件类型。不采用标准的事件名称，而是采用来自DOM第0级模型的相应元素属性的名称：`onclick`、`onmouseover`、`onkeydown`等。

第2个参数是将被建立为处理程序的函数。如同在基本模型里那样，`Event`实例必须从`window.event`属性获取。

97

4

多么混乱！即便是利用相对而立独立于浏览器的DOM第0级模型，还是要在事件处理各个阶段面临依赖于浏览器的众多选择的混乱状态并做出抉择。而当利用更为强大的DOM第2级或IE模型时，甚至一开始建立处理程序时就不得不创建代码分支。

好了，jQuery将竭尽全力把浏览器之间的不一致从我们的眼前隐藏起来，从而使我们的生活更加简单。下面看个究竟！

4.2 jQuery 事件模型

虽然富因特网应用的创建确实需要高度依赖于事件处理，但“处理浏览器差异时要大规模地编写事件处理代码”这种想法，甚至足以使最勇敢无畏的页面作者也变得气馁。

我们可以把差异从页面代码中抽象出来，从而把差异隐藏在API之后。但既然jQuery已为我们做好了这一切，为什么还要操心呢？

jQuery的事件实现，我们非正式地称为jQuery事件模型，它展示如下功能：

- 提供建立事件处理程序的统一方法；
- 允许在每个元素上为每个事件类型建立多个处理程序；
- 采用标准的事件类型名称，例如`click`或`mouseover`；
- 使`Event`实例可用作处理程序的参数；
- 对`Event`实例的最常用的属性进行规范化；
- 为取消事件和阻塞默认操作提供统一方法。

除了明显不支持捕获阶段以外，jQuery事件模型的功能集与第2级模型的功能集极为相似。尽管同时支持IE浏览器和标准兼容的浏览器，却只用同一个API。由于捕获阶段缺乏IE的支持，极大多数的页面作者从不使用捕获阶段（甚至不知道它的存在），因此忽略捕获阶段应该不成问题。

真的那么简单？我们去找出答案。

4.2.1 利用 jQuery 绑定事件处理程序

利用jQuery事件模型，凭借`bind()`命令可以在DOM元素上建立事件处理程序，考虑以下简

单示例：

```
$('img').bind('click',function(event){alert('Hi there!');});
```

这个语句为页面上的各图像绑定已提供的内联函数，作为点击事件处理程序。bind()命令的完整语法如下所示。

98

命令语法：bind

bind(eventType,data,listener)

在匹配集的所有元素上建立函数，作为指定事件类型的事件处理程序。

参数

eventType	（字符串）为将要建立的处理程序指定事件类型的名称。这个事件类型可以添加命名空间的后缀，后缀和事件名称之间以圆点分隔。本节的下文将讲述细节。
data	（对象）调用者提供的数据，作为属性data附加到Event实例，可供事件处理函数使用。如果省略，事件处理函数就被指定为第2个参数。
listener	（函数）将被建立为事件处理程序的函数。

返回

包装集

实践一下bind()函数。以代码清单4-3为例，把它从DOM第2级模型转换为jQuery模型，转换后代码如代码清单4-5所示，可在文件chapter4/jquery.events.html找到。

代码清单4-5　建立事件处理程序，无需浏览器特定代码

```
<html>
  <head>
    <title>DOM Level 2 Events Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $('#vstar')
          .bind('click',function(event) {
            say('Whee once!');
          })
          .bind('click',function(event) {
            say('Whee twice!');
          })
          .bind('click',function(event) {
            say('Whee three times!');
          });
      });

      function say(text) {
        $('#console').append('<div>'+text+'</div>');
      }
```

❶ 绑定3个事件处理程序到图像元素上

99

```
    </script>
  </head>

  <body>
    <img id="vstar" src="vstar.jpg"/>
    <div id="console"></div>
  </body>
</html>
```

这段代码的改动，仅限于就绪处理程序的函数体，虽然细微却意义重大❶。我们创建由目标<img>元素组成的包装集，并应用3个bind()命令到包装集——请记住，jQuery链允许在单个语句里应用多个命令——各bind()命令在元素上分别建立click事件处理程序。

用标准兼容的浏览器打开页面，并点击图像，显示结果如图4-6所示。毫不奇怪的是，和从图4-3看到的结果一模一样（除URL和窗口标题以外）。

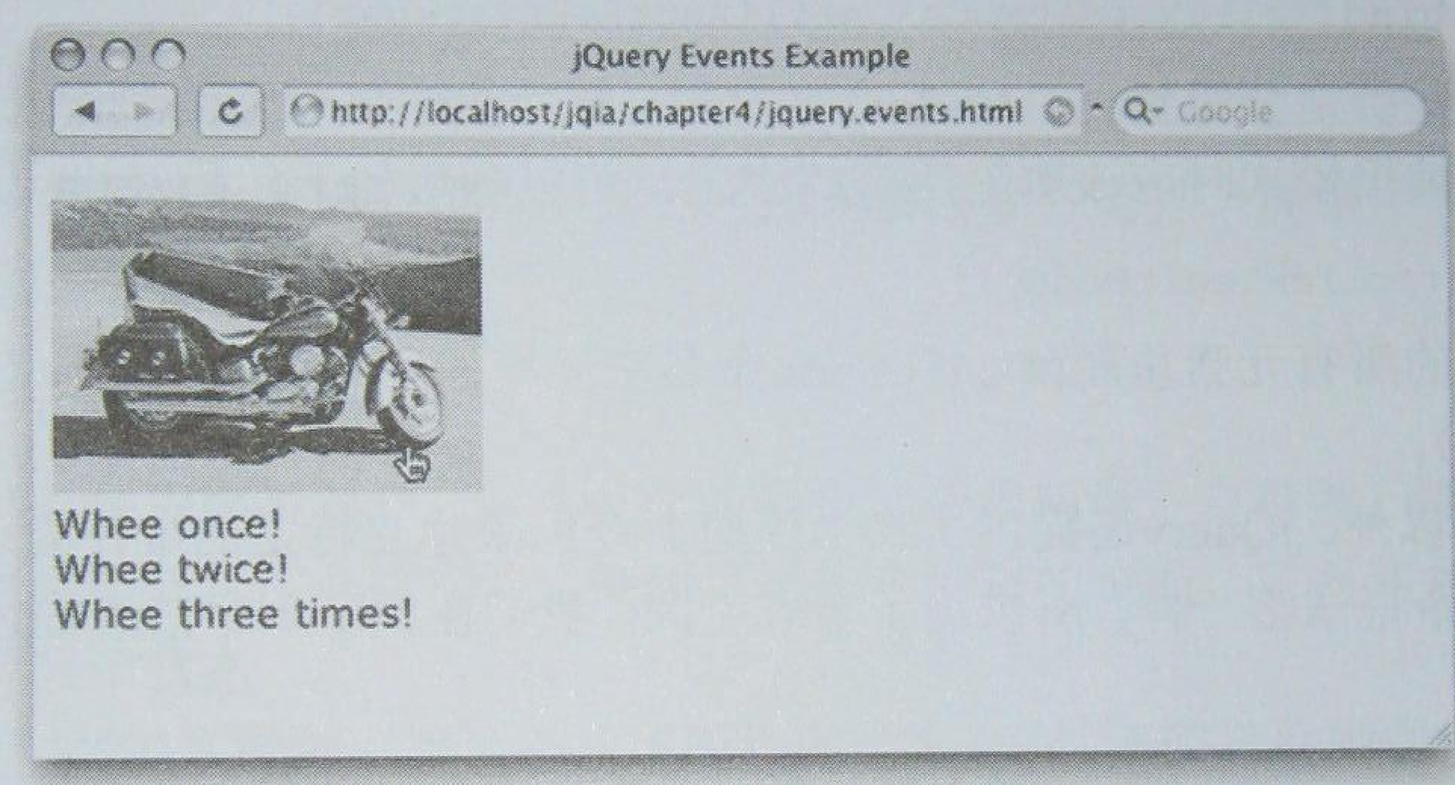


图4-6 如同在DOM第2级模型里那样，利用jQuery事件模型也可以指定多个事件处理程序

也许更为重要的是，即便用IE浏览器打开，页面仍然运行得很好，如图4-7所示。采用代码清单4-3的代码是无法做到的，除非添加大量的浏览器特定的测试和分支代码，以便为当前的浏览器选用正确的事件模型。

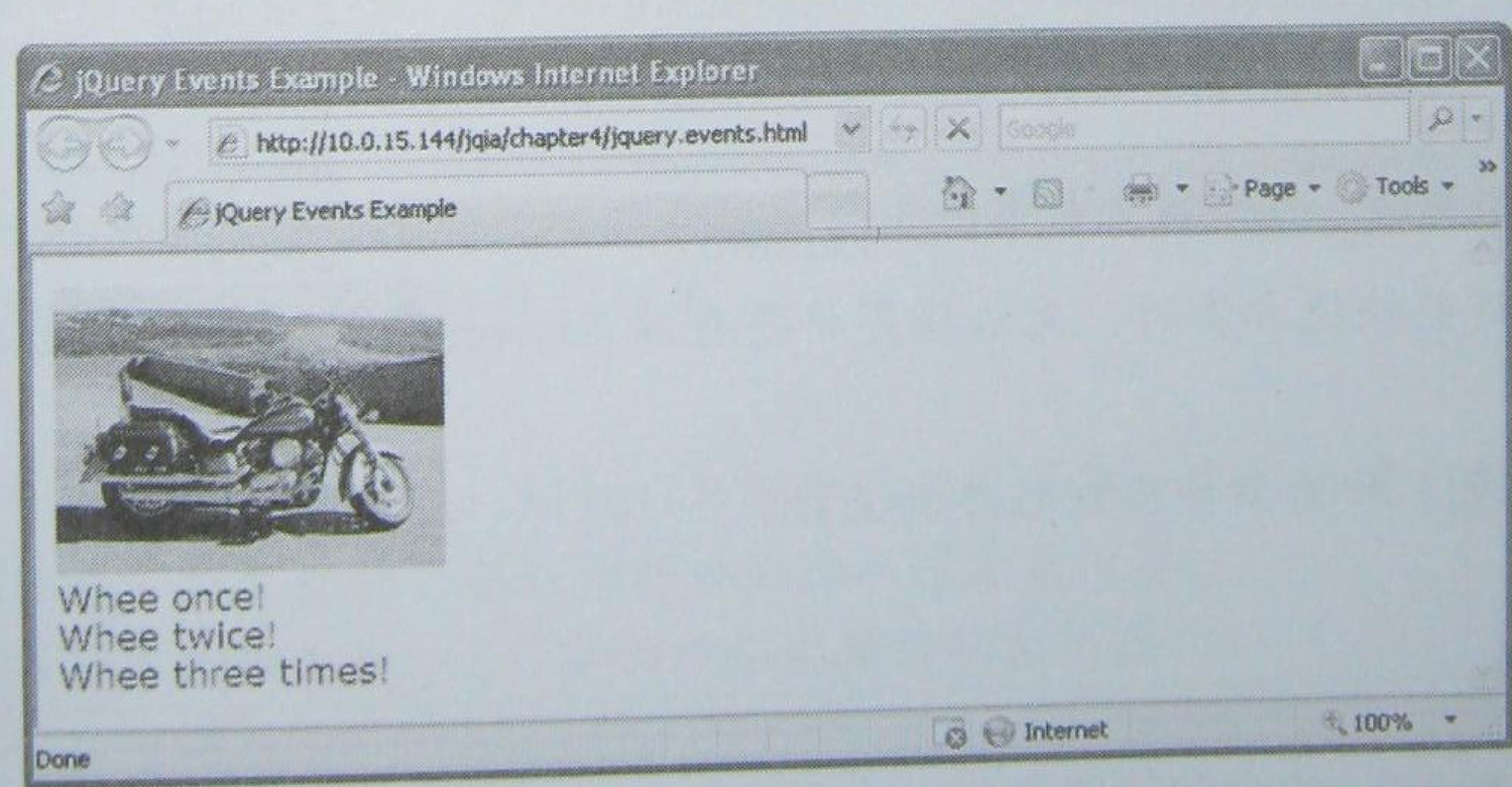


图4-7 jQuery事件模型允许利用统一的代码基础，支持IE里的事件处理

此时此刻，与页面上堆积如山的浏览器特定事件处理代码做过斗争的页面作者，无疑会一边感叹“快乐的日子又回来了”，一边坐在办公转椅上快乐地打转。谁又能责怪他们呢？

100

另一个jQuery提供的事件处理的小巧附赠品是，通过指派命名空间而使事件处理程序分组的能力。不同于指派命名空间的常规方式（通过前缀指派命名空间），这是通过在事件名称后面添加以圆点分隔的后缀来为事件名称指派命名空间。

通过这种方式使事件绑定分组，随后可以作为一个单元在它们上面进行简便的操作。

举个示例来说，页面拥有两个模式：显示模式和编辑模式。处于编辑模式时，事件监听器放在许多页面元素上，但这些监听器并不适合显示模式，因此页面从编辑模式迁出时必须删除。可以利用如下代码给编辑模式的事件指派命名空间：

```
$('#thing1').bind('click.editMode',someListener);
$('#thing2').bind('click.editMode',someOtherListener);
  ...
$('#thingN').bind('click.editMode',stillAnotherListener);
```

通过把所有这些绑定分组到editMode命名空间，随后我们可把它们当作一个单元进行操作。例如，当页面离开编辑模式时应该删除所有的绑定，可以利用如下代码轻松实现：

```
$('*').unbind('click.editMode');
```

这将从页面上的所有元素里删除editMode命名空间里的所有click绑定（unbind()方法的解释在下一节进行）。

101

除bind()命令以外，jQuery还提供了20个快捷命令用来建立特定的事件处理程序。因为除方法名称以外每个命令语法都一样，所以为了节省空间，我们在如下单个语法描述框内呈现所有这些命令。

命令语法：特定的事件绑定

***eventTypeName*(listener)**

建立已指定的函数，作为与方法同名的事件类型的事件处理程序。支持的命令如下：

blur	focus	mousedown	resize
change	keydown	mousemove	scroll
click	keypress	mouseout	select
dblclick	keyup	mouseover	submit
error	load	mouseup	unload

请注意，当利用这些快捷方法时，无法指定将要存储在event.data属性里的data值。

参数

listener （函数）作为事件处理程序的函数。

返回

包装集

102

jQuery也提供bind()命令的特殊版本，命名为one()，建立一次性的事件处理程序。事件处理程序执行第一次后，就作为事件处理程序而自动删除。语法类似于bind()命令，如下所示。

命令语法: one

one(eventType,data,listener)

在匹配集的所有元素上建立作为指定事件类型的事件处理程序的函数。一旦执行，处理程序就被自动删除。

参数

eventType （字符串）为将要建立的处理程序指定事件类型的名称。

data （对象）调用者提供的数据，作为名叫data的属性附加到Event实例，可供事件处理函数使用。如果省略，事件处理函数就被指定为第2个参数。

listener （函数）将被建立为事件处理程序的函数。

返回

包装集

这些命令给予我们绑定事件处理程序到已匹配元素的多种选择。而事件处理程序被绑定后，最终我们可能还要删除它。

4.2.2 删除事件处理程序

通常只要事件处理程序被建立，就在页面的剩余生命周期里一直有效。特殊的交互可能基于一定的标准规定事件处理程序必须删除。例如呈现多个步骤的页面，只要某个步骤完成，这个步骤的控件就还原到只读状态。

在这种情况下，在脚本控制下删除事件处理程序较为有利。我们已看到在处理程序第一次（且唯一一次）执行完成之后，one()命令能够自动删除处理程序。但在更为常见的情况下，我们想在自己的控制下删除事件处理程序[①]，于是jQuery提供了unbind()命令。

unbind()命令语法如下。

命令语法: unbind

unbind(eventType,listener)

unbind(event)

从包装集的所有元素中删除可选的已传递参数所指定的事件处理程序。如果不提供参数，则从元素中删除所有的监听器（即事件处理程序）。

参数

eventType （字符串）如果提供，则说明只删除为指定的事件类型而建立的监听器。

listener （函数）如果提供，则标识将要删除的特定监听器。

event （事件）删除触发Event实例所描述事件的监听器。

① 即在脚本控制下删除事件处理程序。——译者注

返回
包装集

这个命令用于从各个粒度级别的匹配集元素中删除事件处理程序。如果省略参数，就会删除所有的监听器；如果提供事件类型参数，就会删除特定类型的监听器。

通过提供初始化时作为监听器而建立的函数的引用，可以删除特定的监听器。要想使这成为可能，函数的引用必须最初在绑定函数作为事件监听器时就保留。为此，初始化时作为监听器而建立的函数（最终作为处理程序将被删除），要么定义为顶层函数（以便可以通过顶层变量名称取得函数的引用），要么通过其他方式保留函数的引用。如果提供函数作为匿名的内联引用，则使得随后在unbind()调用中无法引用该函数。

103

迄今为止，我们已看到jQuery事件模型使得易于建立（和删除）事件处理程序，而不用担心浏览器差异，但怎样编写事件处理程序呢？

4.2.3 Event 实例

当利用bind()命令建立的事件处理函数被调用时，Event实例作为第一个参数被传递给函数。这样就不必担心IE浏览器下的window.event属性，但怎样存取Event实例的有分歧的属性呢？

即便是利用jQuery建立处理程序，传递到事件处理程序的Event实例也还是浏览器所定义的本地对象的克隆。这意味着在标准兼容的浏览器里，Event实例将遵循标准的属性布局；在IE浏览器里，Event实例将使用专有的属性布局。在专有的Event实例传递到事件处理程序之前，jQuery竭力修正Event实例，以便使Event实例的最频繁存取的属性和方法遵循标准的格式。那么再一次，除了Event最生僻的属性以外，我们可以为事件处理程序编写代码，而无需关心浏览器平台。

表4-1显示以平台独立的方式可以安全存取的Event属性。

表4-1 安全的Event实例属性

属 性	描 述
altKey	当触发事件时，如果Alt键已被按下，设置为true；否则设置为false。在大多数Mac键盘上，Alt键标注为Option
ctrlKey	当触发事件时如果Ctrl键已被按下，设置为true；否则设置为false
data	如果有值的话，就在建立处理程序时，作为第二个参数传递到bind()命令的值
keyCode	对于keyup（键上）和keydown（键下）事件，返回被按下的键。请注意对于字母符号，不管用户输入的是大写还是小写字母，返回的都是字母的大写版本。例如，a和A都返回65。可以用shiftKey属性来确定输入的是大写还是小写。对于keypress（按键）事件，请使用which属性，因为which属性在跨浏览器时依然是可靠的
metaKey	当触发事件时，如果Meta键已被按下，设置为true；否则设置为false。Meta键就是PC机器上的Ctrl键，或Mac（苹果）机器上的Command键
pageX	对于鼠标事件，指定事件的相对于页面原点的水平坐标
pageY	对于鼠标事件，指定事件的相对于页面原点的垂直坐标

104

（续）

属　性	描　述
relatedTarget	对于某些鼠标事件，标识触发事件时光标离开或进入的元素
screenX	对于鼠标事件，指定事件的相对于屏幕原点的水平坐标
screenY	对于鼠标事件，指定事件的相对于屏幕原点的垂直坐标
shiftKey	当触发事件时如果Shift键已被按下，设置为true；否则设置为false
target	标识在哪个元素上事件被触发
type	对于所有的事件，指定已触发的事件的类型（例如click）。如果你利用单个事件处理函数来处理多个事件，这个属性就能派上用场
which	对于键盘事件，指定触发事件的键的数字编码；对于鼠标事件，指定按下的是哪个鼠标键（1为左、2为中、3为右）。应该利用which属性而不是button属性，因为不能指望button属性以一致的方式跨越多种浏览器

重要的是，keypress属性对于非字母符号在跨浏览器时是不可靠的。例如左箭头键的编码是37，在keyup和keydown事件发生时能可靠地运行。但Safari浏览器在keypress事件发生时，返回这些键的非标准的结果。

通过keypress事件的which属性可以获得可靠的、区分大小写的字符编码。通过keyup和keydown事件的which属性，只能获得不区分大小写的键编码（所以a和A都返回65），但可以通过检查shiftKey属性来确定大小写。

Event实例不仅包含关于被处理事件信息的属性，还拥有几个控制事件传播的方法。下面深入探讨这方面的知识。

4.2.4 影响事件传播

除了将Event实例最常用的属性标准化以外，jQuery还提供标准方法用于影响事件传播方面，带来同样的好处。

stopPropagation()方法将防止事件沿着DOM树向上传播（如果需要，请参考前面图4-4，回忆事件怎样传播），而preventDefault()方法将取消可能引起任何语义操作的事件。这种语义操作的示例是，<a>元素的href链接加载、表单提交以及click事件引起复选框的状态切换。

如果想在停止事件传播的同时，取消事件的默认行为，可以返回false作为监听器函数的返回值。

除允许以独立于浏览器的方式建立事件处理以外，jQuery还提供一组命令，赋予我们在脚本控制下触发事件处理程序的能力。下面探讨这些命令。

4.2.5 触发事件处理程序

在相关事件触发通过DOM层次结构的事件传播时，事件处理程序被设计为将被调用。但有时候我们想在脚本控制下触发处理程序的执行。可以把事件处理程序定义为顶层函数，以便通过名称调用它们。但是正如我们所见，把事件处理程序定义为匿名的内联函数更为常见，并且非常简便！

jQuery定义在脚本控制下自动触发事件处理函数的一系列方法，从而帮助我们避免使用顶层函数。这些命令中最通用的是trigger()，语法如下所示。

命令语法：trigger

trigger(eventType)

调用所有已匹配元素的、为已传递事件类型而建立的任何事件处理程序。

参数

eventType （字符串）指定将要调用的处理程序的事件类型名称。

返回

包装集

106

请注意，trigger()命令（还有待会儿介绍的简便命令）不会导致事件触发和通过DOM层次结构进行传播。由于没有跨浏览器的可靠办法用于生成事件，jQuery的trigger()命令把处理程序当作普通函数进行调用。

给每个被trigger()调用的处理程序，传入最低限度已填充的Event实例。因为没有事件，所以用于报告某些值（比如鼠标事件的位置）的属性也就没有值。target属性被设置为匹配集元素的引用，而这些匹配集元素已绑定被trigger()所调用的处理程序。

同样因为没有事件，所以没有发生事件传播。绑定到已匹配元素的处理程序将被调用，但不会调用这些元素的祖先节点上的处理程序。请记住，这些命令是调用事件处理程序的简便方法，而不是为了试着模拟事件。

除trigger()命令以外，jQuery为大多数的事件类型提供简便的命令。所有这些命令的语法几乎一样，除了命令的名称以外。语法描述如下。

命令语法：eventName

eventName()

调用为所有已匹配元素的指名事件类型而建立的任何事件处理程序。

已支持的命令如下：

blur

click

focus

select

submit

参数

无

返回

包装集

除binding（绑定）、unbinding（取消绑定）和triggering（触发）事件处理程序以外，jQuery还提供高级的函数，使得处理页面上的事件尽可能简便。

4.2.6 其他事件相关命令

经常有一般应用于富因特网应用页面的交互样式，是利用行为的组合来实现的。jQuery提供几个事件相关的简便命令，使得在页面上利用交互行为更加轻松。请看如下命令。

107

1. 起切换作用的监听器

第一个简便命令是toggle()命令，建立一对click事件处理程序，每当发生click事件时，就彼此交换。

命令语法：toggle

toggle(listenerOdd,listenerEven)

把已传递函数建立为包装集所有元素的一对click事件处理程序，每当触发click事件就相互切换。

参数

listenerOdd　一个函数，充当所有奇数次点击的click事件处理程序（第1次、第3次、第5次……）。

listenerEven　一个函数，充当所有偶数次点击的click事件处理程序（第2次、第4次、第6次……）。

返回

包装集

这个简便命令的常见用途是，根据元素被点击的次数切换元素的启用状态。可以利用前面示例的图像元素去模拟，改变其不透明度以便反映启用（完全不透明）或禁用（半透明）状态。可以动手做一个示例，切换文本输入控件的只读状态，但这对于演示目的而言，不如可视化语句解释得清楚。下面利用图像示例进行模拟。

图4-8显示的是包含图像的页面随着时间推移而展示的3个状态：

(1) 初始显示；

(2) 点击一次图像之后；

(3) 再次点击图像之后。

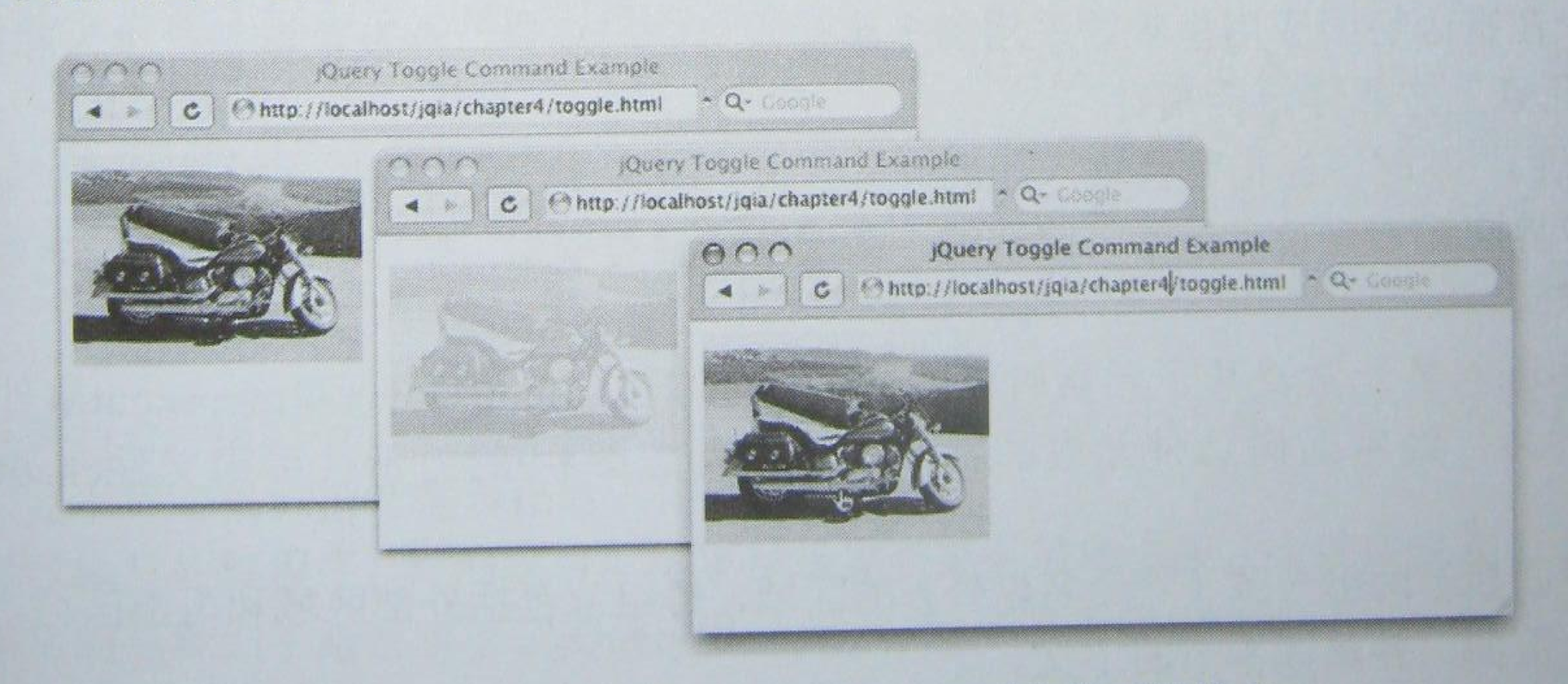


图4-8　toggle()命令使得易于切换图像的可视状态

108 这个示例的代码如代码清单4-6所示，并可在文件chapter4/toggle.html找到。

代码清单4-6 每当点击事件发生时，调用互补的监听器

```
<html>
  <head>
    <title>jQuery Toggle Command Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $('#vstar').toggle(
          function(event) {
            $(event.target)
              .css('opacity',0.4);
          },
          function(event) {
            $(event.target)
              .css('opacity',1.0);
          }
        );
      });
    </script>
  </head>

  <body>
    <img id="vstar" src="vstar.jpg"/>
  </body>
</html>
```

❶ 把toggle()命令应用到图像元素上

❷ 奇数监听器使图像变灰

❸ 偶数监听器使图像还原为完全的不透明度

109

在这个示例中，我们把toggle()命令❶应用到图像上，提供奇数监听器❷用于把不透明度值减小到0.4（使图像变灰，“变灰”是用来表示禁用的常见术语）；提供偶数监听器❸用于把不透明度还原到最大值1.0。因为toggle()命令处理所有的切换，所以我们不必费力地跟踪当前点击是奇数次还是偶数次。多么方便！

在这个示例中我们完成的是图像在完全不透明和半透明状态之间的切换，但很容易就联想到提供切换任何互补状态的监听器，例如启用与禁用。

另一个在富因特网应用里频繁采用的多事件常见应用场景，涉及鼠标指针进入和离开元素。

2. 在元素上方悬停鼠标指针

通知我们“某时鼠标指针已进入或离开某区域”的事件，通常对于在页面上建立呈现给用户的许多用户界面元素来说，是不可或缺的。在这些元素类型中，用作Web应用导航系统的菜单就是常见的示例。

当鼠标进入某元素及其子元素时触发mouseout事件。mouseover和mouseout事件类型的这种恼人的行为，经常妨碍这种元素的轻松创建。考虑图4-9的显示（可用文件chapter4/hover.html）。

这个页面显示两组一致（除命名以外）的区域，每组都包括外部区域和内部区域。请用浏览器加载这个页面，并阅读本节的下文。

110

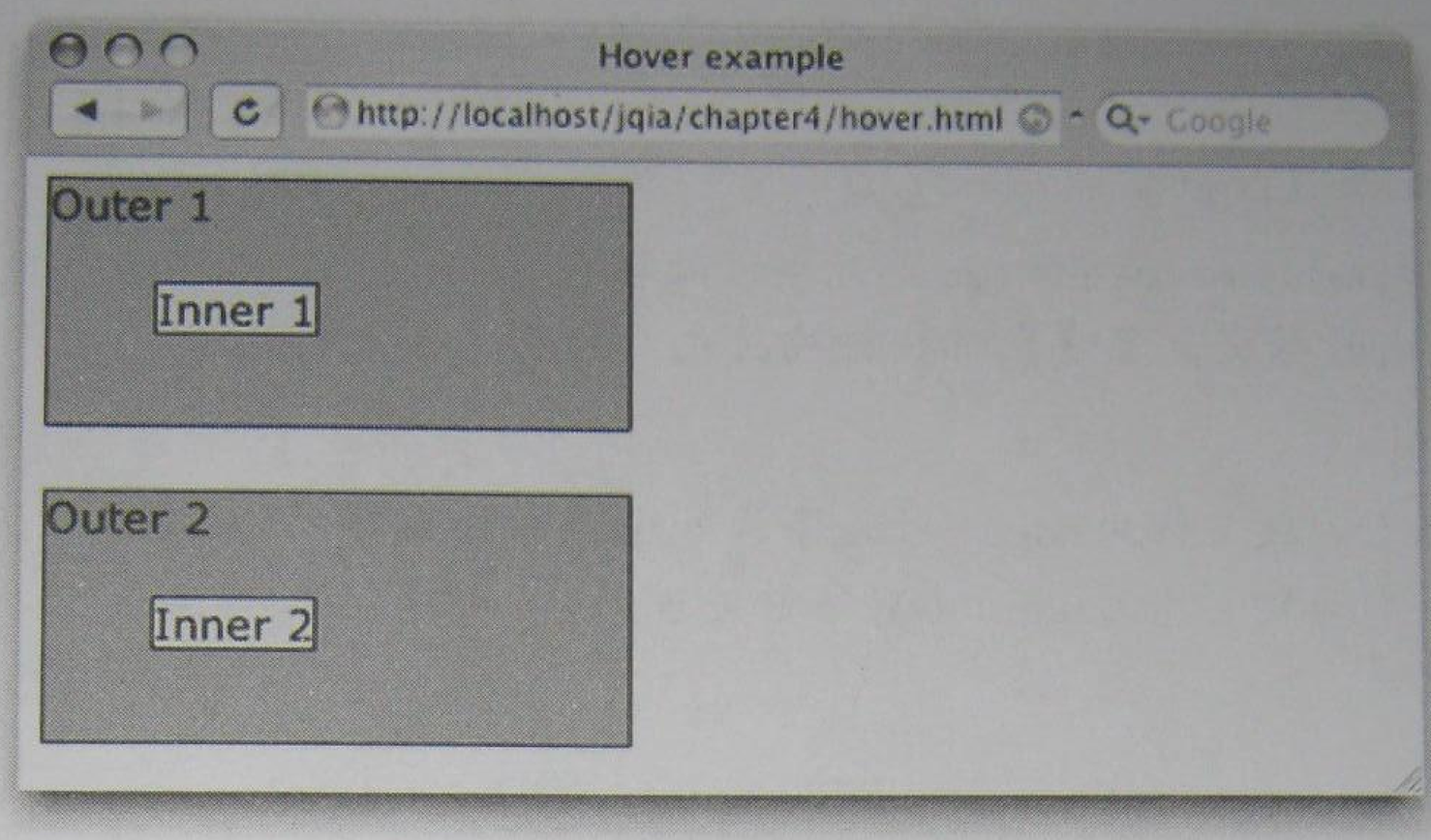


图4-9 这个页面演示当鼠标指针进入某区域及其子区域时触发的鼠标事件

对于上面的那组区域，在就绪处理程序里建立鼠标事件处理程序的脚本如下：

```
$('#outer1')
 .bind('mouseover',report)
 .bind('mouseout',report);
```

这个语句建立名为report的函数作为mouseover和mouseout事件的事件处理程序。report()函数的定义如下：

```
function report(event) {
  $('#console').append('<div>'+event.type+'</div>');
}
```

这个监听器不过是把包含已触发事件名称的 元素，添加到定义在页面下部的名为console的 元素里，以便让我们看到何时触发各个事件。

现在，移动鼠标指针进入标注为Outer 1的区域（小心点，别进入Inner 1）。我们将看到（从页面的下部看到）mouseover事件触发。移动鼠标指针离开Outer 1区域。不出所料，我们将看到mouseout事件触发。

刷新页面以便清空控制台，重新开始。

现在，移动鼠标指针进入Outer 1（请注意事件），但这次继续进入到Inner 1。当鼠标指针进入Inner 1时，Outer 1的mouseout事件触发。如果在内部区域上方晃动鼠标指针，就会看到一阵猛增的mouseout 和mouseover事件。这是预料之中的行为。即便仍处在Outer 1的范围之内，当鼠标指针进入内部元素时，事件模型也把从Outer 1移动到内部元素，看作是离开外部区域。

不管料到与否，我们不一定想要那样的行为。当鼠标指针离开外部区域的边界时我们通常想要被通知，而不关心鼠标指针是否已进入内部区域。

我们可以编写处理程序，检测何时鼠标事件是离开某区域的结果，或只是进入了内部区域的结果，但幸运的是我们不必费事了。jQuery用hover()命令帮了我们大忙。

这个命令的语法如下所示。

命令语法：hover

hover(overListener,outListener)

建立已匹配元素的mouseover和mouseout事件处理程序。这些处理程序当且仅当元素所覆盖区域被进入和退出时触发，忽略鼠标指针从父元素到子元素上的迁移。

参数

overListener　（函数）作为mouseover事件处理程序的函数。

outListener　（函数）作为mouseout事件处理程序的函数。

返回

包装集

我们在hover.html示例页面上，使用下列脚本建立第二组区域（Outer 2及其Inner 2子区域）的鼠标事件处理程序：

```
$('#outer2').hover(report,report);
```

与第一组区域类似，report()函数被建立为Outer 2的mouseover和mouseout事件的处理程序。但与第一组区域不同的是，当鼠标指针通过Outer 2和 Inner 2之间的边界时，不会调用两个处理程序的任何一个。在不必对鼠标指针从子元素上通过而做出反应的情况下，这就非常有用。

拥有所有这些事件处理工具，让我们利用迄今为止所学的知识，研究一下示例页面。该页面利用了事件处理，还利用了在前面章节所学的其他jQuery技巧。

4.3　让事件（以及更多）工作起来

既然已经探讨了jQuery如何消除处理跨浏览器迥异事件模型时的混乱并重建秩序，下面就研究一个示例页面，该页面综合运用了迄今为止我们所学的知识——不仅利用事件，还利用前面几章已探讨的jQuery技巧，包括重量级的jQuery选择器。

本节所创建的页面是名为Bamboo Asian Grille的亚洲餐馆的在线点菜表单的一小部分。为简洁起见，限定在菜单的开胃菜这一节，但你可以把经验教训应用到菜单的其余部分，完成这些才能充分实践你的jQuery技能。

这个示例的目标看起来简单：允许用户选择他们想要的开胃菜的类型和数量以便添加到点菜单。没问题，对吧？一系列的复选框和文本框将干得很漂亮。作为预料中的GUI元素，它们可供做出多项选择和指定数量。

但有个小问题：对于每样开胃菜，必须显示其他子选项。例如当选择Crab Rangoon（炸蟹角）时，用餐者可以选择子选项酸甜酱、辣芥末、或者（对于那些无法做出决定的人来说）二者。这也应该不成问题，因为我们可以把每个开胃菜条目与表示子选项的一系列单选按钮关联起来。

但事实证明，这确实会导致一个小问题。利用少量HTML编码和一些CSS技巧，我们创建如图4-10所示的页面布局。

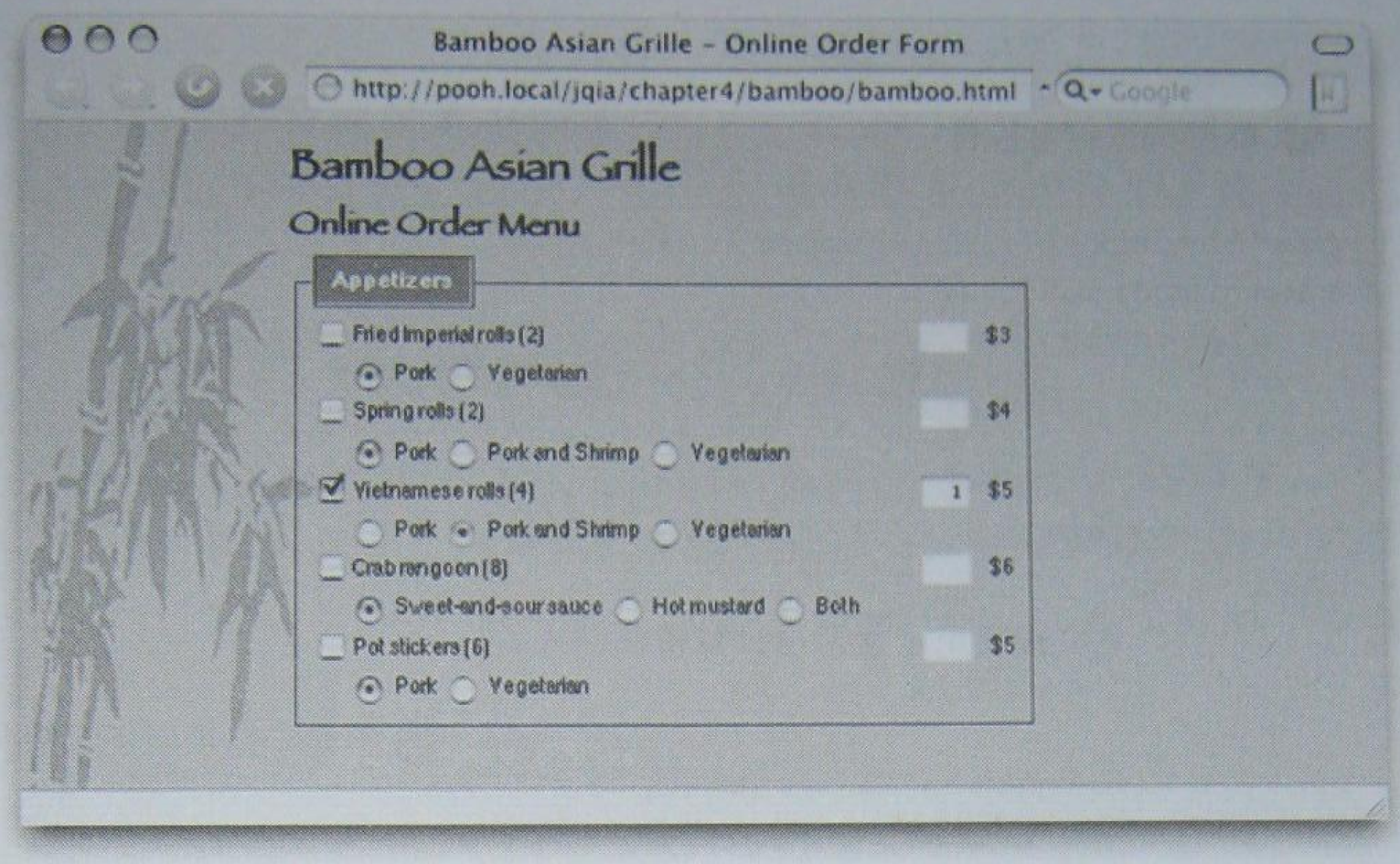


图4-10 所有的开胃菜和子选项都显示了，但屏幕却变得乱七八糟

即使只有5种开胃菜及其相应的子选项，控件的数目就已多得人喘不过气来，甚至可能难以看清迄今为止用餐者所做出的选择。虽然表单按照要求运行，但其可用性还有许多待改进之处。

通过应用称为“渐进式公开（progressive disclosure）”的原则，可以解除这个可用性困境。我们不必为用户当前没有选中的开胃菜显示子选项，因此隐藏单选按钮直到用户需要看到它们。

通过减少易混淆的杂项，根据需要渐进地公开信息将大大地提高表单的可用性，如图4-11所示。

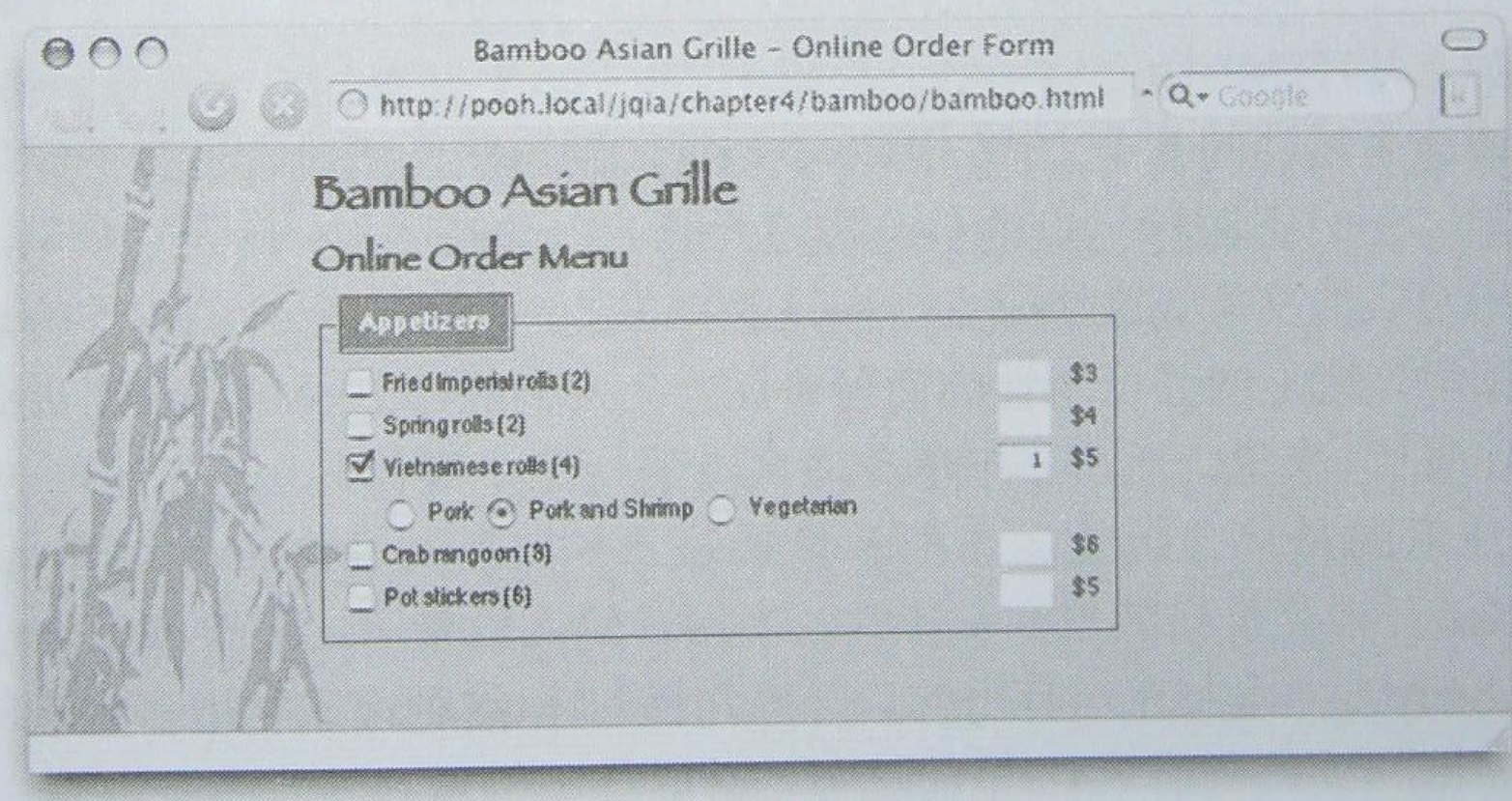


图4-11 隐藏子选项直到需要时才显示，以此减少混淆和混乱，用户也不会被不相关控件弄得喘不过气来

作为附带的好处，我们也设置控件，以便当饥肠辘辘的用户输入数量时，显示的金钱数目能反映与数量相对应的价钱。看看如何实现吧。

这个示例的完整页面代码可从文件chapter4/bamboo/bamboo.html获得。首先看用来显示一个开胃菜条目的HTML结构，如代码清单4-7所示。

代码清单4-7　一个开胃菜条目的HTML结构

```
<div>
  <label>
    <input type="checkbox" name="appetizers"
           value="imperial"/>
    Fried Imperials rolls (2)
  </label>
  <span price="3">
    <input type="text" name="imperial.quantity"
           disabled="disabled" value="1"/>
    $<span></span>
  </span>
  <div>
    <label>
      <input type="radio" name="imperial.option"
             value="pork" checked="checked"/>
      Pork
    </label>
    <label>
      <input type="radio" name="imperial.option"
             value="vegetarian"/>
      Vegetarian
    </label>
  </div>
</div>
```

❶ 标签构造包含控件

❷ 利用自定义特性保持价格数据

❸ 预留用于显示已计算价钱的地方

❹ 包含子选项以便有条件地显示

我们为开胃菜各条目重复这个HTML结构。请注意，这个片段不包含视觉呈现信息。这样的信息已经提出来放到CSS定义文件（可在文件bamboo.css找到，与HTML文件处在同一个文件夹）。

类似地，请注意没有脚本嵌入HTML标记里。页面的行为将遵守“不唐突的JavaScript”原则而定义，所有的脚本从HTML标记里完全分离出来。

我们应该强调这个结构的几个方面，因为添加行为到这些元素时它们变得重要起来。首先请注意，复选框元素（还有下面标记里的单选按钮元素）包含在`<label>`元素❶里，而`<label>`元素也包含与控件相对应的文本。这使得点击标签的文本就能切换已包含的控件的选中状态，就像用户点击了控件那样。这是增强可用性的简便办法，让人们能够更轻松地利用页面（特别是对所谓的“草率的点击者[①]”来说，至少本书的作者之一属于这个群体）。

另一个值得注意的是包含数量文本框和价钱显示的`<span>`元素❷。这个元素被饰以名为price的、用于存储开胃菜价格的自定义特性。当用户输入数量时，需要利用价格计算价钱，而price特性也在jQuery选择器中充当有用的选择器句柄[②]（以这种方式利用自定义的特性，一些人认为是有争议的。请阅读代码清单4-7里的注释，获取更多信息）。

也请注意，为了包含已计算的价钱而创建的`<span>`元素，初始为空❸。我们可以仅静态地填充`<span>`元素，但那意味着我们在两个地方存储价格信息——通常被认为并非最优方法。随后我们查看作为页面初始化行为的一部分，如何填充这个元素。

---

① 指大大咧咧、粗手粗脚的用户，他们在点选页面上的控件时往往要好几次才能选中。——译者注

② 例如'span[price] input[type=text]'匹配带有price特性的span下的text类型的input元素。——译者注

在标记里最后的主要构造是 元素❹，包含着表示开胃菜子选项的单选按钮。这个元素会保持隐藏状态直到一个开胃菜复选框被选中。

既然所有标记已经定下来，就让我们一步一步地开发页面的行为。首先看怎样隐藏单选按钮选项的容器元素。

115

自定义特性：英勇或是可憎

使用非HTML或非XHTML规范所定义的自定义特性去装饰DOM元素，这种做法既有支持者也有批判者。支持者认为，这是对HTML和浏览器所提供的工具的充分利用；批判者认为，这是对所有美好事物的冒犯，因为利用自定义特性会妨碍页面通过有效性测试。

本书作者在这个问题上不偏袒任何一方，而把决定权留给读者。你可以决定自己的主张：利用自定义的特性是有用的机制还是页面的脸上瑕疵。

如果不使用price特性，价格数据可以存储在一个JavaScript变量中，这个变量包含关联开胃菜名称（例如imperial）及其价格的散列对象[1]。

自定义特性策略可以说是优越于JavaScript变量机制，因为添加新的开胃菜条目，意味着添加新的自成一体的构造到页面上，而如果采用JavaScript变量机制的话，就必须记得用新添加的开胃菜的价格更新散列对象。

再一次，我们让你自行决定最适合你自己的办法。

检查每个开胃菜条目的HTML结构，我们能够编写匹配 元素的选择器，并在这些元素上调用hide()命令，如下所示：

```
$('fieldset div div').hide();
```

注意 我们最初可以利用CSS隐藏这些元素，但用脚本实现的话，可以确保关掉JavaScript功能的用户（是的，还有人那样做）也能看到可用的界面，尽管显示页面时进行要付出计算加载的代价。还有其他的理由使我们在就绪处理程序里用脚本实现元素的隐藏，在第5章里当我们讨论包装器方法（如hide()）的细节时再加以探讨。

隐藏开胃菜子选项，留后显示，现在把注意力转移到启用由元素展示的可用性行为。首先把开胃菜关联的单选按钮子选项的显示与否，与开胃菜的选中与否关联起来。

为了对开胃菜复选框状态的变化做出反应，也就是触发包含子选项的 元素可见性的变化，我们在复选框上建立click事件的监听器，通过监听器可以根据复选框的状态来调整子选项的可见性。请看下列建立监听器的语句：

116

```
$(':checkbox').click(function(){
  var checked = this.checked;
  $('div',$(this).parents('div:first'))
```

① 本书中的“散列对象”指“散列关联数组”，例如var myhash = {imperial:3,spring:4,vnrolls:5};
——译者注

```
    .css('display',checked ? 'block':'none');
  $('input[type=text]',$(this).parents('div:first'))
    .attr('disabled',!checked)
    .css('color',checked ? 'black' : '#f0f0f0')
    .val(1)
    .change()
    .each(function(){ if (checked) this.focus();});
});
```

所有这些仅仅为了隐藏和显示一个 ?

噢，不是。隐藏和显示子选项，只是其中一个复选框的状态改变时我们必须做的事情之一。仔细观察这个片段的每一个步骤，看它做了些什么。

首先，click处理程序将复选框的选中状态存储到名为checked的变量，这样易于在代码里引用，并且它建立了可在我们创建的任何闭包里使用的本地变量。

下一步，处理程序找到包含开胃菜子选项的 ，并设置其CSS display值：当复选框未选中时隐藏子选项，当复选框选中时显示子选项。用来找到将被隐藏或显示的元素的jQuery表达式是$('div',$(this).parents('div:first'))，等同于“在this的第一个 祖先元素内查找 元素”。我们从HTML结构得知只有一个匹配，所以不必进一步区分。

精明的你将注意到，最初复选框处于未选中状态而子选项处于隐藏状态，所以通过toggle()命令可以编写更少的代码，而不必查询复选框的状态。这种带有假设条件的代码是脆弱的，当假设条件改变时容易崩溃，因此更好的做法通常是显式地确保子选项的可见性与各自复选框的状态相匹配。

这个处理程序尚未完成，它仍然需要调整数量文本框的状态。这些数量文本框最初是禁止的，当且仅当一个开胃菜被选中的时候才启用。我们利用 $('input[type=text]',$(this).parents('div:first'))找到数量文本框，意思是“在this的第一个 祖先元素内找到text类型的<input>元素”，与我们刚才使用的选择器相似。

17

对这个元素进行以下操作。

- 为了与复选框的状态切换相对应，利用attr()命令设置元素的禁用状态。
- 应用一个CSS颜色值，当控件[①]禁止时使文本不可见（请注意，这并非在所有的浏览器上都起作用，比如Opera和IE浏览器就不允许覆盖已禁止的文本字段的颜色）。
- 设置值为1。当启用控件时，设置默认值为1；当禁止控件时，还原为默认值。
- 调用文本框的change处理程序（我们还没有定义这个处理程序，不过别担心，因为下一步就做这件事）。change处理程序将计算开胃菜的价钱并加以显示。因为我们默认修改文本框的值（为1），所以必须调用这个处理程序，确保价钱显示正确。
- 调用each()方法获取元素的引用，如果对应的复选框处于选中状态，就把焦点放在那个元素上。你不正好喜欢允许存取本地变量checked的闭包吗？

① 即上文提到的“text类型的<input>元素”。控件，是指启用用户交互或输入的一个对象，经常用于初始化一个操作、显示信息或设置值。——译者注

注意 当琢磨处理复选框的哪个事件类型时，你可能最初想到捕捉change事件而不是click事件。为了实现我们的方案，当复选框的状态改变时我们必须立即得到通知。立即的通知发生在Safari和基于Mozilla的浏览器里，但IE浏览器并不触发change事件，直到焦点从控件（复选框）上离开之后，因此在这个示例中不适用change事件。取而代之，我们利用click事件。

现在把注意力转移到文本框的change处理程序。当文本框的值改变时，我们想要重新计算和显示开胃菜的价钱——这是用开胃菜的价格乘以数量的简单计算。

添加change处理程序的语句如下所示：

```
$('span[price] input[type=text]').change(function(){
  $('~ span:first',this).text(
    $(this).val() *
    $(this).parents("span[price]:first").attr('price')
  );
});
```

118

在找到文本框之后（利用选择器，意思是“在拥有price特性的元素内查找text类型的所有<input>元素”），我们指派change处理程序，用来查找要更新的元素并把它的文本内容设置为已计算的值。表达式 $('~span:first',this)找到this的第一个兄弟元素。获取文本框的值，乘以父元素的price特性的值便可完成计算。

如果以上任何一个相当高级的选择器表达式让你挠头的话，就该好好复习一下第2章所讲述的选择器语法了。

在让用户和页面交互之前，我们还有一件要做的事情。记得我们留空元素用来包含已计算的值吗？现在正是填充已计算值的时候。

数量文本框的值预先设置为1，因此我们必须做的就是，当数量文本框的值改变时执行相同的计算。但我们不想重复任何代码，于是触发文本框上的change处理程序让它进行处理。

```
$('span[price] input[type=text]').change();
```

就这样，我们迅速完成了开胃菜点菜表单——至少到目前为止已实现前面规定的目标。这个示例使我们获取一些非常重要的经验。

- 它显示如何在元素上建立click和change处理程序，用于在事件触发时按照我们的意愿实现用户界面的任何改变。
- 我们看到如何在脚本控制下触发处理程序，既避免重复的代码，又避免提取公共代码到全局的命名函数。
- 我们接触到一些强大而灵巧的选择器，用来选择想要操作的元素。

代码清单4-8完整地显示这个页面的代码。

代码清单4-8 开胃菜点菜表单的完整代码

```
<html>
  <head>
    <title>Bamboo Asian Grille - Online Order Form</title>
    <link rel="stylesheet" type="text/css" href="bamboo.css">
```

```
<script type="text/javascript"
        src="../../scripts/jquery-1.2.1.js"></script>
<script type="text/javascript">
  $(function(){
  $('fieldset div div').hide();
  $(':checkbox').click(function(){
    var checked = this.checked;
    $('div',$(this).parents('div:first'))
      .css('display',checked ? 'block':'none');
    $('input[type=text]',$(this).parents('div:first'))
      .attr('disabled',!checked)
      .css('color',checked ? 'black' : '#f0f0f0')
      .val(1)
      .change()
      .each(function(){ if (checked) this.focus();});
  });
  $('span[price] input[type=text]').change(function(){
    $('~ span:first',this).text(
      $(this).val() *
      $(this).parents("span[price]:first").attr('price')
    );
  });
  $('span[price] input[type=text]').change();
 });
</script>
</head>

<body>
  <h1>Bamboo Asian Grille</h1>
  <h2>Online Order Menu</h2>
  <fieldset>
    <legend>Appetizers</legend>

    <div>
      <label>
        <input type="checkbox" name="appetizers"
               value="imperial"/>
        Fried Imperials rolls (2)
      </label>
      <span price="3">
        <input type="text" name="imperial.quantity"
               disabled="disabled" value="1"/>
        $<span></span>
      </span>
      <div>
        <label>
          <input type="radio" name="imperial.option"
                 value="pork" checked="checked"/>
          Pork
        </label>
        <label>
          <input type="radio" name="imperial.option"
                 value="vegetarian"/>
```

```
        Vegetarian
      </label>
    </div>
  </div>

  <div>
    <label>
      <input type="checkbox" name="appetizers" value="spring"/>
      Spring rolls (2)
    </label>
    <span price="4">
      <input type="text" name="spring.quantity"
             disabled="disabled" value="1"/>
    $<span></span>
  </span>
  <div>
    <label>
      <input type="radio" name="spring.option" value="pork"
             checked="checked"/>
      Pork
    </label>
    <label>
      <input type="radio" name="spring.option"
             value="shrimp"/>
      Pork and Shrimp
    </label>
    <label>
      <input type="radio" name="spring.option"
             value="vegetarian"/>
      Vegetarian
    </label>
  </div>
</div>

<div>
  <label>
    <input type="checkbox" name="appetizers" value="vnrolls"/>
    Vietnamese rolls (4)
  </label>
  <span price="5">
    <input type="text" name="vnrolls.quantity"
           disabled="disabled" value="1"/>
    $<span></span>
  </span>
  <div>
    <label>
      <input type="radio" name="vnrolls.option" value="pork"
             checked="checked"/>
      Pork
    </label>
    <label>
      <input type="radio" name="vnrolls.option"
             value="shrimp"/>
```

120

121

```
        Pork and Shrimp
      </label>
      <input type="radio" name="vnrolls.option"
             value="vegetarian"/>
      <label>Vegetarian</label>
    </div>
  </div>

  <div>
    <label>
      <input type="checkbox" name="appetizers" value="rangoon"/>
      Crab rangoon (8)
    </label>
    <span price="6">
      <input type="text" name="rangoon.quantity"
             disabled="disabled" value="1"/>
      $<span></span>
    </span>
    <div>
      <label>
        <input type="radio" name="rangoon.option"
               value="sweet checked="checked"/>
        Sweet-and-sour sauce
      </label>
      <label>
        <input type="radio" name="rangoon.option" value="hot"/>
        Hot mustard
      </label>
      <label>
        <input type="radio" name="rangoon.option" value="both"/>
        Both
      </label>
    </div>
  </div>

  <div>
    <label>
      <input type="checkbox" name="appetizers"
             value="stickers"/>
      Pot stickers (6)
    </label>
    <span price="5">
      <input type="text" name="stickers.quantity"
             disabled="disabled" value="1"/>
      $<span></span>
    </span>
    <div>
      <label>
        <input type="radio" name="stickers.option"
               value="pork" checked="checked"/>
        Pork
      </label>
      <label>
        <input type="radio" name="stickers.option"
```

122

```
                value="vegetarian"/>
            Vegetarian
          </label>
        </div>
      </div>

      <div></div>

    </fieldset>
  </body>
</html>
```

这段代码是健壮的，因为它独立于开胃菜条目的数目。你将注意到没有必要在JavaScript里告诉代码，什么元素与开胃菜条目相对应。jQuery选择器的力量能够自动找到这些元素。可以任意添加新的开胃菜条目——只要遵循规定的格式——这段代码将自动设置新的条目，就像设置前面的现有条目那样。

这段代码可以在许多方面接受改进。为了简洁和重点讨论手头的经验教训，我们简写了好几处代码，在把任何这样的代码投入生产之前，是应该更正过来的。下面详细列举应改进的几个方面（甚至是代码的明显缺点），鼓励你把它们当作练习，探索应改进之处。

- 前面已编写的代码假定用户只将有效的数字输入到数量字段。我们明白过来了！应该添加验证，确保输入的只是有效的数字。当出现无效的输入时，你该做些什么事情？
- 当未选中的开胃菜的子选项被隐藏时，它们依然是启用的，所以会连同其余可见的元素一起提交。这就浪费了带宽，并且服务端代码要筛选更多的数据。怎样在合适的时间启用和禁用单选按钮子选项？
- 表单不完整。事实上，没有<form>元素就根本不是表单！完成这段HTML，实现能够提交到服务器的有效表单。
- 人类不只靠开胃菜活命！如何着手为主菜、饮料和餐后甜点添加新的节？flambé香蕉听起来很诱人！这些新的节会怎样影响JavaScript代码的建立？
- 当用餐者选择（和取消）他们的选择项时，你可以提供即时的点菜数量的合计。你如何着手跟踪点菜合计的数据呢？
- 如果使用自定义特性不合你的心意，请对页面进行重构，删除自定义的特性。但务必做到价格信息始终只定义在一个地方！
- 也许这段代码的最大缺陷是，它非常依赖于开胃菜条目里元素之间的位置关系。一方面使得标记保持简单，另一方面却付出了代价——在条目的结构和支持代码之间创建过强的绑定，以及引入复杂的jQuery选择器。你如何着手修改代码使其更为健壮，以便减小条目的结构改动对代码的影响？添加CSS类名称用来标记元素（而不是依赖于位置关系）是达到目标的一个好办法。你如何着手呢？有什么别的主意吗？

如果你想提出引以为荣的主意，请一定访问本书的Manning出版社网页http://www.manning.com/bibeault，那儿有论坛的链接。欢迎你在论坛上发帖子，共享你的解决方案，供大家阅读和讨论！

4.4 小结

借助迄今为止我们所学到的jQuery知识，本章引领我们进入事件处理的世界。

我们已认识到在网页里实现事件处理会面临痛苦的挑战，但这样的事件处理对于创建富因特网应用的页面来说恰恰是不可或缺的。在那些挑战之中不可忽视的事实是，有3个事件模型各自跨越一组现代通用浏览器以不同的方式进行操作。

遗留下来的基本事件模型，也非正式地称为DOM第0级事件模型，在声明事件监听器方面享有少许独立于浏览器的操作，但监听器函数的实现却要求有分歧的、依赖于浏览器的代码。这个事件模型（大概为页面作者所最熟悉）通过将监听器函数的引用指派给元素的属性（例如onclick属性），实现给DOM元素指派事件监听器。

124

尽管简单，这个模型却遭遇“你只有一次机会”的问题：在特定DOM元素上为任何事件类型只能定义一个监听器。

利用DOM第2级事件模型，可以避免这个缺陷。这是一个更为高级的和标准化的模型，一个API可以绑定多个处理程序到事件类型和DOM元素上。尽管这个模型在多方面适用，但它只享有标准兼容浏览器的支持，如Firefox、Safari、Camino以及Opera。

IE6和IE7浏览器提供基于API的、已实现DOM第2级模型的功能子集的专有事件模型。

利用一系列的if语句编码实现所有的事件处理：一个子句用于标准浏览器，另一个用于IE浏览器——会驱使我们患上早期老年痴呆症。幸运的是，jQuery能够把我们从那种命运中拯救出来。

jQuery提供通用的bind()命令，以便在任何元素上建立任何类型的一个或多个事件监听器，同时提供事件特定的简便命令，比如change()和click()。这些方法以独立于浏览器的方式操作，并利用在事件监听器中最常用的标准属性和方法，对传递到处理程序的Event实例进行规范化。

jQuery也提供删除事件处理程序的办法、在脚本控制下引发事件处理程序的调用，甚至定义一些更为高级的命令，使得执行常见的事件处理任务尽可能简便。

我们已经探讨了几个在页面上利用事件的示例。下一章，看jQuery如何基于这些功能，为我们实现动画和动态效果。

125

第5章

用动画和效果来装扮页面

本章内容

- 显示和隐藏没有动画的元素
- 利用核心jQuery动画效果使元素显示和隐藏
- 其他内建的效果
- 编写自定义动画

你曾经看见用Flash建立的网站，并对Flash开发者布置的所有漂亮的效果羡慕不已吗？糟糕，也许你已经开始心动，纯粹为了那些漂亮的效果而去学习Flash。

不久以前，利用JavaScript实现平滑的效果和动画是不切实际的选择。夹在跨浏览器和浏览器执行缓慢的问题之间，试图使元素淡入淡出或缩小放大，或在屏幕上四处移动元素，困难非比寻常。幸亏那种事态已经过去，并且jQuery提供非常简单的接口用于实现各种很酷的效果。

不过，在我们着手给页面添加很酷的效果之前，必须思考这个问题：应该添加吗？就像一部全是特效毫无情节的电影那样，滥用效果的页面同样会引起意想不到的反作用。不要忘记这一点：效果应该用来增强页面的可用性，而不是妨碍它。

请在脑子里牢记这个警告，下面看jQuery提供了什么。

5.1 使元素显示和隐藏

我们想在一个或一组元素上执行的最普通的动态效果类型，也许是使元素显示或隐藏的简单操作。不久将会接触到更为花哨的动画[1]（比如使元素淡入或淡出），不过有时想要保持简单而使元素突然出现或消失！

使元素显示和隐藏的命令近乎我们所料：`show()`用来显示包装集里的元素，而`hide()`用来隐藏包装集里的元素。我们准备推迟讲述这些命令的正式语法，不久就会明白其中的理由。目前我们专注于不带参数地使用这些命令。

这些命令可能看起来简单，可是我们应该记住几件事情。首先，jQuery通过将`style`属性的`display`值改为`none`来使元素隐藏。如果包装集里一个元素已经隐藏，它将保持隐藏状态，但依然被返回给jQuery链。例如，假定有HTML片段如下：

① 在本书中，“动画”、“动画效果”、“动态效果”和“效果”指的是同一事物。——译者注

```
<div style="display:none;">This will start hidden</div>
<div>This will start shown</div>
```

如果应用$("div").hide().addClass("fun")，将得到结果如下：

127

```
<div style="display:none;" class="fun">This will start hidden</div>
<div style="display:none;" class="fun">This will start shown</div>
```

其次，如果在初始化时将style属性的display值显式地设置为none而使元素隐藏，则调用show()命令之后，元素的style属性的display值总是被设置为block。即便是对于display值默认为inline的典型的元素（例如元素），结果也是一样[①]。如果元素在初始化时没有显式地声明display值而调用jQuery的hide()命令使元素隐藏，则此后调用show()命令，将回忆起原始的值，并把元素还原到display的原始状态。

因此在初始化时不要利用元素的style属性使元素隐藏，而应该在页面的就绪处理程序里将hide()命令应用到想要使其隐藏的元素上。这样既能使元素隐藏，又能确保每个元素的初始状态可知，并且确保随后的隐藏和显示操作像预期的那样进行。

下面看如何恰当地利用这些命令。

5.1.1 实现可折叠的列表

一下子把用户淹没在过多的信息里，是常见的典型的用户界面错误。最好的办法是允许用户自己控制所请求的信息的数量。这是“渐进式公开”大原则的一瞥（请参见4.3节），在这个原则下给用户提供的数据保持在最低的限度，同时会根据需求而提供更多的数据以便执行手头上的任务。

浏览一台计算机的文件系统就是一个很好的示例。文件系统通常以层次结构列表形式显示。在层次结构列表里文件夹的内容嵌套到必需的深度，以便表示系统中所有的文件和文件夹。试图一下子提供系统中所有的文件和文件夹简直荒唐可笑！更好的解决办法是允许列表中的每一层打开或关闭，以便渐进地公开被包含的层次结构信息。在任何应用里，你肯定看到过这种允许浏览文件系统的控件。

在本节里，我们将看到如何利用hide()和show()命令来设置以这种方式来操作的嵌套列表。

注意 有一些很棒的插件提供无需自定义就可以直接使用的这个类型的控件。只要你理解背后的原理，就会利用现成的解决方案，而不是自己编写插件。

首先看列表的HTML结构，它将用于测试代码。

128

```
<body>
  <fieldset>
    <legend>Collapsible List &mdash; Take 1</legend>
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>
```

---

① 根据我的测试，不管在初始化时通过样式或是通过hide()命令使元素隐藏，之后调用show()，都发现元素的style属性的display值变为inline而不是block。因此我认为原书有误。——译者注

```
      Item 3
      <ul>
        <li>Item 3.1</li>
        <li>
          Item 3.2
          <ul>
            <li>Item 3.2.1</li>
            <li>Item 3.2.2</li>
            <li>Item 3.2.3</li>
          </ul>
        </li>
        <li>Item 3.3</li>
      </ul>
    </li>
    <li>
      Item 4
      <ul>
        <li>Item 4.1</li>
        <li>
          Item 4.2
          <ul>
            <li>Item 4.2.1</li>
            <li>Item 4.2.2</li>
          </ul>
        </li>
      </ul>
    </li>
    <li>Item 5</li>
  </ul>
</fieldset>
</body>
```

注意 本节把重点放在效果上，因此假定我们将要设置的列表足够小，以便能够作为页面的一部分一次发送完毕。对于整个文件系统的巨大的数据集来说，这显然是不可能的。在这种情况下，你就想根据需要向服务器请求越来越多的数据，但那不是本章想要关注的主题。在你阅读Ajax那一章（第8章）之后，可以重读这些示例并应用你的技能去增强这些控件，使其拥有这种功能。

这个列表（改造之前）在浏览器里显示的结果如图5-1所示。

这种大小的列表相对容易接受，但不难想象，随着列表越变越长、层次越变越深，在那样的情况下为用户显示整个列表，可能导致他们患上可怕的信息超载综合症。

我们想要设置包含嵌套列表的所有列表项，以便使嵌套列表隐藏直到用户通过点击项来选择查看它。在点击项后，项的子内容将显示出来。接着点击项，将再次使项的子内容隐藏。

如果我们只隐藏包含子列表的项的内容（例如最高一层的第3项、第4项），就很可能把用户弄糊涂，因为他们不可能知道这两项是活动的和可扩展的。我们必须从视觉上把这些可扩展项与（不可扩展的）叶子项区分开来。当光标通过活动项的上方时，把光标变成手形光标加以区分，同时把活动项的列表记号替换为司空见惯的加号（+）或减号（–），指示该项可被扩张或折叠。

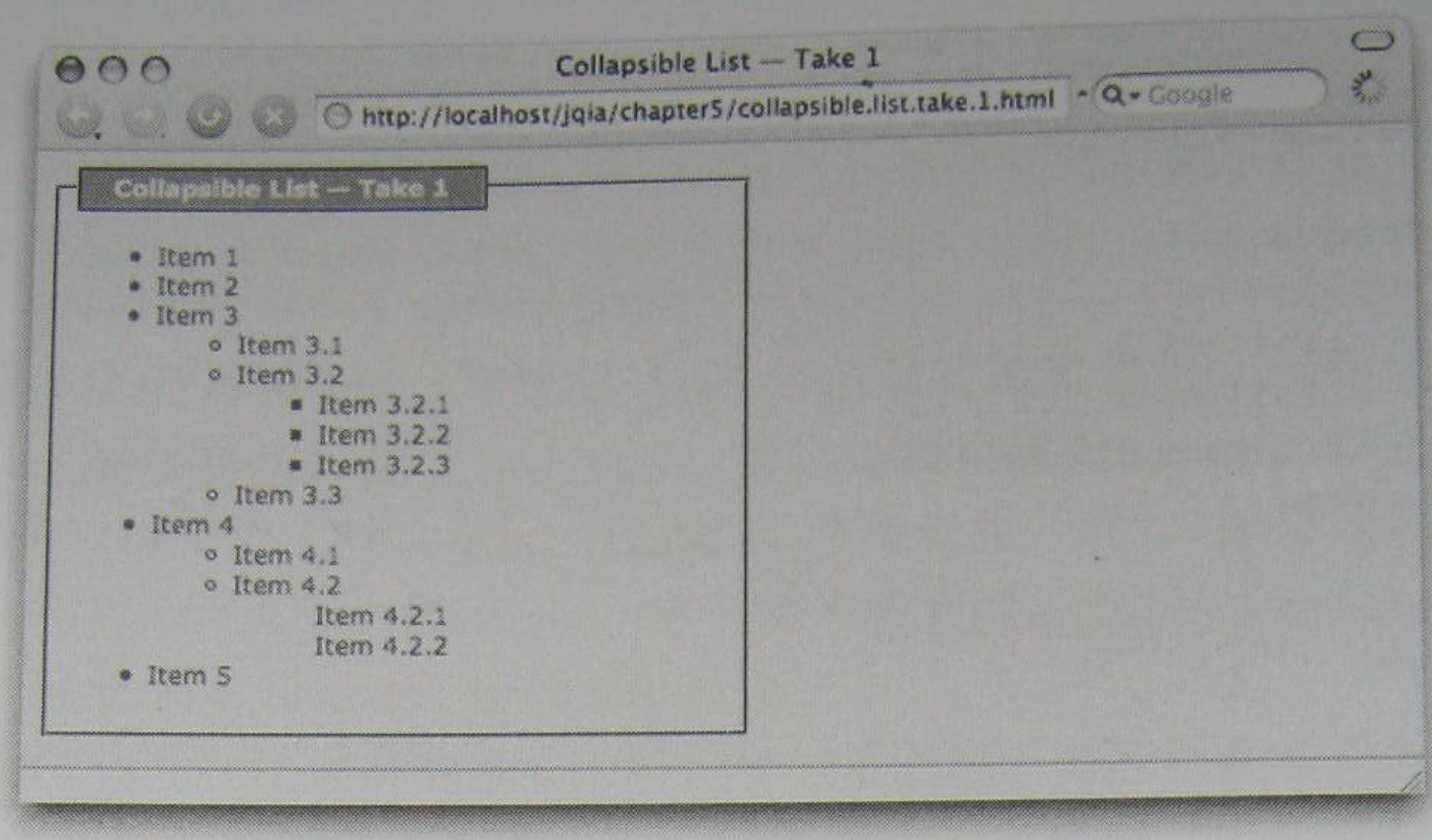


图5-1　改造之前的嵌套列表

我们使用户一开始看到的是处于完全折叠状态的列表。设置之后的列表初始显示时如图5-2所示。

130

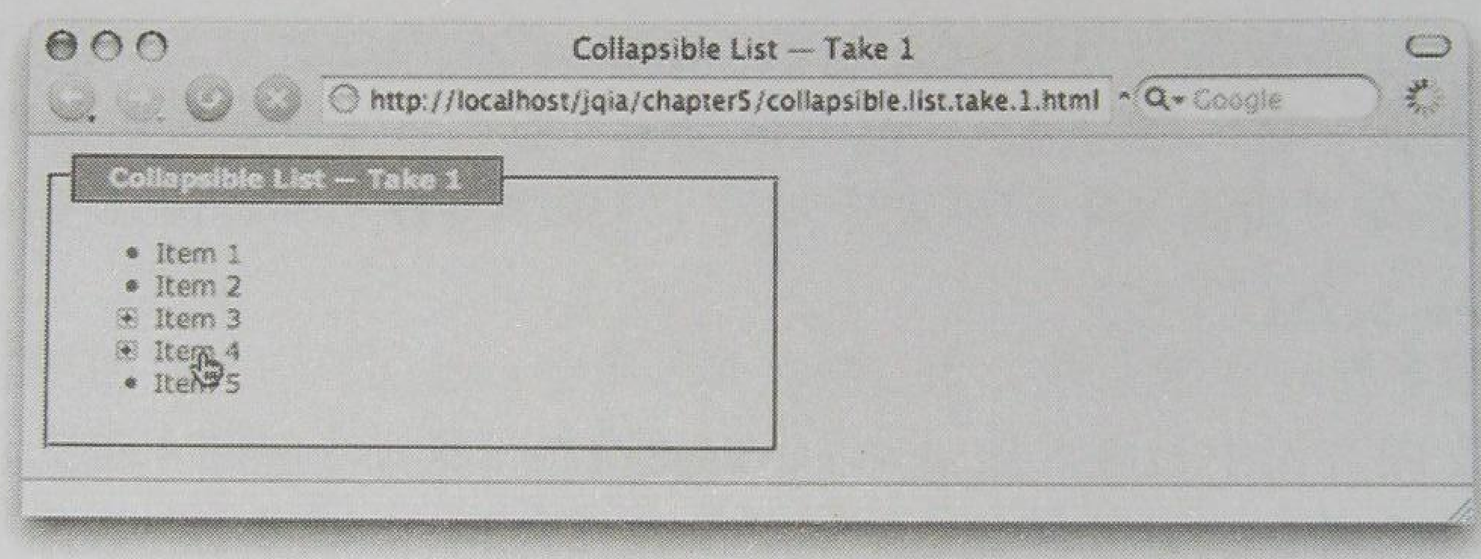


图5-2　设置之后的列表已经完全折叠，并且可扩展项可以从视觉上加以区分

可以看见包含子项的列表项（第3项和第4项）发生了下列的变化。

- 列表项3和列表项4的子项是隐藏的。
- 这两项的列表记号已经替换为加号（+），指示该项可被扩展。
- 鼠标光标在这两项的上方悬停时，变为手形光标。

一点击这些可活动项，就会展现它们的子项，如图5-3的一系列画面所示。

下面看看如何在就绪处理程序中设置这样的行为，将其应用到列表的DOM元素上，如代码清单5-1所示。

代码清单5-1　用可扩展行为来设置列表的就绪处理程序代码

```
$(function(){
  $('li:has(ul)')                    ❶
    .click(function(event){          ❷
      if (this == event.target) {
        if ($(this).children().is(':hidden')) {    ❸
          $(this)
            .css('list-style-image','url(minus.gif)')
```

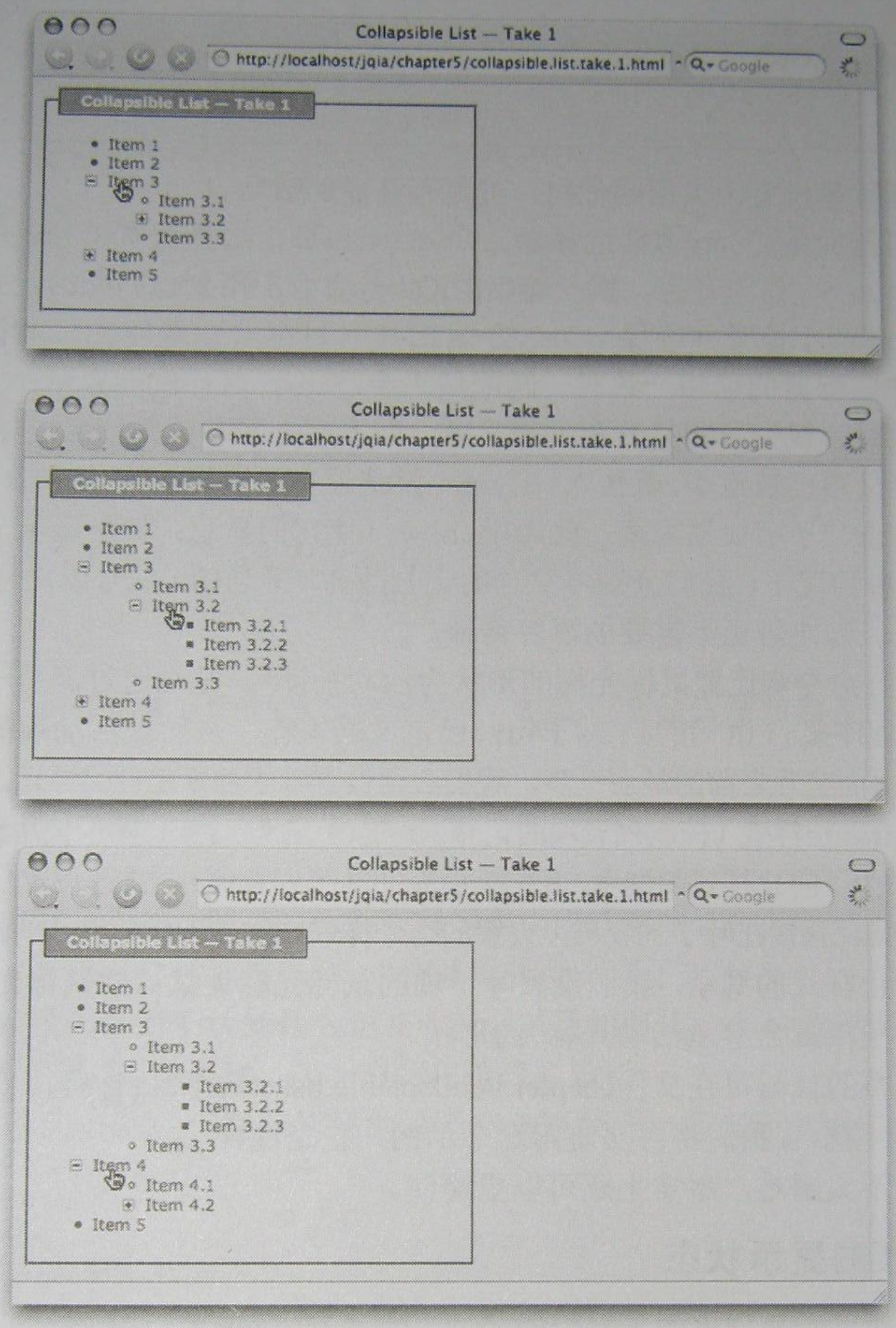


图5-3 点击活动元素使活动元素的内容显示出来

```
            .children().show();
        } else {
            $(this)
              .css('list-style-image','url(plus.gif)')
              .children().hide();
        }
      }
      return false;    ◁ 4
    })
    .css('cursor','pointer')    ◁ 5
    .click();                   ◁ 6
$('li:not(:has(ul))').css({     ◁ 7
```

```
    cursor: 'default',
    'list-style-image':'none'
  });
});
```

这个就绪处理程序没有一大堆的代码，可是有许多的活动。

首先，我们通过应用jQuery容器选择器li:has(ul)❶，选择拥有子列表的所有列表项，然后附加处理程序❷click作为开头，把一连串的jQuery命令应用到已匹配的元素上。

这个click处理程序检查并确保event的target（目标）元素与this相匹配。当且仅当已点击的项与已建立监听器的项相一致时，匹配结果为true。这样就能忽略子元素上的点击。毕竟在用户点击父项时，我们只想打开和关闭父项，而不想打开和关闭任何子项。

如果我们确定父项已被点击，就传入:hidden筛选器❸调用简便的is()命令以便确定子项处于隐藏或是显示状态。子项如果隐藏，就调用show()使它们显示；如果显示，就调用hide()使它们隐藏。在这两种情况下，我们分别将父项的记号改为（适当的）减号或加号，然后返回false作为监听器❹的值，以便防止不需要的事件传播。

我们调用css()命令❺设置鼠标光标的形状为活动指针[1]，并且通过调用click处理程序❻，执行在click处理程序里if语句的else子句内已定义的操作，从而使活动项的子元素隐藏。

作为用户可与页面交互之前的最后一步，我们必须设置叶子项❼元素的样式。我们已经设置活动项的list-style-image样式（控制列表项的记号）为加号或减号的GIF图像，并且不想让活动项的子项继承父项的样式。为了防止样式的继承，我们为所有的叶子列表项，显式地设置list-style-image样式属性的值为none。直接在叶子列表项上设置的这个样式，优先于任何继承的样式。

131 ~ 133

同样是为了防止样式的继承，我们设置叶子项的鼠标光标为默认的鼠标光标的形状，否则包含在活动项里的叶子项就会继承活动光标[2]的形状。

这个页面的完整的代码可在文件chapter5/collapsible.list.take.1.html找到。（也许你猜测文件名的一部分“take.1”预示着我们将再次访问这个示例，正是这样！）

启用这些功能不太困难，事实上还可以更简便。

5.1.2　切换元素的显示状态

切换元素（比如可折叠列表的示例中的子项）的显示或隐藏状态，是如此频繁的操作，因此jQuery定义名为toggle()的命令，使得操作更为简便。

下面把这个命令应用到可折叠列表上，看它如何简化前面的代码。代码清单5-2只显示已经重构的页面的就绪处理程序（不需要任何其他的修改），改动之处用粗体突出显示。完整的页面代码可在文件chapter5/collapsible.list.take.2.html找到。

代码清单5-2　利用toggle()命令修改可折叠列表

```
$(function(){
  $('li:has(ul)')
    .click(function(event){
```

① 即手形光标。——译者注

② 即手形光标。——译者注

```
    if (this == event.target) {
      $(this).children().toggle();
      $(this).css('list-style-image',
        ($(this).children().is(':hidden')) ?
          'url(plus.gif)' : 'url(minus.gif)');
    }
    return false;
  })
  .css('cursor','pointer')
  .click();
$('li:not(:has(ul))').css({
  cursor: 'default',
  'list-style-image':'none'
  });
});
```

134

请注意，我们不再需要条件语句去决定使元素隐藏或显示，而由toggle()负责切换显示状态。我们依然利用.is(':hidden')测试作为更简便的三元表达式的一部分，以便决定使用哪个图像作为合适的列表项记号。

可以轻而易举地使得元素在瞬间内出现或消失，但我们想使这种转换不那么突然。下面看有什么可供利用。

5.2 以动画方式使函数显示和隐藏

人类的认知能力就是那样，使界面元素在瞬间内出现或消失，令人感觉不太舒服。如果我们眨错了眼，就会错过转换，然后惊愕地想："刚才发生什么事了？"

短时间的逐渐转换帮助我们认知什么正在改变，以及如何从一个状态转换为另一个状态。而这就是jQuery核心效果发挥作用的地方。一共有3组核心效果：

- 显示和隐藏（关于在5.1节里所讲述的显示和隐藏命令，其实还有更多的内容）；
- 淡入和淡出；
- 滑上和滑下。

下面逐一研究这3组核心效果。

5.2.1 使元素逐渐地显示和隐藏

show()、hide()和toggle()命令比起在上一节里我们使你相信的要复杂一些。如果不带参数进行调用，这些命令实现包装集元素显示状态的简单操作，使元素从显示器上瞬间显示或隐藏；如果传入参数，这些效果就以动画方式来表现，因此显示状态将在短时间内持续改变。

135

现在我们准备学习这些命令的完整语法。

命令语法：hide

hide(speed,callback)

使包装集里的元素变为隐藏状态。如果不带参数进行调用，就通过把元素的display样式属性的值设置为none使操作突然发生；如果提供speed参数，就通过调整元素的大小和把不透明度

降到0，使元素在短时内逐渐隐藏，完全隐藏之后再把元素的display样式属性的值设置为none，以便从显示器上删除元素。
可以指定在动画结束时调用的回调函数（可选）。

参数

speed　（数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就不产生动画并立即从显示屏上删除元素。

callback　（函数）回调函数（可选），在动画完成时调用。没有参数传递给这个函数，但函数上下文（this）被设置为以动画方式隐藏的元素。

返回

包装集

命令语法：show

show(speed,callback)

使包装集里的元素显示。当元素使用jQuery效果处于隐藏状态时，如果不带参数调用show，就通过把元素的display样式属性的值还原为之前的设置（如block或inline）使操作突然发生；如果元素没有通过jQuery效果隐藏，display样式属性的值就默认为block。
如果提供speed参数，就通过调整元素的大小和提高不透明度，使元素显示。
可以指定在动画结束时调用的回调函数（可选）。

参数

speed　（数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就不产生动画并立即在显示屏上显示元素。

callback　（函数）回调函数（可选），在动画完成时调用。没有参数传递给这个函数，但函数上下文（this）被设置为以动画方式隐藏的元素。

返回

包装集

命令语法：toggle

toggle(speed,callback)

在任何隐藏的包装集元素上执行show()，并且在任何非隐藏的包装集元素上执行hide()。请参见这两个命令的语法说明，了解各自的语义。

参数

speed　（数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就不产生动画。

callback　（函数）回调函数（可选），在动画完成时调用。没有参数传递给这个函数，但函数上下文（this）被设置为以动画方式切换显示状态的元素。

返回

包装集

下面第3次访问可折叠列表，以动画方式使各节打开和关闭。

因为前面的信息，你会想我们必须对take 2可折叠列表的实现代码做出的唯一改动是，把调用toggle()命令修改为：

```
toggle('slow')
```

但是没那么快！如果做出这个改动并测试页面，会发生一些奇怪的事情。首先回忆一下，为了在初始化时使可折叠的元素隐藏，我们调用了活动项的click处理程序。如果处理程序所做的只是立即使子元素隐藏，那当然好。但现在以动画方式进行活动，因此在页面加载时，子项以动画方式逐渐地隐藏起来。因此那根本不顶用！

我们必须地不带参数地显式调用hide()命令，在用户有机会看见子元素之前就把子元素隐藏起来，然后把记号设置为加号。你将回忆起来，在前面的示例中我们没有那样做，因为那样会造成重复的代码。好，有了现在所做的改动，那就不再是个问题[1]了。

136 ~ 137

第二个问题是，记号的图像操作有失妥当。如果切换操作在瞬间内完成，可以在操作执行之后立即安全可靠地检查操作的结果。既然切换操作现在是以动画方式进行，操作结果就不再是同步的，因而随后立即检查子元素隐藏与否（为了弄清楚应该设置哪个图像作为记号）不再安全可靠。

我们颠倒一下顺序，也就是先检查子项的状态，再以动画方式进行切换操作。

新的就绪处理程序的改动之处用粗体突出显示，如代码清单5-3所示。

代码清单5-3 已添加动画效果的可折叠列表的示例

```
$(function(){
  $('li')
    .css('pointer','default')
    .css('list-style-image','none');
  $('li:has(ul)')
    .click(function(event){
      if (this == event.target) {
        $(this).css('list-style-image',
          (!$(this).children().is(':hidden')) ?
            'url(plus.gif)' : 'url(minus.gif)');
        $(this).children().toggle('slow');
      }
      return false;
    })
    .css({cursor:'pointer',
          'list-style-image':'url(plus.gif)'})
    .children().hide();
  $('li:not(:has(ul))').css({
    cursor: 'default',
```

① 是指在这个示例中，在初始化时不带参数地调用hide()命令使子元素隐藏，是必需的操作，所以不再成为问题。——译者注

```
    'list-style-image':'none'
  });
});
```

这个改动之后的页面文件可在chapter5/collapsible.list.take.3.html找到。

知道人们（比如我们）多么喜爱动手实践，因此我们建立简便的工具，用来进一步研究这些命令的操作。

jQuery效果实验室页面

在第2章里我们介绍了实验室页面的概念，以便利用jQuery选择器进行试验。在chapter5/lab.effects.html文件里建立本章的实验室页面，用来探索jQuery效果的操作。

用浏览器加载这个页面，显示结果如图5-4所示。

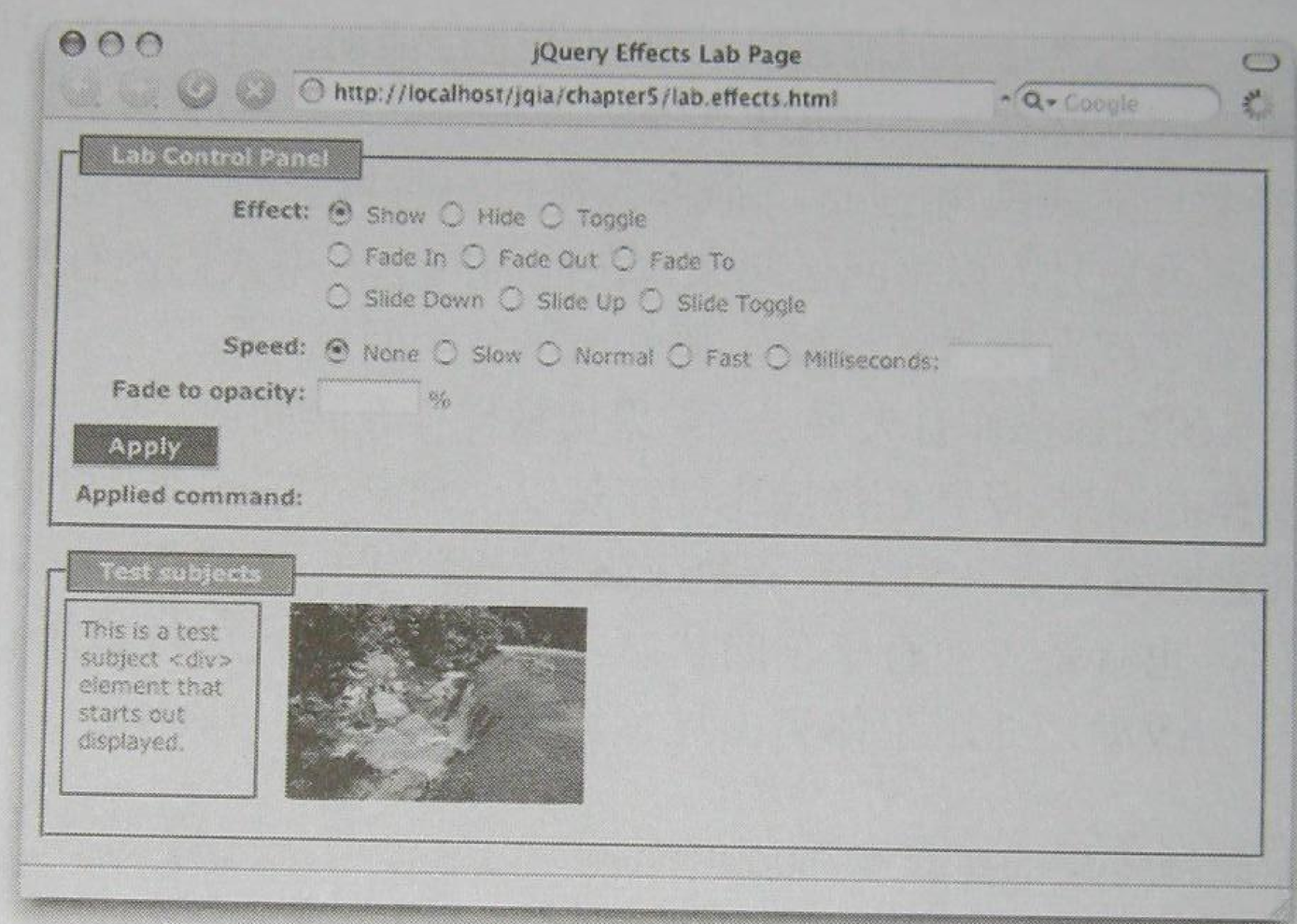


图5-4　jQuery效果实验室页面的初始状态，这个页面帮助我们研究jQuery效果命令的操作

这个实验室页面主要由两个面板组成：控制面板用来指定应用哪个效果；另一个面板包含4个测试对象元素，效果将作用在这些元素上。

138

“他们犯傻了？”你或许在想，“明明只有两个测试对象。”

不，本书作者没有遗漏什么。确实有4个元素，但其中两个（另一个包含文本的 和另一个图像）在初始化时是隐藏的。

我们利用这个页面演示迄今为止已讨论的命令的操作。用浏览器打开页面，进行下面的练习。

- 练习1——初始页面加载之后两个测试控件居于左侧，点击Apply按钮将执行不带参数的show()命令。已应用的命令显示在Apply按钮下面以便为你提供信息。请注意初始隐藏的两个测试对象元素是怎样立即显示的。最右边的图像显示得有点褪色，那是因为它的不透明度被有意地设置为50%。
- 练习2——请选中Hide单选按钮，并点击Apply执行无参数的hide()命令。所有的测试对象都突然消失了。请你特别注意，测试对象所在的字段集的上下边框收紧了。这意味着元素从显示器上完全删除，而不是设置为不可见。

139

注意 当我们说某个元素从显示器上删除（在这里，以及在下文关于效果的讨论中），意思是那个元素不再被浏览器的布局管理器纳入考虑范围，就像元素的CSS样式属性display的值被设置为none那样；而不意味着元素已从DOM树里删除。没有哪个效果会导致元素从DOM里删除。

- 练习3——接下来，选择单选按钮Toggle并点击Apply。再次点击Apply。再一次……你将注意到，每当执行toggle()时测试对象就切换为显示或隐藏状态。
- 练习4——重新加载页面，以便使各元素复位到初始条件（在Firefox浏览器里，请把焦点放在地址栏上并敲击回车键）。选择Toggle并点击Apply。注意两个初始化时可见的对象消失，而两个初始化时隐藏的对象出现。这表明toggle()命令分别应用到包装集的各个元素上，使隐藏的元素显示，与此同时使显示的元素隐藏。
- 练习5——在这个练习里我们进入动画王国。刷新页面，选择Slow用来设置Speed（速度）并点击Apply。请仔细观察测试对象。两个隐藏的元素不是在瞬间内闪现而是从各自的左上角逐渐伸展出来。如果你想“放慢镜头”以便看个究竟，请刷新页面并选中用来设置速度的Milliseconds（毫秒），输入速度值10000。这样使得效果的持续时间延长到（折磨人的）10秒，你就有充足的时间用来观察效果的行为。
- 练习6——选择Show、Hide和Toggle的各种组合以及不同的速度，对各种效果进行试验，直到你觉得对效果的操作已有把握。

配备jQuery效果实验室页面以及关于第一组效果如何操作的知识，下面请看第二组效果。

5.2.2 使元素淡入和淡出

如果你曾仔细观察show()和hide()效果的操作，就会注意到随着元素扩大或缩小，这两个效果按比例放大或缩小元素的尺寸并调整元素的不透明度。第二组效果fadeIn()和fadeOut()只影响元素的不透明度。除缺乏缩小和放大以外，fadeIn()和fadeOut()命令的工作方式分别以类似于动画的方式进行的show()和hide()。这些命令的语法如下。

140

命令语法：fadeOut

fadeOut(speed,callback)

将非隐藏的任何已匹配的元素的不透明度逐渐降低到0%，然后从显示器上删除。不透明度的变化的持续时间由speed参数决定。任何已经隐藏的元素不受影响。

参数

speed （数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就默认为normal。

callback （函数）回调函数（可选），在动画完成时调用。没有参数传递给这个方法，但函数上下文（this）被设置为以动画方式淡出的元素。

返回

包装集

命令语法：fadeIn

fadeIn(speed,callback)

通过把元素的不透明度逐渐提高到初始值，使隐藏的任何已匹配的元素显示出来。这个初始值要么是应用到元素的原始不透明度值，要么是100%。不透明度的变化的持续时间由speed参数决定。任何非隐藏的元素不受影响。

参数

speed　（数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就默认为normal。

callback　（函数）回调函数（可选），在动画完成时调用。没有参数传递给这个函数，但函数上下文（this）被设置为以动画方式淡入的元素。

返回

包装集

我们将从jQuery效果实验室页面找到更多的乐趣。请打开实验室页面，利用Fade In和Fade Out单选按钮，做一组类似上一节的后面的练习（目前别担心Fade To，马上讲到）。

请注意重要的一点，在元素的不透明度受到调整时，jQuery的hide()、show()、fadeIn()和fadeOut()效果会记住元素的原始不透明度值并接受它。在实验室页面里，在使最右边的图像隐藏之前，我们有意地设置这幅图像的原始不透明度为50%。在应用各种jQuery效果时，这幅图像的不透明度会经历所有发生的变化，但原始不透明度的值从未被修改。

在实验室页面里多做些练习，直到你确信上述的那一点，并对淡入和淡出的效果感到满意。

jQuery通过fadeTo()命令提供另一个效果。就像前面已讨论的淡入和淡出效果，这个效果也调整元素的不透明度，但决不从显示器上删除元素。在利用实验室页面的fadeTo()做练习之前，先看这个命令的语法。

命令语法：fadeTo

fadeTo(speed,opacity,callback)

从当前的设置到opacity所指定的新设置，调整已包装的元素的不透明度。

参数

speed　（数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就默认为normal。

Opacity　（数字）指定元素调整的目标不透明度，取值范围从0.0到1.0。

callback　（函数）回调函数（可选），在动画完成时调用。没有参数传递给这个函数，但函数上下文（this）被设置为已添加动画的元素。

返回

包装集

不像在使元素显示或隐藏时调整不透明度的其他效果那样，fadeTo()不会记住元素的原始不透明度。这是合乎逻辑的，因为这个效果的根本用途就是显式地改变不透明度到特定值。

用浏览器打开jQuery效果实验室页面，使所有的元素显示出来（你到现在应该知道怎么办了）。然后做下面的练习。

- 练习1——选择Fade To单选按钮，设置足够小的Speed值以便于观察行为（4000毫秒是很好的选择），然后设置Fade to Opacity（目标不透明度）字段的值为10（要求输入0到100之间的整数，传递到命令时会转换为0.0到1.0之间的数字）并点击Apply。测试对象将在4秒钟内逐渐改变到10%的不透明度。
- 练习2——设置Fade to Opacity为100，并点击Apply。所有的元素（包括在初始化时半透明的图像在内）将调整到完全的不透明度。
- 练习3——设置Fade to Opacity为0，并点击Apply。所有的元素颜色淡化至不可见，不过请注意即使元素消失殆尽，周围的字段集也不会收紧。不像fadeOut()效果那样，fadeTo()决不从显示器上删除元素，即便元素变得完全不可见。

继续用Fade To效果进行试验，直到你掌握它的工作方式。然后我们准备探讨第3组效果。

5.2.3 使元素滑上和滑下

使元素显示或隐藏的第3组效果是slideDown()和slideUp()，工作方式也和效果show()和hide()相似，除了这一点以外：元素显示时似乎从顶部滑下来，而元素隐藏时似乎向顶部滑上去[1]。

就像在hide()和show()之间切换有toggle()命令那样，在效果slideUp()和slideDown()之间切换也有slideToggle()命令。这3个命令的语法如下（迄今为止已经熟悉这类语法了吧）。

命令语法：slideDown

slideDown(speed,callback)

通过从上往下逐渐扩大元素的垂直面积，使隐藏的任何已匹配的元素显示出来。非隐藏的任何元素不受影响。

参数

speed （数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就默认为normal。

callback （函数）回调函数（可选），在动画完成时调用。没有参数传递给这个方法，但函数上下文（this）被设置为以动画方式“滑下”的元素。

返回

包装集

① 其实更像是百叶窗的展落和卷起，或闸门的降下和升起。——译者注

命令语法：slideUp

slideUp(speed,callback)

通过从下往上逐渐减小元素的垂直面积，最终使非隐藏的任何已匹配的元素从显示器上删除。

参数

speed　（数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就默认为normal。

callback　（函数）回调函数（可选），在动画完成时调用。没有参数传递给这个方法，但函数上下文（this）被设置为以动画方式“滑上”的元素。

返回

包装集

命令语法：slideToggle

slideToggle(speed,callback)

在隐藏的任何已包装的元素上执行slideDown()，同时在非隐藏的任何已包装的元素上执行slideUp()。请参见这两个命令的语法说明以便了解各自的语义。

参数

speed　（数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：slow、normal或fast。如果省略，就默认为normal。

callback　（函数）回调函数（可选），在动画完成时调用。没有参数传递给这个方法，但函数上下文（this）被设置为以动画方式“滑动”的元素。

返回

包装集

除了使元素显示或隐藏的方式以外，这些效果的操作与其他的显示或隐藏效果相似。请通过试验确认这一点。打开jQuery效果实验室页面，从头到尾做几组类似于应用到其他两组效果的那些练习。

44

5.2.4　使动画停止

我们可能时不时有理由停止一段已经开始的动画。因为用户事件指示应该处理别的事情，或因为我们想要开始全新的动画。stop()命令可以达到这个目的：

命令语法：stop

stop()

停止当前正在运行的、包装集里的元素的所有动画。

参数

无

返回
包装集

请注意，对于任何已添加动画的元素，任何已发生的变化将一直有效。如果想要还原元素到原始状态，我们就有责任利用`css()`方法或类似的命令还原这些元素的CSS值为原始值。

既然我们已经看了核心jQuery内建的效果，下面研究怎样编写自定义的效果！

5.3 创建自定义的动画

jQuery提供的核心效果的数目有意保持为有限的几个（为了使jQuery的核心覆盖面保持最小化），因为有着这样的预期：页面作者可以自行决定利用插件机制添加更多的效果。

jQuery发布`animate()`包装器方法，允许应用自定义的动画效果到包装集元素上。这个包装器方法的语法如下：

145

命令语法：animate

`animate(properties,duration,easing,callback)`
`animate(properties,options)`

将`properties`和`easing`参数所指定的动画应用到包装集的所有元素上。可以指定回调函数（可选），在动画完成时调用。第二种格式除`properties`以外，还指定一组`options`（选项）。

参数

`properties` （对象）一个散列对象，指定在动画结束时所支持的CSS样式应该达到的终值。通过把元素的样式属性值，从当前值逐渐调整到这个散列对象所指定的终值而产生动画。

`duration` （数字|字符串）把效果的持续时间（可选地）指定为毫秒数或预定义的字符串之一：`slow`、`normal`或`fast`。如果省略，则不会产生动画。

`easing` （字符串）一个函数的名称（可选），用来实现动画的缓和效果。缓和函数（通常由插件提供）必须通过名称注册。核心jQuery提供两个缓和函数，已注册的名称为linear和swing。

`callback` （函数）在动画完成时调用的函数（可选）。没有参数传递到这个函数，但函数上下文（`this`）被设置为已添加动画的元素。

`options` （对象）利用一个散列对象指定动画的参数值。所支持的属性如下。

`duration`——请参见`duration`参数的前面的说明。
`easing`——请参见`easing`参数的前面的说明。
`complete`——当动画完成时调用的函数。
`queue`——如果为`false`，动画不会加入队列而立即开始运行。

返回
包装集

我们通过提供一组CSS样式属性及其目标值而创建自定义的动画。随着动画的进行，那些属性

将融合在一起。动画从元素的原始样式值开始，朝着目标值的方向逐渐进行样式值的调整。在效果的作用期间（由动画引擎自动处理），样式取得的中间值由动画的持续时间和缓和函数所决定。

指定的目标值可以是绝对值或相对值（相对于初始值）。为了指定相对值，要在值的前面加上前缀+=或-=，分别指示正方向或负方向的相对目标值。

146

术语缓和（easing）用来说明动画的帧速率和操作的处理方式。通过在动画的持续时间和当前的时间位置上利用花式数学，使效果发生一些有趣的变化是可能的。编写缓和函数是复杂的有针对性的主题，通常只为最核心的插件作者所关注。在本书中我们不准备研究自定义的缓和函数这个课题。如果除linear（提供线性的连续操作）和swing（临近动画结束时稍微加速）以外，你还想尝试更多的缓和函数，可以利用Easing Plugin（缓和插件）进行试验。这个插件可以从http://gsgd.co.uk/sandbox/jquery.easing.php获取。

动画默认地添加到执行队列。应用多个效果到一个对象上，将使不同效果连续地运行（串行）。如果你想要不同的效果并行地运行，就要将queue选项设置为false。

能够以动画方式表现的CSS样式属性的列表限定于那些要求数字值的属性，因为对于那些属性来说，要从初始值到目标值进行合乎逻辑的连续操作。限定于数字值是完全可以理解的——怎么能够想象非数字的属性（如image-background）从源值到终值进行合乎逻辑的连续操作呢？对于表示尺寸的值，jQuery假定默认的单位是像素，但我们也可以通过包含em或%后缀指定em单位或百分比。

通常以动画方式表现的样式属性包括top、left、width、height和opacity。但只要对想要实现的效果有意义，数字值的样式属性（比如字体大小和边框宽度）也能够以动画方式表现。

注意 如果你想以动画方式表现用来指定颜色的CSS值，就可能对官方的jQuery颜色动画插件（Color Animation Plugin）感兴趣。这个插件可从http://jquery.com/ plugins/project/color获取。

除特定值以外，我们也可以指定以下的字符串之一：hide、 show或toggle。jQuery将按照指定的字符串计算合适的目标值。例如，给opacity属性指定hide将使元素的不透明度减小到0。指定这些特别的字符串的任何一个，都有从显示器上自动地展现或删除元素的附加效果（就像show()和hide()命令那样）。

当我们介绍核心动画时，你注意到淡入和淡出效果之间没有切换命令吗？利用animate()和toggle可以轻松地实现这个切换命令，如下所示：

147

```
$('.animateMe').animate({opacity:'toggle'},'slow');
```

我们再动手编写几个自定义的动画。

5.3.1 一个自定义的放大动画

考虑简单的放大动画：调整元素的大小为原始尺寸的两倍。编写这个动画的代码如下所示：

```
$('.animateMe').each(function(){
  $(this).animate(
    {
      width: $(this).width() * 2,
      height: $(this).height() * 2
```

```
    },
    2000
  );
});
```

为了实现这个动画，在迭代包装集里的所有元素时，我们通过each()将动画分别应用到已匹配的各元素上。这是重要的，因为我们必须根据各元素自身的尺寸而给各元素指定对应的属性值。如果一开始我们就知道，将添加动画到单个的元素上（比如利用id选择器）或把完全相同的一组值应用到各元素上，就可以免除each()而直接添加动画到包装集上。

在迭代器函数里把animate()命令应用到元素上（标识为this），把width和height样式属性的值设置为元素的原始尺寸的两倍。结果是经过两秒钟的过程（因为已经指定duration参数为2000），已包装的元素（可能只有一个）将扩大到原始尺寸的两倍。

下面尝试更令人炫目的动画。

5.3.2 一个自定义的坠落动画

假定我们想要以动画方式显著地表现从显示器上删除元素，多半因为给用户传达这样的信息至关紧要：删除的项一去不复返，因此应该对要删除的项确认无误。我们用动画来表达这个意思，使元素从显示器上消失表现得好像从页面上坠落那样。

对此我们只要思考片刻就能想到，通过调整元素的top位置，就可以使元素在页面上向下移动而模拟坠落的过程；与此同时调整opacity，使元素看起来渐渐地消失。这一切都已完成的时候，最终从显示器上删除元素（类似于以动画方式进行的hide()）。

我们利用下面的代码实现这个坠落效果：

```
$('.animateMe').each(function(){
  $(this)
    .css('position','relative')
    .animate(
      {
        opacity: 0,
        top: $(window).height() - $(this).height() -
             $(this).position().top
      },
      'slow',
      function(){ $(this).hide(); });
});
```

这个效果比前面的效果进行更多的操作。我们再次迭代元素集，同时调整元素的位置和不透明度。但为了调整相对于原始位置的元素的top值，必须首先把元素的CSS样式属性position的值修改为relative。

然后给动画指定opacity的目标值为0以及计算后的top值。我们不想在页面上把元素向下移动到超出窗口底部的那种程度，因为那会导致显示原来没有的滚动条，很可能会分散用户的注意力。我们不想使用户的注意力从动画上抽离——吸引用户的注意力正是我们当初添加动画的理由！因此我们利用元素的高度、垂直位置以及窗口的高度，计算元素应该在页面上向下移动的距离。

注意 在本书的大多数示例中，为了关注核心jQuery，我们尽量避免使用插件。这未必完全反映真实世界的情况，事实上，页面作者通常是同时利用核心jQuery和需要的任何插件去完成任务。编写jQuery插件的简便和丰富的可用插件池，是jQuery的两大优势。在这个示例动画里（还有将要探讨的下一个示例），我们利用Dimensions插件的position()命令以便确定相对于页面的元素的初始位置。在第9章里将详细探讨Dimensions插件（请参见9.2节）。

49 当动画完成时，我们想要从显示器上删除元素，因此我们指定一个回调函数，把非动画方式的hide()命令应用到元素上（该元素作为回调函数的函数上下文而可用）。

注意 比起需要来，我们在这个动画里多做了一些工作，因此能够演示必须等到动画结束时才能在回调函数里做的事情。如果我们刚才指定opacity属性的值为hide而不是0，则在动画结束时将自动地删除元素，也就是说我们本来是不需要编写回调函数的。

为了锦上添花，下面我们再尝试“让它消失”类型的另一个效果。

5.3.3 一个自定义的消散动画

并非使元素从页面上坠落，假定我们想要一种效果，使元素看起来就像一阵轻烟在晴空里消散得无影无踪。为了以动画的方式表现这个效果，我们组合使用放大效果和淡出效果，使元素一边放大一边淡出。由此衍生一个必须处理的问题：如果使元素固定在其左上角的位置进行放大的话[①]，这个消散效果将瞒不过用户的眼睛。在使元素放大的过程中我们想让元素的中心保持在相同的位置，因此除了调整元素的大小以外，作为动画的一部分还必须调整元素的位置。

消散效果的代码如下：

```
$('.animateMe').each(function(){
  var position = $(this).position();
  $(this)
    .css({position:'absolute',top:position.top,
          left:position.left})
    .animate(
      {
        opacity: 'hide',
        width: $(this).width() * 5,
        height: $(this).height() * 5,
        top: position.top - ($(this).height() * 5 / 2),
        left: position.left - ($(this).width() * 5 / 2)
      },
      'normal');
});
```

在这个动画里，我们一边使不透明度减小到0，一边使元素放大到原始大小的5倍，并且根据新的大小的一半调整元素的位置，使元素的中心位置保持不变。在使目标元素放大的过程中，我们不想周围的元素被往外挤，因此通过把元素的位置修改为absolute（绝对位置）以及显式地

① show()命令就是这样，使元素固定在其左上角的位置进行放大。——译者注

设置元素的位置坐标，从而把元素从页面的流布局中提取出来。因为我们指定opacity的值为hide，所以动画结束时元素就自动隐藏（从显示器上删除）。

150

如图5-5所示，用浏览器加载页面chapter5/custom.effects.html之后，可以观察这3种自定义效果的任何一种。

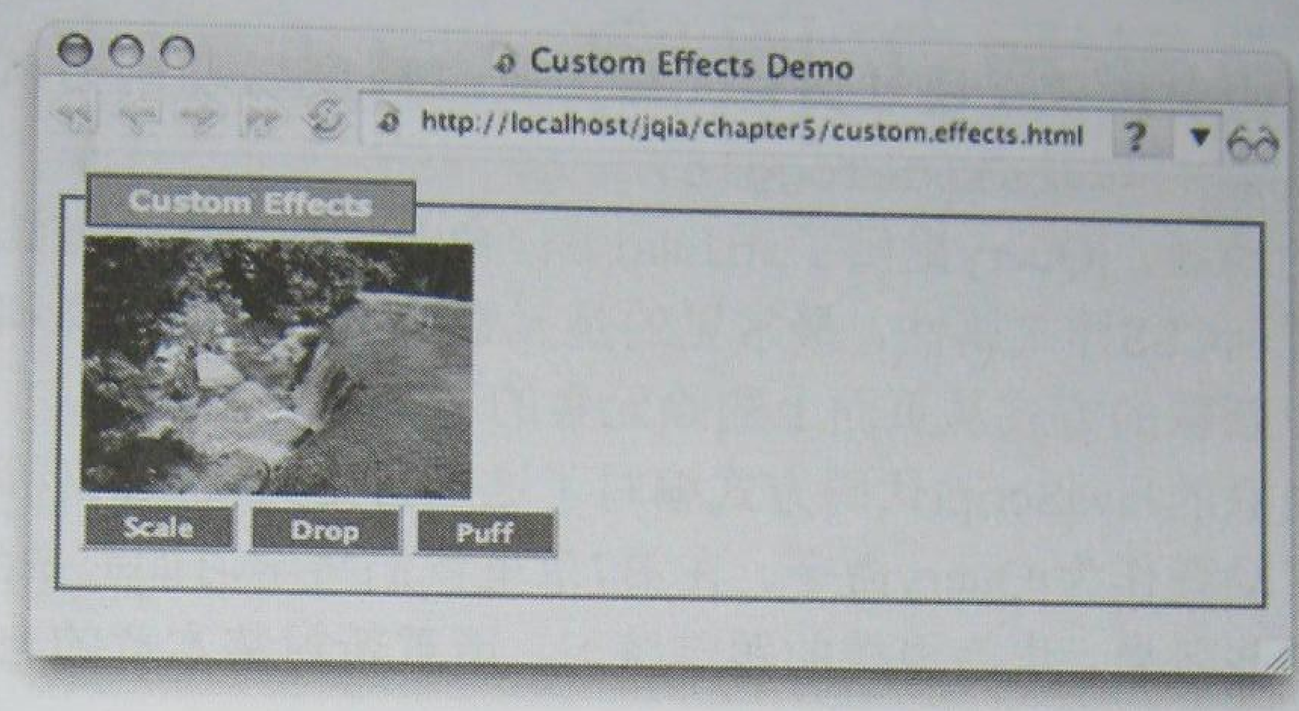


图5-5 演示放大、坠落和消散三种自定义效果的页面的初始显示

我们有意地缩小浏览器窗口以便于截屏。为了便于观察效果的行为，在运行这个页面的时候你可以放大浏览器窗口。尽管我们想要显示这些效果如何操作，但是（静态的）截屏有着明显的局限。不过图5-6显示正在进行中的消散效果。

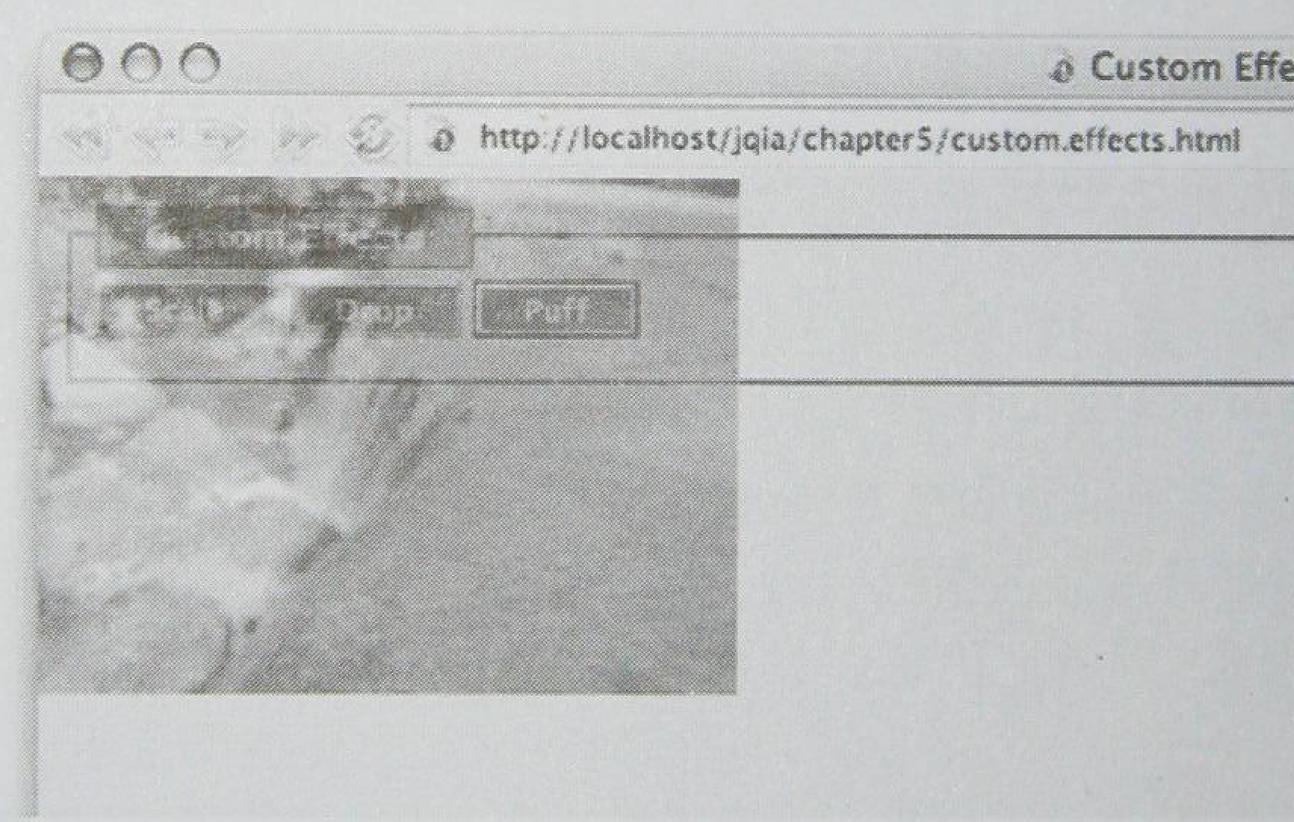


图5-6 消散效果一边放大和移动图像，一边减小图像的不透明度

151

我们把任务留给你：在这个页面上试验各种效果并观察它们的行为。

5.4 小结

本章介绍jQuery提供的可以直接使用的动画效果，以及允许我们创建自定义动画的animate()包装器方法。

不带参数地调用show()和hide()命令时，从显示器上以非动画方式使元素立即显示和隐藏。而通过传递控制动画速度的参数，以及提供可选的回调函数供动画结束之时调用，我们就能

利用这两个命令来实现使元素显示和隐藏的动画方式的版本。toggle()命令用于使元素在显示和隐藏状态之间进行切换。

第2组包装器方法fadeOut()和fadeIn()，则通过调整元素的不透明度最终在显示器里删除元素或展现元素。第3个方法fadeTo()，以动画方式表现已包装元素的不透明度的变化，但不会从显示器上删除元素。

第3组jQuery内建的动画效果通过调整元素的垂直高度以动画方式删除或展现元素：slideUp()、slideDown()以及slideToggle。

为了建立自定义的动画，jQuery提供了animate()命令。利用这个方法，我们能够以动画方式表现接受数字值的任何CSS样式属性，最常见的是元素的不透明度属性和尺寸相关的属性。我们已经探讨如何编写以新颖的方式从页面上删除元素的一些自定义动画。

我们在页面上以内联的JavaScript代码方式编写了这些自定义效果的代码。更为常见和有用的办法是封装这些自定义动画作为jQuery命令。在第7章里我们将学习如何做到这一点，在阅读第7章后鼓励你重新访问这些效果。作为很棒的跟踪练习，请重新封装本章的自定义效果以及任何你能想出的效果。

但在编写自己的jQuery扩展之前，我们要学习jQuery提供的一些高级的函数。

第6章 jQuery实用工具函数

本章内容

- jQuery的浏览器检测标志
- 使用jQuery和其他库
- 用于操作数组的函数
- 扩展对象与合并对象
- 动态地加载新的脚本

到目前为止，我们已经用了好几章内容来探讨jQuery命令。“命令”这个术语是指一系列的方法，用来操作被$()函数包装的一组DOM元素。但是你可能联想到在第1章里我们也介绍了实用工具函数的概念，是指一系列的函数，命名空间为$但不操作包装集。实用工具函数可以看作是顶层函数，不同之处是它们定义在$实例上，而不是定义在window实例[1]上。

通常来说，实用工具函数要么操作除DOM元素（毕竟那属于命令的权限范围）以外的JavaScript对象，要么执行一些非对象相关的操作。

你可能想知道为什么一直等到这一章才介绍实用工具函数。好，我们有两个主要的理由如下。

- 我们想引导你从利用jQuery命令的角度进行思考，而不是诉诸较低级别的操作。较低级别的操作可能让人觉得较为熟悉却不如利用jQuery命令进行编程来得高效或简便。
- 因为在操作页面上的DOM元素时命令处理了我们想做的大多数事情，所以这些较低级别的函数通常在编写命令（以及其他扩展）时最为有用（在第7章里我们将探讨如何编写jQuery插件），而不是用在页面级别的代码里。

在本章里，我们终于开始正式地介绍大多数$级别的实用工具函数以及几个有用的标志。涉及Ajax的实用工具函数的介绍，将推迟到专门讲述jQuery的Ajax功能的那一章（第8章）。

下面从已经提及的那些标志开始讲述。

6.1 利用 jQuery 标志

jQuery给页面作者和插件作者的一些信息，不是通过方法或函数提供的，而是通过定义在$上的变量而提供的。这些标志（定义在$上的变量）通常用于帮助检测当前浏览器的功能，以便

---

① window实例是浏览器对象中的顶层对象，用来表示浏览器所打开的窗口。例如，window对象的alert()方法可以看作是“顶层函数”，因为可以省略前缀window.而直接调用alert()。——译者注

让我们在必要时能够基于这些信息做出决定。

为公共用途而设立的jQuery标志如下。

- `$.browser`
- `$.boxModel`
- `$.styleFloat`

154

首先看jQuery如何通知我们正在使用的是哪一个浏览器。

6.1.1　检测用户代理

到目前为止已介绍的jQuery命令使我们免于处理浏览器差异，即便是在传统的问题区域，比如事件处理，对此我们充满感激与欣喜。但是，如果我们是命令（或其他扩展）的编写者，就经常需要处理不同浏览器的操作方式的差异，让命令（或扩展）的用户不必考虑这些差异。

但在研究jQuery在这点上如何帮助我们之前，首先介绍浏览器检测的概念。

1．为什么浏览器检测是可憎的

好吧，或许“可憎的”这个字眼用得太重了，但除非绝对必要（在没有其他可用选择时），否则不应使用浏览器检测这个办法。

浏览器检测乍看起来似乎是处理浏览器差异的合理办法。毕竟这样说起来很容易：“我知道浏览器X的功能集是什么，因此对浏览器X进行检测绝对行得通，对不对？”其实浏览器检测是充满隐患和问题的。

反对这个技术的主要论点之一是浏览器种类繁多，再加上相同浏览器的不同版本内有着不同的支持级别，使“浏览器检测”成为不具可扩展性的问题解决办法。

你可能在想：“好，我只需对IE和Firefox进行检测。”但是你为什么排斥不断增加的Safari用户呢？怎么对待Opera呢？此外还有一些占有一席之地、并非无足重轻的浏览器，与其他更为流行的浏览器共享功能子集。例如，Camino在其Mac友好的用户界面背后使用与Firefox相同的技术；OmniWeb使用与Safari相同的呈现引擎。

没有必要排斥对这些浏览器的支持，但对这些浏览器进行检测是一种极大的痛苦。而那甚至还没有考虑版本之间的差异，例如IE6与 IE7之间的差异。

反对浏览器检测（有时也称为嗅探）的另一个论点，是弄清楚“谁是谁”正在变得越来越困难。浏览器通过设置名为*用户代理*[①]字符串的请求标头（request header）来标识自身。解析这个字符串就已经让某些人望而却步。况且现在许多浏览器允许用户篡改[②]这个字符串，因此我们即使在解析之后还不能相信那就是正确的结果！

一个名为`navigator`的JavaScript对象提供关于用户代理的一部分信息，但它也存在浏览器差异。我们几乎为了进行浏览器检测而必须进行浏览器检测！

155

浏览器检测具有下列的特征。

- *不精确*——代码在某些浏览器中运行时意外地阻碍某些功能。
- *不可扩展*——为了理清头绪而导致数目巨大的嵌套的`if`和`if-else`语句。

---

① 用户代理是帮助客户端连接服务器的一个程序，在本书中是指浏览器。

② 欺骗（spoof），微软术语网站定义为“使传输看起来像是来自执行操作的用户以外的用户”。——译者注

- 不准确——由于用户篡改（欺骗）而导致假的用户代理信息。

显而易见，无论何时，只要有可能我们就要避免采用浏览器检测这个办法。

但是我们能够做点别的什么来替代呢？

2. 替代方案是什么

如果仔细想想，其实我们并不真的对某人使用哪种浏览器感兴趣，不是吗？我们考虑浏览器检测的唯一理由，是为了弄清楚可以利用浏览器的哪些功能。我们寻求的是浏览器的功能，而利用浏览器检测却不过是确定那些功能的笨拙的方式。

我们应该对那些功能是什么进行识别，而不应该试图通过识别浏览器而对那些功能进行推断。众所周知的对象检测技术允许基于特定的对象、属性甚至方法的存在与否而对代码进行分支。

作为示例，让我们回想关于事件处理的那一章（第4章）。记得有两个高级的事件处理模型：W3C标准的DOM第2级事件模型和专有的IE事件模型。两个模型都在DOM元素上定义方法用来建立监听器，但各自使用不同的方法名称。标准事件模型定义`addEventListener()`方法，而IE事件模型定义`attachEvent()`方法。

假设在（或许正确地）确定当前使用什么浏览器时我们经受了痛苦和懊恼，现在利用浏览器检测编写代码如下：

```
...
complex code to set flags: isIE, isFirefox and isSafari
...
if (isIE) {
  element.attachEvent('onclick',someHandler);
}
else if (isFirefox || isSafari) {
  element.addEventListener('click',someHandler);
}
else {
  throw new Error('event handling not supported');
}
```

暂且不谈以下事实：这个示例省略了用来设置标志`isIE`、`isFirefox`和`isSafari`的任何必定复杂的代码。起码我们不能肯定这些标志是否准确地表示当前使用的浏览器。更有甚者，如果使用Opera、Camino、OmniWeb或许多其他不太知名的浏览器（可能完全支持标准模型），这段代码将抛出一个错误。

考虑这段代码的以下变化：

```
if (element.attachEvent) {
  element.attachEvent('onclick',someHandler);
}
else if (element.addEventListener) {
  element.addEventListener('click',someHandler);
}
else {
  throw new Error('event handling not supported');
}
```

这段代码并不进行大量复杂的并且根本不可靠的浏览器检测，而是自动地支持所有的浏览

器，只要浏览器能够支持两个竞争事件模型的任何一个。好得多！

对象检测较之浏览器检测具有极大的优越性。对象检测更为可靠，并且不会意外地阻碍浏览器（该浏览器支持我们所测试的功能），只不过因为我们不了解那个浏览器的功能。

注意　除非绝对必要，否则就连对象检测也应该尽量避免。如果我们能够想出跨浏览器的解决方案，则较之任何类型的代码分支，应该优先采用跨浏览器的解决方案。

尽管对象检测较之浏览器检测可能具有优越性，但却未必总能为我们解围。有时候我们必须做出特定于浏览器的判断（不久就会看到示例），而且只能利用浏览器检测做出这种判断。

总而言之，我们找时间最终回答这个问题……

3. 烦人的浏览器标志

如果只能利用浏览器检测，则jQuery提供在加载库时就已建立的一组标志用来协助代码分支：甚至在任何就绪处理程序执行之前就使这组标志可用。这组标志被定义为对象实例（其引用为`$.browser`）的属性。这组标志的正式语法如下。

57

标志语法：$.browser

`$.browser`

定义一组标志，用来显示当前的用户代理属于哪一个浏览器家族。这些标志如下：

- `msie`　如果用户代理标头[1]把浏览器标识为微软的IE浏览器，则设置为`true`；
- `mozilla`　如果用户代理标头把浏览器标识为基于Mozilla的任何浏览器，比如Firefox、Camino和Netscape，则设置为`true`；
- `safari`　如果用户代理标头把浏览器标识为Safari或基于Safari的任何浏览器，如OmniWeb，则设置为`true`；
- `opera`　如果用户代理标头把浏览器标识为Opera，则设置为`true`；
- `version`　设置为浏览器的呈现引擎的版本号。

请注意这些标志不会试图标识当前使用的特定的浏览器：jQuery根据用户代理属于哪一个浏览器家族而进行分类。每个家族内的浏览器表现相同的几组特征。特定的浏览器标识应该不是必需的。

绝大多数常用的现代浏览器将归类于这四个浏览器家族中的一个。

需要特别注意`version`属性，因为它不如我们所想的那么方便。设置到这个属性上的值并不是浏览器的版本（我们原先以为那样）而是浏览器的呈现引擎的版本。例如，在Firefox 2.0.0.2里执行时，报告版本为`1.8.1.6`——Gecko呈现引擎的版本。利用`version`属性对IE6和IE7进行区分倒还方便，包含的值分别是`6.0`和`7.0`。

我们在前面已提及有时不能求助于对象检测而必须采取浏览器检测。这种情况的示例之一

① 在本书中也就是HTTP header（HTTP头），是指HTTP请求或响应的顶部的消息列表。——译者注

是，如果浏览器之间的差异不在于它们提供不同的对象类或不同的方法，而在于传递到方法里的参数在浏览器之间执行不同的解释。在这种情况下，没有对象可供执行检测。

请看<select>元素的add()方法，定义如下：

```
selectElement.add(element,before)
```

在这个方法里，第1个参数标识将要添加到<select>元素的<option>或<optgroup>元素；第2个参数标识现有的<option>或<optgroup>元素（新元素将要添加到它的前面）。在标准兼容的浏览器里，第2个参数是现有元素的引用，而在IE浏览器里是现有元素的顺序下标。

158

因为无法执行对象检测以便确定是否应该传递一个对象引用或整数值，所以必须采取浏览器检测，如代码清单6-1所示，示例页面的文件可以在chapter6/$.browser.html找到。

代码清单6-1 浏览器检测

```
<html>
  <head>
    <title>$.browser Example</title>
    <link rel="stylesheet" type="text/css" href="../common.css">
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js"></script>
    <script type="text/javascript">
      $(function(){
        $('#testButton').click(function(event){
          var select = $('#testSubject')[0];
          select.add(
            new Option('Two and \u00BD','2.5'),
            $.browser.msie ? 2 : select.options[2]
          );
        });
      });
    </script>
  </head>

  <body class="plain">
    <select id="testSubject" multiple="multiple" size="5">
      <option value="1">One</option>
      <option value="2">Two</option>
      <option value="3">Three</option>
      <option value="4">Four</option>
    </select>
    <div>
      <button type="button" id="testButton">Click me!</button>
    </div>
  </body>
</html>
```

为第2个参数采取浏览器检测

这个示例建立带有4个选项的<select>元素和简单的按钮元素。在按钮元素上添加click事件处理程序，用于在第2个和第3个原始选项之间添加新的<option>元素。因为IE浏览器要求顺序下标2，而W3C标准浏览器要求第3个现有选项的引用，所以我们利用浏览器检测来建立代码

分支，通过检测的值将作为add()方法的第二个参数。

159　示例在各种浏览器里执行之前和之后的对比如图6-1所示。

图6-1　“添加选项”的功能在多种浏览器里毫无瑕疵地运行

示例代码已在6种现代浏览器（Firefox、Safari、Opera、Camino、IE7以及OmniWeb）里执行，分别代表jQuery的$.browser标志所支持的4个浏览器家族。正如我们所见，“添加选项”的功能

0　在各个浏览器中正确地运行。

毋庸赘言，认识到这一点非常重要：（通过浏览器检测）建立代码分支的这个办法并不令人满意。这个示例的代码假定所有的非IE浏览器在<select>元素的add()方法上遵循W3C标准。经过测试，我们已经确认如图6-1所示的5种非IE浏览器确实遵循W3C标准，但其他的浏览器怎么

样呢？不经测试你能知道Konqueror浏览器怎样运行吗？

浏览器检测采取的是一刀切的方法，因此底线就是为了知道特定的浏览器应该采取哪一个代码分支，要求彻底地调查清楚“代码将要支持的任何浏览器和平台”。

撇开浏览器检测这个主题，我们继续探讨有助于处理浏览器差异的另一个标志。

6.1.2 确定方框模型

通过布尔类型的`$.boxModel`标志可以知道当前页面使用的是哪一个方框模型：如果页面使用W3C标准的方框模型，被设置为`true`；如果页面使用IE浏览器的方框模型（有时称为传统方框模型），被设置为`false`。

“有什么差别呢？”你也许会问。

方框模型规定了如何决定元素（与其内边距和边框相连）的内容大小（外边距虽然也是方框模型的一部分，但不参与确定内容大小）。除IE以外大多数的浏览器，只支持W3C方框模型，然而IE浏览器能够根据页面的呈现模式（严格模式或兼容模式）而使用对应的方框模型。使用哪个呈现模式取决于在页面上声明的DOCTYPE（或者没有DOCTYPE声明）。

详细探讨各种DOCTYPE问题已经超出本书的范围，但一些经验方法在大部分的时间都是有用的。

- 如果页面包含有效的、可识别的DOCTYPE声明，则以严格模式呈现。
- 如果页面没有DOCTYPE声明或包含无法识别的DOCTYPE声明，则以兼容模式呈现。

如果你想要深入研究DOCTYPE问题，一个极好的可供参考的资源是http://www.quirksmode.org/css/quirksmode.html。

简而言之，两个方框模型之间的差异集中在如何解释`width`和`height`样式上。在W3C方框模型里，由这些值来决定元素的内容尺寸，不包括内边距和边框宽度；在传统方框模型里，这些值则包括内边距和边框宽度在内。

假定有一个元素已经应用下列的样式：

```
{
  width: 180px;
  height: 72px;
  padding: 10px;
  border-width: 5px;
}
}
```

在解释元素大小上，两个方框模型的不同方式如图6-2所示。

在W3C方框模型里，元素的内容的大小恰好是`width`和`height`特性所指定的180×72像素。内边距和边框应用在180×72像素的方框之外，使整个元素的总覆盖面积达到210×102像素。

如果使用传统方框模型，则整个元素呈现在`width`和`height`特性所指定的180×72像素的方框里，使内容的大小减小到150×42像素。

双方都有人声称己方的模型比对方的模型更为直观或正确，实际上我们必须忍受这些差异。

如果我们的代码必须处理在元素呈现方式上的这些差异，则`$.boxModel`标志让我们知道哪个方框模型正在生效，因此可以进行相应的计算。

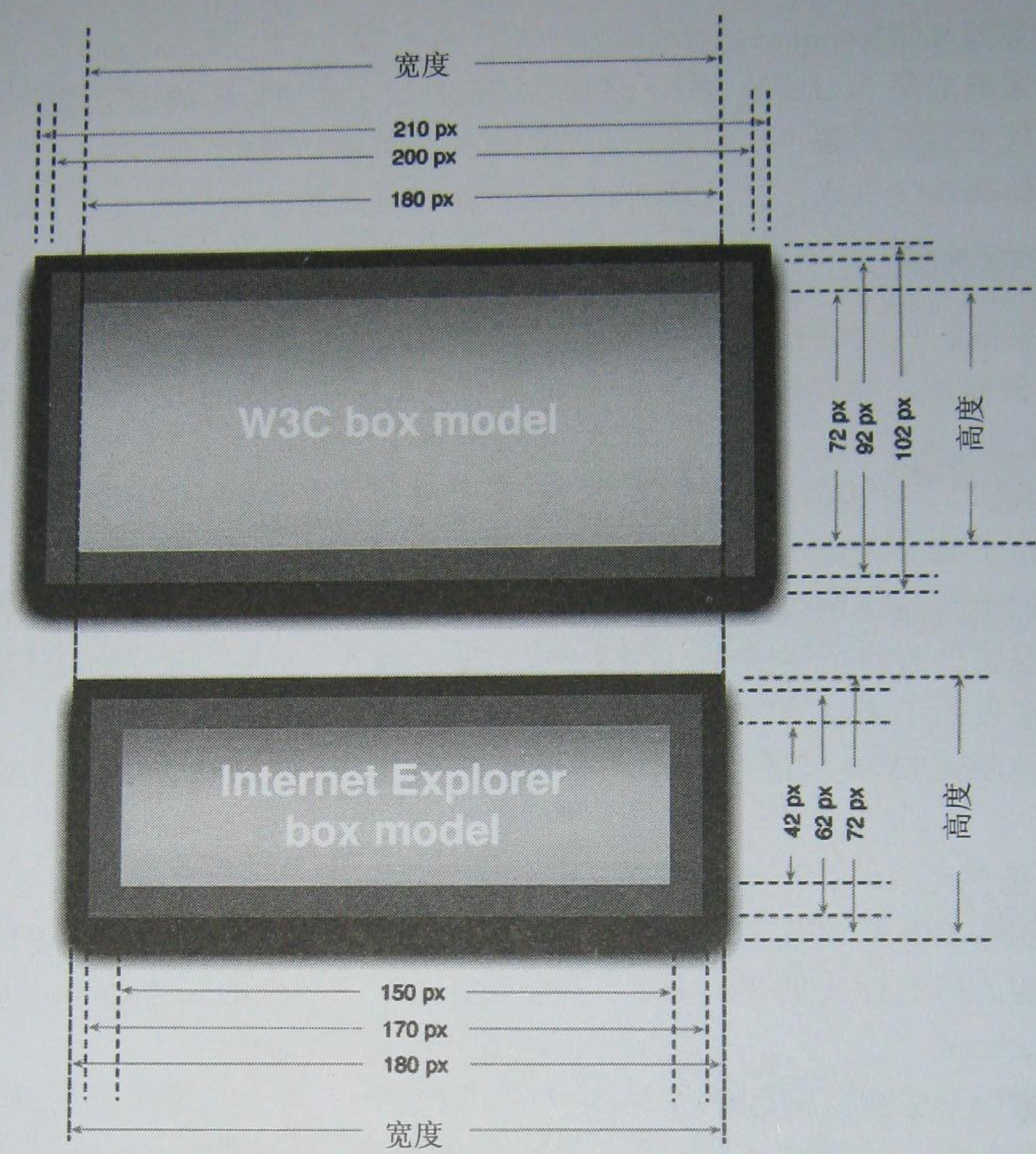


图6-2　在W3C方框模型里，元素的宽度不包括内边距和边框的宽度，而在IE方框模型里则不是那样

6.1.3　检测要用的正确的浮动样式

jQuery为浏览器的另一个功能上的差异提供标志，用来表示在元素的style属性里的float样式的名称。这就是$.styleFloat标志，求值为字符串，供属性名称使用。

例如，为了设置元素的float样式的值，我们使用

```
element.style[$.styleFloat] = 'left';
```

在style属性的浮动样式的命名上，不同的浏览器之间存在差异。上面的语句处理了这个差异，因为$.styleFloat标志在IE浏览器里求值为styleFloat，在其他的浏览器里则求值为cssFloat。

注意　$.styleFloat标志一般没有必要在页面上使用。css()包装器方法在与float属性名称一起使用时，能够自动地（利用$.styleFloat标志）选择正确的属性。插件以及其他扩展的作者才是$.styleFloat标志的目标用户，因为在那些情况下较低级别的控制变得十分重要。

我们马上就要离开标志的世界去查看jQuery提供的实用工具函数。

6.2 使用 jQuery 和其他库

早在1.3.6节就已经介绍jQuery团队所提供的细心体贴的办法，用来在同一个页面上轻松地使用jQuery与其他库。一般说来，$变量的定义是最大的争论焦点：如果在同一个页面上使用jQuery与其他库就会引发冲突。我们知道，jQuery使用$作为jQuery名称的别名，因此jQuery所展示的每个功能都用到$。但其他库，特别是最引人注目的Prototype也使用$名称。

163

jQuery提供$.noConflict()实用工具函数，用于放弃对$名称的控制，让给想要使用$名称的任何其他库。这个函数的语法如下。

函数语法：$.noConflict

$.noConflict()

归还$名称的控制权给另一个库，因此允许在页面上其他库与jQuery混用。
一旦这个函数被执行，jQuery功能就必须利用jQuery名称进行调用，而不能利用$名称。

参数

无

返回

未定义

因为$只不过是jQuery的别名，所以在应用$.noConflict()之后jQuery的全部功能依然可用，尽管是通过使用jQuery标识符。为了对已失去的、简短而惹人喜爱的$进行补偿，我们可以给jQuery定义更短的、却不会引起冲突的别名，如

```
var $j = jQuery;
```

另一个经常采取的习惯用法是创建一个环境，在那里$名称的作用域被设定为引用jQuery对象。在扩展jQuery时常常使用这个技巧，使用者当中尤以插件作者为甚。因为插件作者不能对于页面作者是否已经调用$.noConflict()做出任何假设，当然也不能调用$.noConflict()，以免使页面作者的希望破灭。

这个习惯用法如下所示：

```
(function($) { /* function body here */ })(jQuery);
```

如果这个表示法使你晕头转向，也不用担心！如果是第一次碰到这个习惯用法，可能有人觉得它很奇怪，其实它是相当浅显易懂的。

下面我们剖析这个习惯用法的第一部分：

16

```
(function($) { /* function body here */ })
```

这个部分声明一个函数，并且用圆括号把它括起来，由此构成一个表达式，这个匿名函数的引用作为表达式的值而被返回。这个函数需要名为$的单个参数。在这个函数的体内，可以利用$标识符引用任何传递给这个函数的东西。因为参数声明优先于在全局作用域里的任何同名标识符，所以在这个函数外面为$定义的任何值在这个函数内被传入的参数所取代。

这个习惯用法的第二部分：

```
(jQuery)
```

在匿名函数上执行函数调用（传递jQuery对象作为匿名函数的实参）。

因此在匿名函数体内，jQuery对象被$名称所引用，而不管在匿名函数外面，是否$名称已经为Prototype或其他库所定义。棒极了，不是吗？

如果使用这个技巧，则在匿名函数体内，$的外部声明是不可用的。

除了在1.3.3节里已经探讨的两种定义就绪处理程序的语法以外，这个习惯用法的变化形式也被经常用来构成第三种定义就绪处理程序的语法。考虑下列代码：

```
jQuery(function($) {
  alert("I'm ready!");
});
```

如同在1.3.3节所见，通过传递一个函数作为jQuery函数的参数，由此把这个函数声明为就绪处理程序。但是这一次，我们声明$标识符作为单个参数传递给就绪处理程序。因为jQuery总是把jQuery对象的引用作为唯一的第一个参数传递给就绪处理程序，所以这保证了在就绪处理程序内jQuery对象被$名称所引用，而不管在就绪处理函数体的外面，是否$名称已被定义。

利用简单的测试来证明这一点。作为测试的第一部分请看代码清单6-2所示的HTML文档。

代码清单6-2　就绪处理程序之测试1

```
<html>
  <head>
    <title>Hi!</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js">
    </script>
    <script type="text/javascript">
      var $ = 'Hi!';
      jQuery(function(){
        alert('$ = '+ $);
      });
    </script>
  </head>
  <body>
  </body>
</html>
```

❶ 用自定义的值来重写$名称

❷ 声明就绪处理程序

在这个文档里，我们导入jQuery库（我们知道jQuery库预定义了全局名称jQuery以及它的别名$）。然后把全局名称$重新定义为字符串值❶，由此覆盖了原始定义。在示例中以简单的字符串值取代$的原始定义是为了简洁，其实可以通过导入其他库比如Prototype来重新定义$。

然后我们定义就绪处理程序❷，它的唯一的操作就是弹出警告消息框以便显示$的值。

在加载这个页面时，我们看到弹出的警告消息框，如图6-3所示。

请注意，就绪处理程序处在全局变量$的作用域，因此不出所料，$拥有从字符串赋值语句得来的值。如果我们只想在就绪处理程序里使用jQuery库所定义的$值，就会非常失望。

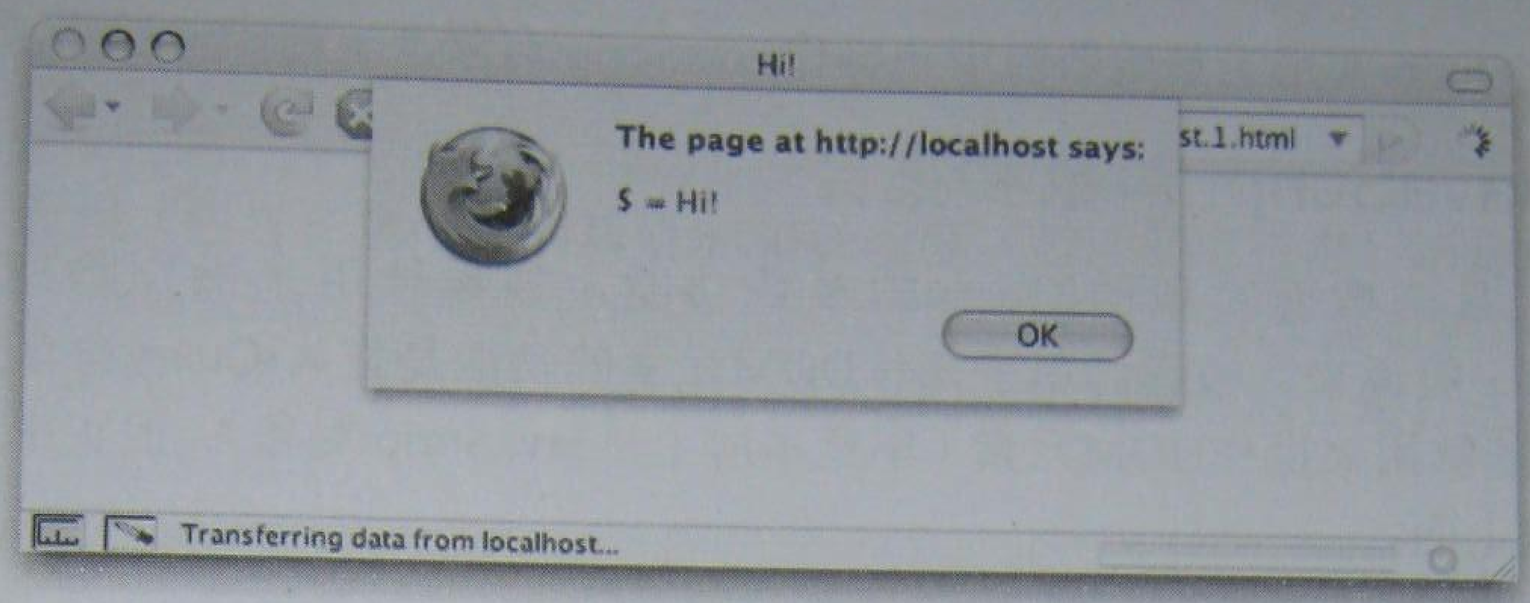


图6-3 由于重定义生效，在就绪处理程序里$的值为“Hi!”

接着修改示例文档的一处地方。代码清单6-3只显示文档的已修改部分。小改动以粗体突出显示。

166

代码清单6-3 就绪处理程序之测试2

```
<script type="text/javascript">
  var $ = 'Hi!';
  jQuery(function($){
    alert('$ = '+ $);
  });
</script>
```

我们所作的唯一改动就是给就绪处理程序添加名为$的参数。加载修改之后的页面，我们看到完全不同的结果，如图6-4所示。

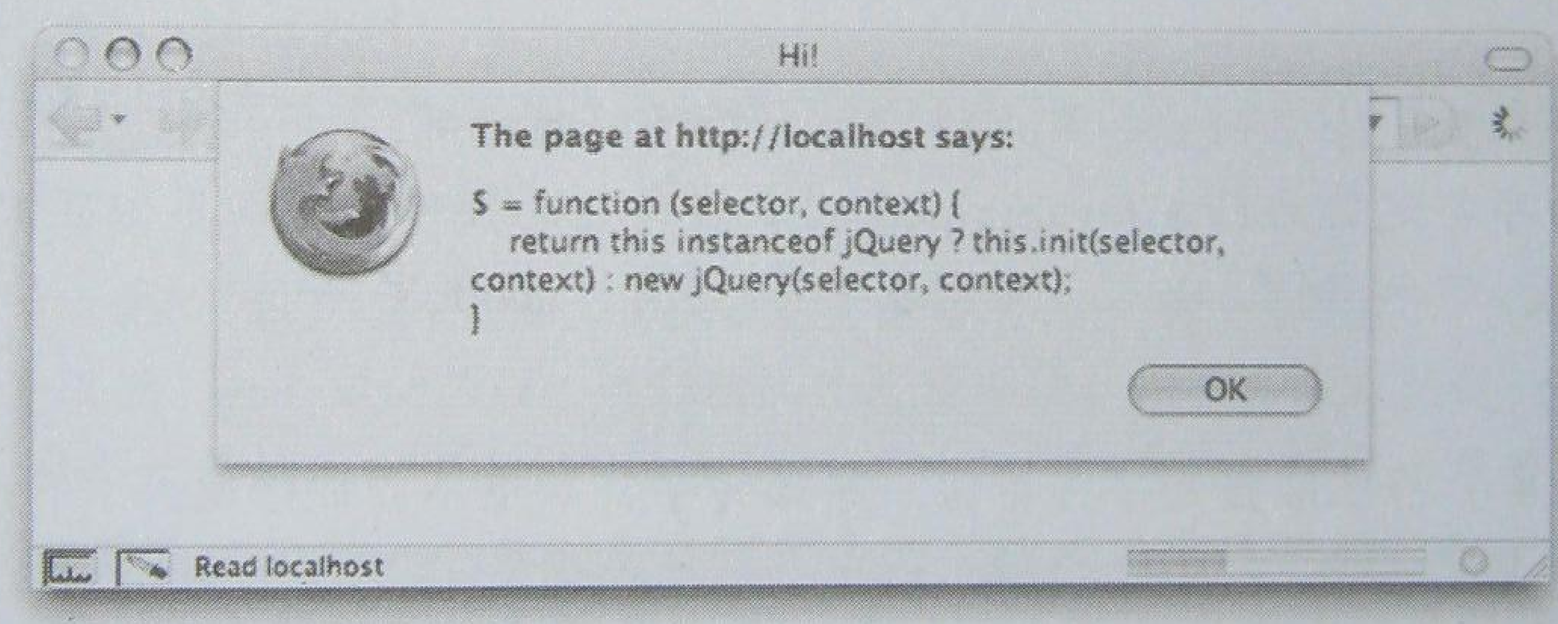


图6-4 现在警告消息框显示jQuery库所定义的$值，因为它的定义在函数内已被强制生效

好，那可能与我们的预言不完全一致。但是快速地阅读一下jQuery库的源代码就会发现，因为我们在外部函数[1]里把就绪处理程序的第一个参数声明为$，而jQuery函数被jQuery库作为唯一的参数传递到所有的就绪处理程序里（因此警告消息框显示jQuery函数的定义），所以jQuery函数被$名称所引用。

在编写可重用的组件时，对$的定义最好采取这种防范措施，因为这些组件有可能被用于那些使用$.noConflict()的页面。

① 即jQuery()。——译者注

还有大量的jQuery实用工具函数被用来操作JavaScript对象。下面探讨这些函数。

6.3　操作 JavaScript 对象和集合

作为实用工具函数而实现的jQuery的多数功能，被设计用来操作除DOM元素以外的JavaScript对象。一般说来，被设计用来操作DOM元素的功能是作为jQuery命令提供的。虽然一些实用工具函数能够用来操作DOM元素（毕竟本质上是JavaScript对象），但实用工具函数的焦点并不聚集在DOM上面。

167

下面从基本的实用工具函数讲起。

6.3.1　修整字符串

简直是让人莫名其妙，JavaScript的`String`实现竟然没有一个方法用来删除字符串实例的开头和结尾的空白字符。一般说来，在其他大多数语言里这种基本功能是`String`类的一部分，但在JavaScript里缺少这个有用的功能，让人觉得不可思议。

然而修整字符串在许多JavaScript应用里是常见的需求。一个显著的示例是表单数据验证。因为空白在屏幕上不可见（“空白”这个名称真是名副其实），用户一不小心就会在文本框或文本区里的有效项的后面（有时甚至是前面）意外地输入多余的空格字符[①]。在验证时我们想悄悄地从数据里修整这些前后空白，而不是警告用户说他们输入的数据项存在一些看不见的错误。

为了帮助我们解决问题，jQuery定义了`$.trim()`函数，语法如下。

函数语法：$.trim

`$.trim(value)`

从已传入的字符串里删除任何前导或尾随空白字符并返回结果。

这个函数把空白字符定义为匹配JavaScript正则表达式`\s`的任何字符，不但匹配空格字符，而且匹配换页、换行、回车、制表，以及垂直制表字符，还有Unicode字符`\u00A0`、`\u2028`和`\u2029`。

参数

`value`　（字符串）将被修整的字符串值。原始的字符串值不会被修改。

返回

修整之后的字符串

利用这个函数对某处文本字段的字符串值进行修整的小示例如下：

```
$('#someField').val($.trim($('#someField').val()));
```

请注意，这个函数并不检查传入的参数以便确保是字符串值。因此，如果我们给这个函数传递任何其他类型的值，则可能得到令人遗憾的未定义的结果（大概是一个JavaScript错误）。

168

下面探讨用来操作数组以及其他对象的一些函数。

---

① 空格字符以“十进制的ASCII码 '转义字符'（英文名称）中文名称”的格式列举如下：9 '\t'(tab)水平制表；10 '\n'(new line)换行；11'\v'(vertical tab)垂直制表；12 '\f'(form feed)换页；13 '\r'(return)回车；32 ' '(space)空格。——译者注

6.3.2 对属性和集合进行迭代

我们时常拥有其他组件所构成的非标量值，并且需要对被包含的每一项进行迭代。不管容器元素是JavaScript数组（包含任意个数的其他JavaScript值，包括数组在内）还是JavaScript对象的实例（包含属性），JavaScript语言都会提供办法对它们进行迭代。如果是数组，则利用for循环对数组的元素进行迭代；如果是对象，则利用for-in循环将对象的属性进行迭代。

可以各编写一个示例如下：

```
var anArray = ['one','two','three'];
for (var n = 0; n < anArray.length; n++) {
  //do something here
}

var anObject = {one:1, two:2, three:3};
for (var p in anObject) {
  //do something here
}
```

以上代码相当简单，但也许有的人认为语法没有必要那么冗长和复杂（这个批评经常针对for循环）。我们知道，jQuery为DOM元素的包装集而定义each()命令，用来轻松地迭代包装集里的元素而无需杂乱的for语法。对于普通的数组和对象，jQuery则提供名为$.each()的类似的实用工具函数。

不管是迭代数组的元素，还是迭代对象的属性，使用的语法是一致的。

函数语法：$.each

$.each(container,callback)

对传入的容器的每一项进行迭代，为每一项调用传入的回调函数。

参数

container　（数组|对象）一个数组，其每一项将被迭代；或一个对象，其属性将被迭代。

callback　（函数）一个回调函数，为容器的各元素而被调用。如果容器是数组，则为数组的每个元素调用回调函数；如果容器是对象，则为对象的每个属性调用回调函数。回调函数的第一个参数是数组元素的下标或对象属性的名称，第二个参数是对应的数组元素或对象属性的值。调用的函数上下文（this）被设置为与第二个参数的值相同。

返回

容器对象

这个统一的语法可以用相同的格式来迭代数组或对象。用该函数来重新编写前面示例，如下所示：

```
var anArray = ['one','two','three'];
$.each(anArray,function(n,value) {
  //do something here
});
```

```
var anObject = {one:1, two:2, three:3};
$.each(anObject,function(name,value) {
  //do something here
});
```

虽然利用带有内联函数的$.each()，在选择语法上看起来似乎是半斤八两的方案，但这个函数使得易于编写可重用的迭代器函数，或易于把循环体提取到另一个函数里（为了代码清晰这个目标），如下所示：

```
$.each(anArray,someComplexFunction);
```

请注意，如果是对数组的元素进行迭代，可以通过从迭代器函数返回false而跳出循环；但如果是将对象的属性进行迭代，即使从迭代器函数返回false也不会跳出循环。

有时候我们对数组进行迭代以便挑选元素构成新的数组。下面看jQuery如何轻松地实现。

6.3.3　对数组进行筛选

遍历数组以便查找匹配特定标准的元素，是处理大量数据的应用的频繁需求。我们也许想要对数据进行筛选，查找落在特定界限之上（或之下）的项，或匹配特定模式的项。对于这个类型的任何筛选操作，jQuery都提供$.grep()实用工具函数。

$.grep()函数的名称也许让人误以为该函数利用正则表达式的功能，就像UNIX的同名grep命令那样。其实$.grep()实用工具函数使用的筛选标准不是正则表达式，而是调用者所提供的回调函数（定义标准以便决定一个数据值应该从结果集里排除或保留）。jQuery不会阻止回调函数利用正则表达式去完成任务，但是正则表达式的利用并不是自动的。

170　$.grep()函数的语法如下。

函数语法：$.grep

$.grep(array,callback,invert)

遍历已传入的数组，为各元素分别调用回调函数。回调函数的返回值决定是否把当前元素收集到新数组（新数组作为$.grep()函数的值而被返回）。如果invert参数被省略或设置为false，则回调函数返回true时导致当前元素被收集；如果设置为true，则回调函数返回false时导致当前元素被收集。原始的数组不会被修改。

参数

array　（数组）被遍历的数组。检查数组的各元素，决定是否收集到新数组。原始数组不会被该操作以任何方式修改。

callback　（函数|字符串）一个回调函数。函数的返回值决定是否收集当前元素。如果invert参数被省略（或设置为false），函数返回true时导致当前元素被收集；如果设置为true，则函数返回false时导致当前元素被收集。

函数被传入两个参数：当前元素以及当前元素在原始数组里的下标。

callback参数也可以是一个字符串，该字符串会被转换为回调函数。下文讨论有关细节。

invert　（布尔型）如果指定为true，则反转函数的正常操作。

返回
被收集的元素所构成的数组

如果想要筛选一个数组，获取所有大于100的值。可以用类似下面的语句来实现：

```
var bigNumbers = $.grep(originalArray,function(value) {
                  return value > 100;
                });
```

我们传递给$.grep()的回调函数可以进行任何必要的处理，以便决定是否应该包含当前的值。这个决定可能像以上示例那样简单，也有可能复杂得如同向服务器发起同步的Ajax调用以便决定是否包含或排除当前值。

如果做出决定就像以上示例那样简单，那么可以使用jQuery所提供的简写方式，使语句更加简洁——提供字符串形式的表达式。例如，可以重新编写上面的代码片段如下：

171

```
var bigNumbers = $.grep(originalArray,'a>100');
```

如果callback参数被指定为字符串，jQuery会自动地利用已传入的字符串作为return语句的值而生成一个回调函数，并且传递两个参数：a作为当前值，i作为当前下标。那么这个示例的生成函数等同于

```
function(a,i){return a>100;}
```

即使$.grep()函数不直接使用正则表达式（别看它的名称），JavaScript正则表达式作为强大的工具，可以在回调函数里使用以便决定当前元素是否从结果数组中排除或包含。考虑这种情况：我们有一个数组并且想要标识其中不匹配美国邮政编码（也称为ZIP编码）模式的任何值。

美国邮政编码由5位十进制数构成，（可选地）后跟短横杠以及另外的4位十进制数。表达这个模式的一个正则表达式为/^\d{5}(-\d{4})?$/，那么可以利用下列语句筛选源数组的不符合标准的项：

6

```
var badZips = $.grep(
                originalArray,
                function(value) {
                  return value.match(/^\d{5}(-\d{4})?$/) != null;
                },
                true);
```

在这个示例中值得注意的是，利用String类的match()方法确定一个值匹配模式与否，以及指定$.grep()的invert参数为true以便排除与模式相匹配的任何值。

获取数据的子集或许不是我们在数组上执行的唯一操作。下面探讨jQuery所提供的另一个以数组为目标的函数。

6.3.4 对数组进行转换

数据不一定符合我们所需要的格式。在以数据为中心的Web应用里另一个经常执行的普通操作是把一组值转换为另一组值。尽管编写一个for循环从一个数组创建另一个数组是简单的事，jQuery还是提供$.map实用工具函数使操作更加简便。

172

函数语法：$.map

$.map(array,callback)

迭代已传入的数组，为数组的各元素分别调用回调函数，并把回调函数的返回值收集到新数组

参数

array （数组）一个数组。数组的各元素将被分别转换为新数组的各元素

callback （函数|字符串）一个回调函数。回调函数的返回值将被收集到新数组（新数组将作为调用$.map()函数的结果而被返回）。回调函数被传入两个参数：当前元素以及当前元素在原始数组里的下标。callback参数也可以是一个字符串，该字符串会被转换为回调函数。下文讨论有关细节。

返回

包装集

请看下面的小示例，显示在实际中如何运用$.map()函数。

```
var oneBased = $.map([0,1,2,3,4],function(value){return value+1;});
```

这个语句把值从0开始的数组，转换为值从1开始的对应的数组。

就像$.grep()那样，对这样的简单表达式，我们可以传递用于表示表达式的字符串使语句更为简洁。在这种情况下，自动生成的函数被传入名为a的参数的值（不同于$.grep()的是，数组的下标不会传递给这个自动生成的函数）。

```
var oneBased = $.map([0,1,2,3,4],'a+1');
```

值得注意的另一个重要的行为是，如果函数返回null或undefined，结果就不被收集。在这种情况下，结果数组的长度将小于原始数组的长度，而结果数组项与原始数组项之间的一对一的对应顺序将被打乱。

下面探讨一个稍微复杂的示例。假定我们有一个可能是从表单字段收集得来的字符串数组，要求是表示数字值的字符串，我们想把这个字符串数组转换为对应的Number实例的数组。因为不能保证不出现无效的数字字符串，所以我们必须采取防范措施。考虑下列代码：

173

```
var strings = ['1','2','3','4','S','6'];

var values = $.map(strings,function(value){
  var result = new Number(value);
  return isNaN(result) ? null : result;
});
```

我们从字符串值的一个数组开始。每个字符串值要求表示一个数字值。但一个排印错误（也可能是用户输入错误）导致S取代了正确的5。这段代码通过检查构造函数所创建的Number实例来处理这种情况，判断字符串是否成功地转换为数字。如果转换失败，返回值将是常量Number.NaN。但有趣的是Number.NaN从定义上来说不等于任何其他值，包括其自身在内。因此表达式Number. NaN==Number.NaN的值为false！

因为不能用比较运算符来测试NaN（顺便说一下，它是Not a Number的缩写，即“不是一个

数字”），所以JavaScript提供isNaN()方法，我们用该方法来测试字符串转换为数字的结果。

在这个示例中，失败的情况下我们返回null，确保结果数组只包含有效的数字值而排除任何错误值。如果想要收集所有的值，可以让转换函数为错误值返回Number.NaN。

$.map()的另一个有用行为是适当地处理以下情况：如果从转换函数返回的是一个数组，$.map()会自动地把数组的各个元素合并到结果数组里。考虑下列语句：

```
var characters = $.map(
  ['this','that','other thing'],
  function(value){return value.split('');}
);
```

这个语句把字符串数组转换为构成字符串的所有字符所形成的数组。执行之后，变量characters的值如下所示：

```
['t','h','i','s','t','h','a','t','o','t','h','e','r','','t','h',
➡ 'i','n','g']
```

这是利用String.split()方法来完成的，如果给这个方法传递空字符串作为分隔符，它就返回当前字符串的字符所形成的数组。这个数组作为转换函数的结果而被返回，并且这个数组的各个元素被合并到结果数组里。

174

jQuery对数组的支持不止于此。还有好几个小函数，你也许会觉得很好用。

6.3.5 从 JavaScript 数组上找到更多乐趣

你曾经需要弄清楚JavaScript数组是否包含特定值吗？或许还想知道那个值在数组里的下标，不是吗？

如果有过这样的经历，你会感激$.inArray()函数的。

函数语法：$.inArray

$.inArray(value,array)

返回已传入的值在数组里第一次出现时的下标。

参数

value （对象）搜索数组时的目标值。

array （数组）将被搜索的数组。

返回

该值在数组里第一次出现时的下标；如果该值搜索不到，就返回-1。

举一个利用这个函数的小示例：

```
var index = $.inArray(2,[1,2,3,4,5]);
```

这个语句的结果是返回下标值1并指派到index变量。

另一个有用的、数组相关的函数从类似数组的其他对象中创建JavaScript数组。

“类似数组的其他对象？到底什么是类似数组的对象？”你会提问。

jQuery认为任何对象只要有长度和索引（下标）项的概念，就是类似数组的对象。这个功能对于处理NodeList（节点列表）对象来说最为有用。考虑以下片段：

```
var images = document.getElementsByTagName("img");
```

这个语句把页面上的所有图像元素所构成的NodeList指派给images变量。

175

处理NodeList有些痛苦，因此把它转换为JavaScript数组之后事情就好办得多。jQuery的$.makeArray函数使得转换NodeList很轻松。

函数语法：$.makeArray

$.makeArray(object)

把已传入的类似数组的对象转换为JavaScript数组。

参数

object　（对象）将被转换的类似数组的对象（比如NodeList）。

返回

作为结果的JavaScript数组

这个函数的设计意图是用于不太使用jQuery的代码中，因为jQuery在其内部已为我们处理这一类事情。在不用jQuery来遍历XML文档并且处理NodeList对象时，这个函数也很有用。

另一个很少用到的函数（也许处理在jQuery外面所建立的数组时用得上）是$.unique()。

函数语法：$.unique

$.unique(array)

传入DOM元素的数组，该函数就返回原始数组中的唯一的元素所构成的数组。

参数

array　（数组）将被检查的DOM元素的数组。

返回

DOM元素的数组，由原始数组中的唯一的元素所构成。

同$.makeArray函数一样，jQuery在其内部用$.unique函数来确保我们所获取的元素列表包含各不相同的元素。该函数的设计意图是用在创建于jQuery的范围之外的元素数组上。

既然已经探讨jQuery如何轻松地处理数组，下面看它如何帮助我们操作普通老旧的JavaScript对象POJO。

6.3.6　扩展对象

虽然众所周知JavaScript提供一些功能使得它在许多方面的行为像面向对象语言，但是JavaScript不是大家所说的纯面向对象，因为它不支持某些功能。这些重要的功能之一是继承，也

176

就是通过扩展现有类的定义而定义新类的方式。

在JavaScript里用来模拟继承的模式是通过复制基础对象的属性到新对象里而扩展对象，也就是以基础对象的功能来扩展新对象。

注意 如果你是面向对象JavaScript的狂热爱好者，毫无疑问你将不仅熟悉怎样扩展对象的实例，还知道如何用对象构造器的prototype（原型）属性来扩展对象的蓝图。$.extend()既可以通过扩展prototype而实现基于构造器的继承，又可以通过扩展现有对象实例而实现基于对象的继承。因为对于有效地利用jQuery来说，理解这样的高级主题不是必要条件，所以这个主题虽然很重要却超出了本书的范围。

编写JavaScript代码来实现借助于复制的这种扩展是相当轻松的。然而，就像对待其他许多过程那样，jQuery预料到这个需求并提供现成的实用工具函数帮助解决问题。在第7章我们将看到这个函数的用途远不止于扩展对象，即便那样它的名称还是叫做$.extend()。函数的语法如下。

函数语法：$.extend

$.extend(target,source1,source2, ... sourceN)

用已传入的source1 ... soureN对象的属性来扩展已传入的target对象。

参数

target　（对象）目标对象，jQuery用源对象的属性来扩展目标对象的属性。jQuery用新的属性来直接修改目标对象，然后把目标对象作为函数的值而返回。与任何一个源对象的属性名称相同的、目标对象的任何属性，被覆盖为源对象的属性值。

source1 ... sourceN　（对象）一个或多个源对象，这些对象的属性被添加到目标对象。如果提供多个源对象，而源对象之间存在同名属性，则处在实参列表后部的源对象优先于处在实参列表前部的源对象。

返回

扩展之后的目标对象

177

我们把这个函数运用于实践。看看代码清单6-4的代码，也可在文件chapter6/$.extend.html找到。

代码清单6-4　对$.extend函数进行测试

```
<html>
  <head>
    <title>$.extend Example</title>
    <link rel="stylesheet" type="text/css" href="../common.css">
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js"></script>
    <script type="text/javascript"
            src="../scripts/support.labs.js"></script>
```

```
<script type="text/javascript">
  var target =  { a: 1, b: 2, c: 3 };
  var source1 = { c: 4, d: 5, e: 6 };
  var source2 = { e: 7, f: 8, g: 9 };

  $(function(){
    $('#targetBeforeDisplay').html($.toSource(target));
    $('#source1Display').html($.toSource(source1));
    $('#source2Display').html($.toSource(source2));
    $.extend(target,source1,source2);
    $('#targetAfterDisplay').html($.toSource(target));
  });
</script>
<style type="text/
 css">
    label { float: left; width: 108px; text-align: right; }
    p { clear: both; }
    label + span { margin-left: 6px; }
</style>
</head>

<body>
  <fieldset>
    <legend>$.extend() Example</legend>
    <p>
      <label>target (before) =</label>
      <span id="targetBeforeDisplay"></span>
    </p>
    <p>
      <label>source1 =</label>
      <span id="source1Display"></span>
    </p>
    <p>
      <label>source2 =</label>
      <span id="source2Display"></span>
    </p>
    <p>
      <label>target (after) =</label>
      <span id="targetAfterDisplay"></span>
    </p>
  </fieldset>
</body>
</html>
```

❶ 定义测试对象

❷ 显示扩展之前对象的状态

❸ 用源对象来扩展目标对象

❹ 显示扩展之后目标对象的状态

❺ 定义HTML元素用来显示信息

在这个简单的示例中，我们定义3个对象：一个目标对象和两个源对象❶。这个示例演示了如果使用`$.extend`把源对象合并到目标对象会发生什么情况。

声明对象之后，我们定义就绪处理程序，在其中操作这些对象。虽然这些对象是立即可用的，但是为了在页面上显示结果，必须等到HTML元素❺已经呈现之后才能开始显示结果。

在就绪处理程序内，在被定义用来保持结果的`<span>`元素里❷显示这3个对象的状态（如果你对`$.toSource()`函数如何工作感兴趣，可以在support.labs.js文件找到它的定义。第7章将探讨如何添加这样的实用工具函数到我们的工具箱里）。

我们用两个源对象❸来扩展目标对象，语句如下：

```
$.extend(target,source1,source2);
```

这个语句把对象source1和source2的属性合并到target对象里。target对象作为函数的值而被返回。然而由于target被适当地修改了，我们不必创建一个变量用来保持它的引用。如果作为语句链的一部分而利用这个函数，则返回target对象这个事实是十分重要的。

然后我们显示修改之后的target对象的值❹。结果如图6-5所示。

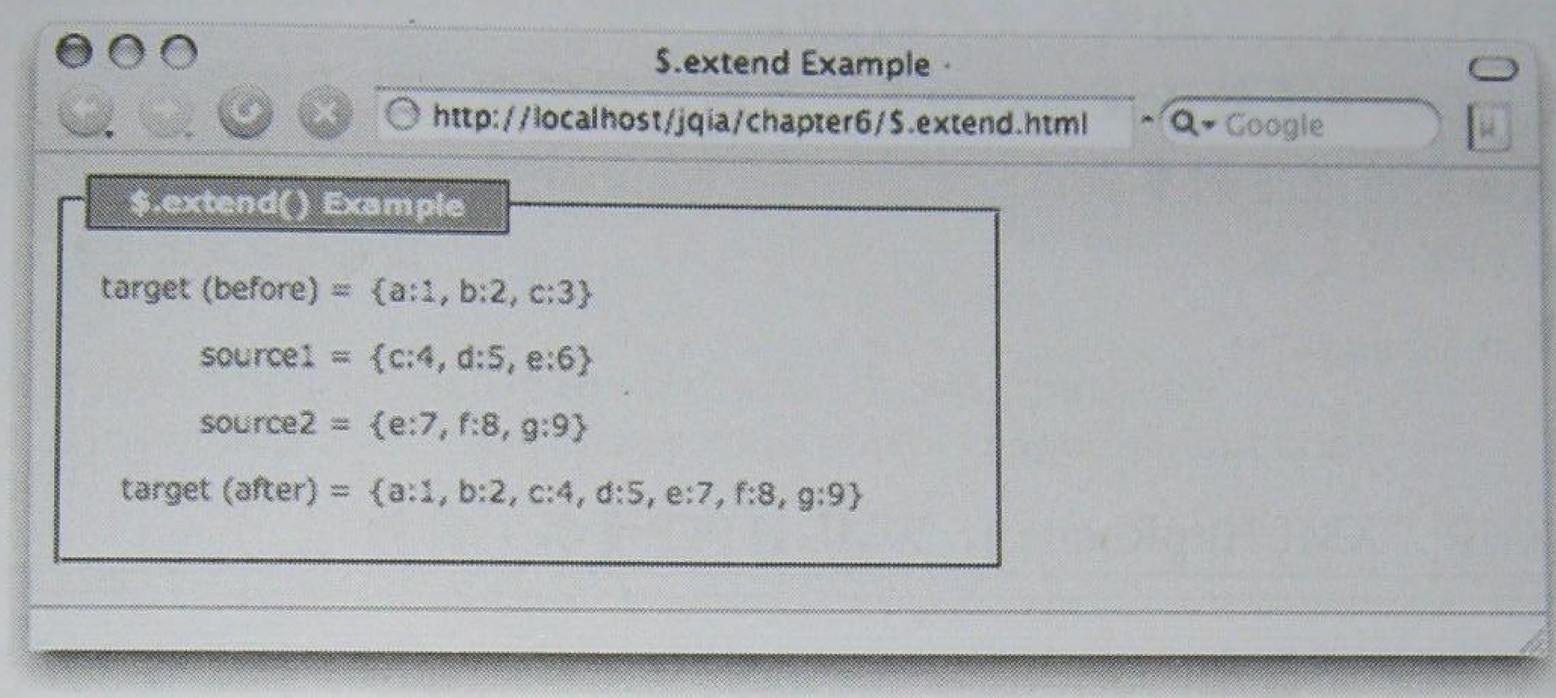


图6-5 利用$.extend合并对象，使所有源对象的属性被复制到目标对象

正如我们所见，源对象的所有属性已被合并到目标对象里。不过请注意下列的重要细节。

- target和source1两者都包含名为c的属性。source1里的c值取代原始目标对象里的c值。
- source1和source2两者都包含名为e的属性。请注意在合并到target时，source2里的e值优先于source1里的e值，这演示了实参列表后部的源对象如何优先于实参列表前部的源对象。

显然这个实用工具函数在许多应用场景里可以用得上，在那里一个对象必须用另一个对象（或一组对象）的属性进行扩展，虽然如此，在第7章里学习如何定义实用工具函数的时候，我们才会看到这个功能的具体和常见的用法。

不过在学习如何定义实用工具函数之前，我们用一个函数把关于实用工具函数的研究成果包装起来。这个函数可以用来把新脚本动态地加载到页面上。

6.4 动态加载脚本

在大部分的时间里（可能几乎总是），当页面加载时我们通过页面<head>里的<script>标签从脚本文件加载页面所需的外部脚本。但时不时地，也许事后我们还想在脚本控制下动态地加载一些脚本。

我们可能会这样做，因为直到特定的用户活动发生之后才知道是否需要某个脚本（然而我们不想静态地包含脚本，除非绝对有必要），也可能因为需要利用在页面加载时无法得到的一些信息（以便在不同的脚本之间做出条件选择）。

不管出于什么理由要把新脚本动态地加载到页面上，jQuery提供$.getScript()实用工具函数让我们轻松地达成这个目标。

函数语法：$.getScript

$.getScript(url,callback)

向指定的服务器发起GET请求，获取url参数所指定的脚本。（可选地）在成功地获取脚本时调用回调函数。

参数

url　（字符串）将要获取的脚本文件的URL。

callback（函数）在脚本资源已被加载和求值后调用的回调函数（可选）。

给回调函数传递下列参数：

从资源加载的文本；

字符串success。

返回

用于获取脚本的XHR（XMLHttpRequest，XML HTTP请求）实例

从本质上来说，这个函数用jQuery的内建Ajax机制来获取脚本文件。第8章会详细讲述这些Ajax设施，但使用这个函数无需了解任何关于Ajax的知识。

在成功地获取文件后，文件里的脚本被求值：内联的任何函数都被执行，而已定义的任何变量或函数都变得可用。

警告　在Safari浏览器里，从已获取的文件中加载的脚本定义不会立即变得可用，甚至在$.getScript的回调函数里也是如此。任何动态加载的脚本元素直到加载脚本文件的脚本块把控制权返还给浏览器之后才变得可用。如果你的页面准备支持Safari，就要做好相应的计划！

我们实践一下。考虑下列脚本文件（可从chapter6/new.stuff.js得到）

```
alert("I'm inline!");

var someVariable = 'Value of someVariable';

function someFunction(value) {
  alert(value);
}
```

这个小脚本文件包含内联语句（这个语句发出一个警告，表明语句何时被执行）、变量声明以及函数声明（执行时发出包含已传入的任何值的警告）。现在编写页面把这个脚本文件动态地包括进来。页面代码如代码清单6-5所示，文件可以在chapter6/$.getScript.html找到。

代码清单6-5　动态地加载脚本文件并检查结果

```
<html>
  <head>
    <title>$.getScript Example</title>
    <link rel="stylesheet" type="text/css" href="../common.css">
    <script type="text/javascript"
```

```
        src="../scripts/jquery-1.2.1.js"></script>
<script type="text/javascript">
  $(function(){
    $('#loadButton').click(function(){
      $.getScript(
        'new.stuff.js'//,function(){$('#inspectButton').click()}
      );
    });
    $('#inspectButton').click(function(){
      someFunction(someVariable);
    });
  });
</script>
</head>
<body>
  <button type="button" id="loadButton">Load</button>
  <button type="button" id="inspectButton">Inspect</button>
</body>
</html>
```

❶ 点击Load（加载）按钮，获取脚本

❷ 点击Inspect（检查）按钮，显示结果

❸ 定义两个按钮

这个页面定义两个按钮❸用来触发示例的活动。标注为Load的第1个按钮，使new.stuff.js文件通过调用`$.getScript()`函数❶而被动态加载。请注意，最初第2个参数（回调函数）是被注释掉的（稍候我们对此加以说明）。

一点击Load按钮，new.stuff.js文件就被加载，并且文件的内容被求值。不出所料，文件里的内联语句触发警告消息，如图6-6所示。

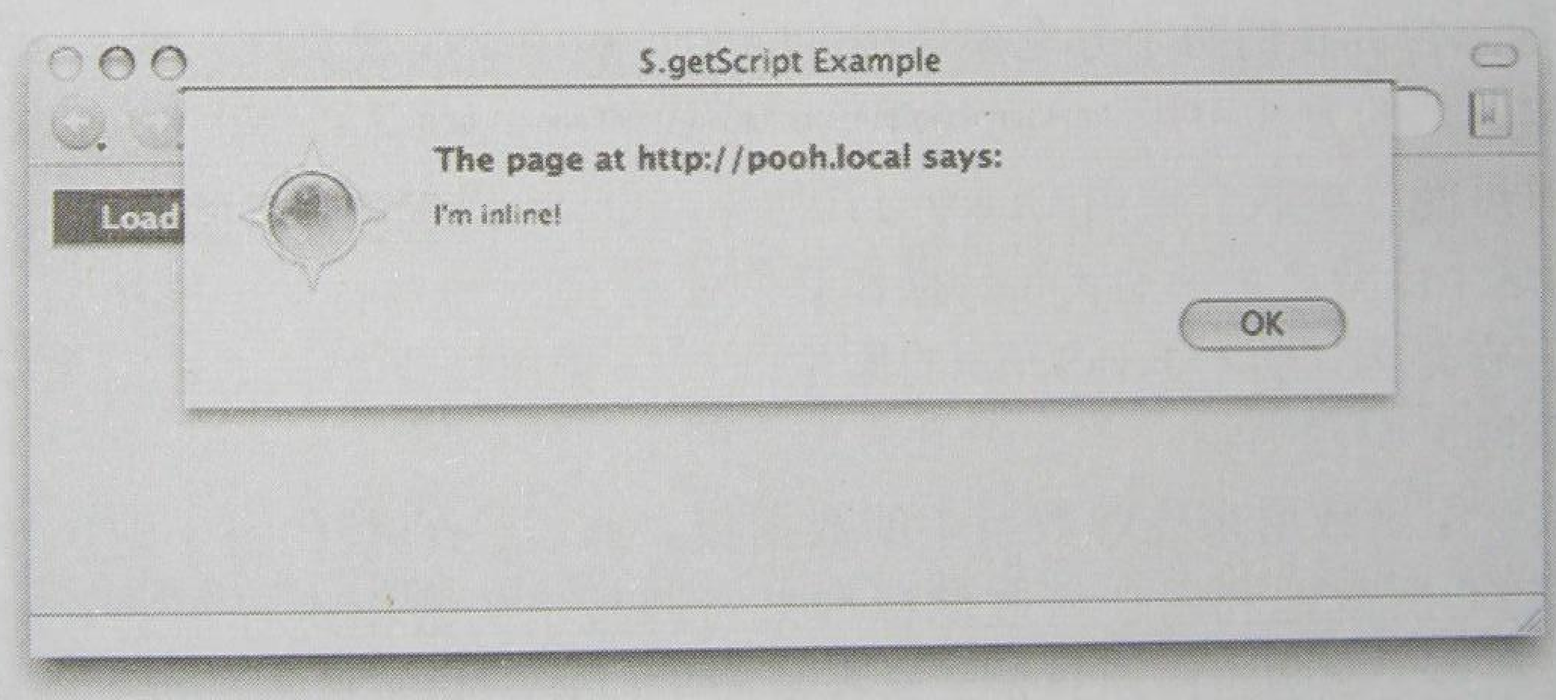


图6-6 脚本文件的动态加载和求值，导致内联的警告语句被执行

点击Inspect按钮，触发其`click`处理程序❷的执行。这个程序执行已动态加载的`someFunction()`函数，用已动态加载的`someVariable`变量的值作为传入参数。如果出现如图6-7所示的警告，就知道变量和函数都已正确地加载。

如果你想观察Safari浏览器的行为（我们在前面已经提醒你），就请复制一份如代码清单6-5所示的HTML文件，并取消针对`$.getScript()`函数的第2个参数（回调函数）的注释。一点击Inspect按钮，这个回调函数就执行`click`处理程序，把已动态加载的变量作为参数去调用已动态加载的函数。

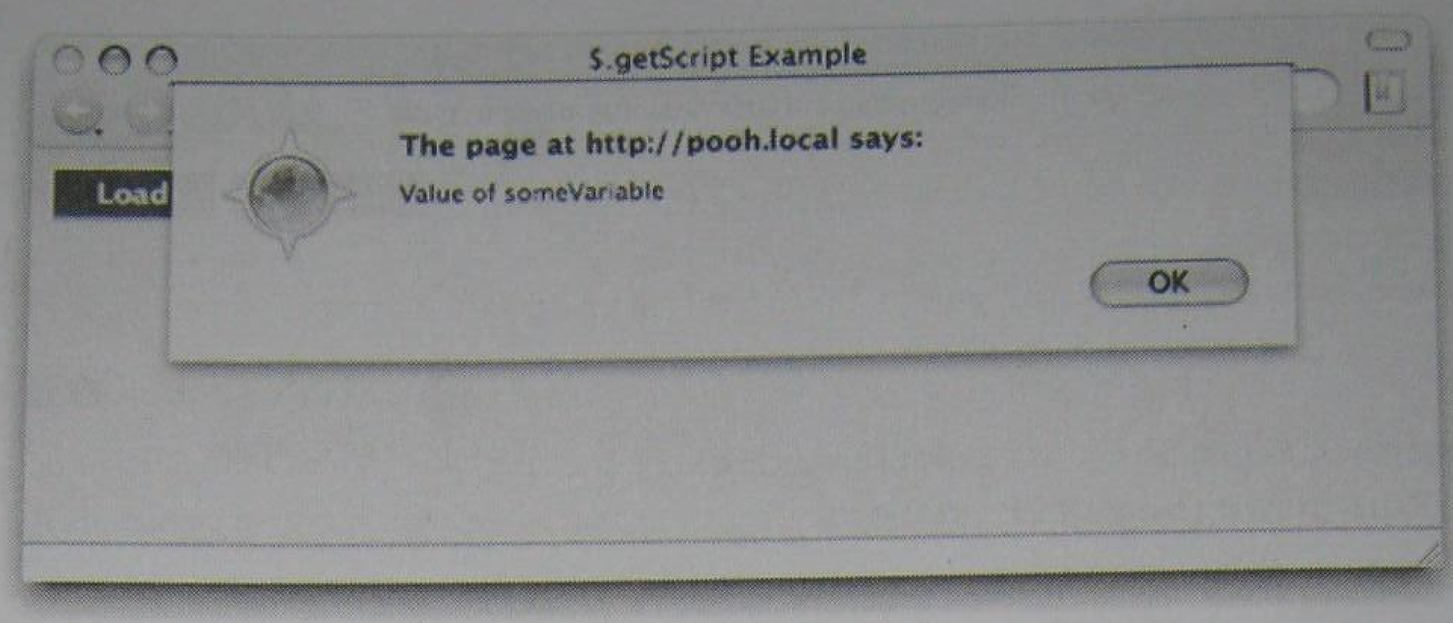


图6-7　警告消息框的出现，显示动态函数加载正确；而正确显示的值，表明变量已经动态加载

在除Safari以外的浏览器里，从脚本中动态加载的函数和变量在回调函数内可用。但如果在Safari上执行，则什么也不会发生！如果使用$.getScript()函数，必须留意在功能上的这个分歧。

83

6.5　小结

在本章里，我们研究除包装器方法（用于操作已匹配的DOM元素的包装集的方法）以外jQuery提供的几个功能，包括实用工具函数和jQuery标志在内。它们被直接地定义在jQuery（及其$别名）这个顶层名称上。

如果必须采取浏览器检测以便处理浏览器的功能和操作的差异，则$.browser这组标志可以确定页面正在哪个浏览器家族中显示。在无法以独立于浏览器的方式编写代码的时候，以及不能采用比浏览器检测更为优先的对象检测办法的时候[①]，浏览器检测只应作为最后采用的手段。

$.boxModel标志告诉我们两个方框模型中的哪个正被用于呈现页面，而$.styleFloat标志让我们以独立于浏览器的方式引用float（浮动）样式的style属性。

页面作者有时可能希望联合使用jQuery与其他库，jQuery提供$.noConflict()，允许其他库使用$别名。调用这个函数之后所有jQuery操作必须使用jQuery名称而不能用$。

$.trim()的存在是为了弥补JavaScript的原生String类所留下的缺陷，用来从字符串值的开头和结尾修整（删除）空白字符。

jQuery也提供一组函数用来处理数组中的数据集。$.each()使得易于遍历数组的每一项；$.grep()让我们随心所欲地利用任何过滤标准来对源数组的数据进行筛选从而创建新数组；$.map()让我们轻松地对源数组应用自定义转换从而生成对应的新数组（包含已转换的值）。

为了合并对象，甚至为了模拟一种继承方案，jQuery也提供$.extend()函数。这个函数允许我们把任意数量的源对象的属性合并到目标对象里。

考虑到我们不时想要动态地加载脚本文件，jQuery定义$.getScript()用来在其他页面脚本执行的任意时刻加载脚本文件并且求值。

把这些附加工具稳稳当当地收藏到工具箱之后，我们准备探讨如何给jQuery添加自定义扩展。我们在下一章讲述这个主题。

4

① 在处理浏览器差异时方案的优先顺序如下：独立于浏览器的方式>对象检测>浏览器检测。——译者注

第7章

用自定义插件来扩展jQuery

本章内容

- 为什么要用自定义代码来扩展jQuery
- 用于有效扩展jQuery的规则
- 编写自定义的实用工具函数
- 编写自定义的包装器方法

在前面的章节中，我们既看到jQuery所提供的有用命令及函数的庞大工具集，也看到可以轻松地把这些工具绑在一起，以便赋予页面任何选定的行为。有时候那些代码遵循我们想要一次又一次使用的常见模式。当这样的模式浮现时，应该捕获这些重复的操作作为可重用工具添加到原始工具集。在本章里，探讨如何捕获可重用代码片段作为jQuery扩展。

在讨论jQuery扩展之前，首先讨论最初为什么想要使自定义代码遵循jQuery扩展的模式。

7.1 为什么要扩展

在通读本书（包括书中所提供的代码示例在内）时只要你曾留心，毫无疑问，你就会注意到，采用jQuery对页面脚本的编写方式有着深刻的影响。

使用jQuery可以促进页面代码的特定风格，常常表现为这样的形式：形成元素的包装集，然后把jQuery命令或命令链应用到包装集上。在编写自己的代码时可以随心所欲，但最有经验的开发者都赞成，使整个网站上的所有或至少绝大部分代码遵循一致的风格是个好习惯。

那么，有个很好的理由要使我们的代码遵循jQuery扩展的模式，就是帮助维护整个网站的一致的代码风格。

理由不充分？需要更多理由？jQuery的整个关注点是提供一组可重用的工具和API。jQuery的创建者仔细地规划了库的设计和如何利用工具来提高可重用性的原理。通过遵循这些工具设计所设置的先例，我们自动收获融入这些设计之中的规划的利益——这是编写我们的代码作为jQuery扩展的非常有说服力的第二个理由。

还不信服？我们认为最后的理由是（尽管别人很可能列举更多的理由）：通过扩展jQuery可以利用jQuery所提供的现有代码基础。例如，通过创建新的jQuery命令（即包装器方法），我们自动继承jQuery的强大选择器机制的使用。如果能够站在jQuery所提供的强大工具的肩膀上，为什么我们要从头开始编写每样东西呢？

186 综上所述，很容易看出编写我们的可重用组件作为jQuery扩展是个好习惯，也是聪明的工作方式。本章的剩余部分探讨创建jQuery插件的准则和模式，然后动手创建几个jQuery插件。在讲述全新主题（Ajax）的第8章里，将会看到更多的证据，表明在真实世界的应用场景中创建我们的可重用组件作为jQuery插件，帮助保持代码一致性以及使得当初编写那些组件时简便得多。

不过首先规则是……

7.2 jQuery插件创建准则

标语，警告，有完没完！遮挡了风景，破坏了情绪。你应该这么做！你不要那么做！难道你看不懂这标语吗？

——Five Man Electric Band（五人电子乐队），1971年

虽然早在1971年五人电子乐队抒情地宣称反规则反正统，但有时规则却是件好事。没有规则，混乱当道。

规则就是这样——更像普通的经验准则——控制着如何用自定义的插件代码来成功地扩展jQuery。这些准则不仅确保我们的新代码能够恰当地插进jQuery体系结构，还确保它能与其他jQuery插件甚至其他JavaScript库一起正常工作。

扩展jQuery有两种形式：

- 在$（jQuery的别名）上直接定义的实用工具函数；
- 对jQuery包装集进行操作的方法（所谓的jQuery命令）。

本节的剩余部分仔细探讨这两种扩展类型共有的一些准则。而后续几节将探讨编写每个插件元素类型的特定准则和技巧。

7.2.1 给文件和函数命名

To Tell the Truth（说出真相）是一个美国游戏表演，最初盛行于20世纪50年代，在游戏中多个竞争者声称是同名的同一个人，然后名人陪审团被分派任务，要求确定哪个人才是真的那个人。
187 虽然对电视观众来说很有趣，但是轮到编程的时候，命名冲突一点也不好玩。

下面讨论在插件中如何避免命名冲突，不过首先让我们关注文件命名，在那里我们将编写插件，以便这些文件与其他文件不发生冲突。

jQuery团队所推荐的准则是简单而有效的，他们提倡下列格式：

- 文件名以jquery为前缀；
- 前缀后接插件的名称；
- 文件名以.js结尾。

例如，如果我们编写一个插件，准备命名为Fred，这个插件的JavaScript文件名是：

```
jquery.fred.js
```

使用前缀jquery.消除与其他库所使用文件发生可能的命名冲突。毕竟编写非jQuery插件的任何人没有理由使用jquery.前缀。

但那样还是让插件名称在jQuery社区内遭遇命名冲突。

如果编写插件给自己使用，需要做的只是避免与我们计划使用的任何其他插件发生命名冲

突。但如果编写插件并计划发布给其他人使用，就必须避免与已经发布的任何其他插件发生命名冲突。

避免命名冲突的最好办法是与jQuery社区的现有插件保持协调，请参照网页http://docs.jquery.com/Plugins。除注意在社区里已存在什么插件以外，还可以采取其他防范措施。

确保插件文件名称不与其他插件冲突的方式之一，是对文件名称采用对我们或组织而言唯一的子前缀。例如，在本书中所开发的所有插件都使用文件名称前缀jquery.jqia（jqia是*jQuery in Action*的首字母缩写），确保不与其他任何人可能在Web应用中使用的插件文件名称发生冲突。同样，jQuery表单插件的文件以前缀jquery.form开头。并非所有的插件都遵循这个惯例，但随着插件数量的增加，遵循这样的惯例将变得越来越重要。

不管是新的实用工具函数还是新的包装器方法，我们给函数取名时需要把类似的事情考虑进去，并且遵循类似的惯例。

如果创建插件给自己用，通常我们清楚还要用别的什么插件，因此避免命名冲突是简单的事。但如果我们创建插件供大众使用呢？或者一开始打算给自己用的插件，后来显得那么有用以至于我们想与社区里的其他人分享？ 188

再次强调，熟知已经存在的插件对避免API冲突大有帮助，但是为了避免命名空间混乱，我们也鼓励把相关函数收集到处于共同前缀（与文件名称的相关建议类似）之下的集合里。

那么，如何处理有关$的冲突？

7.2.2 小心$

“真正的$请站起来？”

写过相当数量的jQuery代码之后，我们已经看到利用$别名代替jQuery多么方便。但如果编写可能供别人在页面上使用的插件，我们就不能那样无忧无虑了。作为插件作者，我们无法知道页面作者是否打算调用$.noConflict()函数，而让另一个库占用$别名。

我们可以采取彻底的措施，使用jQuery名称替换$别名。噢，不！我们喜欢使用$，不会那么轻易就拱手相让。

6.2节已讨论一种习惯用法，用来确保$别名以本地化的方式来引用jQuery名称，而不影响页面的其余部分。这个小技巧也可以在定义jQuery插件时使用，如下所示：

```
(function($){
//
// Plugin definition goes here
//
})(jQuery);
```

通过传递jQuery到定义参数为$的函数，这样就确保$在函数体内引用jQuery。

现在我们可以心满意足地在插件定义中使用$了。

在研究如何添加新元素到jQuery之前，再探讨一个技巧，我们鼓励插件作者使用这一技巧。 189

7.2.3 简化复杂的参数列表

大多数插件倾向于简单，要么不需要几个参数，要么只需要区区几个参数。在绝大多数核心jQuery方法和函数中，能够找到证明这一点的大量证据——这些方法和函数要么只需要有限的几

个参数，要么根本不作要求。如果可选的参数被省略，就会提供智能的默认值，甚至在省略可选的参数时，参数顺序能够表示不同的含义。

bind()方法是个好示例。如果可选的数据参数被省略，通常指定为第3个参数的监听器函数，就可以指定为第2个参数。JavaScript的动态性和解释性允许编写这种灵活的代码，但随着参数数量的增多，这种事情就会变坏或变得复杂（对于页面作者和插件作者来说都是这样）。崩溃的可能性随着可选参数的增多而增加。

考虑一个多少有点复杂的函数，函数的签名如下：

```
function complex(p1,p2,p3,p4,p5,p6,p7) {
```

这个函数要求有7个参数，假定除第一个以外其他参数都是可选的。如果可选参数被省略（像bind()方法那样），有太多的可选参数需要智能地猜测调用者的意图。如果这个函数的调用者只是省略尾随参数，那不成问题，因为能够检测为null的可选尾随参数。但如果调用者想要指定p7而让p2到p6为默认值呢？调用者必须为任何被省略参数使用占位符并写成：

```
complex(valueA,null,null,null,null,null,valueB);
```

可恶！更为糟糕的一个调用例如：

```
complex(valueA,null,valueC,valueD,null,null,valueB);
```

还有类似的其他各种变形。使用这个函数的页面作者被迫小心翼翼地跟踪计算null的数目以及参数的顺序；再说，这样的代码难以阅读和理解。

但是，除了不给调用者有这么多可选项以外，我们还有什么办法？

JavaScript的灵活性再次成为救兵。平息这种混乱的一种模式已在页面创作社区里兴起——那就是选项散列对象。

190

采用这种模式，多个可选参数被集中到单个参数（JavaScript的Object实例），Object实例的“名称/值对”作为多个可选参数。

利用这个技巧，第一个示例改写为：

```
complex(valueA, {p7: valueB});
```

第二个示例改写为：

```
complex(valueA, {
  p3: valueC,
  p4: valueD,
  p7: valueB
});
```

好得多！

就这样，我们不必使用占位符null来说明已省略的参数，也不必计算参数。每个可选参数可以简便地标注，以便能够清楚地看到该参数正好表示什么（如果我们使用更好的参数名称而不是p1到p7，就更加清楚）。

显然这对于复杂函数的调用者来说是个有利条件，尽管那样，对于作为函数作者的我们来说，有什么样的结果呢？结果是看到jQuery所提供的机制，让我们得以轻松地收集这些可选参数，并且把它们与默认值合并起来。重新考虑前面那个要求具备1个必需参数和6个可选参数的复杂示例函数。简化之后的新函数签名是：

```
complex(p1,options)
```

在函数里面，我们可以利用便利的$.extend()实用工具函数把这些选项与默认值合并起来。考虑下面的代码：

```
function complex(p1,options) {
  var settings = $.extend({
    option1: defaultValue1,
    option2: defaultValue2,
    option3: defaultValue3,
    option4: defaultValue4,
    option5: defaultValue5,
    option6: defaultValue6
  },options||{});
  // remainder of function...
}
```

191

通过把页面作者在options里所传递的值与包含所有可用选项及其默认值的一个对象进行合并，最终settings变量的所有可能选项的默认值被替换为页面作者显式指定的任何值。

请注意，我们用||{}来防备值为null或undefined的options对象：如果options求值为false（正如我们所知null和undefined就是如此），就提供一个空对象。

简单、通用，并且对调用者友好！

在本章后面部分和第8章所介绍的jQuery函数里，我们将看到这种模式的使用示例。不过目前，最后探讨一下如何利用自定义的实用工具函数来扩展jQuery。

7.3 编写自定义实用工具函数

在本书中，使用术语实用工具函数描述定义为jQuery（因此也是$）属性的函数。我们不打算用这些函数来操作DOM元素（那是为操作DOM包装集而定义的函数所做的工作），而是用这些函数来操作JavaScript非元素对象或执行其他非对象特定的操作。我们已看到的这种类型的函数示例是$.each()和$.noConflict()。

在本节里，我们将学习如何添加自定义的实用工具函数。

添加一个函数作为Object实例的属性，就像声明函数并指派到object属性那样简单（如果你觉得这像是巫术的话，那你就还没有通读附录，而现在是阅读附录的一个好时机）。创建一个不具实用价值的自定义实用工具函数就像下列语句那样简单：

```
$.say = function(what) { alert('I say '+what); }
```

实在简单。以这个方式来定义实用工具函数不是没有缺陷的。能想起在7.2.2节里关于$的讨论吗？如果页面作者在使用Prototype，且已调用$.noConflict()的页面上包括这个函数，会有什么后果？不是添加了一个jQuery扩展，而是在Prototype的$()函数上创建了一个方法（如果“函数的方法”这个概念让你觉得不知所云的话，赶紧阅读附录）。

对于决不共享的私有函数而言，这不是问题。但如果未来页面上有一些改动占用$，会有什么后果呢？因此选择稳妥的做法才是上策。

确保某个人占用$时不会践踏我们的代码的一个办法是根本不使用$。我们可以这样编写那个不具实用价值的函数：

192

```
jQuery.say = function(what) { alert('I say '+what); }
```

看起来似乎是一条出路，但事实证明对于更为复杂的问题这决不是最理想的解决办法。如果函数体内部利用大量jQuery方法和函数来完成工作，结果会怎样呢？我们将必须贯穿整个函数使用jQuery而不是$。那就相当冗长和不雅。此外，一旦使用了$，我们就爱不释手！

回顾7.2.2节所介绍的习惯用法，我们可以安全地编写函数如下：

```
(function($){
  $.say = function(what) { alert('I say '+what); }
})(jQuery);
```

我们极力推荐使用这种模式（即便对于这样琐碎的函数而言，似乎有“杀鸡用牛刀”之嫌），因为在声明和定义函数的时候它保护了$的使用。就算函数必须变得更为复杂，我们还可以扩展和修改函数，而不用疑惑使用$是否安全。

趁着现在这种模式在我们脑子里印象鲜明，下面实现一个非琐碎的实用工具函数。

7.3.1　创建操作数据的实用工具函数

进行定宽输出的时候，通常有必要输出一个数值并把数值格式化到定宽字段里（在这里“宽”或“宽度”定义为字符的个数）。通常这样的操作将使值在定宽字段内右对齐，并且把填充字符作为前缀添加到值前面，以便弥补值的长度和字段长度之间的差异。

我们编写这样一个实用工具函数，根据如下的语法进行定义。

函数语法：$.toFixedWidth

$.toFixedWidth(value,length,fill)

把已传入值格式化为指定长度的定宽字段。可以传入填充字符（可选）。如果数值超过指定长度，数值的高位数字将被截断以便符合指定长度。

参数

value　（数字）将要格式化的值。

length　（数字）结果字段的长度。

fill　（字符串）对值进行前（左）填充时使用的填充字符。如果省略，就使用0。

返回

定宽字段

193

这个函数的实现如代码清单7-1所示。

代码清单7-1　$.toFixedWidth()实用工具函数的实现

```
(function($){
  $.toFixedWidth = function(value,length,fill) {
    var result = value.toString();
    if (!fill) fill = '0';                    ❶
    var padding = length - result.length;     ❷
    if (padding < 0) {
```

```
      result = result.substr(-padding);    ◁ ❸
    }
    else {
      for (var n = 0; n < padding; n++)    ❹
        result = fill + result;
    }
    return result;    ◁ ❺
  };
})(jQuery);
```

这个函数简单而又直接。已传入的值转换为对应的字符串，填充字符被确定为已传入字符或默认字符0❶。然后计算需要填充的字符个数❷。

如果计算结果为负数（字符串长度大于已传入的字段长度），则截断字符串的开头得到指定长度的字符串❸；否则，在返回字符串（作为函数的结果）之前❺，在字符串开头补足适当个数的填充字符❹。

虽然这是简单的示例，却能够显示如何轻松地添加一个实用工具函数。一如既往，还有改进的余地。请思考下面几个练习。

- 像书中大部分示例那样，错误检查被减到最少以便关注手头的示例。你将如何加强函数，处理调用者的错误（比如不给value和length传递数字类型的值）？如果调用者根本不传递这些值，又如何处理？
- 我们小心地截断过长的数值确保结果始终符合指定的长度。但如果调用者传递非单个字符的字符串作为填充字符，后果将不可预料。你如何处理那种情况呢？
- 如果你不想截断过长的数值字符串，又该如何处理呢？

现在，让我们应对一个更为复杂的函数，在那里可以利用刚才编写的$.toFixedWidth()函数。

194

7.3.2 编写日期格式器

如果你曾经从服务端来到客户端编程的世界，可能你盼望已久的事情之一是方便的日期格式器（JavaScript的Date类型不提供）。这种函数将操作Date实例而不是任何DOM元素，因此它是实用工具函数的完美候选者。下面编写这样的函数，语法如下。

函数语法：$.formatDate

$.formatDate(date,pattern)

对已传入日期根据提供的模式进行格式化。在模式中被替换的标记如下：

yyyy：4位数年

yy：2位数年

MMMM：月份的完整名称

MMM：月份的缩写名称

MM：月份数字，2个字符的字段，不足2个字符时左边补0

M：月份数字

dd: 月份里的天，2个字符的字段，不足2个字符时左边补0
d: 月份里的天
EEEE: 一星期中的天的完整名称（即星期几的完整名称）
EEE: 一星期中的天的缩写名称（即星期几的缩写名称）
a: 上午或下午（AM或PM）
HH: 24小时制的第几小时（即几点），2个字符的字段，不足2个字符时左边补0
H: 24小时制的第几小时（即几点）
hh: 12小时制的第几小时（即几点），2个字符的字段，不足2个字符时左边补0
h: 12小时制的第几小时（即几点）
mm: 小时里的分，2个字符的字段，不足2个字符时左边补0
m: 小时里的分
ss: 分里的秒，2个字符的字段，不足2个字符时左边补0
s: 分里的秒
S: 秒里的毫秒，3个字符的字段，不足3个字符时左边补0

参数

date　（日期）将被格式化的日期。
pattern　（字符串）用来对日期进行格式化的模式。不匹配模式标记的任何字符被原样地复制到结果中。

返回

已格式化的日期

这个函数的实现如代码清单7-2所示。我们不准备讲述执行格式化的算法细节（毕竟本书不是讲算法的），但是准备利用这个实现，指出在创建稍微复杂的实用工具函数时可以采用的几个有趣的策略。

195

代码清单7-2　$.formatDate()实用工具函数的实现

```
(function($){
  $.formatDate = function(date,pattern) {
    var result = [];
    while (pattern.length > 0) {
      $.formatDate.patternParts.lastIndex = 0;
      var matched = $.formatDate.patternParts.exec(pattern);
      if (matched) {
        result.push(
          $.formatDate.patternValue[matched[0]].call(this,date)
        );
        pattern = pattern.slice(matched[0].length);
      } else {
        result.push(pattern.charAt(0));
        pattern = pattern.slice(1);
      }
    }
    return result.join('');
```

❶ 实现函数的主体

```
};

$.formatDate.patternParts =
  /^(yy(yy)?|M(M(M(M)?)?)?|d(d)?|EEE(E)?|a|H(H)?|h(h)?|m(m)?|s(s)?|S)/;

$.formatDate.monthNames = [
  'January','February','March','April','May','June','July',
  'August','September','October','November','December'
];

$.formatDate.dayNames = [
  'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday',
  'Saturday'
];

$.formatDate.patternValue = {
  yy: function(date) {
    return $.toFixedWidth(date.getFullYear(),2);
  },
  yyyy: function(date) {
    return date.getFullYear().toString();
  },
  MMMM: function(date) {
    return $.formatDate.monthNames[date.getMonth()];
  },
  MMM: function(date) {
    return $.formatDate.monthNames[date.getMonth()].substr(0,3);
  },
  MM: function(date) {
        return $.toFixedWidth(date.getMonth() + 1,2);
      },
      M: function(date) {
        return date.getMonth()+1;
      },
      dd: function(date) {
        return $.toFixedWidth(date.getDate(),2);
      },
      d: function(date) {
        return date.getDate();
      },
      EEEE: function(date) {
        return $.formatDate.dayNames[date.getDay()];
      },
      EEE: function(date) {
        return $.formatDate.dayNames[date.getDay()].substr(0,3);
      },
      HH: function(date) {
        return $.toFixedWidth(date.getHours(),2);
      },
      H: function(date) {
        return date.getHours();
      },
      hh: function(date) {
        var hours = date.getHours();
        return $.toFixedWidth(hours > 12 ? hours - 12 : hours,2);
```

❷ 定义正则表达式

❸ 提供月份的名称

❹ 提供星期里的每一天的名称

❺ 把多个“从标记到值”的转换函数集中起来

196

7

```
    },
    h: function(date) {
      return date.getHours() % 12;
    },
    mm: function(date) {
      return $.toFixedWidth(date.getMinutes(),2);
    },
    m: function(date) {
      return date.getMinutes();
    },
    ss: function(date) {
      return $.toFixedWidth(date.getSeconds(),2);
    },
    s: function(date) {
      return date.getSeconds();
    },
    S: function(date) {
      return $.toFixedWidth(date.getMilliseconds(),3);
    },
    a: function(date) {
      return date.getHours() < 12 ? 'AM' : 'PM';
    }
  };
})(jQuery);
```

除了用于保持代码量受控的几个JavaScript技巧以外，这个实现的最为有趣的方面是函数❶需要一些辅助数据来完成工作，特别是以下各方面：

- 正则表达式用来匹配模式中的标记❷；
- 月份的英文名称的列表❸；
- 星期几的英文名称的列表❹；
- 一组子函数，设计为如果给定源日期，就提供各种标记类型的值❺。

我们可以把以上各项作为var定义而包括在函数体内，但那样会搞乱已经有点复杂的算法，并且因为它们是常量，所以从变量数据中分离出来是明智的。

我们不想用一堆只被这个函数所需的名称来污染全局命名空间或者$命名空间，因此我们使这些声明成为新函数的属性。请记住：JavaScript函数是一等对象，而函数可以像任何JavaScript对象那样，拥有自身的属性。

探讨一下格式化算法？简而言之，算法的操作如下。

- 创建一个数组用于保持结果的各个部分。
- 迭代整个模式，消耗已标识的标记以及非标记的字符，直到检查完毕。
- 通过设置lastIndex属性为0，在每一次迭代时使正则表达式复位（存储在$.formatDate.patternParts）。
- 为了匹配模式的当前开头标记，测试正则表达式。
- 如果发生匹配，就调用转换函数的$.formatDate.patternValue集合里的函数，从Date示例中获取适当值。这个值被添加到结果数组的末尾，而已匹配的标记则从模式的开头删除。

- 如果模式的当前开头标记不被匹配，就从模式中删除第一个字符并添加到结果数组的末尾。
- 当整个模式被消耗完毕时，把结果数组连接成一个字符串，并作为函数的值而返回。

请注意，$.formatDate.patternValue集合里的转换函数利用了我们在上一节所创建的$.toFixedWidth()函数。

198

这两个函数可在文件chapter7/jquery.jqia.dateFormat.js找到，而chapter7/test.dateFormat.html是用于测试$.formatDate函数的基本页面。

操作普通JavaScript对象固然好，但jQuery的真正力量在于包装器方法（利用jQuery选择器的力量来收集一组DOM元素进行操作）。下面看如何添加强大的自定义包装器方法。

7.4 添加新的包装器方法

jQuery的真正力量在于简便快捷地选择和操作DOM元素的能力。幸好可以通过添加包装器方法来操作我们认为适当的已选择DOM元素，从而扩展那种能力。通过添加包装器方法，我们自动地获得jQuery选择器的强大能力以便选择将要操作的元素，而不必亲自完成所有的工作。

倘若了解JavaScript，大概可以领会如何添加实用工具函数到$命名空间，但对于包装器方法就不是那样。我们必须了解jQuery特定的一些信息：要添加包装器方法到jQuery，必须把包装器方法指派为$命名空间内名为fn的对象的属性。

创建包装器方法的一般模式是：

```
$.fn.wrapperFunctionName = function(params){function-body};
```

我们编造小的包装器方法，把已匹配DOM元素的颜色设置为蓝色。

```
(function($){
  $.fn.makeItBlue = function() {
    return this.css('color','blue');
  }
})(jQuery);
```

像实用工具函数那样，我们在保证$是jQuery别名的外函数中进行声明。不同的是，我们创建新的包装器方法作为$.fn的属性，而不是$的属性。

注意 如果你熟悉面向对象JavaScript及其基于原型的类声明，又知道$.fn不过是jQuery构造器函数的prototype属性的别名，就会觉得很有趣。

199

在方法体内，函数上下文（this）引用包装集。我们可以在包装集上调用所有预定义的jQuery命令。比如在这个示例中，我们在包装集上调用css()命令来设置所有已匹配DOM元素的颜色为蓝色。

警告 在包装器方法的主体内，函数上下文（this）引用的是包装集，但是如果在包装器方法中声明内联函数，则包装器方法和内联函数拥有各自不同的函数上下文。在这种情况下必须谨慎地使用this，确保引用的是你所认为的函数上下文！例如，如果你调用带有迭代器函数的each()，则在迭代器函数内this引用的是当前迭代的DOM元素。

我们几乎可以随心所欲地操作包装集里的DOM元素，但在定义新的包装器方法时有个非常重要的规则：除非有意地让方法返回特定的值，否则应该总是返回包装集作为包装器方法的返回值。这样使得新命令能够参与任何jQuery命令链。在这个示例中，由于css()命令返回包装集，我们简单地返回css()调用的结果就行。

这个示例通过this把jQuery的css()命令应用到包装集的所有元素上。如果出于某个理由，必须分别处理各个已包装元素（或许因为我们必须做出有条件的处理决定），可以使用下列模式：

```
(function($){
  $.fn.someNewMethod = function() {
    return this.each(function(){
      //
      // Function body goes here -- this refers to individual
      // elements
      //
    });
  }
})(jQuery);
```

在这个模式里，使用each()命令来迭代包装集的每一个单独元素。请注意在迭代器函数内，this引用当前DOM元素而不是整个包装集。然而each()返回包装集作为新的包装器方法的返回值，因此这个新的包装器方法能够参与命令链。

200

以上是关于定义包装器方法的一切，但（不总是有“但”吗）在创建更复杂的jQuery包装器方法时我们应该懂得一些技巧。下面再定义两个更为复杂的插件方法[1]用来检验那些技巧。

7.4.1　在包装器方法中应用多个操作

我们现在来开发在包装集上执行多个操作的另外一个新的插件方法。假定我们必须切换表单里文本字段的只读状态，与此同时一致地改变文本字段的外观。把几个现有jQuery命令链接起来可以轻松实现，但我们想要干净利落，于是把这些命令捆绑起来成为单个方法。

我们给新命令取名为setReadOnly()，语法如下。

函数语法：setReadOnly

setReadOnly(state)

把包装集里文本字段的只读状态设置为state所指定的状态。同时调整不透明度：如果非只读，则100%；如果只读，则50%。忽略包装集里除文本字段以外的任何元素。

参数

State　（布尔型）将要设置的只读状态。如果为true，则使文本字段变为只读；如果为false，则清除只读状态。

返回

包装集

① 即“包装器方法”。——译者注

这个插件的实现如代码清单7-3所示。文件可在chapter7/jquery.jqia.setreadonly.js找到。

代码清单7-3 setReadOnly()插件方法的实现

```
(function($){
  $.fn.setReadOnly = function(readonly) {
    return this.filter('input:text')
      .attr('readonly',readonly)
      .css('opacity', readonly ? 0.5 : 1.0);
  }
})(jQuery);
```

201

这个示例只比初始示例稍微复杂一点，却展示了下列附加的关键概念：

- 传入参数，影响方法的操作；
- 通过使用jQuery链，把3个jQuery命令应用到包装集上；
- 因为新命令的返回值是包装集，所以新命令能够参与jQuery链；
- filter()命令用来确保不管页面作者把这个命令应用到什么包装集，只有文本字段受到影响。

如何把这个方法学以致用呢？

通常在定义在线订单时，可能必须允许用户输入两组地址信息：一组是送货目的地，另一组是账单目的地。在大多数的情况下，这两个地址是一致的。让用户重复输入两次相同的信息导致用户友好度降低，而这并非我们所想。

我们可以假定，如果表单有留空，则账单地址与送货地址一致，并编写对应的服务器端代码。但是假定产品经理有点偏执，要求用户一方明确地表示是否账单地址与送货地址一致。

我们满足产品经理的要求，通过给账单地址添加复选框，指示账单地址是否与送货地址一致。如果选中复选框，就从送货地址复制账单地址并且设为只读状态；如果取消选中复选框，就从文本字段里清空账单地址和只读状态。

图7-1显示一个测试表单在复选框选中之前和之后的状态。

这个测试表单的页面代码如代码清单7-4所示，文件可从chapter7/test.setreadonly.html得到。

202

代码清单7-4 用来测试新命令setReadOnly()的表单的实现

```
<html>
  <head>
    <title>setReadOnly() Test</title>
    <link rel="stylesheet" type="text/css" href="../common.css">
    <link rel="stylesheet" type="text/css"
          href="test.setreadonly.css">
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js"></script>
  <script type="text/javascript"
          src="jquery.jqia.setreadonly.js"></script>
  <script type="text/javascript">
    $(function(){
      $('#sameAddressControl').click(function(){
        var same = this.checked;
        $('#billAddress').val(same ? $('#shipAddress').val():'');
        $('#billCity').val(same ? $('#shipCity').val():'');
```

```
        $('#billState').val(same ? $('#shipState').val():'');
        $('#billZip').val(same ? $('#shipZip').val():'');
        $('#billingAddress input')
          .setReadOnly(same);
      });
    });
  </script>
</head>

<body>
  <fieldset>
    <legend>The Test Form</legend>
    <div>
      <form name="testForm">
        <div>
          <label>First name:</label>
          <input type="text" name="firstName" id="firstName"/>
        </div>
        <div>
          <label>Last name:</label>
          <input type="text" name="lastName" id="lastName"/>
        </div>
        <div id="shippingAddress">
          <h2>Shipping address</h2>
          <div>
            <label>Street address:</label>
            <input type="text" name="shipAddress"
                   id="shipAddress"/>
          </div>
          <div>
            <label>City, state, zip:</label>
            <input type="text" name="shipCity" id="shipCity"/>
            <input type="text" name="shipState" id="shipState"/>
            <input type="text" name="shipZip" id="shipZip"/>
          </div>
        </div>
        <div id="billingAddress">
          <h2>Billing address</h2>
          <div>
            <input type="checkbox" id="sameAddressControl"/>
            Billing address is same as shipping address
          </div>
          <div>
            <label>Street address:</label>
              <input type="text" name="billAddress"
                     id="billAddress"/>
            </div>
            <div>
              <label>City, state, zip:</label>
              <input type="text" name="billCity" id="billCity"/>
              <input type="text" name="billState" id="billState"/>
              <input type="text" name="billZip" id="billZip"/>
            </div>
          </div>
        </form>
```

❶ 应用新插件

203 ~ 204

```
    </div>
  </fieldset>
</body>
</html>
```

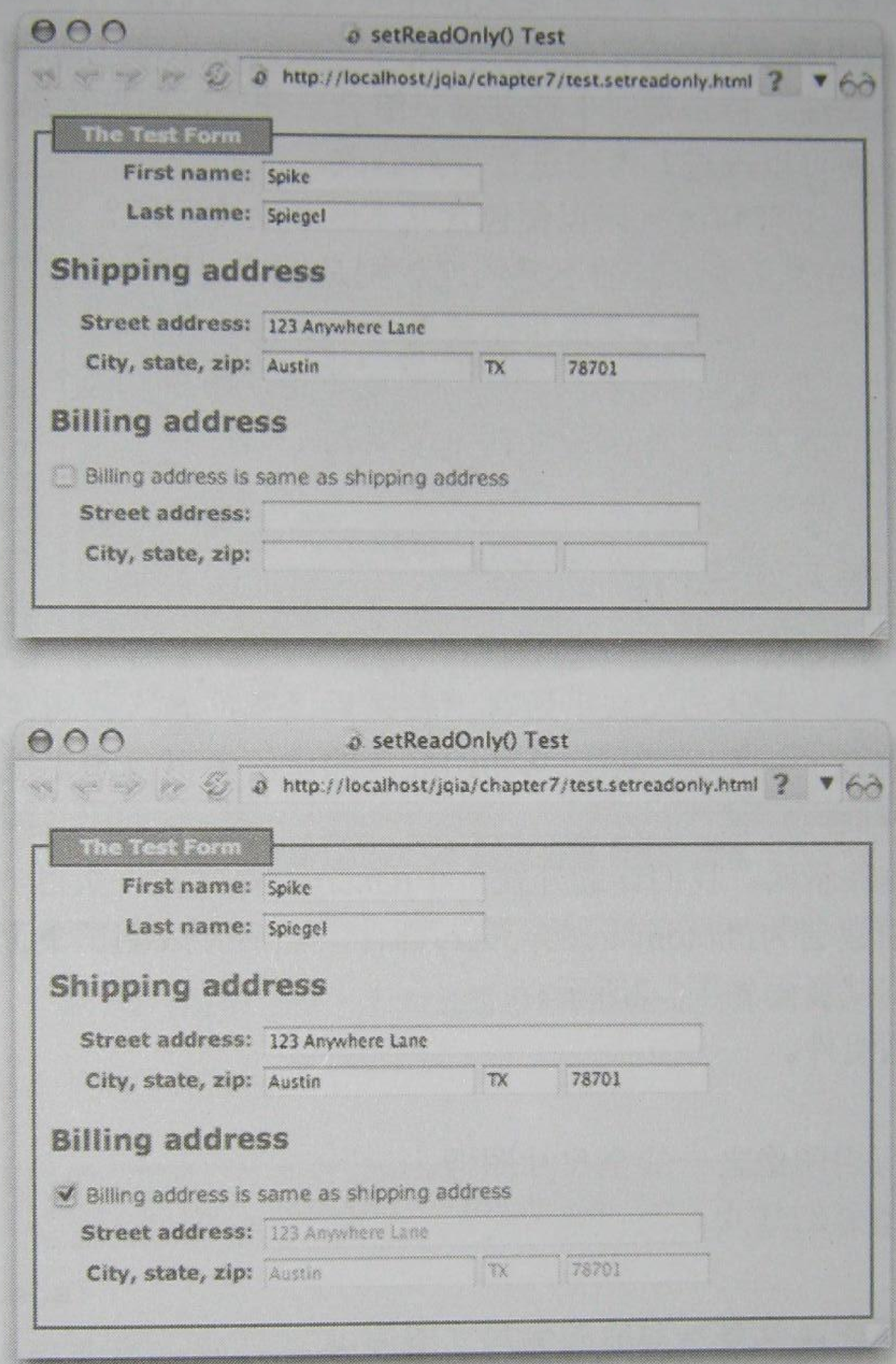


图7-1 测试表单在点击复选框之前和之后的状态

我们不对这个页面的操作进行详细分析，因为它相对来说浅显易懂。这个页面真正有趣的地方只有一个：在就绪处理程序里被附加到复选框的click处理程序。当复选框状态因点击而改变时，我们需要做以下事情。

(1) 把选中状态复制到same变量，以便在监听器的其余代码中易于引用。

(2) 设置账单地址字段的值。如果与送货地址一致，就把送货地址信息对应字段值复制到账单地址字段；否则，清空账单地址字段。

(3) 在账单地址容器里所有的输入字段上调用新命令setReadOnly()❶。

真不巧！我们在最后一步时有点粗心大意。利用$('#billingAddress input')所创建的包装集不仅包含账单地址块里的文本字段，也包含复选框。复选框元素没有只读语义，但它的不透明度可被修改——绝对不是我们的本意！

幸亏这个粗心大意被化解了，因为在定义插件的时候，我们没有粗心大意。回想一下，在setReadOnly方法中应用剩余的命令之前已经把文本字段筛选出来，而排除了其他input元素。我们非常赞同对细节的关注，特别是对于计划提供给公众使用的插件来说。

这个命令在哪些方面可以改进？考虑进行下列修改。

205

- 我们忘了文本区！如何修改代码以便包含文本区和文本字段？
- 在两种状态下应用到文本字段的不透明度级别是硬编码到函数里的。这算不上对调用者友好。修改方法以便允许调用者提供不透明度的级别。
- 噢，糟糕，为什么强迫页面作者只能接受改变不透明度的功能？如何修改方法使页面作者可以决定在两种状态下文本字段如何呈现呢？

下面编写一个更为复杂的插件。

7.4.2　保留在包装器方法之内的状态

人人都喜爱幻灯片！

至少在Web上是这样。不像到人家里做客用餐之后倒霉的客人，得耐着性子看完令人头脑麻木没完没了且不分重点的度假照片展，Web幻灯片的访问者可以想走就走，不伤害任何人的感情！

作为更加复杂的插件示例，我们计划开发一个jQuery命令，允许页面作者轻松地创建敏捷的幻灯片页面。我们将创建名为Photomatic的jQuery插件，然后生成测试页面一步步地测试这个插件。当其完成之时，测试页面如图7-2所示。

这个页面显示下列组件。

- 一组缩略图。
- 缩略图列表里可用图像之一的全尺寸图像。
- 一组按钮用于浏览幻灯片。

页面的行为如下。

- 一点击任何缩略图，就显示对应的全尺寸图像。
- 一点击全尺寸图像，就显示下一幅图像。
- 点击不同按钮，执行不同操作：
  - First——显示第一幅图像；
  - Previous——显示上一幅图像；
  - Next——显示下一幅图像；
  - Last——显示最后一幅图像。
- 任何离开图像列表一端的操作都会折回到另一端。在最后一幅图像上点击Next则显示第一幅图像，反之亦然。

我们也想在页面布局和样式上赋予页面作者尽可能多的自由。我们定义插件，以便页面作者能够以喜欢的任何方式来设置元素，并且告诉我们哪个页面元素应该用于哪个目的。此外，为了

206

给予页面作者尽可能多的回旋余地，我们定义插件，以便页面作者能够提供任何图像包装集作为缩略图。通常来说，缩略图将被集中起来，就像我们的测试页面那样，但是页面作者可以自由地标识在页面上的任何图像作为缩略图。

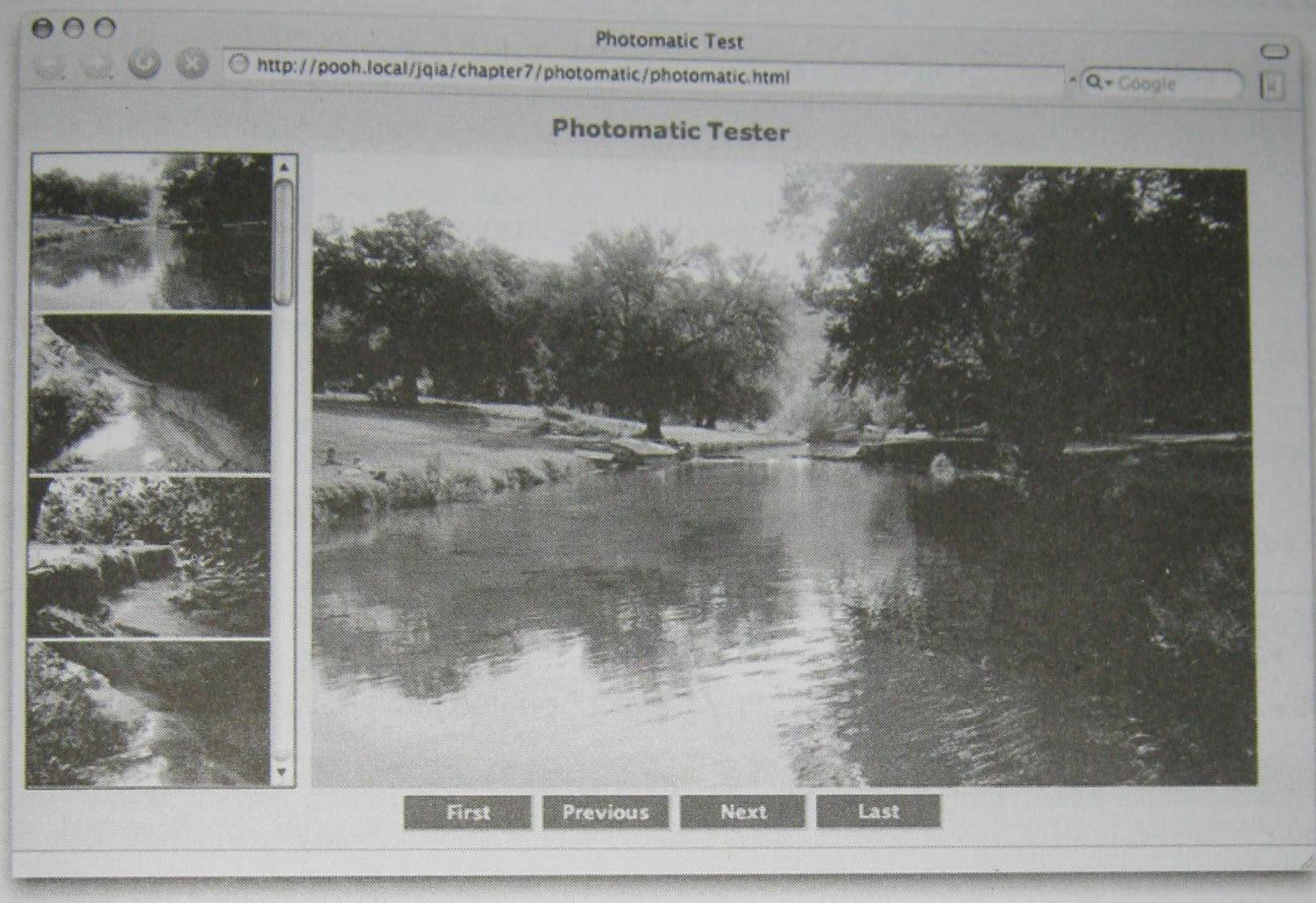


图7-2 用来一步一步浏览图像的Photomatic测试页面

首先介绍Photomatic插件的语法。

命令语法：photomatic

`photomatic(settings)`

设置缩略图的包装集以及在settings散列对象里所标识的页面元素，作为Photomatic[①]控件进行操作。

参数

settings （对象）一个散列对象。指定电子相册的各种设置。参见表7-1了解细节。

返回

包装集

因为拥有不少参数用来控制电子相册的操作（其中许多有默认值），所以我们利用一个散列

① Photomatic，可以理解为“电子相册”。——译者注

对象来传入这些参数，如7.2.3节所概括。可能指定的设置如表7-1所示。

表7-1　Photomatic()插件命令的各种设置属性

设置的名称	说　明
firstControl	（字符串|对象）DOM元素的引用，或标识DOM元素的jQuery选择器。充当第一个控件。如果省略，就不设置控件
lastControl	（字符串|对象）DOM元素的引用，或标识DOM元素的jQuery选择器。充当最后一个控件。如果省略，就不设置控件
nextControl	（字符串|对象）DOM元素的引用，或标识DOM元素的jQuery选择器。充当下一个控件。如果省略，就不设置控件
photoElement	（字符串|对象）<img>元素的引用，或标识<img>元素的jQuery选择器。充当显示的全尺寸图像。如果省略，就默认为jQuery选择器'#photomaticPhoto'
previousControl	（字符串|对象）DOM元素的引用，或标识DOM元素的jQuery选择器。充当上一个控件。如果省略，就不设置控件
transformer	（函数），用于转换缩略图的URL为对应的全尺寸照片图像的URL。如果省略，默认转换会在URL里用photo替换所有的缩略图实例

由于认同“测试驱动开发”理念，在着手创建Photomatic插件之前，我们先创建这个插件的测试页面。页面如代码清单7-5所示，文件可从chapter7/photomatic/photomatic.html得到。

代码清单7-5　测试页面，创建如图7-2所示的Photomatic显示

```
<html>
  <head>
    <title>Photomatic Test</title>
    <link rel="stylesheet" type="text/css" href="../../common.css">
    <link rel="stylesheet" type="text/css" href="photomatic.css">
    <script type="text/javascript"
            src="../../scripts/jquery-1.2.1.js"></script>
    <script type="text/javascript"
            src="jquery.jqia.photomatic.js"></script>
    <script type="text/javascript">
      $(function(){
        $('#thumbnails img').photomatic({
          photoElement: '#photo',
          previousControl: '#previousButton',
          nextControl: '#nextButton',
          firstControl: '#firstButton',
          lastControl: '#lastButton'
        });
      });
    </script>
  </head>

  <body>
    <h1>Photomatic Tester</h1>
    <div id="thumbnails">
      <img src="thumbnails/IMG_0340.jpg"/>
      <img src="thumbnails/IMG_0939.jpg"/>
```

❶ 调用Photomatic插件

❷ 包含缩略图

```
    <img src="thumbnails/IMG_1520.jpg"/>
    <img src="thumbnails/IMG_1904.jpg"/>
    <img src="thumbnails/IMG_2063.jpg"/>
    <img src="thumbnails/IMG_2186.jpg"/>
  </div>
  <div id="photoContainer">
    <img id="photo" src=""/>
  </div>
  <div id="buttonBar">
    <button type="button" id="firstButton">First</button>
    <button type="button" id="previousButton">Previous</button>
    <button type="button" id="nextButton">Next</button>
    <button type="button" id="lastButton">Last</button>
  </div>
</body>
</html>
```

❸ 定义全尺寸的图像元素

❹ 包含用作控件的元素

如果以上代码看起来比你以为的简单，也不应该在此刻感到吃惊。通过应用“不唐突的JavaScript”原则而把所有样式信息保存在外部样式表，我们的标记简洁而易懂。事实上，页面上脚本（on-page script）内存占用极小，由调用插件❶的单个语句所构成。

HTML标记由保持缩略图的容器❷、保持全尺寸照片的图像元素（初始化时src为空）❸，以及控制幻灯片的一组按钮❹所构成。别的每样东西都由我们的新插件进行处理。

下面我们开发新插件。

首先搭建框架（随着进展我们将填充细节）。目前的起点应该看起来相当熟悉，因为它一直以来遵循我们使用的一致模式。

209

```
(function($){
  $.fn.photomatic = function(callerSettings) {
  };
})(jQuery);
```

这段代码定义初始化为空的包装器方法，接受（如同语法说明所要求）名为callerSettings的单个散列对象参数。首先，在方法体内把调用者所传入的这些参数与表7-1所述的默认设置合并起来。这样在方法的剩余代码里我们可以引用单个设置对象。

7

利用下列习惯用法（我们已经看过几次了）执行这个合并操作：

```
var settings = $.extend({
  photoElement: '#photomaticPhoto',
  transformer: function(name) {
                 return name.replace(/thumbnail/,'photo');
               },
  nextControl: null,
  previousControl: null,
  firstControl: null,
  lastControl: null
}, callerSettings||{});
```

这个语句执行之后，settings变量将包含调用者所提供的值，以及内联散列对象所提供的默认值。但到settings这里还没完。考虑photoElement属性，它可能包含指定的jQuery选择器字符串（默认的或页面作者所提供的），或者一个对象引用。添加下列语句把它规范化为我们

知道如何处理的事物：

```
settings.photoElement = $(settings.photoElement);
```

我们创建包含照片元素的包装集（可能是多个照片元素，如果页面作者选择这么做）。现在我们拥有知道如何处理的一致的元素。

我们还必须跟踪几件事情。比如，为了知道“下一个”和“上一个”图像意味着什么，我们不只需要缩略图列表，还需要指示器，用于标识正被显示的当前图像。

缩略图列表是这个方法正在（或至少应该是）操作的包装集。我们不知道页面作者在包装集里收集了什么，因此只想把图像元素筛选出来——利用jQuery选择器可以轻松地做到这一点。但应该在哪里存储这个列表呢？

210

我们可以轻易地创建另一个变量用于保持这个列表，但会被说成是使列表受到局限。那就把这个列表作为另一个属性存储在settings上，如下所示：

```
settings.thumbnails = this.filter('img');
```

从包装集(在方法中通过this可用)里仅筛选图像元素来生成新包装集(只包含<img>元素)。新包装集存储在settings的名为thumbnails的属性里。

我们必须跟踪的另一个状态是当前图像。给settings添加名为current的属性从而维持缩略图列表的当前下标，因而能够跟踪当前图像。

```
settings.current = 0;
```

还有一个必须执行的设置步骤。如果通过跟踪下标来跟踪当前的照片，至少有一种情况，即在给定缩略元素的引用的情况下，我们必须知道对应的下标。处理这种情况的最简便办法是预先考虑这个需求，给缩略元素添加expando（自定义的属性），记录各自的下标。利用下面的语句可以做到这一点：

```
settings.thumbnails.each(function(n){this.index = n;});
```

这个语句迭代每一幅缩略图，给每一幅图像添加index属性用来记录当前图像在列表中的下标。既然初始状态已经设置，我们准备进入插件的中心内容。

等一等！状态？我们怎么能期待在就要结束执行的函数内以本地变量来跟踪状态呢？当函数返回时，变量和所有的设置不都超出作用域了吗？

一般而言确实是那样，但在一种情况下，也就是在变量是闭包的一部分的情况下，这种变量会比通常的作用域保留得更久。我们以前已经看见闭包，但如果你依然对闭包感到不踏实，就请复习附录内容。你必须理解闭包，不仅为了完善Photomatic插件的实现，也是为了创建一些重要的插件。

在考虑剩下的工作时，我们认识到必须附加大量的事件监听器到控件和元素上，到目前为止我们为了标识那些控件和元素而付出了巨大的努力。将要定义的每个监听器，都会创建包含settings变量的闭包，因此我们可以放心，即便settings可能看起来是暂时的，它所表示的状态却将保留并且对我们所定义的所有监听器可用。

211

说起那些监听器，下面是必须附加到不同元素上的click事件监听器清单。

- 点击缩略图将导致对应的全尺寸图像的显示。
- 点击全尺寸图像将导致下一幅图像的显示。

- 点击定义为previous控件的元素将导致上一幅图像的显示。
- 点击next控件将导致下一幅图像的显示。
- 点击first控件将导致列表里的第一幅图像的显示。
- 点击last控件将导致列表里的最后一幅图像的显示。

纵观这个列表，我们立即注意到，所有的这些监听器拥有共同的特征：它们都必须显示与缩略图相应的全尺寸图像。作为优秀而聪明的程序员，我们想把共同的处理提取出来封装在一个函数里，那样就不用一遍又一遍地重复相同的代码。

但怎样实现呢？

如果我们在编写正常的页面上的JavaScript，可以定义顶层函数；如果编写面向对象JavaScript，可以在JavaScript对象上定义方法；但是我们正在编写jQuery插件，应该在哪里定义实现函数？

我们不想为了只应从插件代码内部调用的函数而侵犯全局或$命名空间，那么能做什么呢？可以把函数定义为插件函数的一个方法（类似于我们在代码清单7-2的日期格式器中所做的事情）。作为一等的JavaScript对象，函数也能拥有方法。但还有更为简便的方法。

回想一下，我们的插件函数定义在另一个函数体内（用来确保$表示的是jQuery）。因此定义在外函数的任何本地变量都成为我们的插件函数所定义的闭包的一部分。如果作为本地变量在外函数里定义名为showPhoto()的实现函数，结果怎么样呢？

那就漂亮地解决了问题！showPhoto()将作为插件函数的闭包的一部分而对插件函数可用，但因为它声明在外函数的本地，所以不能为外函数之外所访问，也就没有机会去污染任何其他的命名空间。

212

在插件函数的外面，却在外函数的里面，我们定义的showPhoto()函数如下：

```
var showPhoto = function(index) {
  settings.photoElement
    .attr('src',
          settings.transformer(settings.thumbnails[index].src));
  settings.current = index;
};
```

7

当传入缩略图的index时（对应的全尺寸照片将被显示），这个函数利用settings对象里的值来完成下列事情：

(1) 查找index所标识的缩略图src特性；

(2) 传递src特性值到transformer函数里以便把缩略图URL转换为全尺寸图像URL；

(3) 把转换结果指派到全尺寸图像元素的src特性上；

(4) 记录被显示照片的下标作为新的当前下标。

但在我们为自己的聪明才智而欢欣鼓舞之前，应该知道还有个问题。showPhoto()函数需要存取settings，但settings定义在插件函数内，因此在插件函数外不可用。

“神圣的扩展范围的闭包，蝙蝠侠！我们怎么使settings突破插件函数的范围呢？”

如果给函数传递另一个参数settings，就显得生硬，我们却比那灵活。因为问题在于settings并非定义在包含实现函数和插件函数的外闭包里，所以最简便的解决方案就是把settings移到外闭包里。这样做不会影响所有的内闭包的可用性，但将确保settings对插件函

数和实现函数（或我们可能想定义的任何其他实现函数）都可用（请注意，这个技巧假定在页面上只有一个Photomatic实例——在这种情况下是个有意义的限制。对于更为一般的情况，传递settings作为参数不会施加这种限制）。

那么我们添加下列语句到外函数：

```
var settings;
```

并从插件函数内的下列语句中删除var：

213

```
settings = $.extend({
```

现在settings变量对这两个函数都可用，最终我们准备完成插件函数的实现。定义前面已列举的监听器的代码如下：

```
settings.thumbnails.click(function(){ showPhoto(this.index); });
settings.photoElement.click(function(){
  showPhoto((settings.current + 1) % settings.thumbnails.length);
});
$(settings.nextControl).click(function(){
  showPhoto((settings.current + 1) % settings.thumbnails.length);
});
$(settings.previousControl).click(function(){
  showPhoto((settings.thumbnails.length + settings.current - 1) %
            settings.thumbnails.length);
});
$(settings.firstControl).click(function(){
  showPhoto(0);
});
$(settings.lastControl).click(function(){
  showPhoto(settings.thumbnails.length - 1);
});
```

这些监听器的每一个都带着缩略图下标而调用showPhoto()函数，缩略图下标由下列三者之一来决定：已点击缩略图的下标、列表的长度、已计算的对当前下标的相对值（注意：如何使用取余运算符在下标值落在列表的两端之外时折返下标值）。

在宣布成功之前，我们有两个最后的任务：在任何用户操作之前必须显示第一幅照片，以及必须返回原始包装集，以便插件能够参与jQuery命令链。完成这两个任务的代码如下：

```
showPhoto(0);
return this;
```

花点时间跳个简短的庆功舞：终于完成了！

完整的插件代码如代码清单7-6所示，文件可以在chapter7/photomatic/ jquery.jqia.photomatic.js找到。

代码清单7-6　Photomatic插件（电子相册插件）的完整实现

214

```
(function($){
  var settings;

  $.fn.photomatic = function(callerSettings) {
    settings = $.extend({
      photoElement: '#photomaticPhoto',
      transformer: function(name) {
```

```
                return name.replace(/thumbnail/,'photo');
              },
    nextControl: null,
    previousControl: null,
    firstControl: null,
    lastControl: null
  }, callerSettings || {});
  settings.photoElement = $(settings.photoElement);
  settings.thumbnails = this.filter('img');
  settings.thumbnails.each(function(n){this.index = n;});
  settings.current = 0;
  settings.thumbnails.click(function(){ showPhoto(this.index); });
  settings.photoElement.click(function(){
    showPhoto((settings.current + 1) % settings.thumbnails.length);
  });
  $(settings.nextControl).click(function(){
    showPhoto((settings.current + 1) % settings.thumbnails.length);
  });
  $(settings.previousControl).click(function(){
    showPhoto((settings.thumbnails.length + settings.current - 1) %
              settings.thumbnails.length);
  });
  $(settings.firstControl).click(function(){
    showPhoto(0);
  });
  $(settings.lastControl).click(function(){
    showPhoto(settings.thumbnails.length - 1);
  });
  showPhoto(0);
  return this;
};

var showPhoto = function(index) {
  settings.photoElement
    .attr('src',
          settings.transformer(settings.thumbnails[index].src));
  settings.current = index;
};

})(jQuery);
```

在我们完成这个插件之后，看起来超过了45行代码，不是吗？

这是启用jQuery代码的典型插件，它在简洁的代码里封装了强大的功能。它足以演示一组重要的技术——利用闭包以便跨越jQuery插件的作用域而维持状态，以及启用私有实现函数的创建。插件能够定义和使用私有实现函数，不用侵犯任何命名空间。

现在你可以编写自定义jQuery插件了。如果你想出有用的插件，就考虑与jQuery社区的其他人分享吧。请访问http://jquery.com/plugins/获取更多的信息。

7.5 小结

本章介绍如何编写代码用于扩展jQuery。

编写自定义代码作为jQuery扩展，有许多优势。这不仅使代码贯穿Web应用而保持一致（不管采用jQuery API还是自定义API），还使我们的代码充分利用jQuery所提供的力量。

遵循几个命名规则，帮助避免文件名之间的命名冲突，并避免在$名称被页面重新指派时可能遇到的问题（页面使用我们的插件）。

创建新的实用工具函数就像在$上创建新的函数属性那样简便，而创建新的包装器方法就像创建$.fn的属性那样简便。

如果插件创作激起了你的兴趣，那么推荐你下载和详查现有插件的代码，看作者们如何实现各自的功能。你将看到本章所讲述的技巧如何在插件代码中被广泛地使用，你也将学会超出本书范围的新技巧。

还有更多jQuery知识可供我们运用，下面开始学习jQuery如何把Ajax合并到富因特网应用，
216 就像“不用脑子”那样简单。

第8章 利用Ajax与服务器交谈

8

本章内容

- Ajax概览
- 从服务器加载预格式化的HTML
- 发起通用的GET和POST请求
- 发起带有细化控制的请求
- 设置默认的Ajax属性
- 一个综合示例

可以肯定地说，近年来没有一项比Ajax更有力地改变Web的景观。无需重新加载页面而向服务器发起异步请求的这种能力，已经启用一整套全新的用户交互方式并使富因特网应用成为可能。

Ajax并非像许多人可能认为的那样，是新近添加到Web工具箱的。在1998年，微软引入在脚本控制下（低估`<iframe>`元素用于执行异步请求的能力）执行异步请求的功能，即利用一个ActiveX控件来启用OWA（Outlook Web Access，Outlook Web存取）。虽然OWA取得了一定的成功，但似乎很少有人关注底层的技术。

几年过去了，几个事件使Ajax进入Web开发社区的集体意识：几个非微软浏览器把技术的标准化版本实现为XHR（XMLHttpRequest，XML HTTP请求）对象；Google开始使用XHR；在2005年，Adaptive Path的Jesse James Garrett创造了Ajax（Asynchronous JavaScript and XML，异步的JavaScript和XML）这个术语。

仿佛只在等待给这项技术取个朗朗上口的名称似的，Ajax迅速地赢得Web开发大众的广泛关注，并且成为使我们得以启用富因特网应用的主要工具之一。

在本章里，我们简要地审视Ajax（如果你已经是一个Ajax专家，那就可以跳到8.2节），然后看jQuery如何使得利用Ajax成为“小菜一碟”。

下面首先回顾一下Ajax究竟是什么样的技术。

8.1 温习 Ajax

虽然在这节里我们将快速地审视Ajax，但请注意，这节内容并不打算作为完整的Ajax指南或Ajax初级读本。如果你完全不了解Ajax（甚至更糟，依然以为我们在谈论一种洗洁精或希腊神话

人物），我们鼓励你熟悉这项技术。为讲授Ajax而专门打造的材料中，Manning出版社的图书*Ajax in Action*和*Ajax in Practice*[1]都是其中的佼佼者。

有些人可能会争辩，说Ajax这个术语应用于无需刷新用户所面对的页面而发起服务器请求的任何技术（比如通过将请求提交到一个隐藏的<iframe>元素），但大多数人都把Ajax这个术语和使用XHR或微软的XMLHTTP ActiveX控件相互关联起来。

218

下面开始创建一个对象，看如何使用那些对象来生成服务器请求。

8.1.1　创建一个XHR实例

在完美的世界里，为一个浏览器编写的代码可以在所有常用的浏览器中正常工作。众所周知，我们并非生活在那个完美的世界里——即便有了Ajax，情况仍未改变。标准的办法是通过JavaScript的XHR对象来发起异步请求，而IE浏览器的专有办法是使用ActiveX控件。IE7提供模拟标准接口的包装器，但IE6则要求不同的代码。

一旦创建（幸亏是这样），用于设置、发起和响应请求的代码就是相对地独立于浏览器的，并且创建XHR实例对于任何特定的浏览器而言都很简便。问题是不同的浏览器以不同的方式来实现XHR，因此我们必须以适合于当前浏览器的方式来创建XHR实例。

然而，不要依赖于检测用户正在运行哪一种浏览器以便决定选择哪一条路径，我们使用称为“对象检测”的更受青睐的技术。利用这种技术，我们试图弄清楚浏览器的实际功能是什么，而不是正在运行的是哪一种浏览器。对象检测使得代码更为健壮，因为它能够在支持所测试功能的任何浏览器中工作。

代码清单8-1显示利用对象检测技术来初始化XHR实例的典型用法。

代码清单8-1　对象检测使代码能够应对多种浏览器

```
var xhr;
if (window.XMLHttpRequest) {        ← 测试一下，看是否已经定义XHR
  xhr = new XMLHttpRequest();
}
else if (window.ActiveXObject) {    ← 测试一下，看ActiveX是否存在
  xhr = new ActiveXObject("Msxml2.XMLHTTP");
}
else {                              ← 如果不支持XHR，就抛出错误
  throw new Error("Ajax is not supported by this browser");
}
```

创建之后，XHR实例跨越所有支持的浏览器实例，表现一组方便可靠的属性和方法。这些属性和方法如表8-1所示，最常用的属性和方法在下一节讨论。

219

创建XHR实例之后，下面看如何设置请求以及向服务器发起请求。

表8-1　XHR的方法和属性

方　法	描　述
abort()	导致当前正在执行的请求被取消

[1] 中文版《Ajax实战》和《Ajax实战：实例详解》，人民邮电出版社。——编者注

（续）

方　法	描　述
getAllResponseHeaders()	返回一个字符串，包含所有响应标头的名称和值
getResponseHeader(name)	返回指定的响应标头的值
open(method,url,async,username,password)	设置请求的方法和目标URL。请求可以声明为同步的（可选），也可以给需要基于容器认证的请求而提供用户名和口令（可选）
send(content)	发起带有指定的体内容（可选）的请求
setRequestHeader(name,value)	利用指定的名称和值，设置一个请求标头

属　性	描　述
onreadystatechange	指派在请求的状态发生变化时所使用的事件处理程序
readyState	一个整数值，指示请求的状态如下 0——未初始化 1——正在加载 2——已加载 3——交互 4——完成
responseText	在响应里所返回的体内容
responseXML	如果体内容是XML，就根据体内容而创建XML DOM
status	从服务器所返回的响应状态码。例如：200表示成功，404表示未找到。请参考HTTP规范[1]，以便了解全套的响应状态码
statusText	响应所返回的状态文本消息

220

8.1.2 发起请求

在向服务器发起请求之前，我们必须执行下列设置步骤：

(1) 指定HTTP方法，比如POST或GET；

(2) 提供将要接触的服务器端资源的URL；

(3) 让XHR实例知道如何通报进展；

(4) 为POST请求提供任何体内容。

我们通过调用XHR的open()方法，设置最先的两项如下：

```
xhr.open('GET','/some/resource/url');
```

请注意，这个方法不把请求发送到服务器。它不过是设置URL和HTTP方法以供使用。也可以传递布尔类型的第3个参数给open()方法，将请求指定为异步的（如果true，这是默认值）或同步的（如果false）。很少不需要生成异步请求（即使这意味着不必处理回调函数），毕竟请求的异步本性，通常是以异步方式来生成请求的全部意义所在。

第三，我们给XHR实例提供一个函数，让XHR实例通知我们当前正在进行什么——通过指派回调函数到XHR对象的onreadystatechange属性即可实现这一点。这个函数被称为就绪状态处理程序，可供XHR实例在其不同的处理阶段进行调用。通过查看XHR的其他各种属性设置，可以确切地查明当前的请求正在进行什么处理。在下一节，我们将会看到典型的就绪状态处理程

8

① http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10

序如何操作。

发起请求的最后一步是为POST请求提供体内容并发送到服务器。这两个步骤都是通过send()方法来完成的。对于GET请求（通体没有体）则不传递任何体内容作为参数，如下：

```
xhr.send(null);
```

如果传递请求参数给POST请求，则传递给send()方法的字符串必须符合正确的格式（可以把它看作查询字符串格式），也就是说名称和值必须正确地URI编码。URI编码超出了本节的范围（结果证明，jQuery准备为我们处理那些事情），但如果你觉得好奇，请上网搜索术语encodeURIComponent，你会得到相应的回报。

221

这种调用的示例如下：

```
xhr.send('a=1&b=2&c=3');
```

下面全面地探讨就绪处理程序。

8.1.3　跟踪进展

XHR实例通过就绪状态处理程序通知自身的进展。可以通过将函数的引用指派给XHR实例的onreadystatechange属性作为就绪处理程序，从而把这个处理程序建立起来。

一旦通过send()方法发起请求，随着请求在不同的状态之间转换，这个回调函数会被调用多次。在readyState属性里（请参考表8-1中关于这个属性的描述）可以得到当前请求的数字状态码。

那很好，然而我们多半只关心什么时候请求完成以及成功与否。因此经常看到利用如代码清单8-2所示的模式来实现的就绪处理程序。

代码清单8-2　编写就绪状态处理程序，忽略除完成状态以外的所有状态

```
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status >= 200 &&
        xhr.status < 300) {
      //success
    }
    else {
      //error
    }
  }
}
```

忽略除完成状态以外的所有的状态

根据响应的状态进行代码分支

在成功的时候执行

在失败的时候执行

这个模式忽略除完成状态以外的所有状态，并且只要完成，就检查status属性的值，以便确定请求成功与否。HTTP规范定义200到299范围内的编码为成功，而大于或等于300的那些编码则为各种失败类型。

我们应该注意到一件关于就绪处理程序的事情：它通过顶层变量来引用XHR实例。然而，我们不应该期待XHR实例作为一个参数传递给处理程序吗？

222

好，我们可能有那样的期待，但那样的期待没有成为现实。必须通过一些其他方式来找到XHR实例，通常是顶层（全局）变量。如果我们想要同时发起多个请求，这将成为问题。幸亏jQuery

的Ajax API为我们解决了这个问题。

下面探讨如何处理来源于已完成请求的响应。

8.1.4 获得响应

就绪处理程序确定readyState为已完成并且请求已成功地完成后，就可以从XHR实例获取响应体。

尽管名为Ajax（X代表XML），响应体的格式可以是任何文本格式而不限于XML。事实上，大多数时候，对Ajax请求的响应不是XML格式，而是普通的文本，或多半是HTML片段，甚至是JavaScript对象的文本表示，或JSON（JavaScript Object Notation，JavaScript对象表示法）格式的数组。

不管格式如何，通过XHR实例的responseText属性可以得到响应体（假定请求成功完成）。如果响应通过包含content-type标头而指定MIME类型为text/xml（或任何XML MIME类型），指示响应体的格式为XML，则响应体将被解析为XML，在responseXML属性里可以得到作为结果的DOM。然后可以利用JavaScript（jQuery则利用选择器API）来处理XML DOM。

在客户端上处理XML虽然并非前沿科学，但是就算有jQuery帮忙，依然还是痛苦的事情。虽然有时候只有XML能返回复杂的层次结构数据，但如果不是绝对需要XML的全部力量（XML也带来相应的麻烦），页面作者通常使用其他格式。

但其他格式未必不带来痛苦。如果返回JSON，就必须将它转换为对应的运行时等价物；如果返回HTML，就必须将它加载到适当的目标元素；如果返回的HTML标记包含需要求值的<script>块，又该如何呢？在本节里我们不打算处理这些问题，因为本节不打算作为Ajax的完全指南，更重要的是，我们将会发现jQuery已经处理这些问题的大部分。

整个过程的图表如图8-1所示。

在这个简短的Ajax概述里，我们已标识下列难点，都是利用Ajax的页面作者所必须处理的： 223

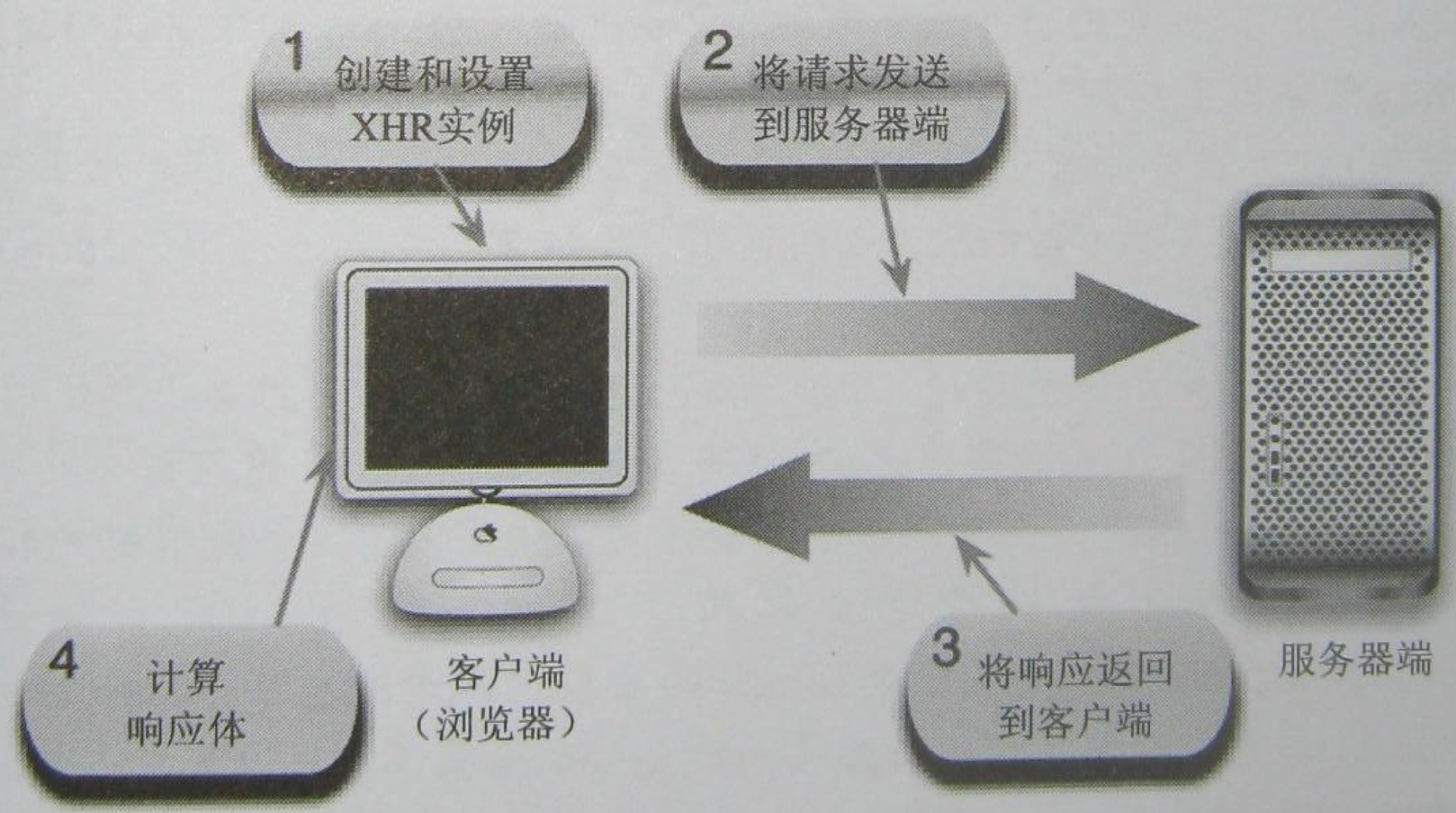


图8-1 Ajax请求的生命周期，即从客户端发送请求到服务器，再从服务器返回响应到客户端

- 初始化一个XHR对象，需要特定于浏览器的代码；

- 就绪处理程序必须过滤大量的无趣的状态变化；
- 就绪处理程序不会自动地获得引用以便调用XHR实例；
- 响应体必须根据其格式以不同方式来进行处理。

本章的剩余部分将说明jQuery的Ajax命令以及实用工具函数如何使得在页面上利用Ajax更为简便（和更为整洁）。在jQuery的Ajax API里有大量的选择，下面从最简单的和最常用的工具开始讲述。

8.2　加载内容到元素上

Ajax的最为常见的用途之一或许是从服务器获取一大块的内容并填充到DOM里某个战略位置上。内容可能是HTML片段，那就成为目标容器元素的子内容；内容也可能是普通的文本，那就成为目标元素的内容。

224

想象一下，在页面加载时我们想要利用名为/serverResource的资源定位符从服务器获取一大块HTML，并使它成为id为someContainer的 元素的内容。现在，在本章里最后一次观看不依靠jQuery的协助怎样做到这一点。利用本章前面所讲述的模式，onload处理程序体如代码清单8-3所示。这个示例的完整HTML文件可以在chapter8/listing.8.3.html找到。

为运行本章的示例而做好准备

与迄今为止在本书中已检查的任何代码示例不同，本章的代码示例需要Web服务器的服务以便接受对服务器端资源的请求。因为讨论服务器端的操作机制完全超出了本书的范围，所以我们将要设置最少限度的服务器端资源，以便发送数据回到客户端而无需担心实际的操作。我们把服务器看成黑盒，因此不必也不想知道服务器怎样工作。

为了启用这些“黑盒”资源的服务，必须建立某个类型的Web服务器。为了给你提供方便，服务器端资源已经以两种格式来设置：JSP（Java Server Page，Java服务器页面）和PHP（Personal Home Page，个人主页）。如果运行（或希望运行）servlet[1]/JSP引擎，可以使用JSP资源；如果想在你所选的Web服务器上启用PHP，可以使用PHP资源。

如果你想使用JSP资源却没有正在运行的适宜的Web服务器，可以查看本章示例代码附带的关于建立Tomcat Web服务器（自由软件）的说明。可以在文件chapter8/tomcat.pdf找到这些说明。别担心，也许比你想象的还要简单！

在可下载代码里找到的示例是为使用JSP资源而建立的。如果你想切换到使用PHP的示例，可以进行查找和替换，把文件里所有的.jsp字符串实例替换为.php。请注意并非所有的服务器端资源都已从JSP转换为PHP，但现有PHP资源应该足以使得PHP高手能够填充其余的资源。

只要你所选择的Web服务器建立起来了，就可以在浏览器的地址栏输入http://localhost:8080/chapter8/test.jsp（检查Tomcat的安装）或者http://localhost/chapter8/test.php（检查PHP的安装）。后者假定你已建立Web服务器（Apache或任何其他选择）并且使用示例代码根文件夹作为文档基础。

225

当你能够成功地看到合适的测试页面时，就做好了运行本章示例的准备。

① servlet是Java EE平台的服务器端小程序。——译者注

代码清单8-3 利用原生的XHR来包含一个HTML片段

```
var xhr;

if (window.XMLHttpRequest) {
  xhr = new XMLHttpRequest();
}
else if (window.ActiveXObject) {
  xhr = new ActiveXObject("Msxml2.XMLHTTP");
}
else {
  throw new Error("Ajax is not supported by this browser");
}

xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status >= 200 && xhr.status < 300) {
      document.getElementById('someContainer')
        .innerHTML = xhr.responseText;
    }
  }
}

xhr.open('GET','/serverResource');
xhr.send();
```

虽然以上代码没有什么复杂之处，但代码数量可不小：大约19行，还不包括为了可读性而添加的空行以及为了适合页面的宽度而人为地一分为二的那一行。

利用jQuery编写作为就绪处理程序体的等效代码如下：

```
$('#someContainer').load('/serverResource');
```

我们敢打赌你乐于编写哪一种代码！下面来看看在这个语句中用到的jQuery命令。

8.2.1 利用 jQuery 加载内容

上一节的简单jQuery语句利用最为基本却有用的jQuery的Ajax命令之一`load()`，从服务器端资源轻松地加载内容。这个命令的完整的语法说明如下。

226

命令语法：load

load(url,parameters,callback)

有可选的参数，向指定的URL发起Ajax请求。可以指定回调函数，在请求完成时调用。响应文本将替换所有已匹配元素的内容。

参数

url	（字符串）服务器端资源的URL，正是向它而发送请求。
parameters	（对象）一个对象，其属性被序列化为正确地编码的参数以便传递到请求里。如果指定，就用POST方法来发送请求；如果省略，就用GET方法来发送请求。
callback	（函数）一个回调函数，在响应数据已经加载到包装集元素之后被调用。传入

8

这个函数的参数是响应文本、状态码，以及XHR实例。

返回
包装集

虽然用起来很方便，但这个命令还是有一些需要注意的重要细节。例如，如果利用parameters参数来提供请求参数，则发送POST请求；如果省略parameters参数，则发送GET请求。如果想要发送带有参数的GET请求，可以把参数作为查询字符串包含在URL里。但要意识到如果这样做，就应确保查询字符串正确地格式化以及请求参数的名称和值为URI编码。

大部分时间内，我们利用load()命令把已完成响应的文本插入到包装集所包含的任何元素中，但有时可能想对作为响应而返回的元素进行筛选。在对响应元素进行筛选时，jQuery允许在URL上指定选择器以便对哪一些响应元素可以插入到已包装元素进行限制，通过给URL添加空格和#作为后缀来完成，后缀的后面紧接选择器。

例如，为了筛选响应元素，以便只插入 实例，编写语句如下：

```
$('.injectMe').load('/someResource #div');
```

227

如果请求参数来自表单控件，帮助建立查询字符串的有用命令是serialize()，语法如下。

命令语法：serialize

serialize()

根据包装集里所有的成功表单元素，创建正确格式化的、经过URI编码的查询字符串。

参数
无

返回
已格式化的查询字符串

serialize()命令足够聪明，只从包装集里所有的成功表单元素（控件）来收集信息。成功控件是指根据HTML规范[1]的规则，作为表单提交的一部分而被包括的控件。未选中的复选框和单选按钮、未选择的下拉列表，以及任何已禁用的控件均被视为不成功控件，因此不参与表单提交。这些控件也被serialize()所忽略。

如果我们宁愿以JavaScript数组形式来获取表单数据（由于反对查询字符串），jQuery提供serializeArray()方法。

命令语法：serializeArray

serializeArray()

把所有的成功表单控件的值都收集到对象的数组中（包含控件的名称和值）。

① http://www.w3.org/TR/html401/interact/forms.html#h-17.13.2。

参数
无

返回
表单数据的数组

serializeArray()所返回的数组由匿名对象实例所组成，每一个均包含name属性和value属性，也就是说包含每一个成功表单控件的名称和值。

有load()命令在握，我们把它付诸实践，解决许多Web开发者都曾遭遇的真实世界的一个常见问题。

228

8.2.2 加载动态的库存数据

常常是在商业应用里，特别是对于零售类网站来说，需要从服务器获取实时数据以便给用户提供最新信息。毕竟不该误导消费者，使他们误以为能够买到已经脱销的东西，不是吗？

在这一节里我们着手开发一个页面，它将在本章各处派上用场。这个页面是名为Boot Closet（靴子储藏室）的虚构企业（摩托车靴子进销存在线零售商）网站的一部分。与其他在线零售商的固定的产品目录不同，这个进销存清单是变动的，根据当天经营者达成了什么交易以及销售了什么货物而改变。因此对我们来说，确保始终显示最新信息是非常重要的。

建立一个页面（忽略网站导航和其他的样板文件以便专注于手头上的示例），我们想给客户提供包含当前可用款式的下拉列表，在下拉列表的一项被选中的时候，给客户显示关于那个款式的详细信息。在初始化显示时，页面如图8-2所示。

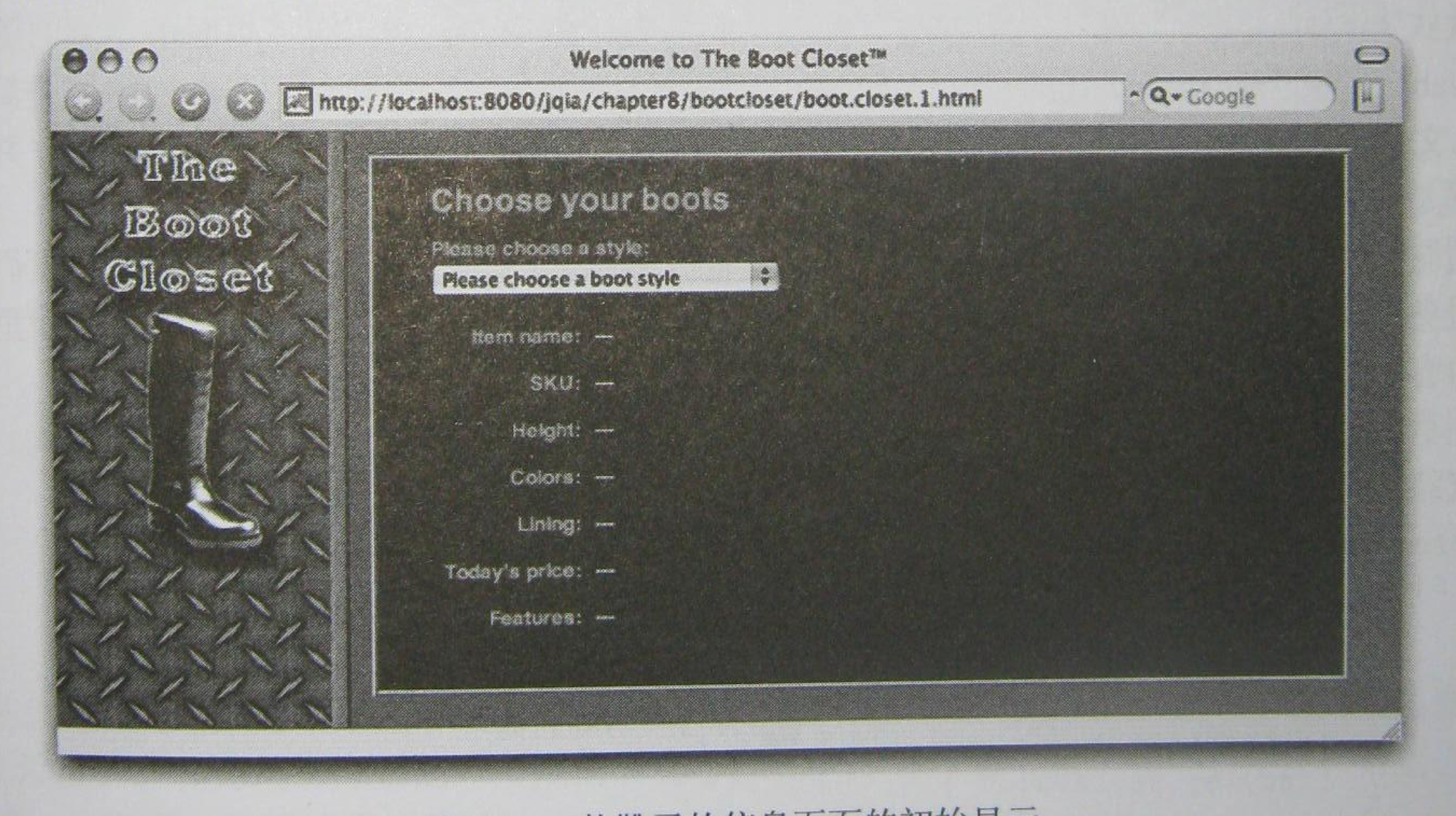


图8-2 一款靴子的信息页面的初始显示

在第一次加载页面后，显示预先加载的、当前库存可用款式的下拉列表，以及用来显示款式数据的带标签的字段。如果没有选中任何款式，就显示横线作为数据的占位符。

229

8

下面从创建这个页面结构的HTML标记的定义开始，如下所示：

```
<body id="bootCloset1">
  <img id="banner" src="boot.closet.branding.png"/>
  <form action="" id="orderForm">
    <div id="detailFormContainer">
      <h1>Choose your boots</h1>
      <div>
        <label>Please choose a style:</label><br/>
        <select id="styleDropdown">
          <option value="">Please choose a boot style</option>
          <option value="7177382">Caterpillar Tradesman Work Boot</option>
          <option value="7269643">Caterpillar Logger Boot</option>
          <option value="7141832">Chippewa 17" Engineer Boot</option>
          <option value="7141833">Chippewa 17" Snakeproof Boot</option>
          <option value="7173656">Chippewa 11" Engineer Boot</option>
          <option value="7141922">Chippewa Harness Boot</option>
          <option value="7141730">Danner Foreman Pro Work Boot</option>
          <option value="7257914">Danner Grouse GTX Boot</option>
        </select>
      </div>
      <div id="detailsDisplay"></div>
    </div>
  </form>
</body>
```

并没有很多代码，不是吗？

为了遵守“不唐突的JavaScript”原则，我们已经把所有的视觉呈现信息定义在外部样式表中，并且在HTML标记里没有包括任何行为信息。

款式下拉列表的选项是预先填充的。在本章的所有示例中，假定正在使用服务器端的资源来驱动Web应用：毕竟与这些资源打交道是Ajax的全部要点所在。因此即便示例使用的仅是简单的HTML文件，我们也假定它原来是由服务器端的模板化资源比如JSP或PHP页面所生成的，并且产品数据是从库存数据库（或任何其他存储之处）动态地包括进来的。

同时，为显示款式明细而定义的`<div>`容器（id为`detailsDisplay`）完全为空！我们打算依靠服务器端的模板化资源来提供动态内容，因此不想在这里以及在JSP（或PHP）页面里指定这些内容。将相同的结构定义在两个地方就不得不使它们保持同步。馊主意！

230

在页面加载时，我们从服务器获取内容的空版本，因此只需在一个地方定义这个结构。下面查看就绪处理程序。

```
$(function(){
  $('#styleDropdown')
    .change(function(){
      var styleValue = $(this).val();
      $('#detailsDisplay').load(
        'getDetails.jsp',
        { style: styleValue }
      );
    })
    .change();
});
```

❶ 对款式下拉列表进行包装以及绑定change处理程序

❷ 为选中的款式而加载数据

❸ 触发change处理程序

在这个就绪处理程序里，我们对靴子款式下拉列表进行包装，并且为它绑定change处理程序❶。无论何时客户改变款式的选择，将回调这个change处理程序，通过对this（在处理程序里this是触发change事件的<select>元素）应用val()命令而获取当前选项值。然后对detailsDisplay元素应用load()命令❷而发起服务器端资源getDetails.jsp的Ajax回调（传递款式的值作为名为style的参数）。

就绪处理程序的最后操作是调用change()命令❸以便调用change处理程序。这将带着默认为""（空字符串）的款式选项的值而发起一个请求，使服务器端资源返回构造，导致如图8-2所示的画面。

在客户选择可用的靴子款式之后，页面的外观如图8-3所示。

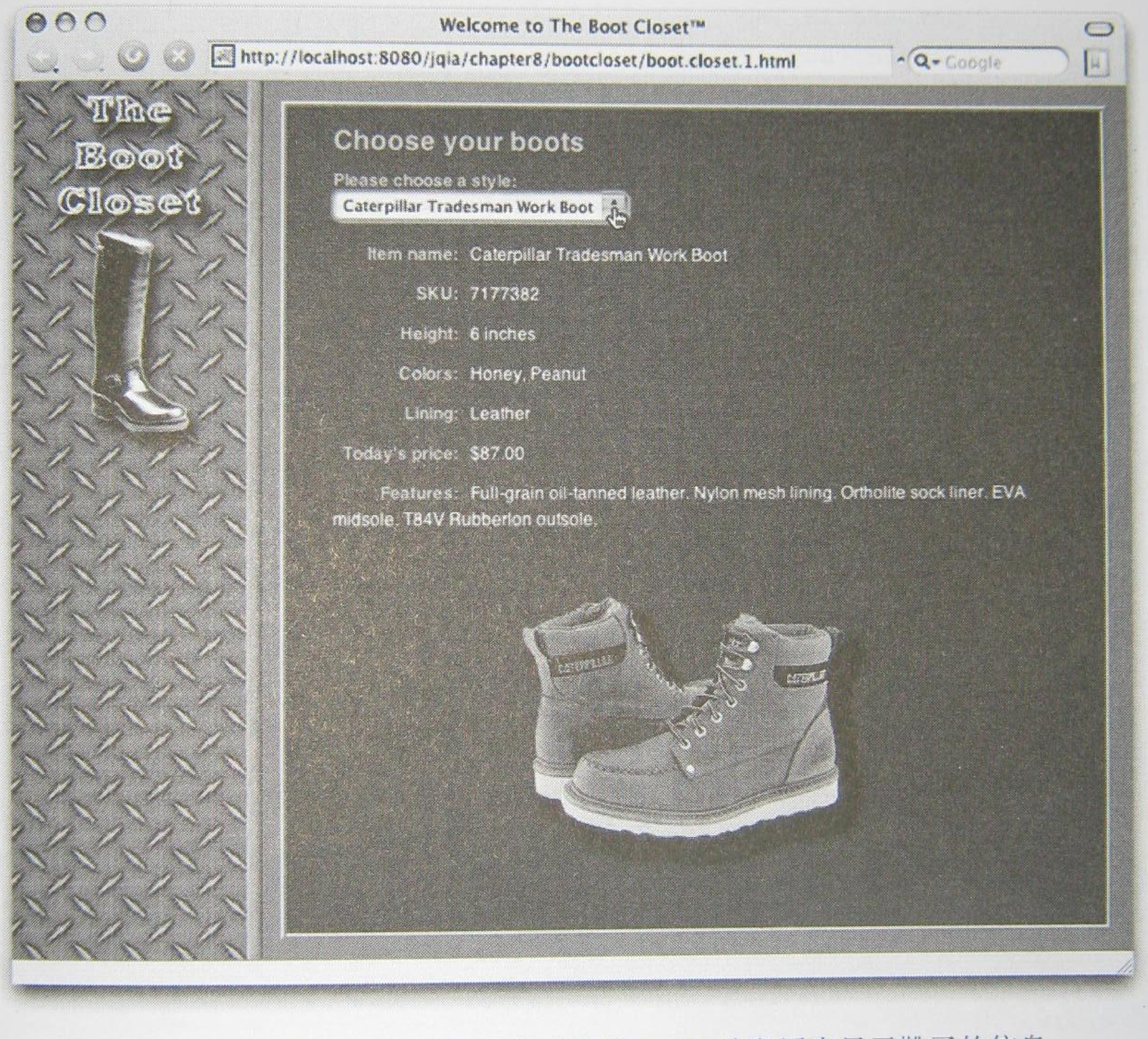


图8-3 服务器端资源返回预先格式化的HTML片段用来显示靴子的信息

在就绪处理程序里所执行的最值得注意的操作是利用load()命令轻松快速地从服务器获取HTML片段并且作为现有元素的子节点而放置于DOM中。这个命令极为简便，并且非常适合于由服务器（具有服务器端的模板化功能比如JSP和PHP）所驱动的Web应用。

代码清单8-4显示Boot Closet页面的完整代码，文件可以在chapter8/bootcloset/boot.closet.1.html找到。随着本章的进展我们将重新访问这个页面并添加更为高级的功能。

代码清单8-4　第一阶段的Boot Closet零售商页面

```
<html>
  <head>
    <title>Welcome to The Boot Closet&trade;</title>
    <link rel="stylesheet" type="text/css" href="boot.closet.css">
    <script type="text/javascript"
            src="../../scripts/jquery-1.2.1.js"></script>
    <script type="text/javascript">
      $(function(){
        $('#styleDropdown')
          .change(function(){
            var styleValue = $(this).val();
            $('#detailsDisplay').load(
              'getDetails.jsp',
              { style: styleValue }
            );
          })
          .change();
      });
    </script>
  </head>

  <body id="bootCloset1">
    <img id="banner" src="boot.closet.branding.png"/>
    <form action="" id="orderForm">
      <div id="detailFormContainer">
        <h1>Choose your boots</h1>
        <div>
          <label>Please choose a style:</label><br/>
          <select id="styleDropdown">
            <option value="">Please choose a boot style</option>
            <option value="7177382">
             Caterpillar Tradesman Work Boot</option>
            <option value="7269643">Caterpillar Logger Boot</option>
            <option value="7141832">Chippewa 17" Engineer Boot</option>
            <option value="7141833">Chippewa 17" Snakeproof Boot</option>
            <option value="7173656">Chippewa 11" Engineer Boot</option>
            <option value="7141922">Chippewa Harness Boot</option>
            <option value="7141730">Danner Foreman Pro Work Boot</option>
            <option value="7257914">Danner Grouse GTX Boot</option>
          </select>
        </div>
        <div id="detailsDisplay"></div>
      </div>
    </form>
  </body>
</html>
```

232

如果想要获取HTML片段用来填充一个元素（或一组元素）的内容，`load()`命令就非常有用。但有时候我们或许想对Ajax请求的完成方式施加更多的控制，或必须对返回的响应体的数据进行更为复杂的处理。

下面继续研究jQuery为这些更为复杂的情况提供了什么应对措施。

8.3 发起 GET 和 POST 请求

load()命令发起GET或POST请求，取决于是否带请求数据进行调用，但有时候我们想对使用哪种HTTP方法进行更多的控制。为什么我们应该关心这件事情？因为，也许是服务器在乎这件事情。 233

Web作者过去使用GET和POST方法很轻松随意，使用这个或那个而不太留意HTTP协议规定如何使用这些方法。关于这两个方法的规定如下。

- GET请求——规定为幂等的[①]：服务器的状态和应用的模型数据应该不受GET操作的影响。同样的GET操作，进行第一次第二次第三次……应该返回完全一致的结果（假定没有其他力量在运作从而改变服务器的状态）。
- POST请求——可以是非幂等的：发送到服务器的数据可以用来修改应用的模型状态。例如，给数据库添加记录或从服务器删除信息。

因此，GET请求应该用来获取数据（正如它的名称所暗示的那样）。可能需要利用GET请求给服务器发送一些数据。例如，发送一个款式代号以便获取颜色信息。但如果发送数据到服务器是为了实现一个修改，就应该使用POST请求。

警告 这是超出理论之外的事情。浏览器基于使用的HTTP方法而做出关于缓存的决定：GET请求非常容易遭受缓存。利用适当的HTTP方法可以确保你不会违背浏览器对请求意图的要求。

虽说是那样，jQuery仍然给我们提供几个办法用于生成GET请求，与load()不同，不是以操作包装集的jQuery命令方式来实现的，而是提供几个实用工具函数用来生成不同类型的GET请求。我们在第1章就已指出，jQuery实用工具函数是顶层函数，它们的命名空间为全局名称jQuery或别名$。

下面逐一查看这些函数。

8.3.1 利用 jQuery 获取数据

如果我们想要从服务器获取一些数据然后决定进行什么处理（而不是由load()命令把它设置为一个HTML元素的内容），可以利用$.get()实用工具函数。它的语法如下所示。 234

8

函数语法：$.get

$.get(url,parameters,callback)

利用指定的URL、带着任何已传入的参数作为查询字符串而向服务器发起GET请求。

参数

url	（字符串）将要通过GET方法进行交互的服务器端资源的URL。

① 不管执行一次或多次都起到相同的作用，操作的这种特征叫做幂等。——译者注

parameters	（对象/字符串）一个对象，其属性作为“名称/值对”用于构造查询字符串并追加到URL；或者一个预先格式化的和URI编码的查询字符串。
callback	（函数）回调函数，在请求完成时被调用。响应体作为第一个参数传递到这个回调函数，响应状态则作为第二个参数传递到这个回调函数。

返回

XHR实例

下面看这个函数的一个简单应用，如代码清单8-5所示（文件可以在chapter8/$.get.html找到）。

代码清单8-5　利用实用工具函数$.get()从服务器获取数据

```
<html>
  <head>
    <title>$.get() Example</title>
    <link rel="stylesheet" type="text/css" href="../common.css">
    <script type="text/javascript"
            src="../scripts/jquery-1.2.1.js"></script>
    <script type="text/javascript">
      $(function(){
        $('#testButton').click(function(){
          $.get(
            'reflectData.jsp',
            {a:1, b:2, c:3},
            function(data) { alert(data); }
          );
        });
      });
    </script>
  </head>

  <body>
    <button type="button" id="testButton">Click me!</button>
  </body>
</html>
```

❶ 从服务器获取数据

在这个简单的页面里，我们创建并设置按钮，一旦点击就让它发起$.get()函数调用❶。带着指定的请求参数的值a、b和c，向服务器资源reflectData.jsp（返回文本片段，显示作为请求参数而传递的值）发起GET请求。返回的数据被传递给回调函数，在回调函数里可以对这些数据进行任何处理。在这个示例中，它只是弹出一个警告消息框，显示所取回的数据。

如果把这个HTML页面加载到浏览器并点击按钮，可以看到如图8-4所示的画面。

如果响应包含一个XML文档，则文本将被解析，而传递给回调函数的data参数将是作为结果的DOM。

如果需要灵活性并且我们的数据在本质上是层次结构的，则XML很适合。但是XML解析起来是相当痛苦的。下面查看另一个jQuery实用工具函数，当我们需要更为基本的数据时这个函数相当有用。

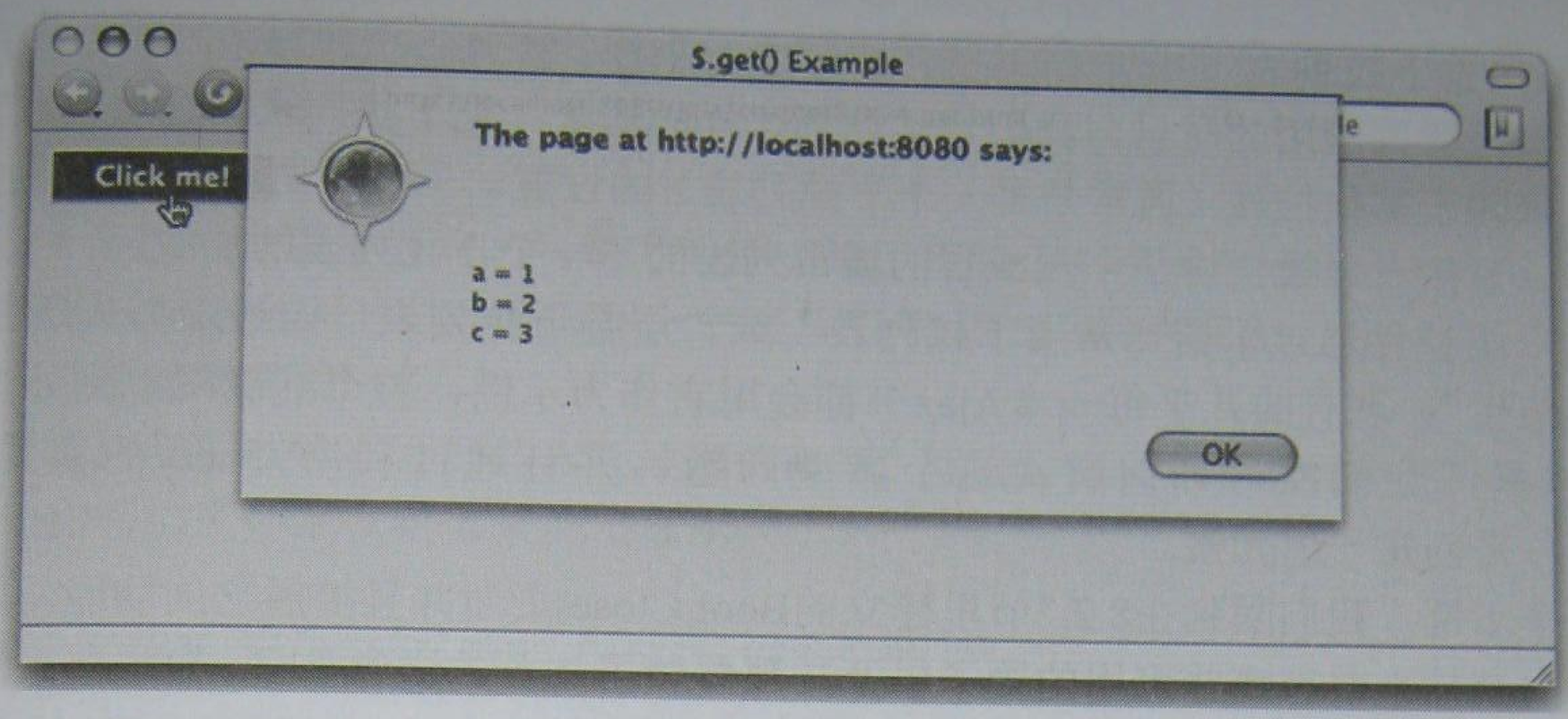


图8-4 实用工具函数$.get()从服务器获取数据（可供我们任意操作），仅在警告消息框里加以显示

8.3.2 获取 JSON 数据

在上一节里已经谈到，如果一个XML文档从服务器返回，XML文档会被自动解析，然后作为结果的DOM对回调函数是可用的。如果XML显得大材小用或者相反，不宜作为数据传送机制，JSON常常被用来取而代之。一个理由就是JSON易于用客户端脚本进行解析。好，jQuery使得这件事情更加简便。

有时如果我们知道响应是JSON，则调用实用工具函数$.getJSON()可以自动解析返回的JSON字符串，使得作为结果的JavaScript数据项在回调函数里可用。这个函数的语法如下。 236

函数语法：$.getJSON

$.getJSON(url,parameters,callback)

利用指定的URL、带着任何已传入参数作为查询字符串而向服务器发起GET请求。响应被解析为JSON字符串，而作为结果的数据将被传递给回调函数。

参数

url	（字符串）将要通过GET方法进行交互的服务器端资源的URL。
parameters	（对象/字符串）一个对象，其属性作为“名称/值对”用于构造查询字符串并追加到URL；或者一个预格式化和URI编码的查询字符串。
callback	（函数）回调函数，在请求完成时被调用。把响应体解析为JSON字符串，这个字符串的值作为第一个参数传递到这个回调函数，响应状态作为第二个参数传递到这个回调函数。

返回

XHR实例

有时如果我们想从服务器获取数据而无需付出处理XML的代价，则这个函数非常有用。下面看一个示例，把这个函数投入使用。

1. 加载层叠下拉列表

在创建富因特网应用的时候，我们常常遇到这样的需求：把下拉列表的选项设置为一些值，而这些值又依赖于其他控件（通常是另一个下拉列表）的设置。一个常见示例是从下拉列表选择一个州或省，紧接着导致包含那个州或省的城市列表的另一个下拉列表的加载。

这样的一组控件通过术语“从属下拉列表”或“层叠下拉列表”而逐渐为人所知，并成为Ajax的“模范生”：现有的几乎每一本Ajax书都会用它作为示例，整个互联网上到处都有它的身影。在本节里，我们探讨如何解决这个典型问题，并且通过利用jQuery的实用工具函数`$.getJSON()`来创建一组元素。

为了这个示例，我们回到在8.2.2节里建立的Boot Closet页面并且扩展它的功能。原先编写的页面允许客户查找当前有哪些可用的靴子以及与那些靴子相关的详细信息。但客户无法挑选一对靴子以便购买。我们喜欢做买卖，那么下一步就是添加控件让客户挑选颜色和尺寸。

请记住我们是进销存企业——那意味着我们不一定拥有任何一个可用款式的全部系列。每天都只有特定的颜色可供购买，而对于那些颜色也只有特定的尺寸可供购买。因此我们不能对颜色和尺寸进行硬编码，而必须从实时库存数据库动态地获取这些列表。

为了让客户能够选择颜色和尺寸，我们给表单添加两个下拉列表：一个用于颜色，另一个用于尺寸。增强之后表单的初始外观如图8-5所示。你可以加载这个示例HTML页面，文件可以在chapter8/bootcloset/boot.closet.2.html找到。

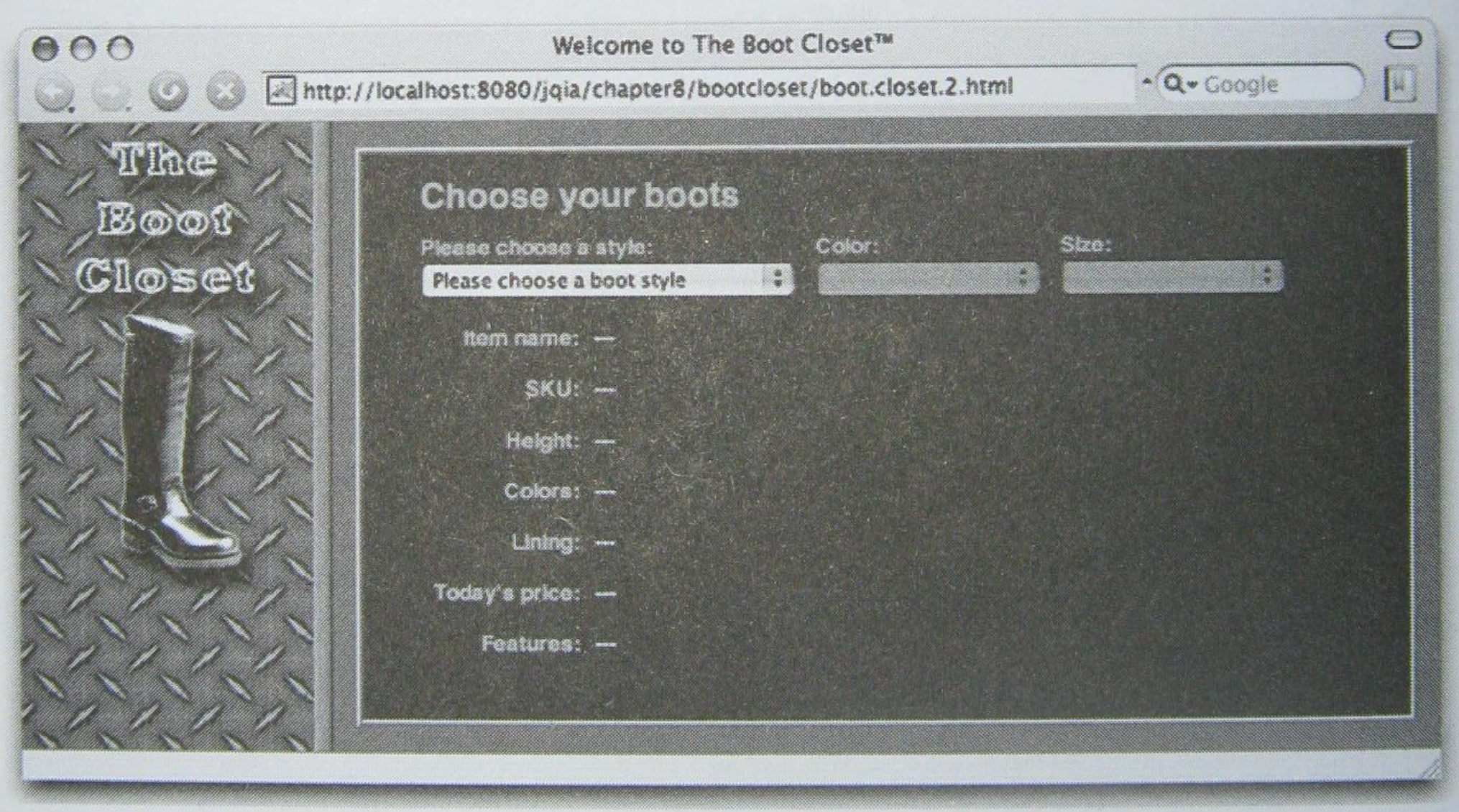


图8-5　订货单的初始状态，带有处于禁止状态的空的从属下拉列表

用于选择靴子款式的下拉列表是启用的（并且预先填充了可用款式，这一点我们在前面已讨论），但颜色和尺寸的下拉列表是禁止的并且是空的。我们无法预先填充这两个下拉列表，因为不知道在一个款式被客户选中之前应该显示什么颜色，更不知道在客户选择款式和颜色之前应该显示什么尺寸。

基于以上条件，下面是我们想要为这些控件实现的功能。

- 如果一个款式被选中，应该启用颜色下拉列表并且为选中的款式而填充可用的颜色选项。 238
- 如果款式和颜色都已选中，就应该启用尺寸的下拉列表，并且为选中的款式和颜色的组合显示可用的尺寸选项。
- 这3个下拉列表需要彼此之间始终保持一致。我们必须确保这一点，不管用户动手操作这些列表的顺序如何，决不能显示一种在库存里无法提供的组合。

让我们卷起袖子开始工作。

作为开头，我们展示在页面体内新添加的HTML标记。下面是新页面的HTML，其中重要的代码的改动和添加之处都以粗体来突出显示：

```
<body id="bootCloset2">
  <img id="banner" src="boot.closet.branding.png"/>
  <form action="" id="orderForm">
    <div id="detailFormContainer">
      <h1>Choose your boots</h1>
      <div id="cascadingDropdowns">
        <div>
          <label>Please choose a style:</label><br/>
          <select id="styleDropdown">
            <option value="">Please choose a boot style</option>
            <option value="7177382">
             Caterpillar Tradesman Work Boot</option>
            <option value="7269643">Caterpillar Logger Boot</option>
            <option value="7141832">Chippewa 17" Engineer Boot</option>
            <option value="7141833">Chippewa 17" Snakeproof Boot</option>
            <option value="7173656">Chippewa 11" Engineer Boot</option>
            <option value="7141922">Chippewa Harness Boot</option>
            <option value="7141730">Danner Foreman Pro Work Boot</option>
            <option value="7257914">Danner Grouse GTX Boot</option>
          </select>
        </div>
        <div>
          <label>Color:</label><br/>
          <select id="colorDropdown" disabled="disabled"></select>
        </div>
        <div>
          <label>Size:</label><br/>
          <select id="sizeDropdown" disabled="disabled"></select>
        </div>
      </div>
      <div id="detailsDisplay"></div>
    </div>
  </form>
</body>
```


我们已经修改<body>标签的id（主要是为了能够在CSS样式表内把它作为开关使用，这个CSS样式表被页面的多个版本所使用），并添加两个已禁止的空下拉列表。

为了给页面添加与这些控件相关的新的行为，我们还需要增强就绪处理程序。新的就绪处理程序，改动之处也用粗体来突出显示，如下所示：

```
$(function(){
  $('#styleDropdown')
    .change(function(){
      var styleValue = $(this).val();
      $('#detailsDisplay').load(
        'getDetails.jsp',
        { style: styleValue }
      );
      adjustColorDropdown();
    })
    .change();
  $('#colorDropdown')
    .change(adjustSizeDropdown);
});
```

❶ 触发颜色的下拉列表的状态调整

❷ 给颜色的下拉列表绑定change监听器

这些改动虽然细小却意义重大。首先，我们在款式下拉列表的change监听器内调用命名为adjustColorDropdown()的函数❶。这将触发基于已选中款式值的状态和内容所要求的任何变化。

然后我们给新的颜色下拉列表添加监听器❷以便在颜色下拉列表的值改变时调整尺寸下拉列表的状态和内容。这个监听器被设置为名为adjustSizeDropdown()的函数。

这一切非常简单，但我们还没有编写函数用来实现从属下拉列表的状态和内容的变化。下面首先定义函数adjustColorDropdown()来处理颜色下拉列表。

```
function adjustColorDropdown() {
  var styleValue = $('#styleDropdown').val();
  var dropdownSet = $('#colorDropdown');
  if (styleValue.length == 0) {
    dropdownSet.attr("disabled",true);
    dropdownSet.emptySelect();
    adjustSizeDropdown();
  }
  else {
    dropdownSet.attr("disabled",false);
    $.getJSON(
      'getColors.jsp',
      {style:styleValue},
      function(data){
        dropdownSet.loadSelect(data);
        adjustSizeDropdown();
      }
    );
  }
}
```

❶ 启用或禁用使颜色下拉列表

❷ 清空已禁用的下拉列表并清除从属下拉列表

❸ 根据款式获取颜色的值

❹ 触发从属下拉列表的调整

获取款式下拉列表的值并指派到styleValue变量以供稍后引用，然后形成颜色下拉列表的包装集并指派到dropdown变量。我们打算在函数的剩余部分里反复引用这个包装集，而不想在每次需要这个包装集时导致重新创建的开销。

下一步根据款式下拉列表的值来决定是否启用颜色下拉列表❶：如果值为空，则禁用；如果值不为空，则启用。如果禁用颜色下拉列表，则同时要通过调用emptySelect()命令使其为空❷。

等一等！emptySelect()命令是什么样的？

在你开始兴冲冲地翻查前面各章搜索这个命令之前，我说别徒劳了。至少迄今为止这个命令根本不存在。我们将在下一节创建这个命令。目前只需知道这个命令将删除颜色下拉列表的所有选项。

在禁用并清空颜色下拉列表之后，我们调用adjustSizeDropdown()函数❷以便确保对从属的尺寸下拉列表进行适当的调整。

想要启用颜色下拉列表，就要根据款式下拉列表已选中的值而给颜色下拉列表填充适当值。为了获取那些值，我们利用$.getJSON()函数的服务❸指定服务器端的资源getColors.jsp，并通过名为style的请求参数来传递已选中款式的值。

在回调函数被调用时（在Ajax请求返回响应之后），传递到回调函数的参数是对响应（作为一个JSON构造）进行求值而得到的JavaScript值。从getColors.jsp返回的典型JSON构造如下所示：

```
[
  {value:'',caption:'choose color'},
  {value:'bk',caption:'Black Oil-tanned'},
  {value:'br',caption:'Black Polishable'}
]
```

241

这种表示法定义对象数组，每个对象都包含两个属性：value和caption。这些对象定义选项的值和显示文本以便添加到颜色下拉列表。我们将求值后的JSON值传递给应用到颜色下拉列表的loadSelect()命令传递，以此实现颜色下拉列表的填充。

最后，无论何时颜色下拉列表的值改变了，都要确保尺寸下拉列表反映新的值：调用函数adjustSizeDropdown()❹应用一系列的操作到尺寸下拉列表，与颜色下拉列表的例程相似。

函数adjustSizeDropdown()的定义如下：

```
function adjustSizeDropdown() {
  var styleValue = $('#styleDropdown').val();
  var colorValue = $('#colorDropdown').val();
  var dropdownSet = $('#sizeDropdown');
  if ((styleValue.length == 0)||(colorValue.length == 0) ) {
    dropdownSet.attr("disabled",true);
    dropdownSet.emptySelect();
  }else {
    dropdownSet.attr("disabled",false);
    $.getJSON(
      'getSizes.jsp',
      {style:styleValue,color:colorValue},
      function(data){dropdownSet.loadSelect(data)}
    );
  }
}
```

8

这个函数的结构与adjustColorDropdown()函数的结构非常相似，但是不应大惊小怪。除了操作尺寸下拉列表而不是颜色下拉列表以外，这个函数同时查看款式下拉列表和颜色下拉列表的值，以便决定是否启用尺寸下拉列表。在这种情况下，款式和颜色的值都必须不为空，才能启用尺寸下拉列表。

如果启用，就再次调用$.getJSON()从名为getSizes.jsp（传递款式和颜色的值）的服务器端资源来加载尺寸下拉列表的可用选项。

这些函数各就其位，现在我们的层叠下拉列表可以使用了。图8-5显示初始化之后的页面外观，图8-6显示在款式被选中之后页面的相关部分（图的上部）以及在颜色被选中之后的相关部分（图的下部）。

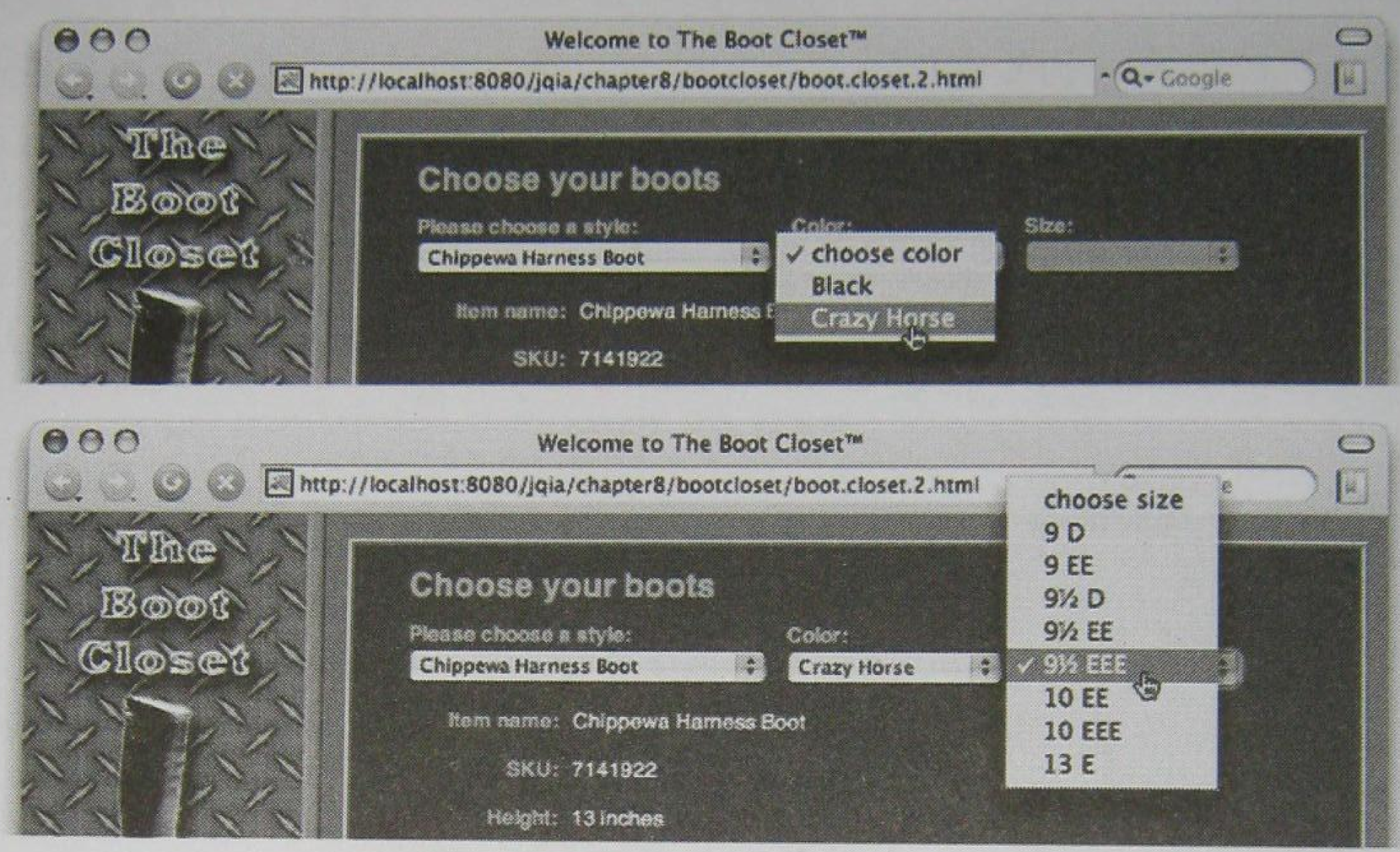


图8-6　选中款式使颜色下拉列表启用（上部），而选中颜色则使尺寸下拉列表启用（下部）

这个页面的完整代码如代码清单8-6所示，文件可在chapter8/bootcloset/boot.closet.2.html找到。

代码清单8-6　添加层叠下拉列表之后的Boot Closet（靴子储藏室）页面

```
<html>
  <head>
    <title>Welcome to The Boot Closet&trade;</title>
    <link rel="stylesheet" type="text/css" href="boot.closet.css">
    <script type="text/javascript" src="../../scripts/jquery-1.2.1.js"></script>
    <script type="text/javascript" src="jquery.jqia.selects.js"></script>
    <script type="text/javascript" src="jquery.jqia.termifier.js"></script>
    <script type="text/javascript">
      $(function(){
        $('#styleDropdown')
          .change(function(){
            var styleValue = $(this).val();
            $('#detailsDisplay').load(
              'getDetails.jsp',
              { style: styleValue },
              function(){
                $('abbr').termifier({
                  lookupResource: 'getTerm.jsp'
                });
              }
            );
            adjustColorDropdown();
```

243

```
    })
    .change();
  $('#colorDropdown')
    .change(function(){ adjustSizeDropdown();});
});

function adjustColorDropdown() {
  var styleValue = $('#styleDropdown').val();
  var dropdownSet = $('#colorDropdown');
  if (styleValue.length == 0) {
    dropdownSet.attr("disabled",true);
    dropdownSet.emptySelect();
    adjustSizeDropdown();
  }
  else {
    dropdownSet.attr("disabled",false);
    $.getJSON(
      'getColors.jsp',
      {style:styleValue},
      function(data){
        dropdownSet.loadSelect(data);
        adjustSizeDropdown();
      }
    );
  }
}

function adjustSizeDropdown() {
  var styleValue = $('#styleDropdown').val();
  var colorValue = $('#colorDropdown').val();
  var dropdownSet = $('#sizeDropdown');
  if ((styleValue.length == 0)||(colorValue.length == 0) ) {
    dropdownSet.attr("disabled",true);
    dropdownSet.emptySelect();
  }
  else {
    dropdownSet.attr("disabled",false);
    $.getJSON(
      'getSizes.jsp',
      {style:styleValue,color:colorValue},
      function(data){dropdownSet.loadSelect(data)}
    );
  }
}
</script>
</head>
<body id="bootCloset3">
  <img id="banner" src="boot.closet.branding.png"/>
  <form action="" id="orderForm">
    <div id="detailFormContainer">
      <h1>Choose your boots</h1>
      <div id="cascadingDropdowns">
        <div>
          <label>Please choose a style:</label><br/>
          <select id="styleDropdown">
```

244

```
        <option value="">Please choose a boot style</option>
        <option value="7177382">Caterpillar Tradesman Work Boot</option>
        <option value="7269643">Caterpillar Logger Boot</option>
        <option value="7141832">Chippewa 17" Engineer Boot</option>
        <option value="7141833">Chippewa 17" Snakeproof Boot</option>
        <option value="7173656">Chippewa 11" Engineer Boot</option>
        <option value="7141922">Chippewa Harness Boot</option>
        <option value="7141730">Danner Foreman Pro Work Boot</option>
        <option value="7257914">Danner Grouse GTX Boot</option>
      </select>
    </div>
    <div>
      <label>Color:</label><br/>
      <select id="colorDropdown" disabled="disabled"></select>
    </div>
    <div>
      <label>Size:</label><br/>
      <select id="sizeDropdown" disabled="disabled"></select>
    </div>
  </div>
  <div id="detailsDisplay"></div>
 </div>
</form>
</body>
</html>
```

在用力地拍拍自己的后背（大功告成）之前，我们应该知道尚未完工。在函数里使用了尚未编写的自定义命令。赶紧返工去……

2. 编写自定义命令

层叠下拉列表的示例需要在页面上的select元素上执行两个操作：从下拉列表里删除所有的选项；加载在一个JavaScript数据结构内已定义的选项。

出于以下两个主要的理由，我们决定把这两个操作实现成jQuery命令。

- 我们知道在页面上的多个地方需要执行这两个操作，因此至少应该把这两个操作分离出来，形成两个函数而不应该在内联代码中重复这两个操作。
- 这两个操作十分普遍，在网站乃至其他Web应用中都能用上。因此把它们定义并构造成jQuery命令是非常有意义的。

我们首先实现emptySelect()命令。

```
$.fn.emptySelect = function() {
  return this.each(function(){
    if (this.tagName=='SELECT') this.options.length = 0;
  });
}
```

在第7章里已经学习如何添加新的jQuery命令，在这里我们应用那些技巧以便给$.fn添加名为emptySelect的新函数。请记住，在这样的函数被调用时，函数上下文（this）是匹配集。通过将each()应用到匹配集，可以迭代匹配集的所有元素：调用指定为each()的参数的迭代器函数。

在迭代器函数内，函数上下文是当前迭代回合的单个元素。我们检查这个元素以便确保它是<select>元素，同时忽略任何其他元素类型，并且设置<select>元素的options数组的长度为0。

这是跨浏览器的、用于从下拉列表里删除所有选项的受到支持的办法。

请注意：我们返回被操作包装集，将它作为函数的值，确保这个命令能够参与任何jQuery命令链。

十分简便！下面实现loadSelect()命令。

我们给$.fn命名空间添加下面的函数：

```
$.fn.loadSelect = function(optionsDataArray) {
return this.emptySelect().each(function(){
    if (this.tagName=='SELECT') {
      var selectElement = this;
      $.each(optionsDataArray,function(index,optionData){
        var option = new Option(optionData.caption,
                                optionData.value);
        if ($.browser.msie) {
          selectElement.add(option);
        }
        else {
          selectElement.add(option,null);
        }
      });
    }
  });
}
```

这个命令稍微复杂一些。

作为这个命令的单个参数，我们要求在上一节所定义的JavaScript构造，即对象数组，每个对象都拥有value和caption属性，以便定义将要添加到<select>元素的选项。 246

我们再一次迭代匹配集里所有的元素。在那之前，通过调用刚才定义的emptySelect()命令，清空匹配集里的<select>元素的选项。这将在添加新的选项之前从<select>元素里删除任何可能存在的选项。

在迭代器函数内，我们检查元素的标签名称而忽略除了<select>以外的所有元素。对于通过这个测试的元素，我们遍历传递给命令的数据数组，创建每个数组项所对应的Option新实例并添加到<select>元素中。

这种添加操作是有问题的，因为必须以浏览器特定的方式来执行。W3C标准定义add()方法如下：为了添加选项到<select>的末尾，null必须作为第二个参数传递给add()。我们原以为省略第二个参数就可以轻松完成了，语句如下：

```
selectElement.add(option);
```

然而生活不会那么轻松！Safari和Opera浏览器不管省略第二个参数还是显式地提供第二个参数，均工作良好，但是基于Mozilla的浏览器在省略第二个参数时则会抛出too-few-arguments（参数太少）异常。显式地添加null作为第二个参数也无济于事，因为这样做会使IE浏览器不再添加新的选项。

左右为难！

因为既没有跨越浏览器的一致方式用来执行操作，也没有对象可供执行检测以便使用受到青睐的对象检测技术，我们迫于无奈只好采取浏览器检测技术。哎，至少我们还有$.browser实用

工具标志用来轻松地执行浏览器检测。

再一次请注意：包装集作为函数的结果而被返回。

这两个命令的完整实现代码的文件可在chapter8/bootcloset/jquery.jqia.selects.js找到，如代码清单8-7所示。

代码清单8-7　实现自定义的选择命令

```
(function($) {
  $.fn.emptySelect = function() {
    return this.each(function(){
      if (this.tagName=='SELECT') this.options.length = 0;
    });
  }

  $.fn.loadSelect = function(optionsDataArray) {
    return this.emptySelect().each(function(){
      if (this.tagName=='SELECT') {
        var selectElement = this;
        $.each(optionsDataArray,function(index,optionData){
          var option = new Option(optionData.caption,
                                  optionData.value);
          if ($.browser.msie) {
            selectElement.add(option);
          }
          else {
            selectElement.add(option,null);
          }
        });
      }
    });
  }
})(jQuery);
```

当提到生成GET请求时，jQuery在`$.get()`和`$.getJSON()`之间给予我们一些强大的工具，但人们不是光靠GET请求而活着。

8.3.3　发起 POST 请求

“有时候选择是一个难题，有时候则不是。”在Almond Joy和Mounds两种糖果之间选择一个时，你的真实反应就跟在选择向服务器发起哪一种请求时的反应一样。有时候我们想要发起GET请求，但有时候我们又想要（或必须）发起POST请求。

有不胜枚举的理由说明我们为什么可能选择POST优先于GET。首先，HTTP协议的意图是POST用于任何非幂等的请求。因此，如果请求可能导致服务器端的状态变化，就应该选择POST（至少依照HTTP纯化论者的说法）。先别说公认的惯例和约定，有时候如果将要向服务器传递的数据超过了一个不大的数量（可以通过URL在查询字符串里进行传递，限制的数量因浏览器而异），就必须使用POST操作。而有时候我们所接触的服务器端资源可能只支持POST操作，甚至根据请求所使用的是GET还是POST方法而执行不同的函数。

对于那些要求或强制使用POST请求的场合，jQuery提供了$.post()实用工具函数。这个函数除使用的HTTP方法不同以外，操作的方式和$.get()一模一样，语法如下。 248

函数语法：$.post

$.post(url,parameters,callback)

利用指定的URL、带着任何包含在请求体内的已传递参数向服务器发起POST请求。

参数

url	（字符串）将要通过POST方法进行交互的服务器端资源的URL。
parameters	（对象/字符串）一个对象，其属性作为“名称/值对”用于构造请求体；或者一个预先格式化和URI编码的查询字符串。
callback	（函数）回调函数，在请求完成时被调用。响应体作为第一个参数传递到回调函数，状态码作为第二个参数传递到回调函数。

返回

XHR实例

在load()命令和各种各样的GET和POST的jQuery Ajax函数之间，对于如何发起请求以及如何通知请求的完成，我们可以施加某种程度的控制。但有时我们需要对Ajax请求进行完全的控制，为此jQuery提供了一个办法供我们选择。

8.4 完全控制 Ajax 请求

到目前为止，我们已经看到适用于各种情况的函数和命令，但有时候我们想要亲手把握关键的细节。

在这一节里，探讨jQuery如何赋予我们这样的支配权。

8.4.1 带着所有的修整发起 Ajax 请求

有时候如果想要（或必须）在如何进行Ajax请求上施加细粒度级别的控制，可以利用jQuery提供的通用的实用工具函数$.ajax()来发起Ajax请求[1]。实际上，所有其他生成Ajax请求的jQuery功能最终会利用这个函数去发起请求。这个函数的语法如下。 249

函数语法：$.ajax

$.ajax(options)

利用已传递的选项（控制如何生成请求以及如何通知回调函数）来发起Ajax请求。

参数

options （对象）一个对象实例，其属性定义这个操作的参数。参见表8-2了解细节。

① 8.4.1节标题中的“修整”是指可以自定义Ajax请求的各个细节。——译者注

返回
XHR实例

看起来简单，不是吗？可不要上当。参数options可以指定很大范围的值用来调整这个函数的操作。这些选项（按照使用的可能性大小来排序）的定义如表8-2所示。

表 8-2　实用工具函数$.ajax()的各种选项

名　称	类　型	描　述
url	字符串	请求的 URL
type	字符串	将要使用的 HTTP 方法。通常是 POST 或 GET。如果省略，则默认为 GET
data	对象	一个对象，其属性作为查询参数而传递给请求。如果是 GET 请求，则把数据作为查询字符串传递；如果是 POST 请求，则把数据作为请求体传递。在这两种情况下，都是由$.ajax()实用工具函数来处理值的编码
dataType	字符串	一个关键字，用来标识预期将被响应所返回的数据的类型。这个值决定在把数据传递给回调函数之前（如果有）进行什么后续处理。有效值如下 xml——响应文本被解析为 XML 文档，而作为结果的 XML DOM 被传递给回调函数 html——响应文本未经处理就被传递给回调函数。在已返回 HTML 片段内的任何<script>块将被求值 json——响应文本被求值为 JSON 字符串，而作为结果的对象被传递给回调函数 jsonp——与json相似，不同之处是提供远程脚本支持（假定远程服务器支持） script——响应文本被传递给回调函数。在任何回调函数被调用之前，响应被作为一个或多个 JavaScript 语句而进行处理 text——响应文本被假定为普通文本。服务器资源负责设置适当的内容类型响应标头。如果省略这个属性，则不对响应文本进行任何处理或求值就传递给回调函数
timeout	数值	设置 Ajax 请求的超时值（毫秒）。如果请求在超时值到期之前仍未完成，则中止请求并且调用错误回调函数（如果已定义）
global	布尔型	启用（如果为 true）或禁用（如果为 false）所谓全局函数的触发。这些函数可以附加到元素上，并且在 Ajax 调用的不同时刻或状态下触发。在 8.8 节详细讨论全局函数。如果省略，默认启用全局函数的触发
contentType	字符串	将要在请求上指定的内容类型。如果省略，默认为 application/x-www-form-urlencoded（与表单提交所使用的默认类型相同）
success	函数	一个函数。如果请求的响应指示成功状态码，则这个函数被调用。响应体作为第一个参数被返回给这个函数，并且根据指定的 dataType 属性进行格式化。第二个参数是包含状态码的字符串——在这种情况下永远为成功状态码
error	函数	一个函数。如果请求的响应返回错误状态码，则这个函数被调用。三个实参被转递给这个函数：XHR 实例、状态消息字符串（在这种情况下永远为错误状态码）以及 XHR 实例所返回的异常对象（可选）
complete	函数	一个函数，在请求完成时被调用。两个实参被传递：XHR 实例和状态消息字符串（成功状态码或错误状态码）。如果也指定了 success 或 error 回调函数，则这个函数在 success 或 error 回调函数调用之后被调用

（续）

名　称	类　型	描　述
beforeSend	函数	一个函数，在发起请求之前被调用。这个函数被传递 XHR 实例，并且可以用来设置自定义的标头或执行其他预请求操作
async	布尔型	如果指定为 false，则请求被提交为同步请求。在默认的情况下，请求是异步的
processData	布尔型	如果设置为 false，则阻止已传递数据被加工为 URL 编码格式。默认情况下，数据被加工为 URL 编码格式（适用于类型为 application/x-www-form-urlencoded 的请求）
ifModified	布尔型	如果设置为 true，则自从上一次请求以来，只有在响应内容没有改变的情况下（根据 Last-Modified 标头）才允许请求成功。如果省略，则不执行标头检查

以上是需要跟踪的大量选项，但不可能任何一个请求都使用多数的那些选项。即便这样，如果能够为计划生成大量请求的页面而设置这些选项的默认值，岂不方便？

8.4.2 设置请求的默认值

显然上一节里最后的问题是关于设置的问题。如你所想，jQuery提供办法用于设置一组默认的Ajax属性，如果不替换值，则使用默认的Ajax属性。如果发起大量类似的Ajax调用，这可以使页面简单得多。

用来设置Ajax默认值列表的函数是$.ajaxSetup()。语法如下。

函数语法：$.ajaxSetup

$.ajaxSetup(properties)

为后续的$.ajax()调用，把传入的一组属性设置为默认值。

参数

properties　（对象）对象实例，其属性定义一组默认的Ajax属性。这些属性与表8-2里所描述的$.ajax()函数的属性相同。

返回

未定义

在脚本处理中的任意时刻，通常在页面加载时（其实可以是页面作者所选的任意时刻），可以用这个函数来设置被所有后续$.ajax()调用所使用的默认值。

注意　用$.ajaxSetup()函数所设置的默认值不会应用到load()命令上。对于实用工具函数，如$.get()和$.post()，其HTTP方法不会因为使用这些默认值而被覆盖。设置GET的默认类型不会导致$.post()使用HTTP的GET方法。

假定正在建立一个页面，对于大多数的Ajax请求（用$.ajax()实用工具函数来发起，而不是用load()命令），我们想要建立一些默认值，以免每个调用都要指定这些值。作为标头<script>

元素里的第一个语句，我们可以这样编写：

```
$.ajaxSetup({
  type: 'POST',
  timeout: 5000,
  dataType: 'html',
  error: function(xhr) {
          $('#errorDisplay)
            .html('Error: ' + xhr.status + ' ' + xhr.statusText);
        }
})
```

252

这将确保后续的每个Ajax调用（再次提醒你，不包括load()在内）使用这些默认值，除非对当前使用的Ajax实用工具函数的属性进行显式地替换。请注意error回调函数的默认设置。对应该应用到所有Ajax调用上的error、complete甚至beforeSend回调函数，以这种方式进行指定是相当常见的。

现在，被global属性控制的那些全局函数怎么样呢？

8.4.3 全局函数

利用$.ajaxSetup()来建立默认的函数，从而为所有的Ajax请求指定将被执行的默认函数。除了这个功能以外，jQuery还允许我们把函数附加到特定的DOM元素。这些函数在Ajax请求处理的不同阶段或在请求最终成功或失败时将被触发。

例如，为了把函数附加到用来显示错误消息的、id为errorConsole的元素上，编写

```
$('#errorConsole').ajaxError(reportError);
```

在任何Ajax请求失败的事件中，函数reportError都将被调用。

当这个或任何其他全局函数被调用时，传递给回调函数的第一个参数由JavaScript的Object实例构成，该实例带有以下两个属性：

- type——包含全局函数的类型的字符串，如ajaxError；
- target——DOM元素（被附加了全局函数）的引用。在前面示例的情况下，那就是id为errorConsole的元素。

我们把这个构造称为全局回调信息（Global Callback Info）对象。一些全局函数类型被传递附加参数（不久就会看到），这个共同的第一个参数可以用来标识什么全局函数类型触发了回调函数以及全局函数被附加到哪一个元素上。

用来附加全局函数的命令为：ajaxStart()、ajaxSend()、ajaxSuccess()、ajaxError()、ajaxComplete()和ajaxStop()。因为用来附加这些函数类型的任何一个命令的语法都是一致的，所以在一个语法描述框内一起加以说明。

253

命令语法：Ajax全局函数
ajaxStart(callback) **ajaxSend(callback)** **ajaxSuccess(callback)**

ajaxError(callback)
ajaxComplete(callback)
ajaxStop(callback)

把传入的回调函数附加到所有匹配元素上，一旦到达Ajax请求处理的指定时刻就触发回调函数。

参数

callback （函数）将被附加的回调函数。请参照表8-3，了解何时回调函数被触发以及什么参数将被传递。

返回

包装集

这些全局回调函数的每一个都在Ajax请求的处理期间的特定时刻被触发，或者根据响应状态有条件地触发，假定全局函数已经为Ajax请求而被启用。表8-3描述每一个全局回调函数类型何时被触发以及什么参数被传递给全局回调函数。

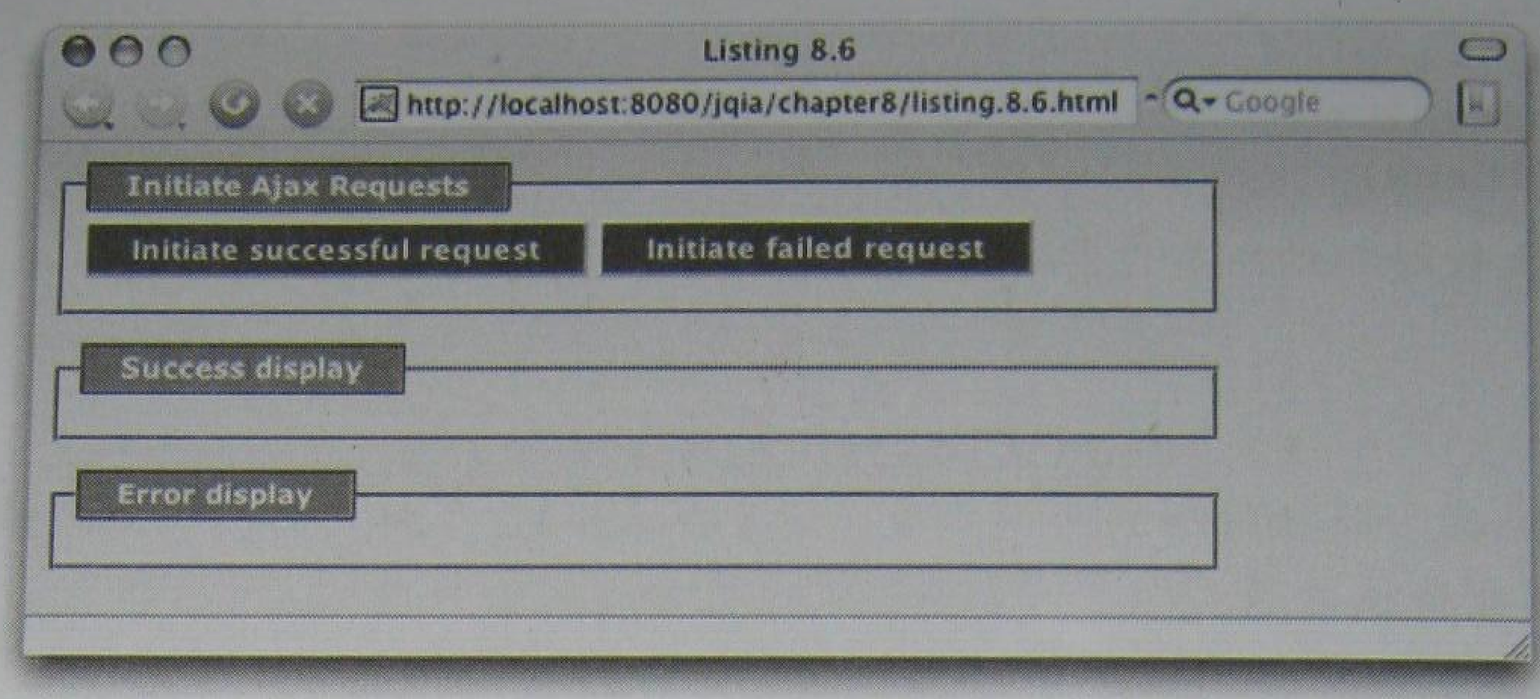
表8-3 Ajax全局回调函数（按触发顺序排列）

全局函数类型	何时被触发	参 数
ajaxStart	在jQuery Ajax函数或命令发起时，但在XHR实例被创建之前	类型被设置为ajaxStart的全局回调信息对象
ajaxSend	在XHR实例被创建之后，但在XHR实例被发送给服务器之前	类型被设置为ajaxSend的全局回调信息对象；XHR实例；$.ajax()函数使用的属性
ajaxSuccess	在请求已从服务器返回之后，并且响应包含成功状态码	类型被设置为ajaxSuccess的全局回调信息对象；XHR实例；$.ajax()函数使用的属性
ajaxError	在请求已从服务器返回之后，并且响应包含失败状态码	类型被设置为ajaxError的全局回调信息对象；XHR实例；$.ajax()函数使用的属性；被XHR实例返回的异常对象（如果有的话）
ajaxComplete	在请求已从服务器返回之后，并且在任何已声明的ajaxSuccess或ajaxError回调函数已被调用之后	类型被设置为ajaxComplete的全局回调信息对象；XHR实例；$.ajax()函数使用的属性
ajaxStop	在所有其他Ajax处理完成以及任何其他适用的全局回调函数已被调用之后	类型被设置为ajaxStop的全局回调信息对象

254

我们组建一个简单示例，看如何利用这些命令轻松地报告Ajax请求的成功或失败。测试页面（太简单了而不能称为实验室页面）的布局如图8-7所示，文件可以从chapter8/listing.8.6.html得到。

我们在这个页面上定义了3个字段集：一个包含发起Ajax请求的按钮，一个包含成功消息区，还有一个包含错误消息区。设置这个结构的HTML标记相应地也比较简单。



255

图8-7 页面（用来检查Ajax全局回调函数的用途）的初始布局

```
<body>
  <fieldset>
    <legend>Initiate Ajax Requests</legend>
    <div>
      <button type="button" id="goodButton">
        Initiate successful request
      </button>
      <button type="button" id="badButton">
        Initiate failed request
      </button>
    </div>
  </fieldset>

  <fieldset>
    <legend>Success display</legend>
    <div id="successDisplay"></div>
  </fieldset>

  <fieldset>
    <legend>Error display</legend>
    <div id="errorDisplay"></div>
  </fieldset>
</body>
```

这个页面的就绪处理程序有3个任务：

(1) 建立两个按钮的click处理程序；

(2) 建立一个全局函数作为附加到成功区的成功监听器；

(3) 建立一个全局函数作为附加到错误区的失败监听器。

建立两个按钮的click处理程序是简单易行的。

```
$('#goodButton').click(function(){
  $.get('reflectData.jsp');
});
$('#badButton').click(function(){
  $.get('returnError.jsp');
});
```

建立“好按钮”，用来向一个资源发起Ajax请求，该资源将返回成功状态；而“坏按钮”用

来向另一个资源发起Ajax请求，该资源将返回错误状态。

下面用ajaxSuccess()命令来建立成功监听器，把这个监听器附加到id为successDisplay的 元素上，如下所示：

```
$('#successDisplay').ajaxSuccess(function(info){
  $(info.target)
    .append('<div>Success at '+new Date()+'</div>');
});
```

256

这就建立一个函数，在Ajax请求成功地完成时这个函数将被调用。这个回调函数被传递全局回调信息实例，其目标属性标识已绑定元素（在这种情况下，目标是successDisplay）。我们用已绑定元素的引用来构造成功消息并使它在成功区显示。

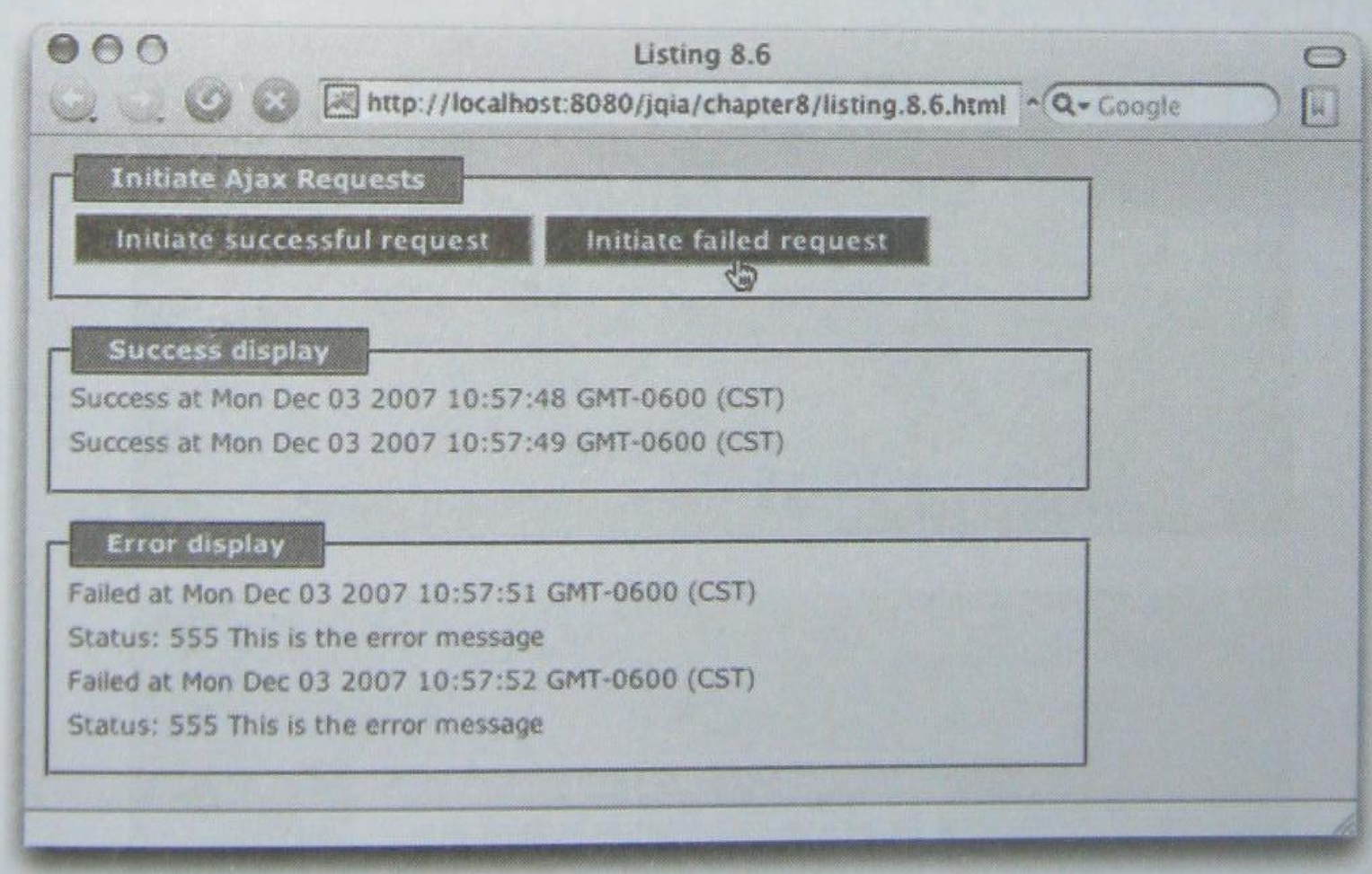
失败回调函数的绑定与此类似。

```
$('#errorDisplay').ajaxError(function(info,xhr){
  $(info.target)
    .append('<div>Failed at '+new Date()+'</div>')
    .append('<div>Status: ' + xhr.status + ' ' +
            xhr.statusText+'</div>');
});
```

利用ajaxError()命令，绑定在Ajax请求失败时将被触发的回调函数。在这种情况下，XHR实例被传递给回调函数，我们用XHR实例来告诉用户有关错误性质的信息。

因为每个函数只绑定了一个全局函数类型，所以不用全局回调信息实例的type字段。不过请注意这两个函数执行的处理有多么相似？我们怎么才能把这两个函数合并为利用type字段的一个函数使操作更加高效？

用浏览器来加载页面，每个按钮分别点击几次（消息里有时间戳，因此很容易看清楚消息显示的顺序）。最终你可能得到的显示与图8-8相似。



257

图8-8 点击按钮，显示回调函数如何获得信息以及如何知道绑定到哪个元素上

在进入下一章之前，我们把所有这些极其重要的知识用于实践，好吗？

8.5　整合一切

该到另一个综合示例了。我们把迄今为止所学知识的一部分运用起来：选择器、DOM操作、高级JavaScript、事件、效果和Ajax。然后在这之上，我们实现另外一个jQuery插件！

为了这个示例，我们再一次回到Boot Closet页面。我们准备继续增强这个页面，因此为了复习，请回顾图8-2、图8-3、图8-5和图8-6。

在列举出来的可供出售的靴子详细信息中（在图8-3显而易见），使用的术语可能不为客户所熟悉，比如Goodyear welt和stitch-down construction。我们想让客户轻松地了解这些术语的含义，因为了解情况的客户往往是快乐的客户，而快乐的客户乐于花钱购物！

如果是在1998年，我们可能只是提供术语表页面，用户可以导航到术语表页面以便参考，但那样做会使焦点从页面上移走（用户可以在那些页面上购买商品）。更为现代的做法就是打开弹出窗口显示术语表，甚至显示用户正想了解的术语定义，虽然那样还是有点过时。

如果你进一步思考，就应该知道当客户的鼠标光标悬停在术语上方时能否用DOM元素的`title`特性来显示包含术语定义的工具提示（tooltip，有时也称为flyout）。好主意！这样使术语定义在恰当的地方显示而不必让客户不由自主地把焦点移到别处。

可是利用`title`特性这个办法会带来一些问题。首先，只有鼠标光标在元素上悬停几秒之后，才会出现工具提示。我们想要更为明显的办法，在点击术语之后立即显示信息。但更为要紧的是，某些浏览器会截断`title`工具提示的文本，限定为远远达不到目标的长度。

那么我们建立自定义的工具提示！

我们设法标识拥有定义的术语、改变术语的外观以便用户能够轻松地识别这样的术语，设置这些术语以便在鼠标点击之后显示包含术语说明的工具提示。随后点击工具提示使它从显示器上删除。

图8-9所示为页面的一部分，显示我们希望添加的行为。

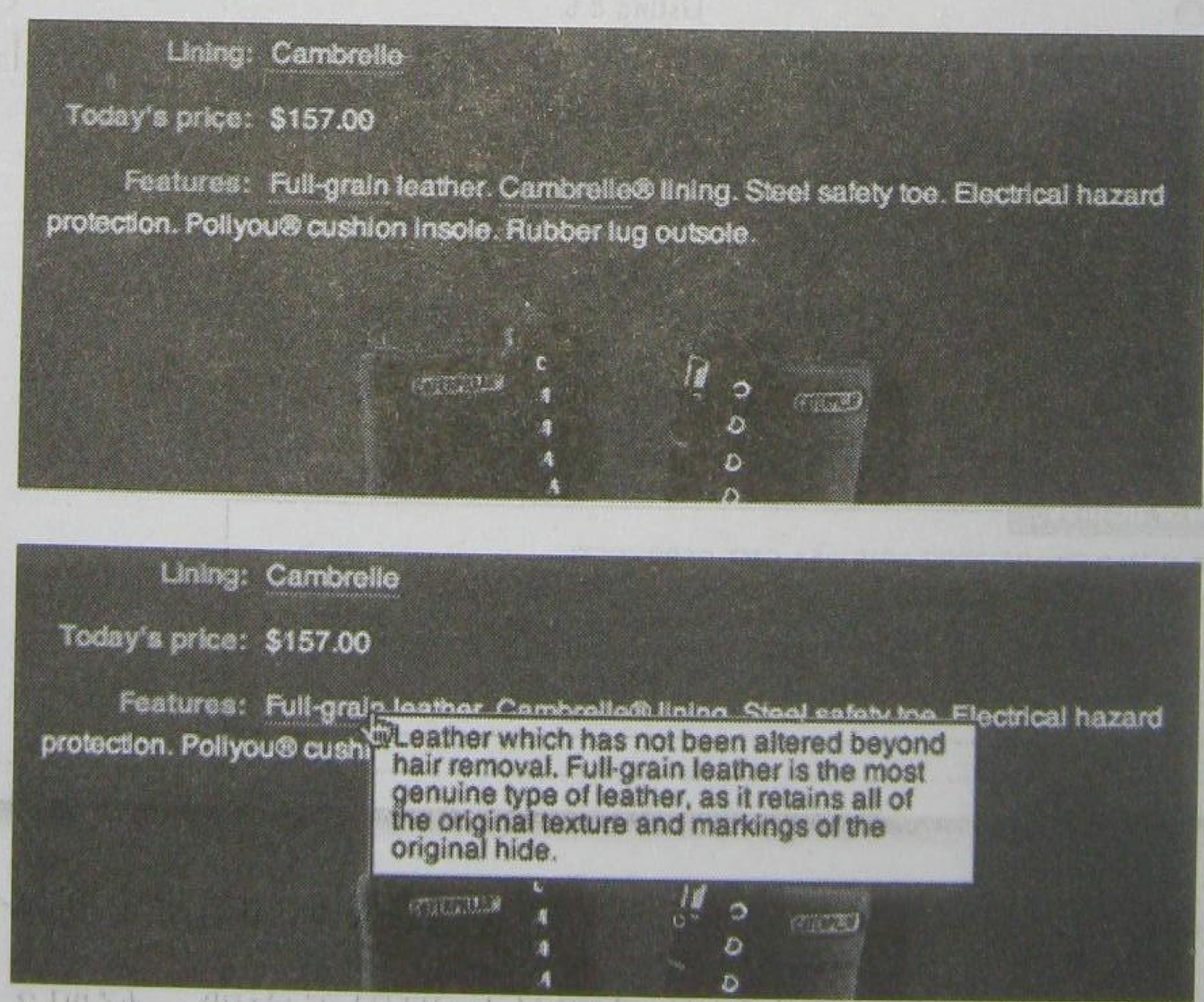


图8-9　添加工具提示之前和之后的页面截图

在上一半幅图里我们看到产品的特征说明，其中的术语Full-grain和Cambrelle被突出显示。点击Full-grain使包含术语定义的工具提示显示，如下一半幅图所示。

本来可以通过硬编码在页面上实现这个功能，不过我们更为明智一些。像对待操作<select>元素的扩展那样，我们想要创建可重用组件，可以用在这个网站或其他网站的任何地方。所以，作为jQuery的精通者，我们再一次将它实现为一个jQuery命令。

8.5.1 实现工具提示行为

回忆一下，添加jQuery命令是利用$.fn属性来完成的。我们把查找术语定义的新插件称为Termifier（术语查找器），并且命名为termifier()。

termifier()命令负责设置匹配集的各个元素以便实现下列目标。

- 在每一个匹配元素上建立用于发起Termifier工具提示显示的click处理程序。
- 一旦点击，当前元素所定义的术语将利用服务器端资源进行查找。
- 一旦接收，术语的定义将利用淡入效果在工具提示里进行显示。
- 工具提示将被设置为一旦在其范围内点击就会淡出。
- 服务器端资源的URL以及指派给工具提示元素的的CSS类，可以由页面作者指派，如果不指定则有合理的默认值。

用来创建termifier()命令以便实现以上目标的代码如代码清单8-8所示，文件可以在chapter8/bootcloset/jquery.jqia.termifier.js找到。

代码清单8-8 termifier()命令的实现

```
(function($) {
 $.fn.termifier = function(options) {
  options = $.extend({
    lookupResource: 'getTerm',
    flyoutClass: 'lookerUpperFlyout'
  },options||{});
  this.attr('title','Click me for my definition!');
  return this.click(function(event){
    $.ajax({
      url: options.lookupResource,
      type: 'GET',
      data: {term: this.innerHTML},
      dataType: 'html',
      success: function(data) {
        $('<div></div>')
          .css({
            position: 'absolute',
            left: event.pageX,
            top: event.pageY,
            cursor: 'pointer',
            display: 'none'
          })
          .html(data)
          .addClass(options.flyoutClass)
          .click(function(){
```

❶ 定义命令的API

❷ 把传入的选项与默认值进行合并

❸ 在术语上建立click处理程序

❹ 为获取术语定义而发起请求

❺ 操作响应数据

❻ 在工具提示上建立click处理程序

259

8

```
                $(this).fadeOut(1500,function(){$(this).remove();});
              })
              .appendTo('body')
              .fadeIn();
            }
          });
          return false;
        });
      }
    })(jQuery);
```

❼ 把工具提示附加到DOM上并使其淡入

260

代码可能没有预料的那样多，但是这里进行着大量的事情！下面逐一进行剖析。

首先，我们利用在第7章所学的模式来建立termifier()命令的API❶。只要求一个对象参数，对象的各个属性作为各个选项。为了更为友好，我们提供一组默认值，可以利用$.extend()实用工具函数，把这组默认值与传入的选项进行合并❷。已定义的选项如下：

- lookupResource——指定要用的服务器端资源的URL；
- flyoutClass——应用到新建的工具提示元素上的CSS类名称。

作为对客户有用的提示，我们给目标元素添加title特性以便客户把鼠标光标悬停于突出显示的术语上方时，能够看到术语定义的消息。这让客户知道点击术语就会发生极好的事情。

我们在匹配集的各个元素上建立click处理程序❸。请记住：jQuery命令的函数上下文(this)是匹配集，因此，要把其他jQuery命令应用到匹配集上，只要用this来调用命令就行了。

在click事件的监听器里，我们发起Ajax调用以便取回术语定义。为了使控制最大化，我们利用$.ajax()函数❹并给它传递用于定义下列选项的一个对象：

- 命令的选项所指定的URL（默认值或页面作者所提供的URL）；
- GET类型的HTTP方法（显然由于请求是幂等的）；
- 名为term的请求参数，将被设置为事件目标（在监听器内的函数上下文）的内容；
- 把预期的响应数据的格式标识为HTML；
- success回调函数❺，利用响应数据来创建工具提示。

在Ajax请求的success回调函数里会发生许多更为有趣的事情。首先，新建空的 元素，然后在元素上执行下列操作（再一次利用神奇的jQuery链）。

- 给 元素添加CSS样式以便使 元素绝对定位于鼠标点击事件的那一点、修改鼠标光标为手形，并且使元素从视图上隐藏。
- 包含术语定义的响应数据作为第一个参数传递给success回调函数，作为内容而插入到 元素中。
- flyoutClass选项所标识的CSS类被添加到 元素。
- 在工具提示 上建立click处理程序❻。一旦点击，就使工具提示 慢慢淡出并且在淡出效果结束时从DOM树上删除工具提示 。
- 把新建的工具提示 追加到<body>元素，从而添加到DOM树上。
- 最后，利用默认速度的淡入效果来显示工具提示 ❼。

261

termifier()命令的实现确保返回包装集作为命令的结果（通过返回click()命令所返回的包装集），因此termifier()命令能够参与任何jQuery命令链。

下面看如何把这个命令应用到Boot Closet页面。

8.5.2 利用术语提示器

因为我们把创建和操作Termifier（术语提示器）工具提示的所有复杂的逻辑都封装到termifier()命令中，所以在Boot Closet页面上使用这个新的jQuery命令相对比较简单。不过首先做出一些有趣的决定。

我们必须决定如何标识页面上的术语。请记住：我们必须构造元素的包装集，各元素的内容包含术语元素以供命令操作。可以利用带有特定的类名称的元素，多半类似下列标记：

```
<span class="term">Goodyear welt</span>
```

创建这些元素的包装集非常简单：$('span.term')。

但可能有人觉得标记有点冗长。取而代之，我们利用很少使用的HTML标签<abbr>。标签<abbr>被添加到HTML 4帮助在文档中标识缩写。因为<abbr>标签纯粹地被设计用来标识文档元素，所以没有一个浏览器在语义或视觉呈现方面对这个标签做过多的处理，所以我们利用<abbr>真是好极了。

注意 HTML 4[①]又定义了几个以文档为中心的标签，如<cite>、<dfn>和<acronym>。HTML 5[②]规范建议（草案）添加更多的以文档为中心的标签，其目的是为了提供语义而不是提供布局或视觉呈现指令。这些标签是：<section>、<article>和<aside>。 262

因此我们必须做的第一件事情就是修改服务器端资源，使其返回商品细节，把术语包含在<abbr>标签里（这些术语拥有术语表定义）。好，果然资源getDetails.jsp已经那样做了。因为浏览器不对<abbr>标签进行任何处理，所以除非已经查看JSP或PHP文件内容，否则我们也不会注意到这一点。这个资源返回JSON数据，示例商品的JSON数据如下所示：

```
{
  name: 'Chippewa Harness Boot',
  sku: '7141922',
  height: '13"',
  lining: 'leather',
  colors: 'Black, Crazy Horse',
  price: '$188.00',
  features: '<abbr>Full-grain</abbr> leather uppers. Leather
  ➡ lining. <abbr>Vibram</abbr> sole. <abbr>Goodyear welt</abbr>.'
}
```

请注意这样的情况：术语Full-grain、Vibram和Goodyear welt是用<abbr>标签来定义的。

下面探讨Boot Closet页面。以代码清单8-6的代码作为出发点，探讨为了使用Termifier必须把什么添加到页面上。我们必须把新命令引入页面，因此给页面的<head>节添加下列语句（在jQuery加载以后）：

---

① http://www.w3.org/TR/html4/。

② http://www.w3.org/html/wg/html5/。

```
<script type="text/javascript"
        src="jquery.jqia.termifier.js"></script>
```

在商品信息加载时，我们必须把termifier()命令应用到已添加到页面的任何<abbr>标签上。因此给load()命令添加回调函数。那个回调函数取回商品信息，用Termifier来设置所有<abbr>元素。增强之后的load()命令如下（改动部分用粗体来突出显示）：

```
$('#detailsDisplay').load(
  'getDetails.jsp',
  { style: styleValue },
  function(){
    $('abbr').termifier({
      lookupResource: 'getTerm.jsp'
    });
  }
);
```

263

已添加的回调函数创建所有<abbr>元素的包装集，并对包装集应用termifier()命令：指定服务器端资源getTerm.jsp覆盖命令的默认值。

仅此而已。

因为我们明智地把所有复杂的逻辑都封装在可重用的jQuery命令中，所以在页面上使用就简便得如同小菜一碟！而且我们可以轻松地在任何其他页面或网站上使用这个命令。现在看到工程设计的威力了吧！

最后剩下的任务是改变文本元素的外观以便让用户知道哪一些是可点击的术语。我们为<abbr>标签添加下列CSS属性到CSS文件：

```
color: aqua;
cursor: pointer;
border-bottom: 1px aqua dotted;
```

这些样式赋予术语类似链接的外观，但是有点细微的差别：利用点式下划线。这将引诱用户点击术语，还能与页面上任何其他的真正的链接区分开来。

新页面的文件可以在chapter8/bootcloset/boot.closet.3.html找到。因为我们对清单8-6的代码改动极少（正如以上所述），所以为了节省纸张在这里就不再包括整个页面清单。

带有新功能的、运行中的新页面如图8-10所示。

我们的新命令既有用又强大，不过总有……

8.5.3　改进的空间

我们的初出茅庐的jQuery新命令虽然有用，不过确实有些小问题和可以大大改进的潜力。为了磨练你的技能，下面列举可能的改进之处，你可以对termifier()命令或Boot Closet页面进行改进。

- 通过命名为term的请求参数给服务器端资源传递术语。给termifier()命令添加一个选项就能赋予页面作者指定查询参数名称的能力。客户端命令不应规定服务器端代码如何编写。
- 添加一个或多个选项，允许页面作者控制淡入淡出效果的持续时间甚至利用替代效果。

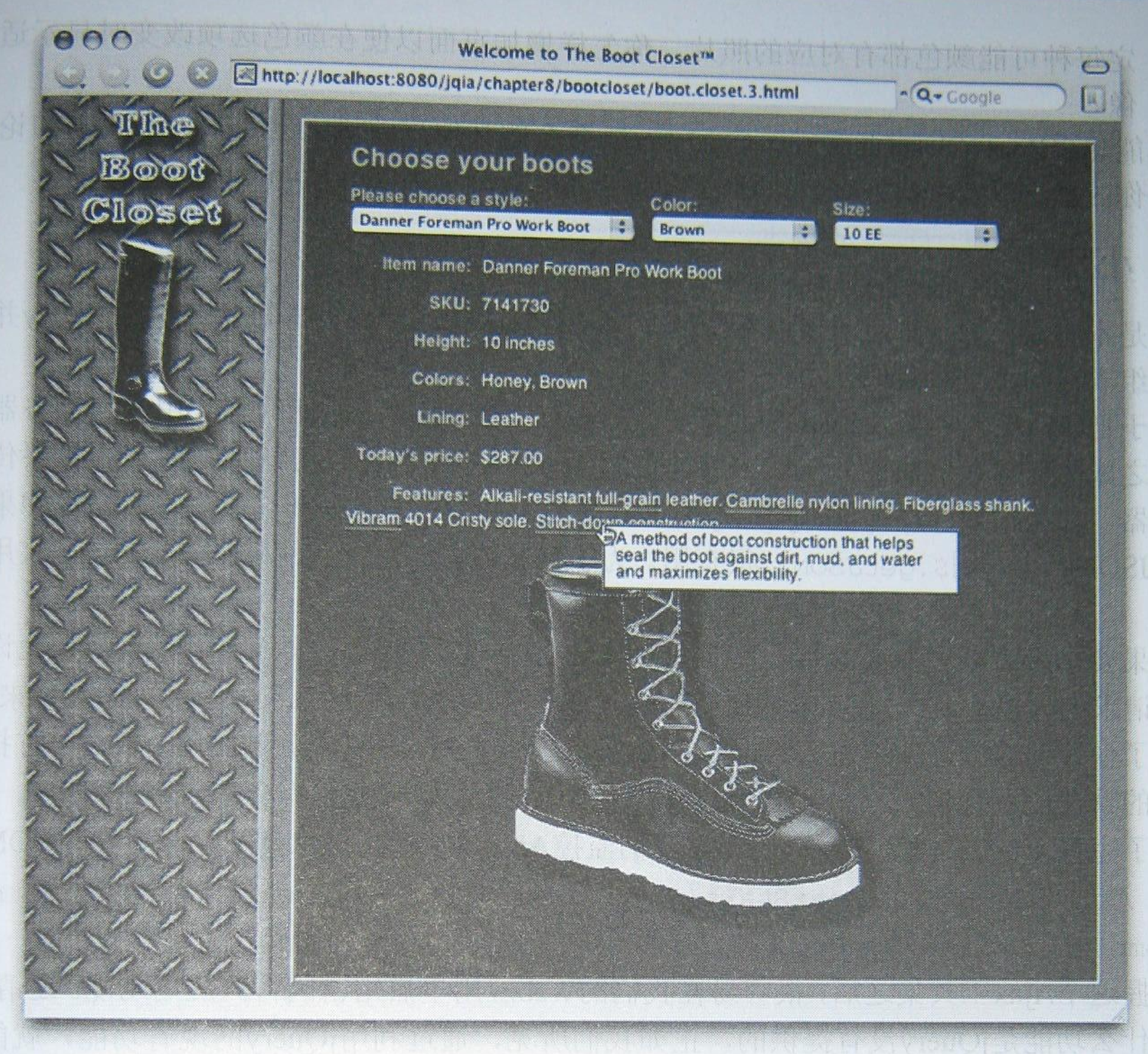


图8-10 我们的客户了解到究竟什么是外翻鞋结构

- Termifier工具提示保持不变，直到用户点击它或页面被卸载。给termifier()命令添加超时选项，以便在到超时过期之后自动地使工具提示消失。 264
- 点击工具提示使它关闭，这引入一个可用性问题，因为工具提示的文本不能选中用于剪切和复制。修改代码以便在用户点击除工具提示以外页面上的任何地方时使工具提示关闭。
- 显示多个工具提示是可能的，如果用户在点击另一个术语之前没有消除现有的工具提示，甚至在靴子的新款式被选择的时候也是如此。添加代码以便在显示新的工具提示之前以及在新款式被选中的时候，删除任何前面的工具提示。 8
- 我们在命令中不做任何错误处理。你如何增强命令以便正常地处理服务器端的错误？ 265
- 我们利用部分透明的PNG文件来实现很吸引人的图像的投影。虽然大多数浏览器对PNG文件处理得很好，但是IE6并非如此，而是显示带有白色背景的PNG文件。为了处理这个问题，可以给没有投影的图像提供GIF格式。如何增强页面以便检测出何时正在使用IE6从而以对应的GIF来替换所有的PNG引用呢？
- 说到图像，我们对于每个靴子款式只有一张照片，即使该款式有多种颜色可供挑选。假

定每种可能颜色都有对应的照片，你怎样增加页面以便在颜色选项改变时显示适当的图像？

你能想出Boot Closet页面或termifier()命令的其他可改进之处吗？请在本书的论坛和我们分享你的想法以及解决方案。论坛的网址是http://www.manning.com/bibeault。

8.6 小结

这是本书最长的一章，一点也不奇怪。Ajax是富因特网应用的关键部分，而jQuery并不懒于提供一组丰富的工具给我们使用。

对于把HTML内容加载到DOM元素里，load()命令提供了简便的方式用来从服务器获取内容并使之成为任何包装集的元素的内容。使用GET方法还是POST方法取决于是否需要传递数据到服务器。如果需要GET方法，jQuery提供实用工具函数$.get()和$.getJSON()。如果从服务器返回JSON数据，则$.getJSON()很有用。为了强制POST方法，可以使用$.post()实用工具函数。

如果要求最大限度的灵活性，则带有丰富的分类选项的$.ajax()实用工具函数允许我们控制Ajax请求的大多数方面。jQuery的所有其他Ajax功能，其实是用$.ajax()函数的服务来实现的。

为了使管理这堆选项不那么繁琐，jQuery提供$.ajaxSetup()实用工具函数，允许把任何常用选项的默认值设置到$.ajax()函数（以及使用$.ajax()服务的所有其他Ajax函数）。

为了完善Ajax工具集，jQuery也允许我们监控Ajax请求的过程以及把Ajax事件与DOM元素相关联，通过以下命令：ajaxStart()、ajaxSend()、ajaxSuccess()、ajaxError()、ajaxComplete()和ajaxStop()来完成。

266

掌握这个Ajax工具集之后，很容易在我们的Web应用里启用富因特网应用功能。还请你记住：如果有什么功能是jQuery没有提供的，正如我们所见，通过利用jQuery的现有功能，就能够轻松地扩展jQuery。或者，多半已经存在这样的插件——官方或非官方的——添加的正是你所需要的功能！

267

这就是下一章的主题……

第9章 卓越、强大和实用的插件

本章内容

- jQuery插件的概览
- 官方的表单插件
- 官方的尺寸插件
- 实时查询插件
- UI插件

在本书的前8章，我们重点讨论了核心jQuery库给页面作者所提供的功能。但那只是冰山的一角！可用jQuery插件的庞大集合令人印象深刻，让我们得以利用更多的工具（全部基于jQuery核心）来进行工作。

核心jQuery库的创建者精心地选择被绝大多数的页面作者所需要的功能并且创建框架，使得在框架之上可以平稳地建立插件。这样使核心的内存占用尽可能地小，并且由页面作者挑选十分重要的附加功能以便添加到页面上，从而决定如何花费剩下的带宽。

在一章里，甚至一本书里想要涵盖所有的jQuery插件是个无法完成的任务，因此在这里我们不得不有选择地讲述其中的一些插件。这是个艰难的决定，因此所选的插件是我们觉得对于大多数的Web应用开发者来说要么十分重要、要么十分有用，值得介绍的那些插件。

如果一个插件没有入选本章，当然并不是控诉这个插件的有用性或质量！只是我们必须做出一些艰难的抉择。

通过访问http://docs.jquery.com/Plugins或http://jquery.com/plugins/most_popular，你可以查找全部可用插件的相关信息。

我们也不能完全涵盖将要讨论的插件的方方面面，但本章应该使你对各种插件有很好的基本理解，知道何时可以应用哪些插件。请查阅各个插件的官方文档以便填补这里涵盖不到的空白。

在前面好几处都提到了某个插件，下面就从那个插件开始探讨。

9.1 表单插件

处理表单可真是个麻烦事。每一个控件类型都有各自不同的特点，并且表单提交常常走意外的途径。核心jQuery拥有许多方法帮助简化表单，但也只能为我们做那么多了。官方表单插件的目的是填补这些空白以及帮助控制表单控件。

269 表单插件可以在http://jquery.com/plugins/project/form找到，其所在的文件是jquery.form.js。它从3个方面增加表单功能：

- 获得表单控件的值；
- 使表单控件清空和复位；
- 通过Ajax提交表单（包括文件上传在内）。

下面从获得表单控件值讲起。

9.1.1　获得表单控件的值

表单插件提供两种途径来获得表单控件的值：作为值的数组，或作为序列化的字符串。表单插件提供3个方法用来获得表单控件的值：fieldValue()、formSerialize()和fieldSerialize()。

首先看如何获取字段的值。

1. 获得控件值

我们可以用fieldValue()命令来获得表单控件的值。可能看第一眼时你认为fieldValue()和val()是互相冗余的。但在jQuery 1.2之前，val()命令的功能相当弱，因而fieldValue()被设计用来弥补val()的不足。

第一个主要的差别是fieldValue()返回包装集里表单控件的全部的值所构成的数组，而val()只返回第一个元素的值（只有在第一个元素是表单控件的情况下）。事实上，fieldValue()始终返回一个数组，即使只有一个值或没有任何值可供返回。

另一个差别是fieldValue()忽略包装集里的任何非控件元素。如果创建页面上的全部元素所构成的集合，则fieldValue()只查找所有控件的值，并以数组的形式返回这些值。但是并非所有的控件都在返回的数组里有值：就像val()命令那样，在默认的情况下fieldValue()只返回被视为成功控件的值。

那么，什么是成功控件？并不是能够提供稳定的花式跑车的控件，而是HTML规范[1]里一个正式的定义：决定某个控件的值是否有意义，以及是否应该作为表单的一部分而被提交。

我们在这里不探讨详尽的细节，不过简而言之，成功控件是指那些拥有name特性、不被禁用的、已选中（对于可选中的控件如复选框和单选按钮）的控件。一些控件，比如复位控件（<input type='reset'>）和按钮控件（<button>），始终被认为是不成功的并且决不参与表单提交。其他控件，如<select>，必须有已选中的值才被视为成功控件。

270

命令fieldValue()允许我们选择是否包括不成功的值。这个命令的语法如下。

命令语法：fieldValue
fieldValue(excludeUnsuccessful) 收集包装集里所有成功表单控件的值，并且返回这些值所构成的字符串数组。如果未找到任何值，就返回一个空数组。

① http://www.w3.org/TR/REC-html40/。

参数

excludeUnsuccessful （布尔型）如果省略（或设置为true），则指定包装集里任何不成功控件被忽略。

返回

已收集的值所构成的字符串数组

我们已建立另一个方便的实验室页面用来演示fieldValue()命令的工作方式。你可以在文件chapter9/form/lab.get.values.html中找到这个页面。在浏览器中显示时，页面外观如图9-1所示。

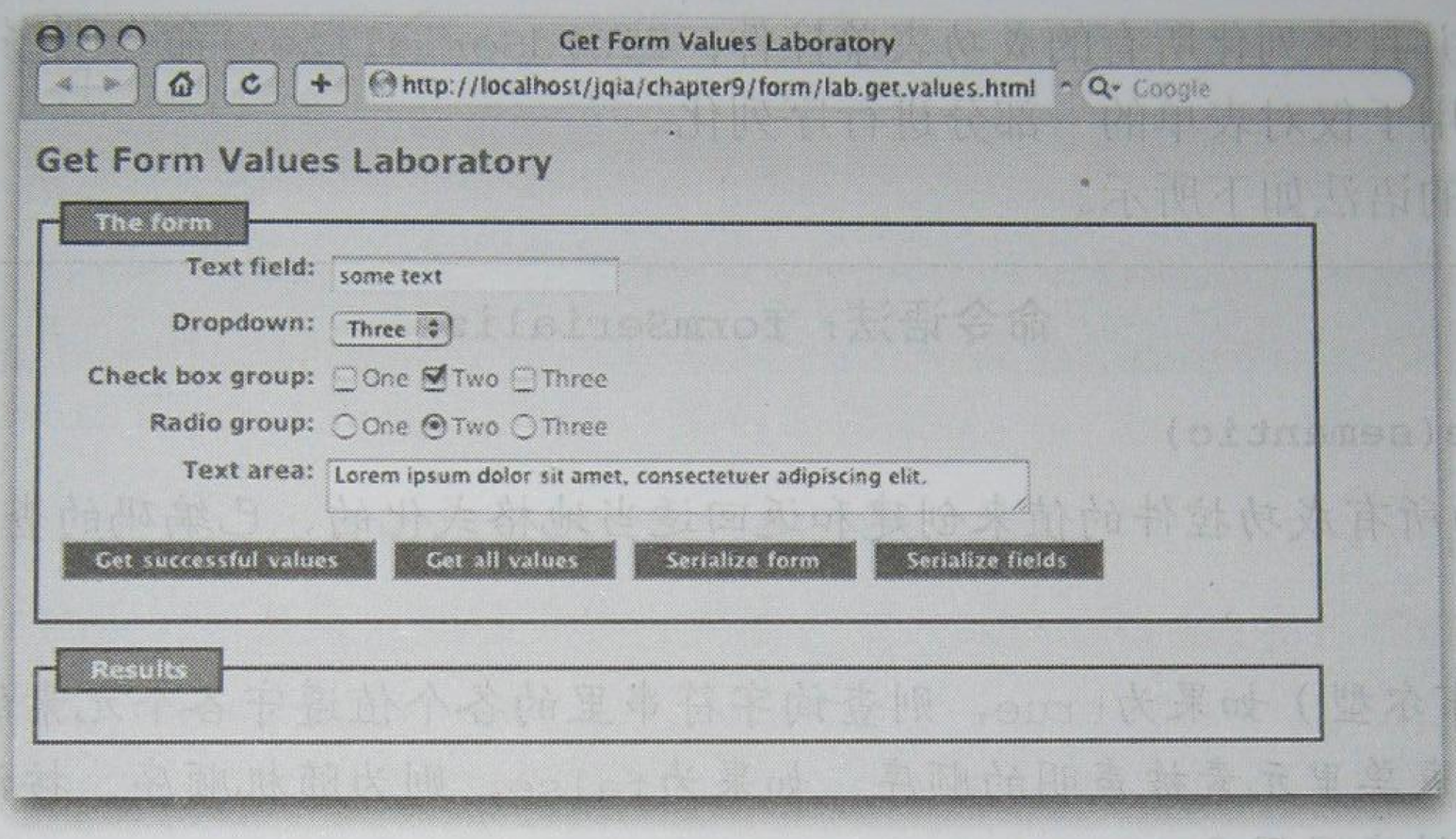


图9-1 “获得表单值实验室”页面帮助我们理解fieldValue()和serialize()命令的操作

利用浏览器加载这个页面，保持控件的初始状态，然后点击按钮Get Successful Values（获得成功值）。这将引发下列命令的执行：

```
$('#testForm *').fieldValue()
```

这将创建测试表单的所有子节点所构成的包装集，包括所有的<label>和 元素，并在包装集上执行fieldValue()命令。因为这个命令不带参数，所以忽略除成功表单控件以外的所有控件，在页面上显示的结果如下：

```
['some text','Three','cb.2','radio.2','Lorem ipsum dolor sit amet,
➡ consectetuer adipiscing elit.']
```

不出所料，文本字段、下拉列表、已选中的复选框、已选中的单选按钮以及文本区的值被收集到数组里。

现在继续进行。请点击按钮Get All Values（获得所有的值），执行下列命令：

```
$('#testForm *').fieldValue(false)
```

这个命令的参数false指示不排斥不成功控件，因此可以看到包含更多的结果如下：

```
['some text','Three','One','Two','Three','Four','Five','cb.1',
➡ 'cb.2','cb.3','radio.1','radio.2','radio.3','Lorem ipsum dolor
➡ sit amet, consectetuer adipiscing elit.','','','','']
```

请注意：不但包括未选中的复选框和单选按钮的值，还包括4个按钮的空字符串值。

接下来请尽情地使用这些控件的值并观察两种形式的fieldValue()命令的行为，直到你完全理解这个命令。

如果想要以某种方式来处理数据，则获得控件值的数组是很有用的；如果想要根据表单数据创建查询字符串，则序列化命令将完成这个任务。下面请看详情。

2. 序列化控件值

如果我们想要从表单控件的值来构造适当地格式化的、已编码的查询字符串，则应求助于formSerialize()和fieldSerialize()命令。这两个包装器方法都从包装集里收集值并且返回已将名称和值适当地进行URL编码的、格式化的查询字符串：formSerialize()方法接受包装集里的一个表单，并且序列化所有的成功表单控件；fieldSerialize()命令则序列化包装集里的所有控件，并可用于仅对表单的一部分进行序列化。

72　这两个命令的语法如下所示：

命令语法：formSerialize

formSerialize(semantic)

从已包装表单的所有成功控件的值来创建和返回适当地格式化的、已编码的查询字符串。

参数

semantic　（布尔型）如果为true，则查询字符串里的各个值遵守各个元素的语义顺序，即在表单里元素被声明的顺序。如果为false，则为随机顺序。按语义排序将比随机排序慢得多。

返回

生成的查询字符串

命令语法：fieldSerialize

fieldSerialize(excludeUnsuccessful)

从已包装表单的控件的值来创建和返回适当地格式化的、已编码的查询字符串。

参数

excludeUnsuccessful　（布尔型）如果省略（或设置为true），则指定包装集的任何不成功控件被忽略。

返回

生成的查询字符串

formSerialize()的semantic参数应该受到特别注意。如果指定为true，序列化值的排序将与通过常规方式提交表单时的顺序相一致，使这些值的提交准确地模拟浏览器提交。只有在绝对必要时（通常没有必要）才指定为true，因为这样做要付出性能上的代价。

警告 semantic标志将使提交数据里参数的顺序被指定为语义顺序，但服务器端的代码如何处理这个语义顺序是不受客户端代码所控制的。例如，如果使用servlet，则请求实例调用getParameterMap()方法时，不会维持提交时的顺序。

我们可以用获取表单值实验室页面来观察这些命令的行为。加载页面，保持控件的初始状态，然后点击按钮Serialize Form（序列化表单）。这将在测试表单上执行formSerialize()命令如下：

```
$('#testForm').formSerialize()
```

273

这个语句导致的结果是：

```
text=some%20text&dropdown=Three&cb=cb.2&radio=radio.2&
➡ textarea=Lorem%20ipsum%20dolor%20sit%20amet%2C%20conse
➡ ctetuer%20adipiscing%20elit.
```

请注意：所有成功表单控件的名称和值被收集，并且利用这些数据所创建的查询字符串已被URL编码。

点击按钮Serialize Fields（序列化字段），执行下列命令：

```
$('#testForm input').fieldSerialize()
```

这个选择器所创建的包装集只包括表单控件的子集：input类型的那些控件。结果查询字符串（只包括成功的已包装控件元素）如下：

```
text=some%20text&cb=cb.2&radio=radio.2
```

也许我们想要序列化表单控件为查询字符串的理由，是为了用作Ajax请求的提交数据。等一等！如果想要通过Ajax而不是正规过程来提交表单，可以借助于表单插件的更多功能。不过在学习那些知识之前，首先检查几个允许我们操作表单控件值的命令。

9.1.2 清除和复位表单控件

表单插件提供两个命令以便影响表单控件的值。clearForm()命令使包装表单的所有字段清空，而resetForm()命令使字段复位。

“唔，区别何在？”你问。

在调用clearForm()来清空表单控件时，表单控件受到的影响如下：

- 文本、口令以及文本区控件被设置为空值；
- <selec>元素被取消选择；
- 复选框和单选按钮被取消选中。

如果调用resetForm()来使控件复位，则表单的原生reset()方法被调用。这将使控件的值还原为原始HTML标记所指定的值。控件（如文本字段）还原为其value特性所指定的值，而其他控件类型则还原为checked或selected特性所指定的设置。

我们再一次建立实验室页面来演示这个差别。找到文件chapter9/form/lab.reset.and.clear.html，并在浏览器里进行显示。你应该看到页面如图9-2所示。

274

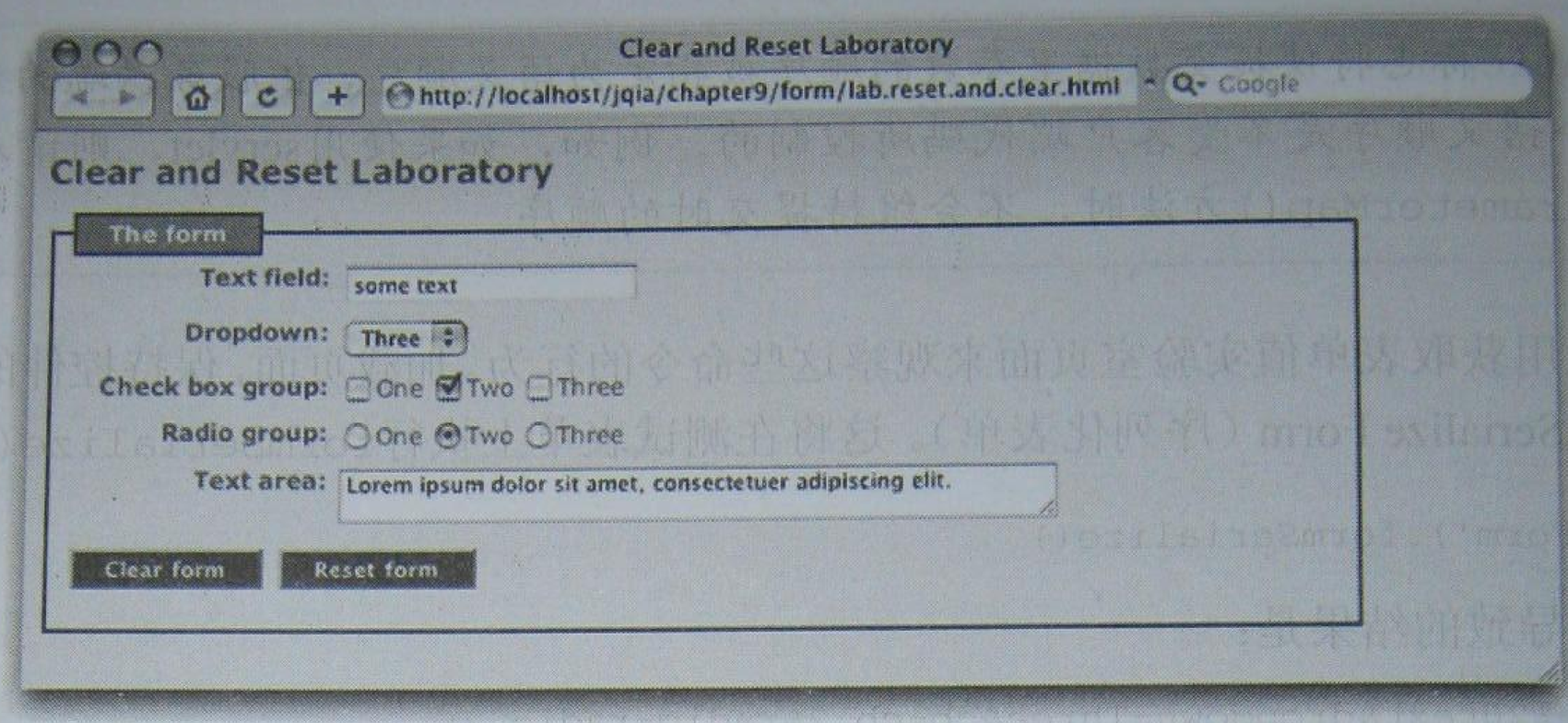


图9-2 “清除和复位实验室”页面显示“清除操作”与“复位操作”之间的差别

请注意：这个熟悉的表单已经通过其HTML标记的值进行初始化：文本字段和文本区已经通过其value特性进行初始化，而下拉列表的某个选项在初始化时被选择，一个复选框和一个单选按钮也在初始化时被选中。

点击按钮Clear Form（清空表单），并观察发生什么情况。文本字段和文本区被清空，下拉列表中没有任何选项被选择，而所有的复选框和单选按钮都被取消选中。

接着点击按钮Reset Form（复位表单），请注意各个控件如何还原为初始值。改变各个控件的值，并点击按钮Reset Form，请注意各个控件如何再一次还原为初始值。

这些命令的语法如下。

命令语法：clearForm

clearForm()

清除包装集里任何控件的值，或包装集元素的后代节点。

参数

无

返回

包装集

命令语法：resetForm

resetForm()

调用包装集里表单元素的原生reset()方法。

参数

无

返回

包装集

下面看表单插件如何帮助我们通过Ajax请求来提交表单。

9.1.3 通过 Ajax 提交表单

早在第8章里我们就看到jQuery使得发起Ajax请求那么简便，不过表单插件使得事情更加简便。我们可以利用9.1.1节所介绍的序列化命令。等一等，还有更多！表单插件使得拦截表单请求更为简便。

表单插件引入两个经由Ajax提交表单的新命令：一个在脚本控制下发起Ajax请求（传递目标表单的数据作为请求的参数），另一个则设置任何表单使表单提交重路由为Ajax请求。

这两个方法都使用jQuery的Ajax核心函数去执行Ajax请求，因此所有的全局jQuery挂钩继续被应用，即便是在利用这些方法来替换核心jQuery的Ajax API的时候。

下面开始研究第一个方法。

1. 获取表单数据用于Ajax请求

在开发第8章的电子商务示例时，我们遇到了许多情况，在那里我们需要从表单控件获取数据以便经由Ajax请求而发送到服务器上。这是真实世界的常见需求。我们看到核心Ajax函数使这成为小事一桩，尤其是只需获取少数表单值的时候。

表单插件的`serializeForm()`方法与核心Ajax函数相结合，使得提交表单的所有控件更为简便。比那更加简便的是，表单插件利用`ajaxSubmit()`命令使得经由Ajax提交整个表单非常简单。

`ajaxSubmit()`命令，如果应用到包含表单的包装集，就获取目标表单的所有成功控件的名称和值并且作为Ajax请求进行提交。可以给这个方法提供关于如何生成请求的信息，或让其默认地根据目标表单上的设置而生成请求。 276

请看这个命令的语法。

命令语法：ajaxSubmit

ajaxSubmit(options)

利用包装集里表单的成功控件来生成Ajax请求。参数options可以用来指定可选的设置，或者这些设置可以默认为下表所描述的值。

参数

options （对象|函数）一个散列对象（可选），包含如表9-1所描述的属性；如果想要设置的选项只是成功回调函数，可以直接传递成功回调函数而无需传递散列对象。

返回

包装集

参数options可以用来精确地指定如何生成请求。可选的属性如表9-1所示。所有的属性都有默认值，这样设计是为了易于生成请求并且尽量减少麻烦。通常不带参数地调用这个方法而让它应用所有的默认值。

表9-1　**ajaxSubmit()**命令的可选的属性，根据使用的可能性大小进行排序

名　称	描　述
url	（字符串）Ajax请求将要提交到该URL。如果省略，URL将从目标表单的action特性来获取
type	（字符串）在提交请求时使用的HTTP方法，如GET或POST。如果省略，则使用目标表单的method特性所指定的值。如果不指定type并且不指定表单的method特性，则使用GET方法
dataType	（字符串）响应的预期数据类型，决定如何对响应体进行后续处理。如果指定，则必须是下列的类型之一 xml——作为XML数据进行处理。任何成功回调函数将被传递responseXML文档 json——作为JSON构造进行处理。对JSON求值，其结果将传递给任何成功回调函数 script——作为JavaScript进行处理。脚本将在全局上下文中进行求值 如果省略，则不对数据进行后续处理（除非其他选项，如target已指定）
target	（字符串｜对象｜元素）指定一个或多个DOM元素，这个（些）元素接收响应体作为内容。这个属性可以是字符串，描述jQuery选择器；可以是jQuery包装器，包含目标元素；或者直接是一个元素引用。如果省略，则没有任何元素接收响应体
beforeSubmit	（函数）指定在发起Ajax请求之前调用的回调函数。这个回调函数可以用来执行任何预处理操作，包括表单数据的验证在内。如果这个回调函数返回false值，则表单提交被取消 下面是传递给这个回调函数的3个参数 数据值的数组，作为参数传递给请求。这是对象数组，每个对象都包含两个属性name和value（分别包含请求参数的名称和值） 已应用ajaxSubmit()命令的jQuery匹配集 已传递给ajaxSubmit()命令的options对象 如果省略，则不调用任何预处理回调函数
success	（函数）指定回调函数，在请求已经完成并且返回带着成功状态的响应之后调用。传递给这个回调函数的3个参数如下 根据dataType选项进行解释（处理）之后的响应体 包含success的字符串 已应用ajaxSubmit()命令的jQuery匹配集 如果省略，则不调用任何成功回调函数；如果这是将要指定的唯一选项，则这个函数可以替换options散列对象而直接传递给命令。请注意：关于在错误情况下调用的回调函数，尚无任何规定
clearForm	（布尔型）如果指定为true，则在成功提交之后清空表单。参见clearForm()的语义说明
resetForm	（布尔型）如果指定为true，则在成功提交之后使表单复位。参见resetForm()的语义说明
semantic	（布尔型）如果指定为true，则表单参数按照语义顺序来排列。这将造成的唯一差别在于为图像类型的input元素而提交的参数的位置（当通过点击那个元素来提交表单时）。因为存在与这种处理相关的开销，所以只有在参数顺序对于服务器端的处理非常重要并且图像类型的input元素被用于表单时，才应该启用semantic选项
其他选项	可以指定对核心jQuery的$.ajax()函数来说可用的任何选项（如表8-2所描述），这些选项将通过较低级别的调用[1]

尽管选项的数量很多，但是调用ajaxSubmit()通常是相当简单的。如果我们只需提交表单到服务器（并且在请求完成时没有什么要做的事情），调用起来就非常简单：

```
$('#targetForm').ajaxSubmit();
```

① 这里的“较低级别的调用”是指$.ajax()的调用。——译者注

如果我们想要把响应加载到一个或多个目标元素：

```
$('#targetForm').ajaxSubmit( { target: '.target' } );
```

如果我们想要在回调函数里对响应进行处理：

```
$('#targetForm').ajaxSubmit(function(response){
  /* do something with the response */
});
```

如此等等。因为所有的选项都有合理的默认值，所以只需指定所需的那些信息就能调整表单提交，使它符合我们的要求。

警告 因为options散列对象被传递给beforeSubmit回调函数，你可能想试着修改这个回调函数。慎重！这时修改beforeSubmit回调函数显然太晚了，因为已经执行，不过你可以添加或修改其他的简单设置，如resetForm或clearForm。对于任何其他修改也需要谨慎，因为可能导致操作出错。请注意：你不可以添加或修改semantic属性，因为在beforeSubmit回调函数被调用时semantic属性的工作已经完成。

如果你想知道是否为这个命令建立了实验室页面，不用再想了！在浏览器里加载页面chapter9/form/lab.ajaxSubmit.html，就会看到如图9-3所示的画面。

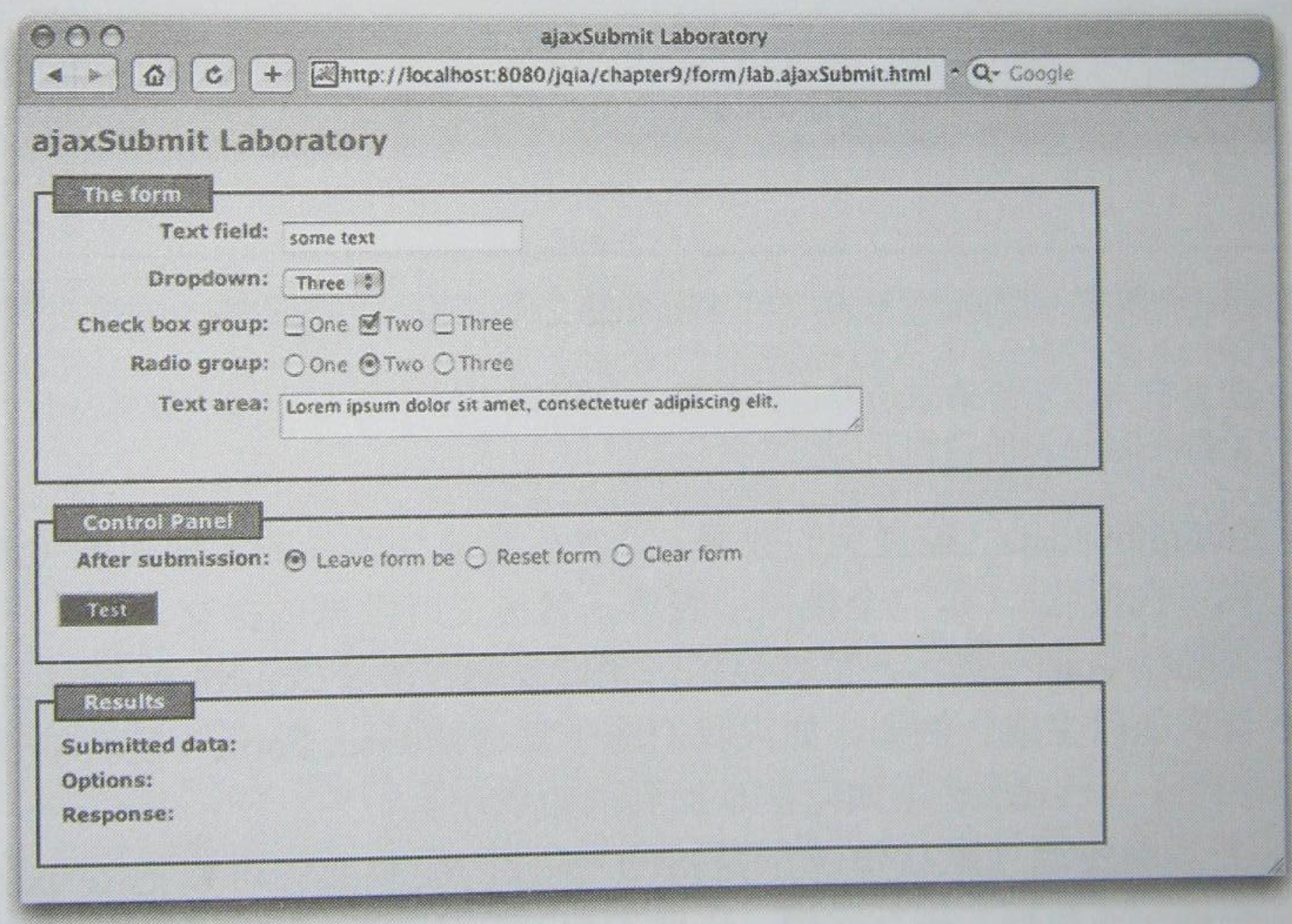


图9-3 ajaxSubmit实验室页面允许我们试验ajaxSubmit()方法的工作方式

注意 因为我们准备向服务器提交请求，所以你必须在活动的Web服务器（如同8.2节里为第8章的示例所描述）上运行这个页面。

这个实验室页面提供已经熟悉的表单供我们利用ajaxSubmit()命令进行操作。最上面的窗格包含表单；中间窗格包含控制面板，允许给这个命令调用添加resetForm或clearForm选项；结果窗格将在命令被调用时显示3项重要的信息：已提交给请求的参数数据、已传递给命令的选项散列对象以及响应体。

如果你留心检查实验室页面的代码，就会注意到前两项信息由beforeSubmit回调函数显示，而第3项信息由target选项显示（为了简洁，beforeSubmit函数不作为选项的一部分进行显示）。

279

在点击按钮Test时，通过应用到包装集（包含第一个窗格的表单）的ajaxSubmit()命令来发起请求。请求的URL默认为那个表单的action特性值reflectData.jsp，这个JSP对HTML响应（描述传递给请求的参数）进行格式化。

使所有的控件保持加载时的初始状态，点击按钮Test。你将会看到如图9-4所示的结果。

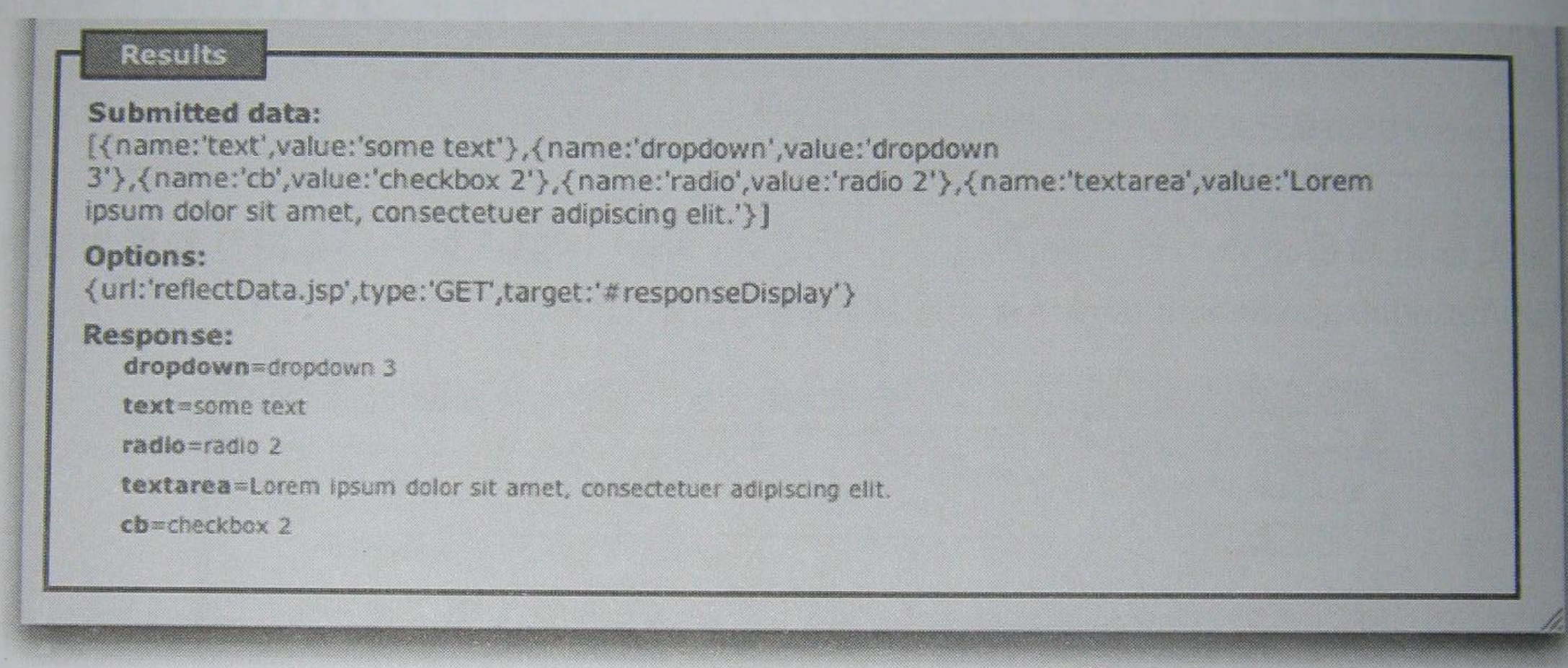


图9-4　结果窗格显示给请求所发送的数据、用于调用命令的选项和响应体（反映传递给服务器资源的数据）

不出所料，Submitted data（被提交的数据）反映所有成功控件的名称和值（请注意：不包括未选中的复选框和单选按钮）。这完全模拟了表单正常提交时数据的提交。

用来生成请求的Options（选项）也被显示，允许我们在改变Control Panel（控制面板）的选项的同时，观察请求如何生成。例如，如果我们选中复选框Reset Form（使表单复位）并点击按钮Test，我们将看到resetForm选项如何被添加到函数调用。

280

最后显示服务器端资源（默认情况下是JSP）所检测的参数。可以把Response（响应）与Submitted data（被提交的数据）进行比较，以便确信它们始终在谈话[1]。

在实验室页面中运行各种不同的脚本，按照你的即兴想法改变表单数据和选项，并观察结果。这样你就会对ajaxSubmit()方法如何操作有很好的理解。

在本节中，假定想要在脚本控制下利用表单数据来发起请求。我们想要在发生除了正常语义

① 指“响应”与“被提交的数据”密切相关。——译者注

提交事件以外的事件时这样做：可能在点击除submit按钮以外的按钮（正如实验室页面）时，或在鼠标事件发生时（比如在第8章的示例中我们用鼠标事件来调用Termifier请求）。但有时，或许非常频繁，请求的提交将会是正常语义提交事件的结果。

下面看表单插件如何帮助我们进行那种设置。

2. 拦截表单提交

如果想要作为表单提交事件以外事件的结果而在脚本控制下发起请求，则ajaxSubmit()方法正好派上用场。不过，我们常常想要进行常规的表单提交并且拦截它（把表单提交作为Ajax请求发送到服务器，而不是像通常那样，在提交表单之后引起整个页面的刷新）。

281

我们可以利用事件处理和ajaxSubmit()命令的知识来使提交重路由。果然无需那样做，因为表单插件预料到这个需求并且提供ajaxForm()方法。

这个方法会设置表单，当表单通过普通语义事件提交时阻止通过正常语义事件（比如点击提交按钮，或在表单拥有焦点时按下回车键）来提交表单，并发起模拟普通Web请求的Ajax请求。

ajaxForm()在其表面之下使用ajaxSubmit()，因此它们语法相似毫不令人稀奇。

命令语法：ajaxForm

ajaxForm(options)

设置目标表单以便在触发表单提交时，把提交重路由为ajaxSubmit()命令所发起的Ajax请求。传递给这个方法的参数options被传递给ajaxSubmit()调用。

参数

options （对象 | 函数）一个散列对象（可选），包含如表9-1所描述的属性；如果想要的选项只是成功回调函数，可以传递成功回调函数而不必传递散列对象。

返回

包装集

一般来说，在就绪处理程序里把ajaxForm()命令应用到表单上，然后可以放心地让命令替我们设置目标表单而使表单提交重路由。

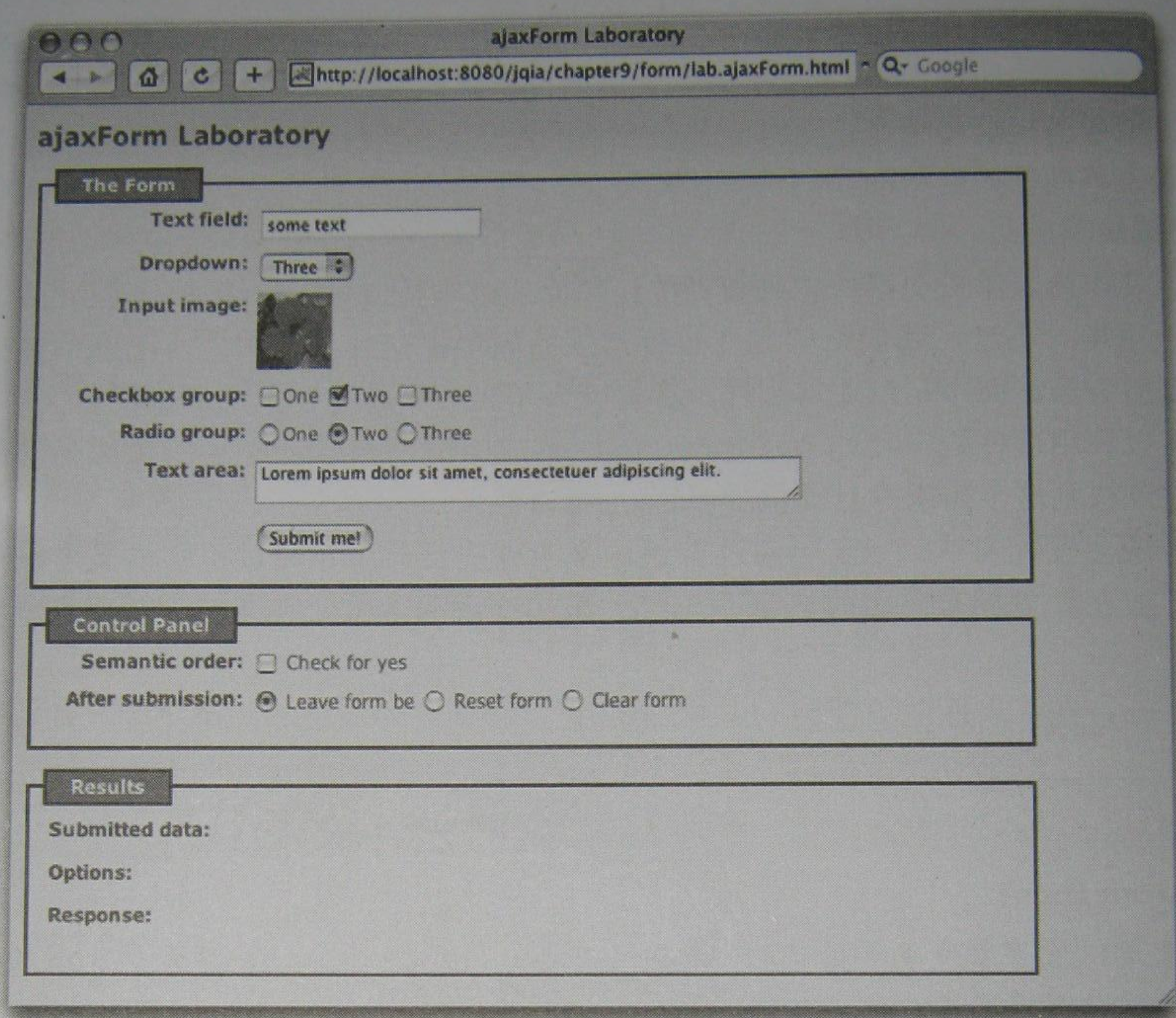
声明HTML表单的标记（仿佛表单将要正常提交）并且让ajaxForm()从表单的声明中获得这些值，这确实是可行的惯例。有时用户已经禁止JavaScript，这时表单适当地降级为正常提交，而我们不必做任何特别的事情。多么方便！

如果给表单绑定ajaxForm()之后的某个时刻必须删除设置，以便让表单正常提交，则ajaxFormUnbind()命令将会完成这个任务。

对于实验室页面的爱好者来说，ajaxForm实验室页面可以在chapter9/form/lab.ajaxForm.html找到。利用浏览器加载这个页面，外观如图9-5所示。

9

282



283

图9-5　ajaxForm实验室页面让我们观察如何把表单提交拦截为Ajax请求

命令语法：ajaxFormUnbind

ajaxFormUnbind()

删除应用到包装集里表单上的设置，以便表单提交能够以正常方式进行。

参数

无

返回

包装集

这个实验室看起来、工作起来与ajaxSubmit实验室非常相似，除了几个重要的不同点以外：

- 删除按钮Test，添加按钮Submit me!到表单上；
- Control Panel（控制面板）允许我们指定是否把semantic属性添加到options散列对象；
- 添加图像类型的input元素以便观察在semantic设置为true时行为上的差别。

这个表单可以通过下列3个方式提交：

- 点击按钮Submit me!；

- 当焦点在可聚焦元素上时按下回车键；
- 点击控件Input image（木槿花图像）。

在以上任何一种情况下，你将看到页面不会被刷新，因为表单提交被重路由为Ajax请求，其结果在页面的底部窗格里显示。再一次使用页面上的各个控件，直到你熟悉ajaxForm()命令如何操作。在这之后还有一个表单插件的主题需要探讨。

9.1.4 上传文件

插件表单的一个有点隐蔽的有用功能是可以自动地检测和处理需要上传文件（通过文件类型的input元素来指定）的表单。因为XHR无法容纳文件上传的请求，所以ajaxSubmit()命令（以及通过代理ajaxForm()）把请求重路由为动态创建和隐藏的<iframe>，同时把请求的内容类型正确地设置为multipart/form-data（多部分/表单数据）。

必须编写服务器代码来处理这样的文件上传请求和多部分表单。不过，从服务器的角度看来，该请求与任何其他常规表单提交所生成的多部分请求一样；从页面代码的角度看来，这工作起来与常规的ajaxSubmit()完全一样。

妙！

下面把目光放到另一个有用的jQuery插件。

284

9.2 尺寸插件

有时知道元素的精确位置和尺寸是创建富因特网应用的关键。例如在实现下拉菜单时，我们想要菜单在相对于当前触发元素的精确位置出现。

核心jQuery拥有width()、height()和offset()命令，但缺少在所有情况下都能准确地找到元素的功能。这就是尺寸插件派得上用场的地方。

下面首先探讨尺寸插件的API。

9.2.1 宽度和高度的已扩展方法

尺寸插件扩展核心width()和height()命令，因此这些命令可以用来获取窗口和文档对象的宽度和高度，这是核心命令办不到的事情。这些已扩展命令的语法如下。

命令语法：width

width()

返回包装集里第一个元素、窗口或文档对象的宽度。如果第一个包装元素不是窗口或文档，就调用核心jQuery命令。

参数

无

返回

窗口、文档或元素的宽度

9

命令语法：height

height()

返回包装集里第一个元素、窗口或文档对象的高度。如果第一个包装元素不是窗口或文档，就调用核心jQuery命令。

参数

无

返回

285 窗口、文档或元素的高度

在传递一个值作为参数时（为了设置元素的宽度和高度），这两个扩展命令并不干扰对应的核心命令，除非第一个元素是窗口或文档元素。如果第一个元素是窗口或文档元素，这两个扩展命令就像没有传入参数那样操作，即返回窗口或文档的宽度或高度。保持警惕，并相应地编码。

命令width()和height()返回指派给元素的内容尺寸，但有时我们想要处理方框模型的其他方面，比如应用到元素上的内边距或边框。对于这种情况，尺寸插件提供两组命令，把内边距或边框等其他尺寸考虑进去。

第一组命令innerWidth()和innerHeight()不仅测量元素的内容，还测量应用到元素上的内边距。第二组命令outerWidth()和outerHeight()不仅包括内边距，还包括边框甚至可选的外边距。

命令语法：innerWidth和innerHeight

innerWidth()
innerHeight()

返回包装集里第一个元素的内部宽度或高度。内部尺寸包括内容和应用到元素上的内边距。

参数

无

返回

包装集里第一个元素的内部宽度或高度

命令语法：outerWidth和outerHeight

outerWidth()

outerHeight()

返回包装集里第一个元素的外部宽度或高度。外部尺寸包括内容、应用到元素上的内边距和边框。

参数

options　（对象）一个散列对象。这个散列对象接受单个选项，即外边距，指定是否应该把外边距计算进去，默认为false。

返回
包装集里第一个元素的外部宽度或高度

286

请注意：对于以上所有内部和外部方法，指定window或document，都得到相同的结果。

下面探讨尺寸插件能够找到的其他尺寸。

9.2.2 获得滚动尺寸

对于所有类型的用户界面来说确实是这样：内容不一定与已分配的空间相吻合。这个问题已经通过使用滚动条而得以解决：允许用户滚动显示全部内容，即使不能一次在一个视区内看到全部内容。Web没有什么不同，其内容常常超出范围。

在试着相对于窗口或滚动元素而放置新内容（或移动现有内容）时，我们可能需要知道窗口或内容元素（允许滚动）的滚动状态。此外，我们可能想要影响窗口或滚动元素的滚动位置。

尺寸插件允许用scrollTop()和scrollLeft()方法来获取或设置元素的滚动位置。

命令语法：scrollTop和scrollLeft

scrollTop(value)

scrollLeft(value)

获取或设置窗口、文档或可滚动内容元素的尺寸。滚动元素是包含内容的元素（带有CSS样式值为overflow、overflow-x或overflow-y的scroll或auto特性）。

参数

value （数值）设置滚动顶边或左边尺寸的值，以像素为单位。无法识别的值被默认为0。如果省略，则获取并返回顶边或左边滚动尺寸的当前值。

返回

如果提供value参数，则返回包装集。否则，返回被请求的尺寸。

用这两个命令来包装window或document将产生一样的结果。

你想在实验室页面里使用这些命令吗？如果想，请用浏览器加载文件chapter9/dimensions/lab.scroll.html，你会看到如图9-6所示的画面。

287

这个实验室页面允许把顶边和左边滚动值应用到测试对象和窗口上。利用scrollTop()和scrollLeft()命令的getter版本来持续显示滚动尺寸的滚动值。

这个页面的实验室控制面板包含两个文本框（在那里输入数字值，传递给scrollTop()和scrollLeft()命令）以及3个单选按钮（用来选择把命令应用到哪个目标上）。可以选择窗口、文档或测试对象 元素。请注意：我们把页面的<body>元素的高度和宽度设置为荒谬的尺寸2000×2000像素，以便使窗口显示滚动条。

9

按钮Apply把指定的值应用到指定的目标上，而按钮Restore把所有目标的滚动值还原为0。在这两个按钮下面，有页面的一节用于实时地显示3个目标的当前值。

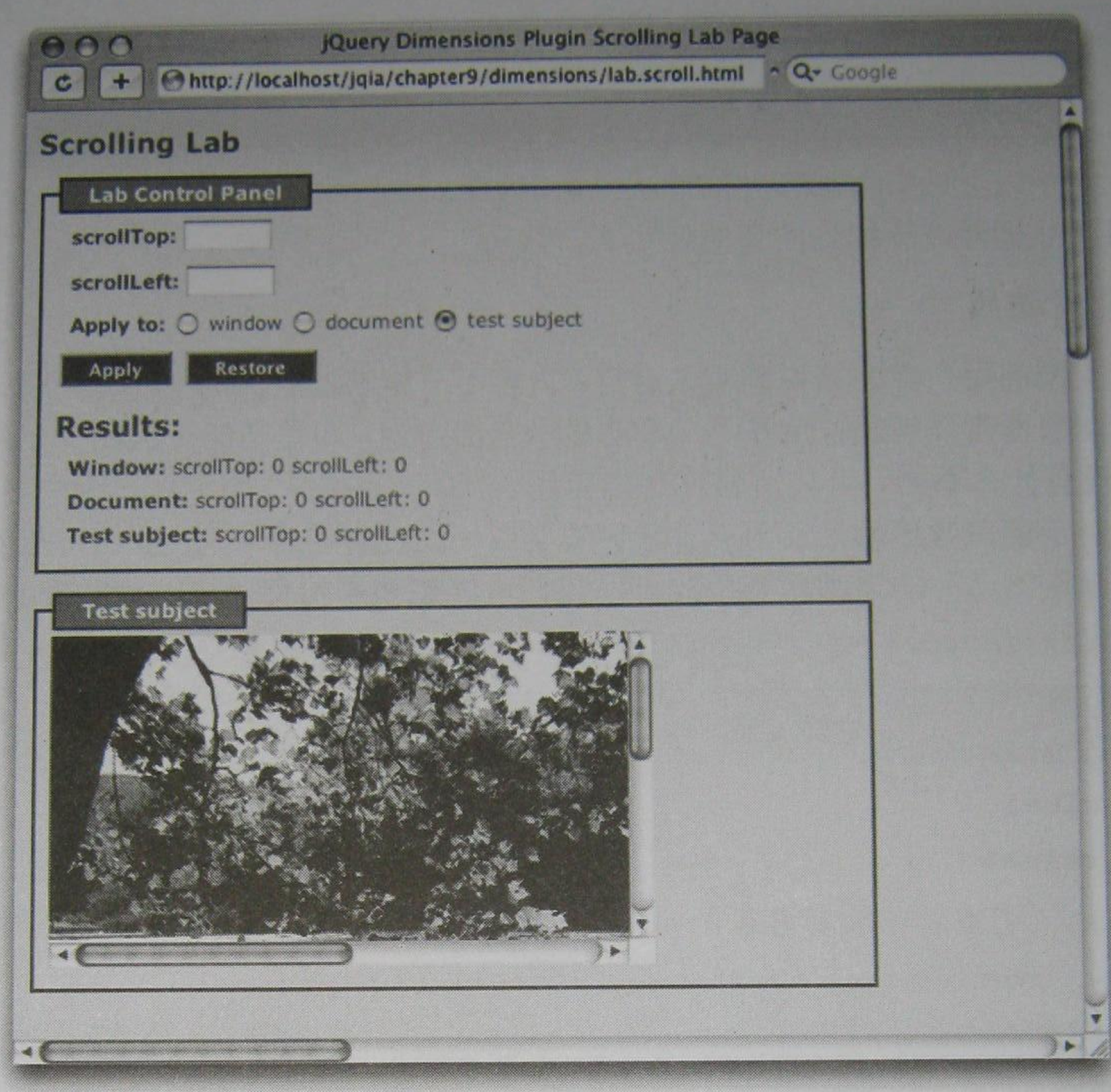


图9-6　滚动实验室让我们观察scrollTop()和scrollLeft()方法的效果

288 Test subject（测试对象）窗格包含测试对象：360×200像素的 元素（包含比那个尺寸所能显示的面积大得多的一个图像）。

这个元素的CSS样式overflow的值被设置为scroll，导致出现滚动条以便能够操作图像。

利用这个页面从头到尾进行下面的练习。

- 练习1——利用Test subject（测试对象）上的滚动条，使图像四处滚动。观察Results（结果）显示，请注意当前滚动值如何保持最新。建立在那个元素上的scroll事件处理程序调用scrollTop()和scrollLeft()方法来获取用于显示的这些值。
- 练习2——重复练习1的步骤，不同之处是这次利用浏览器窗口的滚动条来使页面四处滚动。观察Results（除非你使它们滚出了屏幕），请注意随着你使页面四处滚动，浏览器窗口的滚动值如何变化。也请注意Document的值与Window的值保持一致。强调一点：对于滚动方法，指定窗口或文档，都执行相同的操作。页面既包含窗口也包含文档作为目标，是为了使你相信这一点。
- 练习3——点击按钮Restore，把各个元素设置为正常。选择Test subject（测试对象）作为目标，然后把两个滚动值输入文本框，比如100和100。点击按钮Apply。哇！多么美丽的瀑布！测试其他的值，在按钮Apply调用scrollTop()和scrollLeft()方法时，请看测试

对象受到怎样的影响。

- 练习4——以Window作为目标，重复练习3。
- 练习5——以Document作为目标，重复练习3。请确信这一点：不管指定Window还是Document作为目标，在所有的情况下都产生相同的结果。

9.2.3 关于偏移和位置

我们刚开始可能认为获取元素的位置是简单的任务。只需计算出元素相对于窗口原点的位置，对不？噢，不对。

在CSS值top和left应用到元素上时，top和left是相对于元素的偏移父元素的值。在简单的情况下，偏移父元素是窗口（或者更确切地说，是加载到窗口的<body>元素）。但是，如果元素的任何祖先元素拥有relative或absolute属性的CSS值position，离得最近的这样的祖先元素就是这个元素的偏移父元素。这个概念也被称为元素的定位上下文。

在决定元素的位置时，知道我们正在寻找的是哪一个位置是很重要的。想知道元素相对于窗口原点或偏移父元素的位置吗？也可能涉及其他因素。例如，我们想要计算边框的尺寸。

289

尺寸插件以获取偏移父元素作为起点，为我们解决所有那些问题。

命令语法：offsetParent

offsetParent

返回包装集里第一个元素的偏移父元素（定位上下文）。这是带有relative或absolute的position值、离得最近的祖先元素，或<body>元素（如果没有找到那样的祖先元素）。这个方法只该应用到可见元素上。

参数

无

返回

偏移父元素

如果我们想要获取元素相对于偏移父元素的位置，可以使用position()命令。

命令语法：position

position()

返回包装集里第一个元素相对于偏移父元素的位置值（top和left）。

参数

无

返回

带有两个属性top和left（包含元素相对于定位上下文即偏移父元素的位置值）的一个对象。

如果想要相对于元素的当前位置而使元素重新定位，这个命令很有用。但有时想知道元素（不管它的偏移父元素是什么）相对于<body>元素的位置，并且想在位置计算方式上拥有更多的控制权。对于那种情况下，尺寸插件提供offset()命令。

290

命令语法：offset

offset(options,results)

返回包装集里第一个元素的偏移信息。在默认情况下，返回相对于<body>元素的偏移信息，而偏移信息的计算方式由options参数的设置进行控制。

参数

options　（对象）一个散列对象，包含控制offset方法如何执行计算的设置。可能的取值如下。

- relativeTo——（元素）指定包装集元素的祖先元素，使相对偏移基于祖先元素。这个元素应该拥有relative或absolute的position值。如果省略，则默认为<body>元素。
- lite——（布尔型）指定某些浏览器特定的优化在计算中跳过。这将以精确性为代价而增加性能。默认值为false。
- scroll——（布尔型）指定是否应该把滚动偏移考虑进去。默认值为true。
- padding——（布尔型）指定是否在计算中应该把内边距包括进去。默认值为false。
- border——（布尔型）指定是否在计算中应该把边框包括进去。默认值为false。
- margin——（布尔型）指定是否在计算中应该把外边距包括进去。默认值为true。

results　（对象）一个可选的对象。用来接收offset方法的结果。如果省略，则创建新对象、填充结果并且作为offset方法的值而返回；如果指定，则给传入的对象增加结果属性，并从offset方法返回包装集。如果方法参与jQuery命令链，这就很有用。

返回

如果指定results对象，则返回包装集；如果不指定results对象，则返回results对象。results对象包含属性top和left以及scrollTop和scrollLeft（除非scroll选项被显式地设置为false）。

为了确保从这个方法返回精确值，我们用像素值来表达页面元素的尺寸和位置。这个方法的默认设置通常给予精确的表示。不过比起速度来，如果对精确度更加感兴趣（例如，如果在紧密的循环里对许多元素使用这个方法），就可能想要探索lite选项以及其他设置在这种情况下如何工作。

我们离开位置和度量的王国，下面探讨另一个插件，这个插件在有着大量动态元素和事件处理的页面上非常有用。

291

9.3 实时查询插件

如果你已经对本书所提出的一些较高级别的概念有所体会，就会注意到jQuery在我们所编写页面的结构上有着深刻的影响，并且把页面的结构利用到极致。采用“不唐突的JavaScript”原则，我们的页面通常由<body>里的HTML标记以及就绪处理程序所构成。就绪处理程序建立页面的行为，包括为<body>内所定义的元素而建立的事件处理程序。

jQuery不仅使得以这种方式建立页面轻松得令人难以置信，还使得易于在页面加载期间从根本上修改页面。在页面的生命期内，当就绪处理程序执行时不存在的许多DOM元素，可以随后添加到DOM树上。在添加这样的元素时，我们常常必须随着元素的创建而建立事件处理程序以便定义元素的行为，如同我们为初始化时加载的元素所做。作为新手的Web作者可能艰苦地完成大量重复的、剪切和粘贴的代码，但是更有经验的开发者会把公共的元素提取到函数或JavaScript类中。

不过，如果我们能够在就绪处理程序中声明将在页面上存在的所有元素的行为，而不管是否这些元素在页面加载时是否存在，这样不是很棒吗？

看起来像白日梦，不是吗？但这并非白日梦，实时查询插件（Live Query Plugin）正好让我们能够做到那些事情！

Live Query[1]基于DOM元素匹配我们所定义的jQuery选择器而建立元素的下列行为：

- 为匹配选择器的元素而建立事件处理程序；
- 在任何元素匹配选择器时，触发函数的执行；
- 在任何元素不再匹配选择器时，触发函数的执行。

实时查询插件也允许我们在任何时间内取消绑定这些行为中的任何一个。

下面从Live Query的概览开始，探讨如何为DOM元素而建立事件处理程序，不管这些元素存在与否。

9.3.1 建立主动事件处理程序

实时查询插件允许我们以主动方式来建立事件处理程序：在现在或将来的任何时间内，在匹配jQuery选择器的元素上建立事件处理程序。已建立的处理程序不仅应用到处理程序建立时匹配选择器的现有元素，还应用到随后在页面的生命周期内匹配选择器的任何元素——包括将来被改变而匹配选择器的现有元素，以及匹配选择器的新建元素。 292

如果这些元素被改变了，因此不再匹配选择器，则Live Query所建立的处理程序会从那些元素上自动删除。

某些变化会影响元素是否匹配选择器，这些变化以使用jQuery方法为转移。如果把元素置于jQuery的领域之外，显然Live Query会失去用来跟踪元素的挂钩。如果我们绝对必要在jQuery控制之外生成变化，Live Query确实有办法帮助我们处理那种情况，稍后就会讲到。

所有的Live Query行为，包括事件监听器，都是在元素上利用`livequery()`方法而建立的。建立主动事件处理程序的格式如下。

---

① Live Query Plugin（实时查询插件）简称Live Query，下文同。——译者注

命令语法：livequery

livequery(event,listener)

建立一个函数作为指定事件类型的事件处理程序，既在匹配集里所有的元素上，也在随后与匹配集的选择器相匹配的任何元素上。

参数

event　（字符串）为哪一个事件类型建立监听器。与jQuery的bind()命令所使用的那组事件类型相同。

listener　（函数）将要建立为事件监听器的函数。每个调用的函数上下文（this）是当前的匹配元素。

返回

包装集

这种形式的livequery()调用起来真的很像jQuery的bind()命令。像bind()那样，它为匹配集里所有的元素建立处理程序。但是它也在页面加载的任何时间内在匹配选择器模式的任何元素上自动地建立处理程序。它也会从不再匹配选择器的任何元素（包括来自原始匹配集的那些元素）取消绑定监听器。

这是非常强大的功能。它允许在就绪处理程序里一次性地为匹配选定的选择器的元素而设置行为，随后无需为跟踪这样的事情（比如元素被改变或添加到页面）而操心。那多酷啊？

事件处理程序的建立既是一个特殊的案例，也是常见的案例（因此它获得格外的关注）：在元素因被改变（或添加）而不再匹配原始选择器时执行操作。在这样的时刻我们可能想做许多其他的事情，而Live Query不会令我们失望。

9.3.2　定义匹配和不匹配监听器

如果想要在元素进入（或脱离）匹配特定选择器的状态时执行操作（不同于绑定或取消绑定事件处理程序），可以使用livequery()命令的另一种形式。

命令语法：livequery

livequery(onmatch,onmismatch)

为匹配集建立回调函数，在元素进入或脱离匹配选择器的状态时调用。

参数

onmatch　（函数）指定一个函数作为匹配监听器。这个函数为进入匹配状态的任何元素（建立为函数上下文）而被调用。如果在调用这个方法时任何现有元素匹配选择器，则这个函数会为每个这样的元素而被立即调用。

onmismatch　（函数）指定一个（可选的）函数作为不匹配监听器。这个函数为脱离匹配状态的任何元素（建立为函数上下文）而被调用。如果省略，则不建立任何不匹配监听器。

返回

包装集

如果只想建立可选的不匹配监听器，则通过传递null作为这个方法的第一个参数无法达到目的，因为这将导致第二个参数被建立为匹配监听器，仿佛它被传递作为第一个实参那样。取而代之，我们传入无操作（no-op）函数作为第一个参数：

```
$('div.whatever').livequery(
  function(){},
  function(){ /* mismatch actions go here */ }
);
```

因为事件监听器为Live Query所建立，所以导致进入或脱离匹配状态的变化将会自动地触发这些函数（事件监听器），前提是变化由jQuery方法所执行。但是，如果不能利用jQuery来执行那些变化，那时怎么办呢？

9.3.3 强制 Live Query 求值

如果我们通过jQuery函数以外的手段，为Live Query监听器而实现元素的变化（导致元素进入或脱离匹配状态的那些变化），就可以利用实用工具函数来强制Live Query触发其监听器。 294

函数语法：$.livequery.run

$.livequery.run()

强制Live Query执行元素的全局求值：触发任何适当的监听器。如果变化是在jQuery方法的控制之外实现的，这个函数就能派上用场。

参数

无

返回

未定义

我们已经看到Live Query在元素脱离匹配状态时自动地删除事件监听器，还看到可以建立不匹配监听器以便在元素脱离匹配状态时执行我们所要的操作。但是，如果想要使监听器过期，应该怎么办呢？

9.3.4 使 Live Query 监听器过期

jQuery提供unbind()命令以便撤销bind()的操作。与此相似，Live Query提供办法以便撤销Live Query事件处理程序以及匹配/不匹配处理程序的建立：expire()命令，表现为下列参数格式。

命令语法：expire

```
expire()
expire(event,listener)
expire(onmatch,onmismatch)
```

删除与匹配集的选择器相关联的监听器。不带参数的格式则删除与选择器相关联的所有监听器。请参见下面的参数说明，了解参数如何影响这个命令。

参数

event　（字符串）指定要取消绑定的事件类型的事件监听器。如果不指定监听器，所有该事件类型的监听器将被删除。

listener　（函数）如果指定，则只有特定的监听器从选择器中取消绑定（为指定的事件类型）。

onmatch　（函数）指定将从选择器中取消绑定的匹配监听器。

mismatch　（函数）（可选地）指定将从选择器中取消绑定的不匹配监听器。

返回

295 包装集

我们已经提供实验室页面来帮助说明这些技术的使用。用浏览器加载文件chapter9/livequery/lab.livequery.html，你将会看到如图9-7所示的画面。

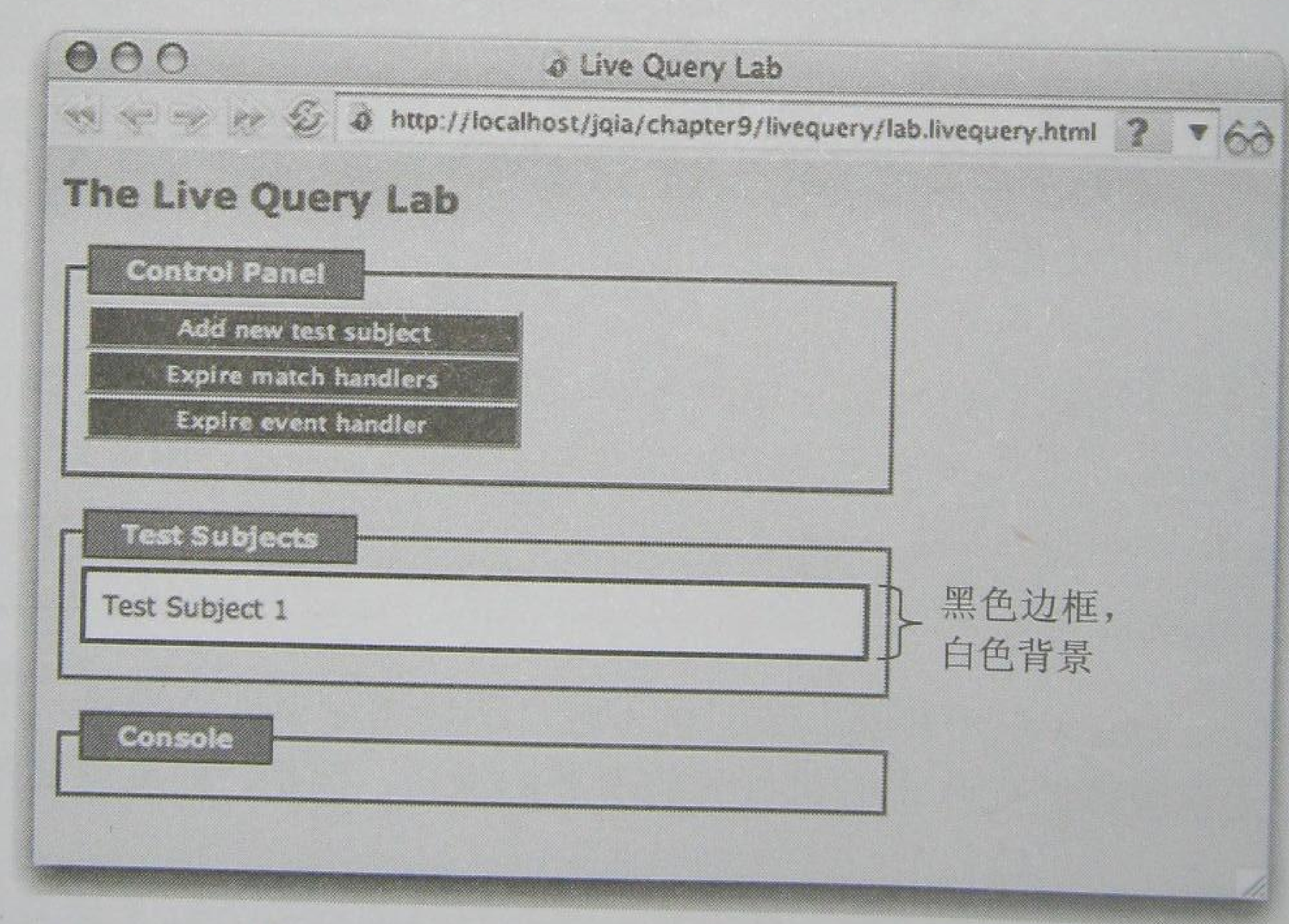


图9-7　实时查询实验室页面帮助我们观察Live Query的实际操作

这个实验室页面显示3个窗格：Control Panel（控制面板，带有用来触发好玩事情的按钮）、Test Subjects（测试对象容器）以及Console（控制台，用来显示有关当前操作的消息）。

这个页面的设置需要一些解释。在就绪处理程序里，控制面板各个按钮被设置为一旦点击之后执行各自的操作（如果感兴趣，请参考文件了解细节）。然后执行两个Live Query语句：

```
$('div.testSubject').livequery('click',
  function(){
    $(this).toggleClass('matched');
  }
);
$('div.matched').livequery(
  function(){ reportMatch(this,true); },
  function(){ reportMatch(this,false); }
);
```

❶ 建立一个主动click事件处理程序

❷ 建立匹配和不匹配的处理程序

第一个语句为带有类testSubject的所有 元素建立Live Query事件处理程序❶。这些元素包括在页面加载时已经驻留的一个测试对象（如图9-7所示），以及将来点击Add New Test Subject（添加新的测试对象）按钮之后创建的所有测试对象元素。click事件处理程序（稍后讨论其活动）不仅立即建立在现有测试对象元素上，而且自动地添加到将来动态地添加到页面的任何测试对象上（这些测试对象都被创建为带有类testSubject的 元素）。 296

click处理程序为event的target而切换类matched。为了易于观察哪个测试对象带有类matched而哪些对象不带有，我们设置CSS规则，因此不带有类的元素拥有黑色边框和白色背景，而带有类的元素呈现更粗的栗色边框和黄褐色背景。这全是时髦的Web 2.0色彩！

第二个语句为带有类matched的所有 元素建立匹配和不匹配处理程序❷。每个这样的处理程序都输出消息到控制台：各自宣布一个元素已经变为匹配或不匹配。因为在页面加载时页面上没有元素拥有类matched，所以在页面初始化显示时，控制台是一片空白。

下面查看控制面板的按钮做些什么。

- Add New Test Subject（添加新的测试对象）——这个按钮添加新的测试对象 元素到页面上。所创建的元素带有id为testSubject#（#是一个不断增长的计数）和类testSubject。这样的一个元素通过HTML标记预先填充在Test Subjects中。
- Expire Match Handlers（使匹配处理程序过期）——这个按钮执行语句$('div.matched').expire();，使就绪处理程序里所建立的匹配和不匹配处理程序过期。
- Expire Event Handler（使事件处理程序过期）——这个按钮执行语句$('div.test-Subject').expire();，使就绪处理程序里所建立的主动事件处理程序过期。

既然我们理解这个实验室页面是如何贯穿起来的，下面做一些练习。

- 练习1——加载页面，并且在Test Subject 1的范围内点击。因为我们为click事件所建立的Live Query事件处理程序导致matched类的切换（在这种情况下是添加类matched），所以我们看到元素改变颜色如图9-8所示。此外，因为matched类的添加导致元素匹配用来建立匹配和不匹配处理程序的选择器，所以我们看到匹配处理程序已经触发，并输出消息到控制台。 297
- 练习2——再次点击Test Subject 1。这将切换matched类（从元素中删除类matched）。元素还原到原始外观，因为元素不再匹配选择器，所以不匹配处理程序被触发而输出对应的消息到控制台。
- 练习3——点击按钮Add New Test Subject。新的测试对象 元素被添加到页面上。因为所创建的新元素带有类testSubject。现在这个元素匹配选择器（用来建立切换类

matched的主动click处理程序）。这将触发Live Query自动地绑定click处理程序到这个新元素上（用来添加这个元素的代码并不绑定任何处理程序到新建的元素上）。为了测试这个假设，点击新建的Test Subject 2。我们看到Test Subject 2改变呈现，并且匹配处理程序为它而被调用。这就证明了用来使元素切换类matched的click处理程序被自动地添加到新建的元素上。有图为证，请看图9-9。

298

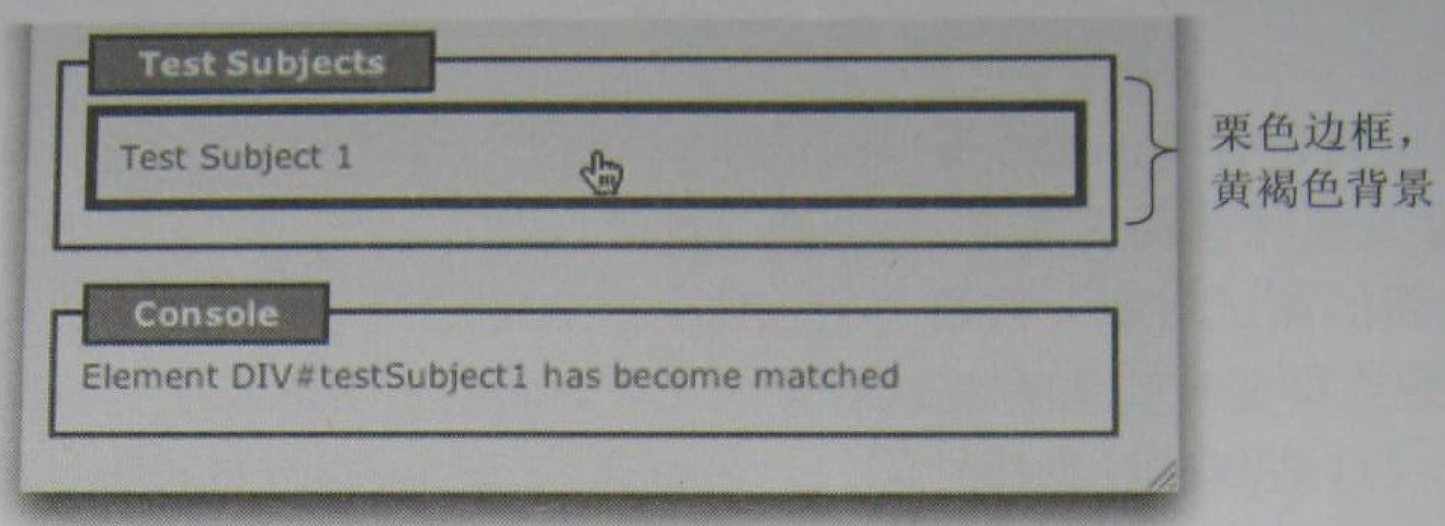


图9-8　给测试对象所添加的matched类触发页面呈现的变化，并触发已建立的匹配处理程序

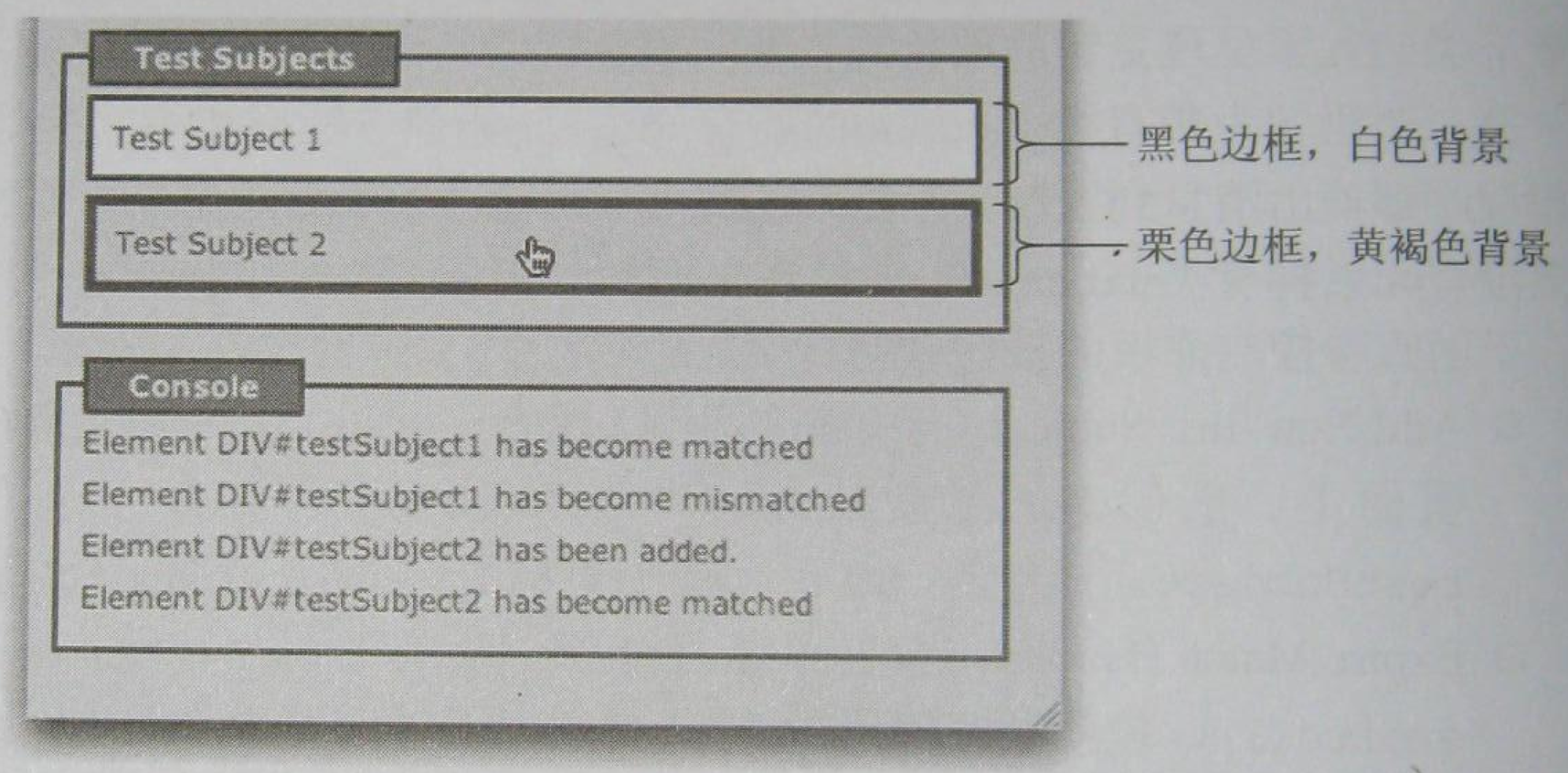


图9-9　新建的Test Subject 2对点击做出与Test Subject 1相同的反应，这一事实证明了Live Query已经自动地给新建的元素添加了click处理程序以及匹配和不匹配的处理程序

- 练习4——用按钮Add New Test Subject和Test Subjects进行试验直到你信服：无论何时只要合适，事件、匹配和不匹配处理程序始终会自动地添加到测试对象元素上。
- 练习5——做试验使Live Query处理程序过期。重新加载页面以便从原始的设置开始。用按钮Add New Test Subject来添加一个或两个测试对象到页面上。现在点击按钮Expire Match Handlers，使在Test Subjects上所建立的匹配和不匹配处理程序过期。请注意：当你点击Test Subjects时，如果匹配和不匹配处理程序已被删除，就没有任何匹配/不匹配消息出现在控制台里。受到点击时元素依然改变外观，这证明了click事件处理程序依然切换类matched。
- 练习6——点击按钮Expire Event Handler，使Test Subjects的Live Query点击处理程序过期。请注意：现在Test Subjects对鼠标点击没有反应，而是保留在你点击按钮时的已有状态。

不难想象实时查询插件给富因特网应用页面所带来的力量，在那里元素时时刻刻都在变化（包括突然出现和消失）。通过预先建立这些事件的行为，在改变和添加元素到页面上时，实时查询插件帮助我们将所需编写的代码减到最少。

下面继续探讨另一个更加重要的有用插件。

9.4 UI 插件

如果说到富因特网应用，那么UI为王。这毫不奇怪，许多jQuery插件都关注于启用富用户界面。在这一节里，我们介绍官方的UI插件，这是最近添加到jQuery家族的一个重要组件。因为它是重要组件，所以我们想要探讨这个插件达到研究核心jQuery的那般深度，不过事实上受限于篇幅，无法进行更为广泛的涵盖。

我们将广泛地涵盖UI插件所定义的两类不可或缺的方法——几个方法为页面上的拖放而提供支持（让人很好地感受到插件的剩余部分如何操作）。然后提供插件的其余部分的概览，演示它可以做些什么给页面带来富因特网应用的功能。想要了解有关这个方面的更多细节，请访问http://docs.jquery.com/ui。

UI插件为三个主要的方面提供支持：鼠标交互、小部件以及可视效果。拖放操作属于鼠标交互这一分类，下面首先探讨鼠标交互。

9.4.1 鼠标交互

与鼠标指针进行交互是任何GUI的核心组成部分。虽然许多简单鼠标交互内建在Web界面里（例如点击），但Web本身并不支持对桌面应用可用的一些高级交互方式。这种不足的最好示例是缺少对拖放的支持。

拖放对于桌面用户界面来说是无处不在的交互技术。例如，在任何桌面系统的GUI文件管理器中，在文件夹之间拖放文件从而轻松地复制文件或在文件系统中四处移动文件，甚至把文件拖放到回收站或废纸篓图标从而轻松地删除文件。尽管这种交互方式在桌面系统中非常普遍，在Web应用里却是少见的，主要因为现代浏览器本身并不支持拖放。正确地实现拖放是令人望而生畏的任务。

“令人望而生畏？”你嘲笑道，“捕获几个鼠标事件加上一些微不足道的CSS，有什么大不了的？”

虽然高级概念并不是那么难以把握，但事实上实现拖放支持的细微差别，尤其是以稳健和独立于浏览器的方式，会迅速地变得痛苦起来。不过，jQuery及其插件以前曾经解除我们的痛苦，同样地，jQuery及其插件为Web启用拖放而提供支持，再一次解除了我们的痛苦。

在拖放之前，首先必须学会如何拖动。

1. 四处拖动界面元素

虽然在大多数的词典中我们很难查到术语“可拖动”，但这个术语常常应用于在拖放操作中可以四处拖动的项。同样地，UI插件用这个术语来描述那样的元素，而且用这个术语来命名把拖动功能应用到匹配集元素的方法。

命令语法：draggable

draggable(options)

根据指定的选项，使包装集的元素变得可拖动。

参数

options　（对象）包含多个选项的散列对象。这些选项被应用于可拖动元素，如表9-2所描述。如果不指定选项，则元素可以在窗口内各处自由地拖动。

返回

包装集

为了使用draggable()命令，必须至少包括下列两个文件（除了核心jQuery脚本文件以外）：

ui.mouse.js

ui.draggable.js

为了获得已扩展的选项，还必须包括下面的文件：

ui.draggable.ext.js

基本的和已扩展的选项都包括在内，这个方法所支持的选项如表9-2所示。

表9-2　**draggable()命令的基本的和已扩展的选项**

名　称	描　述
基本的选项	
helper	（字符串丨函数）精确地指定什么将被拖放。如果指定为original（默认值），则原始项被拖放；如果指定为clone，则创建元素的副本用于拖放。也可以指定一个函数，接受原始DOM元素作为其参数并且返回一个元素用于拖放。最为常见的是应用某种转换之后的原始元素的副本。例如，function(e){return $(e).clone().css('color','green')}
ghosting	（布尔型）如果为true，则是helper:'clone'的同义词
handle	（jQuery丨元素）一个备选元素或包含一个元素的jQuery包装集，作为拖动把柄（被点击的项，用于发起拖动操作）。这个元素经常是可拖动元素的子元素，但也可以是页面上的任何元素
preventionDistance	（数值）在点击之后开始拖动操作之前，鼠标指针必须移动的像素数。这可以用来帮助防止意外拖动。如果省略，则默认值为0
dragPrevention	（数组）选择器数组，匹配在点击时不应该发起拖动操作的子元素。默认值为['input','textarea','button','select','option']。例如，在内嵌控件被点击时，这个选项用来阻止开始拖动操作
cursorAt	（对象）指定在拖动操作进行时鼠标指针与可拖动对象之间的空间关系。指定的对象可以定义下列属性：top、left、bottom或right。例如，{top:5,left:5}这个对象使鼠标指针被定位于距离被拖动元素的左上角5个像素的地方。如果省略，则鼠标指针维持其原始相对位置（发起拖动操作的鼠标点击的位置）
appendTo	（字符串丨元素）指定在拖动操作结束时被拖动的helper被追加到哪一个元素。如果指定字符串parent（默认值），则使helper留在原始的层次结构中

（续）

名　称	描　述
基本的选项	
start	（函数）在拖动操作开始时调用的一个回调函数。其函数上下文（this）被设置为可拖动元素，并且传递两个参数：event实例（其target属性被设置为可拖动元素）和包含下列属性的一个对象： helper——当前的helper元素； position——对象，包含属性top和left（指定在拖动操作开始时鼠标的位置）； offset——为选项cursorAt而指定的对象； draggable——内部JavaScript可拖动对象（不太有用）； options——选项的散列对象，用来创建可拖动元素
stop	（函数）在拖动操作完成时调用的一个回调函数。它与start回调函数一样，被传递同样的两个参数。第二参数的position属性报告可拖动元素的左上角的位置
drag	（函数）在拖动操作进行时持续调用的一个回调函数。它与start回调函数一样，被传递同样的两个参数。第二个参数的position属性报告可拖动元素的左上角的位置
已扩展的选项	
axis	（字符串）限制可拖动元素沿着哪个轴而移动：x代表横轴而y代表纵轴。如果省略，则不施加任何轴限制
containment（包含）	（字符串 | 对象 | 元素）指定可拖动元素可以在哪个范围内移动。如果省略，则不施加任何限制。可以指定为下列值： 父元素——在父元素内包含可拖动元素，因此不必给父元素添加任何滚动条； 文档——在当前文档内包含可拖动元素，因此不必给窗口添加任何滚动条； 一个选择器——标识包含元素； 一个对象——用属性left、right、top和bottom来指定相对于父元素的矩形范围
effect	（数组）由两个字符串所构成的一个数组，用来把褪色（淡入淡出）效果应用到被克隆的helper(s)。可以指定为['fade','fade']、['fade','']或['','fade']。用这种方式来指定效果可能看起来奇怪，但它允许将来支持附加的效果类型，不过当前只支持褪色效果
grid	（数组）由两个数字所构成的数组，用来指定矩形网格（定义可拖动元素能够移动到的离散区域）。例如[100,100]。网格的原点相对于可拖动对象的原始位置。如果省略，则不对移动施加任何网格约束
opacity	（数值）指定在拖动操作期间被拖动helper的不透明度为0.0到1.0之间的一个值（包括0.0和1.0）。如果省略，则不改变被拖动helper的不透明度
revert	（布尔型）如果为true，在拖动操作结束时可拖动对象被移回其原始位置。如果省略或指定为false，则不移回其原始位置

302

如果你原以为我们准备放过用有趣的实验室页面来演示这些选项的机会，再猜猜看！不过在查看那个实验室页面之前，首先探讨另外3个可拖动元素的相关方法。

如果想要使可拖动元素不再可拖动，则draggableDestroy()命令可以删除可拖动性。

命令语法：draggableDestroy

draggableDestroy()

删除包装集元素的可拖动性。

参数
无

返回
303 包装集

如果我们只想暂时挂起元素的可拖动性并且在随后的某个时间内还原可拖动性，可以用命令draggableDisable()来禁用可拖动性，而用draggableEnable()来还原可拖动性。

命令语法：draggableDisable

draggableDisable()

挂起包装集的可拖动元素的可拖动性，而不删除可拖动性信息或选项。

参数
无

返回
包装集

命令语法：draggableEnable

draggableEnable()

对于被draggableDisable()禁用了可拖动性的包装集里任何可拖动元素，还原可拖动性。

参数
无

返回
包装集

通过使用UI可拖动元素实验室，检查各种可拖动选项。用浏览器来加载页面chapter9/ui/lab.draggables.html，你将看到如图9-10所示的画面。

这个实验室显示已经熟悉的页面布局：控制面板（Control Panel）窗格包含用来指定draggable()各种选项的控件，测试对象（Test Subject）窗格包含作为可拖动测试对象的图像元素，而控制台（Console）窗格报告有关正在进行的拖动操作的信息。

如果点击按钮Apply（在控制面板上可以找到），则收集指定的选项并且发出draggable()命令。命令格式显示在按钮Apply的下方（为了清晰起见，只显示通过控制面板所指定的选项，不显示给选项所添加的、用于在控制台窗格里实现显示的几个回调函数，但回调函数被包含在所发出的命令中）。在练习3里可以观察到Disabe和Enable按钮的操作。Reset按钮把各个选项控件还原
304 到初始状态，并且取消在测试对象上所设置的任何可拖动功能。

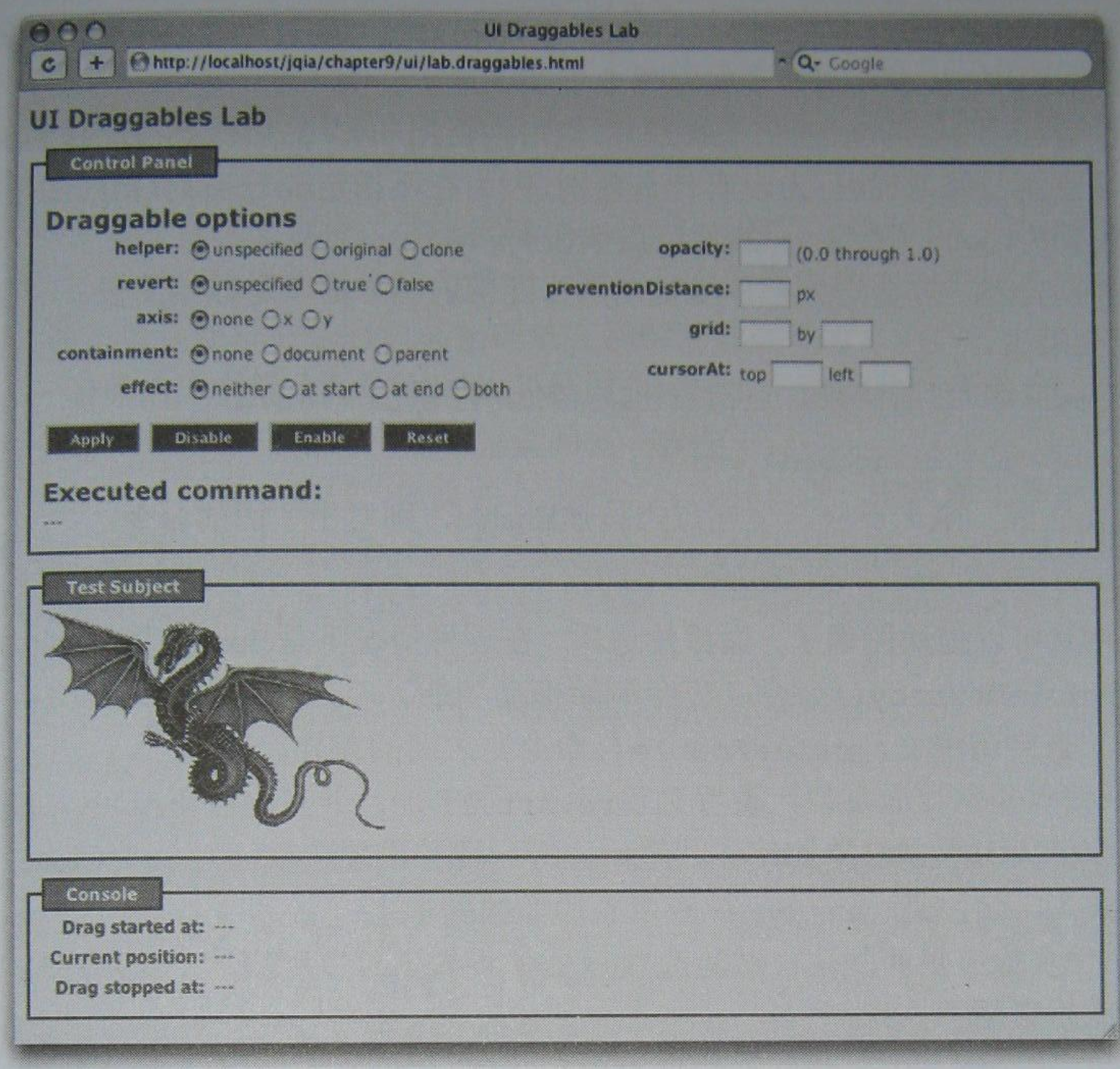


图9-10 UI可拖动元素实验室页面允许试验给可拖动元素所提供的大多数选项

下面钻研这个实验室页面并且做一些练习！

练习1——作为第一个练习，我们创建带有所有默认选项的简单的可拖动元素。用浏览器来加载UI可拖动元素实验室页面，让所有选项控件在加载之后保持不变。控件的初始状态被设置为不指定任何选项。

试着点击和拖动作为测试对象的龙图像。没有发生什么事情，除非你在使用OS X，在那种情况下，你将看到发生一些事情。OS X允许通过拖放而把图像从网页上拖出去，复制到本地文件系统。不要把这种系统启用的拖放与我们准备在页面上启用的拖放操作相混淆。

现在，请点击按钮Apply，并观察如下所示的已执行的命令：

```
$('#testSubject').draggable({});
```

再次试着拖动龙图像。注意如何能使龙图像在浏览器窗口中四处飞动（没有拍打翅膀，尽管那是更为高级的动画）。同样注意如何能使龙图像移动到窗口的边缘，而且在移动到最右边时，导致出现原来所没有的水平滚动条（在大多数浏览器里）。

放下龙图像，又再次把它拖动起来。四处拖动龙图像，只要你喜欢，不管多少次都可以。观察在拖动操作期间控制台里的值如何保持最新。这是利用回调函数start、stop和drag来实现的。

练习2——重新加载页面以便还原初始状态。如果你在使用Firefox或Camino，则这些浏览器

有个烦人的功能：如果用工具栏按钮Reload来重新加载页面，则表单控件不会还原到初始状态。为了使页面还原到初始状态，请移动文本焦点到URL地址栏上，然后敲击回车键。

现在设置选项helper为Original（原始），点击按钮Apply，然后四处拖动作为测试对象的龙图像。你应该注意到与练习1的行为没有什么差别，因为在不指定helper时Original是默认的设置。把选项helper设置为Clone（克隆），然后点击按钮Apply。

现在当你拖动图像时，会看到被四处拖动的是图像的副本而不是原始图像。一旦使拖动结束，图像的副本就会消失。

练习3——保持练习2的结果，点击按钮Disable，注意已发出的命令：

```
$('#testSubject').draggableDisable();
```

试着拖动龙图像。毫无反应。现在点击按钮Enable，则发出下面的命令：

```
$('#testSubject').draggableEnable();
```

请注意：你又可以拖动图像了，而原始选项（在这种情况下是clone类型的helper）依然有效。这演示了draggableDestroy()命令（完全删除拖动功能）与draggableDisable()命令（只是挂起拖动功能，直到调用draggableEnable()命令）之间的差异。

练习4——使实验室页面复位，设置选项revert为True，并点击按钮Apply。拖动测试对象，注意在拖动操作结束时测试对象如何移回原始位置。现在设置选项helper为Clone，点击Apply，然后重复这个练习。请注意：revert被应用到图像的副本上。

练习5——使实验室页面复位，试验选项axis的设置。可以利用这个选项在拖动期间使移动限制在水平面或垂直面上。

练习6——在这个练习中，我们把注意力放在选项containment上。你会想增加浏览器窗口的高度到屏幕所允许的最大限度。希望你的屏幕分辨率使控制台窗格的下面显示另外一些空间。

到目前为止，我们放着选项containment没有指定。请回忆如何能使龙图像在浏览器窗口内四处移动。现在设置选项containment为Document，并点击按钮Apply。在你拖动图像的时候注意两件重要的事情。

- 你不再能够移动图像超出窗口的边缘，因此原先没有滚动条的地方现在也不出现滚动条。
- 你不能移动图像到控制台窗格的底部之下，在那儿document止住了但window没有止住。

现在设置选项containment为Parent，并点击按钮Apply。如果开始拖动操作，注意你只能在测试对象（Test Subject）窗格内（<fieldset>元素，作为测试对象元素的父元素）四处拖动图像。也请注意：即使在父元素外面开始拖动操作（作为上一个拖动操作的结果而没有限制在父元素里面），结果依然是那样。

练习7——设置选项helper为Clone，并给helper设置不同的Effect选项值，观察其效果。

练习8——使实验室页面复位，并指定opacity（不透明度）为0.5。观察被拖动元素的不透明度受到怎样的影响，不管被拖动的是原始元素还是helper副本。

练习9——使实验室页面复位，并设置选项preventionDistance为较大的值比如200。点击按钮Apply之后，通过点击龙的左翼边缘并且向右移动鼠标指针从而开始拖动操作。在拖动操作开始之前，你几乎遍历了龙的全部宽度（龙图像宽度为250像素）。很少会把这个选项设置为这么大的值，但在此处我们这样做是为了说明选项preventionDistance的行为。更为常见的是，这

个选项被设置为小得多的值以便防止意外移动几个像素就导致发起不需要的拖动操作。 307

练习10——使实验室页面复位，并指定选项grid的值为100×100。点击按钮Apply之后，注意在各个方向上现在只能以100像素的离散移动方式来拖动元素。试验其他的值以便观察在拖动时元素的行为。

练习11——使实验室页面复位，并设置选项cursorAt的值为10和10。点击按钮Apply。在拖动操作开始时，鼠标指针的位置被设置为相对于图像左上角向下和向右各10个像素的地方（接近于龙的翼尖），而不管在拖动开始时鼠标指针放在图像的什么地方。

练习12——请尽情地利用实验室页面的各种设置，利用个别以及综合的方式，直到你感到自信：理解了各个选项分别如何影响可拖动元素的拖动操作。

在屏幕上四处拖动元素虽然好，但真的有用吗？一度很有趣，但就像玩溜溜球（除非我们真的是狂热爱好者）那样，很快就玩腻了。在实际应用中，可以利用拖动来让用户在屏幕上、在游戏或谜题中四处移动模块式元素（如果我们细心体贴，还会通过cookie或其他持久化机制来记住用户所选择的位置）。

把被拖动元素放置在一些有趣的东西上时，拖动操作才真的焕发光彩。下面看怎样使可放置元素与可拖动元素相互配合。

2. 放下被拖动元素

翻转硬币的可拖动元素的这一面，就看到可放置元素（能够接纳被拖动元素，并在发生这种事件时做一些有趣的事情）的另一面。根据页面元素而创建可放置元素与创建可拖动元素相似，事实上创建可放置元素更加简便，因为只有较少的选项需要关心。

就像可拖动元素那样，可放置元素也被分为两个脚本文件：定义droppable()命令及其基本选项的基本文件，以及包含已扩展选项的选项文件。这两个文件是

```
ui.droppable.js
ui.droppable.ext.js
```

308

命令droppable()的语法如下所示。

命令语法：droppable

droppable(options)

把包装集里的元素建立为可放置元素（可放置元素，是指上面可以放置可拖动元素的元素）。

参数

options （对象）应用于可放置元素的各种选项。请参见表9-3了解细节。

返回

包装集

9

只要元素被设置为可放置元素，它就存在于下列3种状态之一：不活动态、活动态和武装态。

不活动态是可放置元素在大部分的时间内所保持的常态，等待着以便检测拖动操作开始的时

刻。在拖动开始时，可放置元素决定可拖动元素是否为适宜放置的元素（稍后讨论“适宜”的概念），如果是（且只有是）适宜放置的元素，就进入*活动态*。在活动态里，可放置元素监视拖动操作，等待着直到拖动操作终止（在这种情况下可放置元素还原到不活动态），或者直到可拖动元素悬停在可放置元素的上方。当适宜的可拖动元素悬停在可放置元素上方时，可放置元素就进入*武装态*。

当可放置元素处于武装态时，如果拖动操作终止，可拖动元素被认为已被放置在可放置元素上。如果可拖动元素继续移动，因而不再悬停在可放置元素上方，此时可放置元素还原到活动态。

哇，那就要跟踪一大堆的状态变化啊！图9-11的图应该能够帮你理清思路。

就像可拖动元素那样，在主要脚本文件里提供一组基本选项，而在附加脚本文件里提供一组已扩展选项。这两组选项如表9-3所述。

如果查看可拖动元素的创建，就会看到可以创建非常有用的可拖动元素而不必指定任何选项到`draggable()`方法——`droppable()`方法则并非如此。在调用`droppable()`时，如果不指定任何选项，虽然不会发生什么坏事，但也不会发生什么好事。如果不带`accept`选项而创建可放置元素，就是“冒充好人”。

在默认情况下，可放置元素不认为任何可拖动元素是适宜的。如果不能放置任何元素在其上面，那么可放置元素就没有用，对吧？为了创建可以放置某个元素的可放置元素，必须指定`accept`选项（定义将要被可放置元素认可的可拖动元素）。

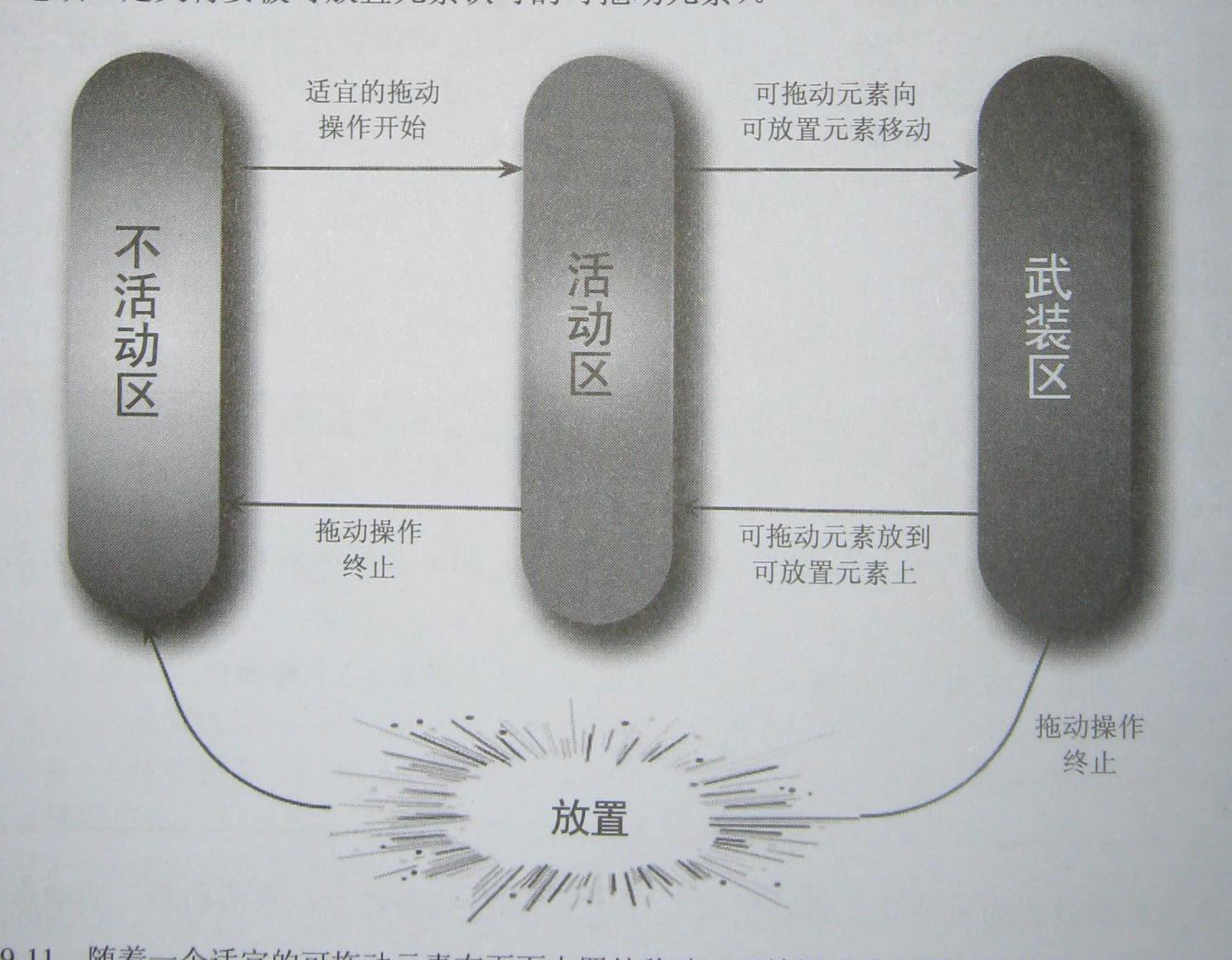


图9-11　随着一个适宜的可拖动元素在页面上四处移动，可放置元素在不同的状态之间进行转换

表9-3 命令**droppable()**的基本选项和已扩展选项

名 称	描 述
基本选项	
accept	（字符串丨函数）指定哪些可拖动元素适宜于放置到可放置元素上。这个选项可以是字符串（用来描述jQuery选择器）或函数（返回true，指定可拖动元素是可接受的）。如果指定为函数，则传递候选可拖动元素作为唯一参数进行函数调用
tolerance	（字符串）一个字符串值，定义可拖动元素必须怎样定位相对于可放置元素的位置，以便把可放置元素武装起来。可能的取值如下 touch——如果可拖动元素触及可放置元素，或可拖动元素的任何部分与可放置元素相重叠，则把可放置元素武装起来 pointer——在拖动操作期间如果鼠标指针进入可放置元素，则把可放置元素武装起来 intersect——如果可拖动元素的50%与可放置元素相交，则把可放置元素武装起来 fit——如果可拖动元素被完全包含在可放置元素内，则把可放置元素武装起来
activate	（函数）一个回调函数，在可接受的可拖动元素开始拖动操作时（在可放置元素由不活动态转变为活动态时）被调用。函数上下文（this）被设置为可放置元素。这个函数被传入event实例以及一个对象，包含关于操作信息的下列属性 draggable——可拖动实例 droppable——可放置实例 element——可拖动元素 helper——可拖动助手 options——传递给droppable()的各个选项
deactivate	（函数）一个回调函数，在可放置元素从活动态或武装态还原为不活动态时被调用。函数上下文（this）被设置为可放置元素。正如activate回调函数的说明，这个函数被传入相同的参数
over	（函数）一个回调函数，在可放置元素从活动态转变为武装态时被调用（由于可拖动元素满足tolerance选项所定义的标准）。函数上下文（this）被设置为可放置元素。正如activate回调函数的说明，这个函数被传入相同的参数
out	（函数）一个回调函数，在可放置元素从武装态转变为活动态时被调用（由于可拖动元素离开可放置元素，如同tolerance选项所指定标准的定义）。函数上下文（this）被设置为可放置元素。正如activate回调函数的说明，这个函数被传入相同的参数
drop	（函数）一个回调函数，在可拖动元素被放置在武装态的可放置元素上时被调用。函数上下文（this）被设置为可放置元素。正如activate回调函数的说明，这个函数被传入相同的参数
已扩展选项	
activeClass	（字符串）在可放置元素处于活动态时应用于可放置元素的一个CSS类名称
hoverClass	（字符串）当适宜的可拖动元素悬停在可放置元素的上方时（即可放置元素处于武装态时），应用于可放置元素的一个CSS类名称

310

9

没有什么比动手实践更能让人深刻地理解概念，因此我们创建了可放置元素实验室页面。用浏览器打开文件chapter9/ui/lab.droppables.html，你会看到如图9-12所示的画面。

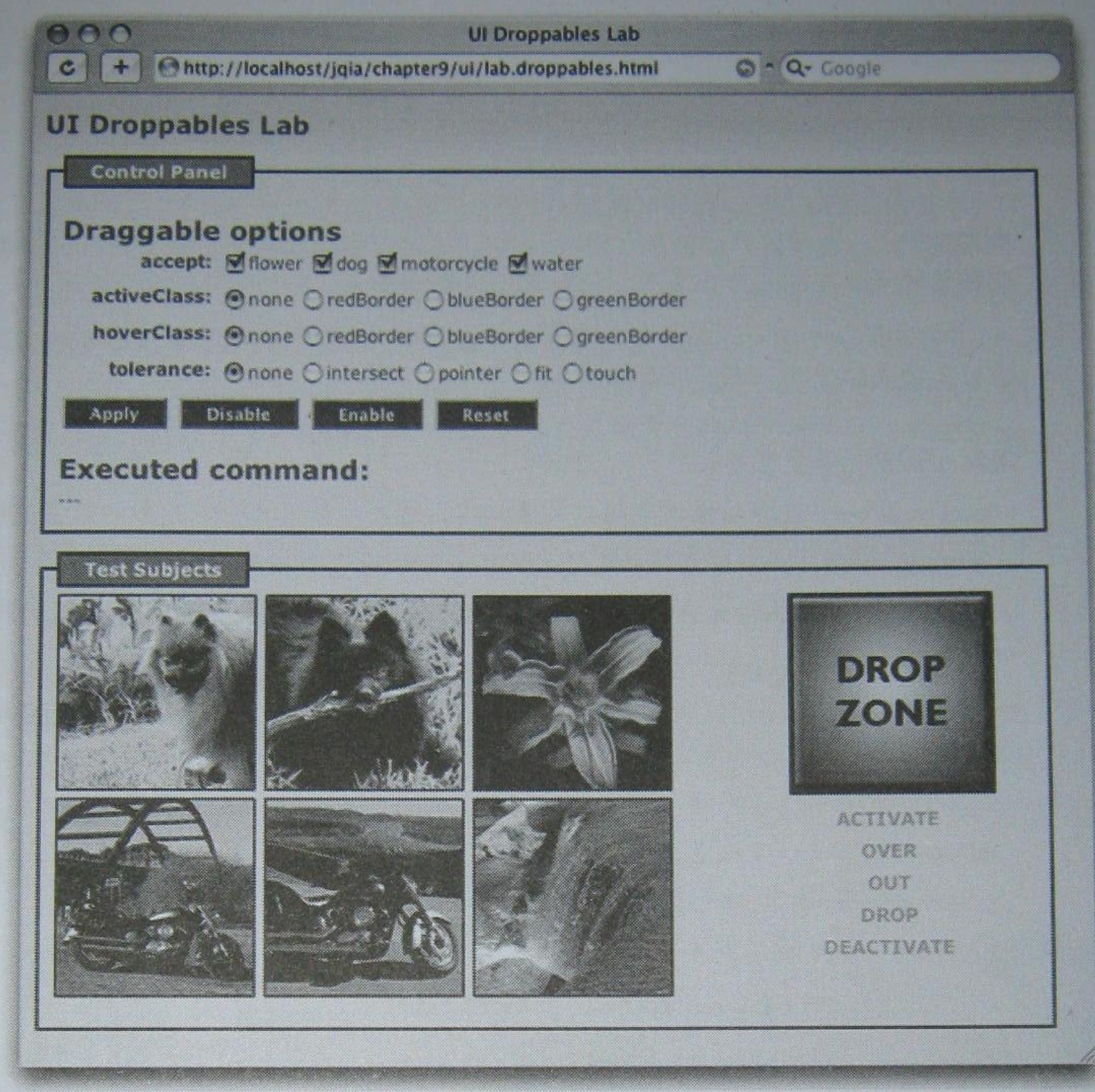


图9-12　UI可放置元素实验室页面用来检查不同的选项对拖放操作所起的作用

与其他实验室页面相似，控制面板（Control Panel）允许指定点击按钮Apply之后将要应用到可放置元素的各种选项。按钮Disable、Enable和Reset与可拖动元素实验室页面的对应按钮所提供的功能是一样的。

311

在测试对象窗格（Test Subjects Pane）里，有6个可拖动元素和一个元素（称之为Drop Zone，这个元素在点击按钮Apply之后将变为可放置元素）。拖放区域（Drop Zone）的下方是已变灰色的文本元素，用来显示Activate、Over、Out、Drop和Deactivate。当可放置元素的对应回调函数（在幕后被实验室添加到可放置元素上）被调用时，适当的文本元素（我们称之为回调函数指示器）即刻被突出显示以便指示某个回调函数已经触发。

312

下面利用这个实验室页面来挖掘可放置元素的本质。

练习1——在这个练习里，我们开始熟悉accept选项，这个选项告诉可放置元素，什么构成适宜（或可接受）的可拖动元素。虽然这个选项可以被设置为任何jQuery选择器（或是一个函数，能够以编程方式来测定适宜性）。为了实验室页面，我们专注于拥有特定类名称的元素。具体地说，通过选中accept选项控件的适当的复选框，可以指定选择器（包含类名称flower、dog、motorcycle和water中的任何一个或多个）。

测试对象（Test Subject）窗格的左边是6个可拖动图像元素，根据在图像里出现的事物，每

个图像元素都被指派这些类名称中的一个或两个。例如，左上方的可拖动元素拥有类名称dog和flower（因为狗和花都出现在图像里），然而中下方的图像被定义为带有类名称motorcycle和water（准确地说是雅马哈V-Star摩托车和科罗拉多河）。

在点击按钮Apply之前，试着把这些图像元素中的任何一个拖放到拖放区域（Drop Zone）。除拖动以外，没有发生别的什么事情。仔细观察回调函数指示器，注意其没有什么变化。这毫不奇怪，因为从一开始在页面上就不存在可放置元素。

现在使所有的控件保持初始状态（包括使所有的accept复选框处于选中状态），点击按钮Apply。已执行的命令包含accept选项，指定匹配所有4个类名称的选择器。

再次试着拖动任何一个图像到拖放区域，同时观察回调函数指示器。这一次将会看到：当你开始移动任何一个图像时，Activate指示器即刻突出显示，或称为跳动，表明了可放置元素已注意到拖动操作开始，也表明了可拖动元素的放置是可接受的。

在拖放区域进进出出地拖动图像多次。在适当的时候over和out回调函数被调用（如对应的指示器所示）。接着在拖放区域范围之外把图像放下，并观察Deactivate指示器的跳动。

最后，重复拖动操作，不同之处是这次把图像放置到拖放区域上面。Drop指示器发生跳动（指示drop回调函数已被调用）。也请注意：拖放区域被设置为显示最近放置到它上面的图像。

练习2——取消选中所有的accept复选框，并点击按钮Apply。无论你选择把哪个图像放置到拖放区域上面，没有回调函数指示器发生跳动，也没有什么事情发生。如果没有针对性的accept选项，拖放区域就变成一个（冒充的）好人。 313

练习3——试着选中至少一个accept复选框，假定是Flower，请注意只有包含花朵的图像（这些图像为页面所知，因为它们的类名称里包含flower字样）被构造为可接受元素。

再一次尽情地尝试可接受类名称的各种组合，直到你对accept选项的概念感到轻松。

练习4——使控件复位，选中activeClass的单选按钮greenBorder，然后点击按钮Apply。这给可放置元素提供activeClass选项，即指定（你猜对了）用来定义绿色边框的类名称。

现在，当你开始往可放置元素拖动可接受元素（被选项accept所定义）时，拖放区域周围的黑色边框被替换为绿色边框。如果你在这个练习中遇到麻烦，请注意这一点：必须小心CSS优先规则。当activeClass类名称被应用时，必须能够覆盖你所要替换的、用来指派默认可视化呈现的CSS规则。对于hoverClass来说也是这样。

练习5——选中hoverClass的标注为RedBorder的单选按钮，并点击按钮Apply。在可接受图像被拖动经过拖放区域的上方时，边框从绿色（由于把activeClass设置为greenBorder）变为红色[①]。

试验activeClass和hoverClass这两个选项的不同设置，直到你轻松地掌握了在拖动操作中这两个选项各自触发CSS类变换的时机。

练习6——在这个练习里，分别选择不同的tolerance单选按钮，并且注意当可放置元素从活动态转换到武装态时（如同表9-3所描述的tolerance选项的定义）这些设置产生什么影响。

9

① 原书的这个练习实验效果不明显，建议把activeClass设置为redBorder，而把hoverClass设置为blueBorder。这样设置之后，在IE7和Firefox 2.0里测试效果则很明显。——译者注

通过设置选项hoverClass或在Over回调函数指示器跳动时，可以轻松地观察到状态转换。

就像对应的可拖动元素那样，可放置元素可以被毁坏、暂时禁用，然后重新启用。实现这些操作的方法如下。

命令语法：droppableDestroy

droppableDestroy()

删除包装集里所有元素的作为可放置元素的功能。

参数

无

返回

包装集

命令语法：droppableDisable

droppableDisable()

挂起包装集里可放置元素的可放置性，而不删除可放置性的信息或选项。

参数

无

返回

包装集

命令语法：droppableEnable

droppableEnable()

还原已被droppableDisable()所禁用的包装集里任何可放置元素的可放置性。

参数

无

返回

包装集

拖放是多用途的有用交互技术。拖放常被用于指示关联关系，也可以用于重排元素的顺序。后一种用法非常普遍，因此理所当然得到UI插件的直接支持。请看下文。

3．其他鼠标交互方法

在UI插件的鼠标交互分类里还有几类命令：sortable、selectable和resizable。这些方法利用拖放功能，允许在容器内将元素重新排序以及分别调整大小。

到目前为止，我们已讨论的UI插件命令，每一个都通过应用方法到包装集上并且传入散列对

象（用来指定各个选项）而进行操作。欲知详情，请参考http://docs.jquery.com/UI。

除了动态拖放以及其他鼠标交互操作以外，UI插件还提供许多小部件，用来扩展HTML本身提供的基本用户界面元素集。

315

9.4.2 UI 小部件和可视化效果

虽然我们乐意探讨UI插件所提供的众多的用户界面小部件的有关细节，但现实不允许（限于书本的篇幅）。取代任何深入的讨论，我们至少让你知道什么可用，因此你就知道从在线文档里查找些什么。

下面以简短的说明来列举可用的小部件。请访问http://docs.jquery.com/ui了解更多细节。

可折叠项——简单的小部件，利用简单标记比如列表或嵌套的 元素来创建可扩展的和可收缩的多个层次。

选项卡——小部件，利用相当有趣和具有潜在复杂性的一组选项来创建选项卡。<a>元素用来指定选项卡，<a>的href特性用来标识选项卡节（利用页内散列引用）。这个小部件可以用于回退按钮，也可以在页面加载时配置为打开特定的选项卡。

日历——小部件，提供动态的日期挑选器（与一个<input>元素相关联）。这个小部件是高度可配置的，可以显示在页面上或显示在对话框内。

对话框——“模式对话框”小部件，带有移动和调整大小的功能。

滑块——创建滑块的控件（与桌面应用所提供的滑块相似），可以通过隐藏元素而集成到表单里。该控件可以面向多种方式，其最小值和最大值是完全可配置的。

表格——可排序的表格小部件，被认为是健壮和快速的。

除以上小部件以外，提供下列可视化效果：

阴影——为指定的元素而生成投影。

放大镜——当鼠标光标贴近时使元素的内容放大。

有以上这些UI元素在握，我们在创建大的富因特网应用时获得许多选择权。正如他们所说：“等一等！还有更多！”jQuery社区乐于分享对jQuery进行扩展和增强的成果。请访问http://jquery.com/plugins，了解其他jQuery用户所发布的许许多多的插件。

9.5 小结

从一开始，jQuery的创建者从设计上支持简便却稳健的插件体系结构。其想法是核心jQuery下载将保持小而灵活，只提供大多数的Web应用作者所需的那些核心功能，而把其他功能留给在需要时可以包含进来的插件。这个策略很好地服务于jQuery，因为jQuery用户社区已经创建和发布任何jQuery用户都可以下载和使用的一大批插件。

316

对更为常用的一些插件进行调查，我们看到增强核心jQuery功能集的广泛功能。

表单插件提供包装器方法，允许以受控的方式来处理表单元素，甚至允许把表单设置为通过Ajax请求而简便地提交，而不是整页刷新的传统表单提交方式。

获得DOM元素的精确位置和尺寸（或粗略的估计数据），是尺寸插件所提供的功能。如果试图相对于另一元素或页面原点而在页面上精确地放置元素，则这个功能是不可或缺的。

另一个不可或缺的插件是实时查询插件，允许给尚不存在的元素注册事件处理程序。这个看起来非因果关系的功能对于在页面生存期内要求频繁地创建和销毁DOM元素的页面来说，是一个巨大的优势。

令人遗憾的是我们没能深入探讨UI插件。UI插件提供那种不可或缺的用户界面功能，如拖放、排序，以及几个有用的用户界面小部件。

而那仅仅是个开端。请访问http://jquery.com/plugins，了解一系列其他可用插件。插件数目时时刻刻都在不停地增长！

9.6　尾声

尾声？完全不是！

即使在本书范围内我们已经提供全部的jQuery API，也无法向你逐一展示在页面上使用这个广泛API的所有方式。我们所提供的示例都是精心挑选的，以便引导你一步一步探索如何使用jQuery来解决每一天在Web应用页面上所遇到的问题。

jQuery是一个充满活力的项目。糟糕，在编写本书的过程中，本书的作者为了跟上jQuery的飞速发展步伐，付出了相当多的艰辛和努力。核心jQuery库正在不断发展，以成为更加有用的资源，而且几乎每天都在涌现越来越多的插件。

我们鼓励你跟踪jQuery社区的发展，并且真诚地希望本书能够给你很大的帮助，促使你以更短的时间和更少的代码来编写出更好的富因特网应用（比你认为可能的代码数量还要少）。

我们祝你健康幸福，并祝愿你轻松地解决掉所有的bug！

附录 A

JavaScript必知必会

附录内容

- 哪些JavaScript概念对于有效地利用jQuery来说很重要
- 关于JavaScript `Object`的基础知识
- 为什么说函数是一等对象
- 决定（以及控制）`this`表示什么
- 什么是闭包

jQuery给Web应用带来的最大好处之一是实现大量的脚本启用行为而无需亲自编写一大堆的脚本。jQuery处理具体细节，以便我们能够专注于使Web应用为其所应为！

在本书的最初几章里，我们只需要初级的JavaScript技巧以便理解和编写那些示例。讲到关于高级主题的各章比如事件处理、动画和Ajax，我们必须理解几个根本的JavaScript概念，以便有效地利用jQuery库。可能你会发现在JavaScript里原以为理所当然（或盲目相信）的许多事情开始变得更有意义。

我们不准备事无巨细地研究所有的JavaScript概念——那可不是本书的目的。本书的宗旨是帮助读者在尽可能短的时间内有效地运用jQuery。归根结底，我们将专注于在Web应用里为了最有效地利用jQuery而必知必会的根本概念。

这些最为重要的概念围绕着JavaScript以哪一种方式定义和处理函数而展开，尤其是以什么方式定义和处理JavaScript里的“一等对象”（函数）。这意味着什么呢？好，为了理解一个函数如何成为一个对象，先抛开“一等对象”不说，我们必须确保首先理解JavaScript对象究竟是指什么。下面开始深入探讨。

A.1 JavaScript 对象的基本原理

大多数的面向对象语言都定义一个特定的根本`Object`类型，所有其他对象都起源于这个`Object`。同样地在JavaScript中，根本的`Object`也作为所有其他对象的基础，不过在这里就应该停止这种比较了。从基本的层面来说，JavaScript的`Object`与其他兄弟面向对象语言所定义的根本对象，几乎没有什么共同之处。

看第一眼，JavaScript的`Object`看起来似乎令人厌烦并且平淡无奇。一旦创建，它不持有任何数据，而且几乎不表现什么语义。不过那些有限的语义确实给予它很大的潜力。

下面看个究竟。

A.1.1 对象怎样成为对象

新对象经由new操作符以及与其相伴的Object构造器而产生。创建一个对象就像下列语句那么简单：

320

```
var shinyAndNew = new Object();
```

还可以更简便（不久就会看到），不过目前这样就行了。

但利用这个新对象能够做些什么呢？它看起来什么也没有包含：没有信息、没有复杂的语义，一切皆无。我们的闪亮的全新对象不会变得有趣，直到我们开始给它添加东西——称为属性的东西。

A.1.2 对象的属性

就像服务器端的对象，JavaScript对象也可以包含数据和方法（哦……有点超前了）。不像那些服务器端的同胞，这些元素不是为对象而预先声明的。我们随着需要动态地创建它们。

请看下面的代码片段：

```
var ride = new Object();
ride.make = 'Yamaha';
ride.model = 'V-Star Silverado 1100';
ride.year = 2005;
ride.purchased = new Date(2005,3,12);
```

我们创建新的Object实例并把它指派到名为ride的变量。然后用几个不同类型的属性来填充这个变量：两个字符串、一个数字以及一个Date类型的实例。

我们无需在赋值之前声明这些属性，这些属性不过是通过赋值而产生的。那是给予我们高度灵活性的非常强大的“符咒”。在我们“得意忘形”之前，请记住：灵活性总是与代价为伴！

例如，假定在页面的后续部分，我们想要改变购买日期的值：

```
ride.purchased = new Date(2005,2,1);
```

没问题……除非我们无意中打错字了，比如

```
ride.purcahsed = new Date(2005,2,1);
```

没有编译器会警告：已经犯了一个错误。名为purcahsed的新属性被我们高高兴兴地创建了，可后来却使我们纳闷：在引用正确拼写的purchased属性时为什么新日期无效呢？

与更大的权力相伴的是更大的责任。（我们在哪里似曾听闻？）那么打字就要小心点！

注意 JavaScript调试器（如Firefox的Firebug）在处理这种问题时可以说是“救生员”。因为使用Firebug调试器，类似这样的打字错误通常不会导致JavaScript错误，而依赖于JavaScript控制台或错误对话框通常收效甚微。

从示例中我们得知JavaScript Object的实例（从此往后简称为对象）是属性的集合，每一个属性都由名称和值构成。属性的名称是字符串，而属性的值可以是任何JavaScript对象，可以是

Number、String、Date、Array、基本的Object或任何其他JavaScript对象类型（很快就会看到，也可以是函数）。

这就使得Object实例的主要目的就是用作其他对象的已命名集合。这可能使你想起在其他语言里的概念：例如Java里的映射，或其他语言里的字典或散列。

在引用属性时，可以连续地引用对象（作为父对象的属性）的属性。假设给ride实例添加新属性以便保存机车的所有者。这个属性是另一个JavaScript对象，包含属性，如所有者的姓名和职业：

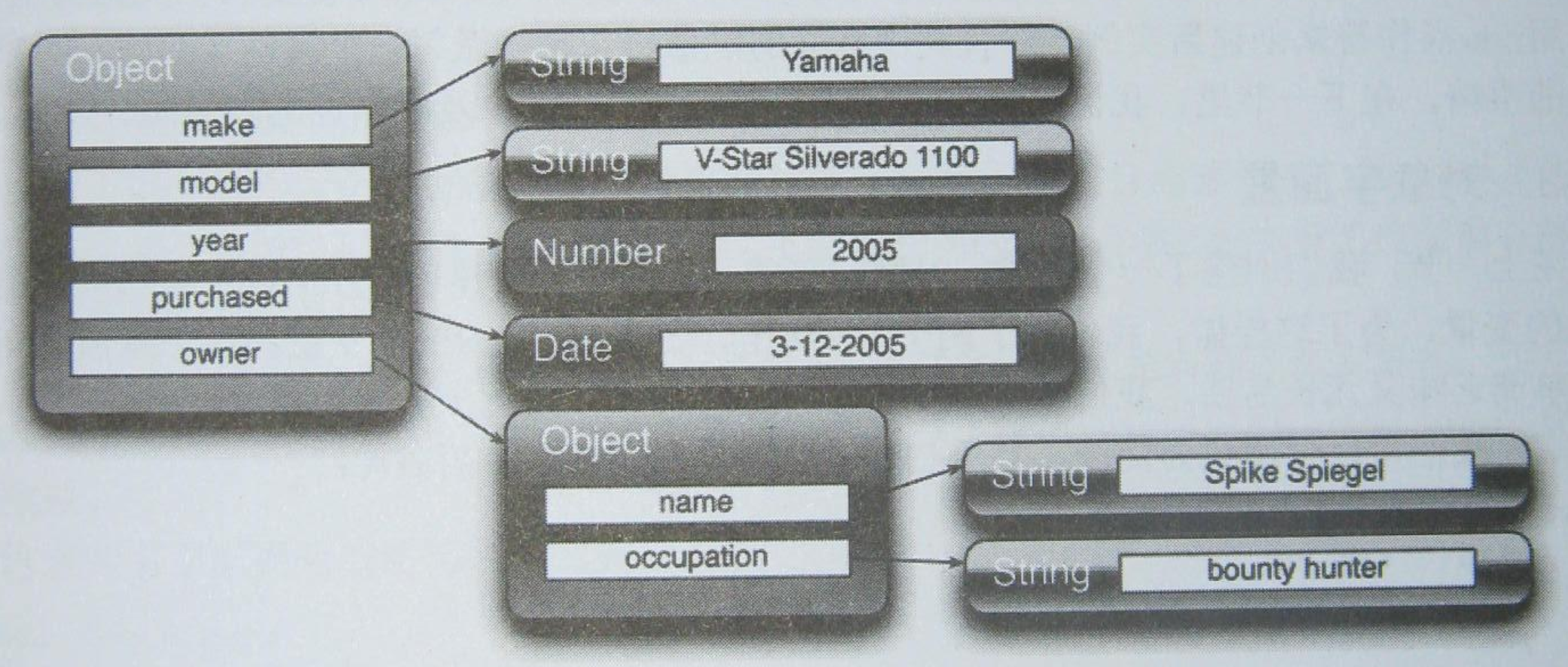
```
var owner = new Object();
owner.name = 'Spike Spiegel';
owner.occupation = 'bounty hunter';
ride.owner = owner;
```

为了访问嵌套的属性，我们编写

```
var ownerName = ride.owner.name;
```

可以使用的嵌套层次是没有限制的（只要不超出常识的限度）。到目前为止，已完成的对象层次结构看起来如图A-1所示。

注意每一个值都是JavaScript类型的一个截然不同的实例。



图A-1　对象层次结构显示Object是其他Object或JavaScript内建对象的已命名引用的容器

顺便说一下，在这些代码片段里创建用于说明目的的所有中间变量（如owner）不是必须的。不久我们将看到声明对象及其属性的更为高效和简洁的方式。

到目前为止，我们用点（英文句号）操作符来引用对象的属性。但那原来是执行属性引用的更为通用的操作符的同义词。

例如，如果有一个属性名为color.scheme，会怎样呢？你注意到这个名称里的圆点了吗？它在操作中抛出破坏性的错误，因为JavaScript解释器将试着把scheme当成color的内嵌属性进行查找。

你说："好，只要不那样做就行！"不过空格符如何呢？可能被误认为是分隔符而不是名称的一部分的其他字符呢？最重要的是，如果我们不知道属性名称是什么，却把它当作另一个变量里

的值，或把它当作表达式的求值结果？

对于所有这些情况，点操作符是不足胜任的，因此必须以更为通用的表示法来访问属性。通用的属性引用操作符的格式为

```
object[propertyNameExpression]
```

在那里propertyNameExpression是JavaScript表达式，其求值结果作为字符串而形成将被引用的属性的名称。例如，下列3个引用是等效的：

```
ride.make
ride['make']
ride['m'+'a'+'k'+'e']
```

也等价于

```
var p = 'make';
ride[p];
```

对于其名称并不构成有效JavaScript标识符的属性来说，利用通用的引用操作符是引用该属性的唯一途径，比如

```
ride["a property name that's rather odd!"]
```

其属性名称包含对于JavaScript标识符来说不合法的字符（或属性名称是其他变量的值）。

用new操作符来创建新实例，并且利用独立的赋值语句来指派每一个属性从而建立对象，是繁琐的事情。在下一节里，我们探讨更为简洁和易读的声明对象及其属性的表示法。

A.1.3 对象字面量

在上一节，我们创建了一个对象，对摩托车的一些属性进行建模，并把这个对象指派给名为ride的变量。为了这样做，我们用了两个new操作、名为owner的中间变量和几个赋值语句。这样既单调乏味又冗长易错，我们难以在快速地检查代码时从视觉上把握对象的结构。

幸亏我们可以使用更为简洁、易于扫读的表示法。思考下面的语句：

```
var ride = {
  make: 'Yamaha',
  model: 'V-Star Silverado 1100',
  year: 2005,
  purchased: new Date(2005,3,12),
  owner: {
    name: 'Spike Spiegel',
    occupation: 'bounty hunter'
  }
};
```

这个片段利用对象字面量来创建ride对象，与上一节里用赋值语句所创建的ride对象相同。

这个表示法称为JSON（JavaScript Object Notation，JavaScript对象表示法[1]），大多数页面作者对JSON的偏爱远远超过利用多个赋值语句来创建对象的办法。JSON结构简单：一对花括号所表示的对象，在其内部列举以逗号分隔的多个属性。通过列举以冒号分隔的名称和值来表示每一个属性。

---

① 你可以访问http://www.json.org/，以便了解更多信息。——译者注

注意　从技术上来说，JSON无法表达日期值，主要是因为JavaScript缺少任何种类的日期字面量。如同前面示例，通常在脚本中使用Date构造器来表达日期值。如果用作交换格式，日期常被表达为包含ISO 8601格式的字符串或数字，即Date.getTime()所返回的表达日期的毫秒数。

324

从owner属性的声明可见，对象声明可被嵌套。

顺便说一下，也可以在JSON里通过在方括号内放置逗号分隔的元素列表来表达数组，如下：

```
var someValues = [2,3,5,7,11,13,17,19,23,29,31,37];
```

从这一节所提供的示例可见，对象引用经常存储在变量或其他对象的属性里。下面探究后一个应用场景的特殊情况。

A.1.4　对象作为窗口属性

到目前为止，我们已经看到把一个引用存储到JavaScript对象有两种方式：变量和属性。存储引用的这两种方式使用不同的表示法，如同下面的片段所示：

```
var aVariable =
  'Before I teamed up with you, I led quite a normal life.';

someObject.aProperty =
  'You move that line as you see fit for yourself.';
```

这两个语句利用赋值操作给变量和对象属性分别指派由字面量创建的String实例（如果你能够识别这两个晦涩的引文的出处，荣誉归你，不可以利用Google作弊！在本章的前面留有线索）。

不过这两个语句真的执行不同的操作吗？结果证明不是！

如果在顶层（在任何包含函数体之外）使用var关键字，那只是对程序员友好的、引用预定义JavaScript window对象的属性的表示法。在顶层作用域里生成的任何引用，在window实例上隐式地生成。

这意味着下面所有的语句是等效的：

```
var foo = bar;
```

和

```
window.foo = bar;
```

以及

```
foo = bar;
```

不管使用哪一个表示法，都创建了window的名为foo的属性（如果foo尚未存在），并指派bar的值。也请注意：因为未限定bar，所以假定它是window的一个属性。

325

把顶层作用域认为是window作用域，大概不会使我们陷入概念上的麻烦，因为处在顶层的任何未限定的引用被假定为window属性。当我们深入钻研函数体时，用来决定作用域的规则变得更为复杂——实际上复杂得多——不过我们很快就会解决这个问题。

以上涵盖了JavaScript Object概览的大部分。从这个讨论中总结出来的重要概念是：

- JavaScript对象是属性的无序集合；
- 属性由名称和值所构成；
- 对象可以利用对象字面量来声明；
- 顶层变量是window的属性。

如果我们把JavaScript函数称为一等对象，这是什么意思呢？请看下文。

A.2 函数作为一等公民

在许多传统面向对象语言里，对象可以包含数据，并且可以拥有方法。在这些语言里数据和方法通常是截然不同的概念，而JavaScript走的是不同的道路。

在JavaScript里函数被认为是对象，与JavaScript里所定义的任何其他对象类型一样，比如String、Number或Date。就像其他对象那样，函数也是通过JavaScript构造器来定义。通过这种方式，Function可以

- 被指派给变量；
- 被指派为对象的属性；
- 被传入参数；
- 作为函数结果返回；
- 用字面量来创建。

因为对待函数的方式与JavaScript语言里的其他对象相同，所以我们说函数是一等对象。

不过你可能暗自寻思：函数与其他对象类型比如String或Number根本不相同，因为函数不仅拥有值（Function实例的值是函数体），还拥有名称。

26 噢，没那么快呢！

A.2.1 名称里面是什么

大部分的JavaScript程序员在错误的假设（认为函数是已命名实体）下操作。函数并非已命名实体。如果你是这样的程序员之一，你已被绝地武士的精神控制所愚弄了[1]。正如对象的其他实例（例如String、Date或Number），函数只有在被指派给变量、属性或者参数时，才被引用。

思考Number类型的对象。我们经常通过字面量表示法来表达Number实例。下面的语句

```
213;
```

是完全有效的，但它也是完全无用的。Number实例不是那么有用，除非把它指派给属性或变量，或绑定到参数名称上。否则，我们无法引用无实体的实例。

同样的规则也适用于Function对象的实例。

“但，但，但是……”你可能说，“下面的代码如何？”

```
function doSomethingWonderful() {
  alert('does something wonderful');
}
```

“那不是创建名为doSomethingWonderful的函数吗？”

---

① Jedi mind trick，即电影《星球大战》的绝地武士的精神控制。——译者注

不，不是。虽然那个表示法可能看起来熟悉，并且普遍用于创建顶层函数，但同样的语法糖也被var用于创建window的属性。function关键字自动地创建Function实例并且把它指派给利用函数“名称”所创建的window的属性（前面称之为绝地武士之精神控制）如下：

```
doSomethingWonderful = function() {
  alert('does something wonderful');
}
```

如果你觉得不可思议，就请考虑另一个语句，使用完全相同的语法，除了这一次使用Number字面量以外：

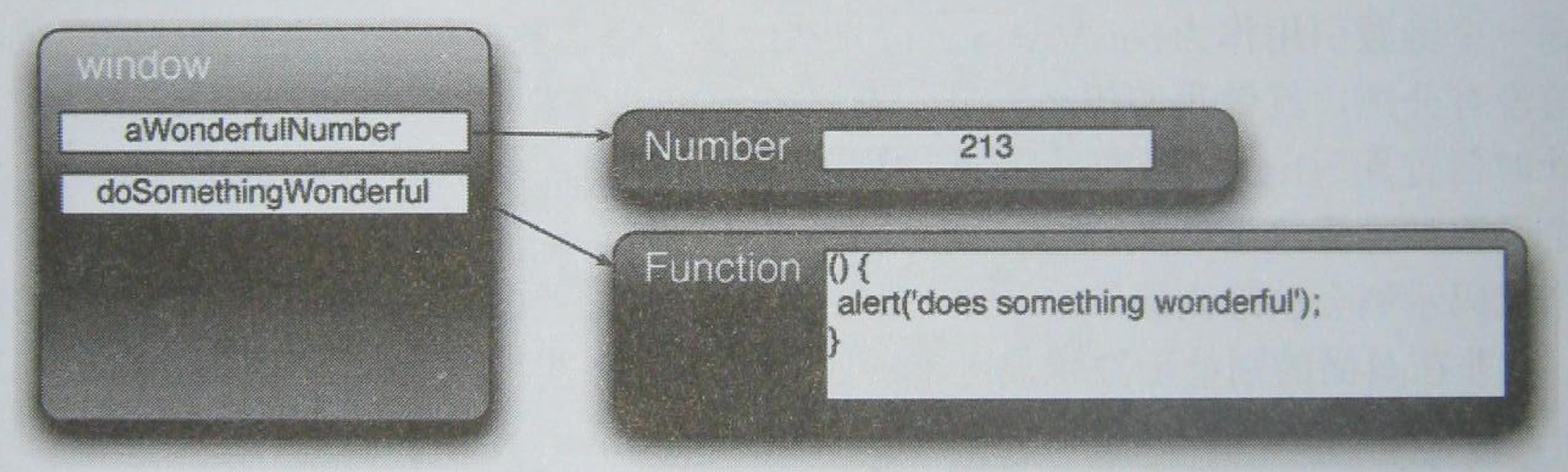
```
aWonderfulNumber = 213;
```

这个语句毫不稀奇，把函数指派给顶层变量（众所周知的window的属性）与这个语句没有什么不同：函数字面量被用来创建Function实例，然后指派给doSomethingWonderful变量，与数字字面量213被用来指派Number实例给aWonderfulNumber变量的方式是一样的。

如果你从没看过函数字面量的语法，也许觉得奇怪。它是由关键字function、紧跟的圆括号所包括的参数列表以及函数体而构成的。

327

如果我们声明已命名的顶层函数，则Function实例被创建并指派给window的属性（利用所谓的函数名称而自动地创建这个属性）。Function实例不再拥有名称，而是拥有数字字面量或字符串字面量。图A-2说明了这个概念。



图A-2　函数实例是无名对象，就像数字213或任何其他JavaScript值。只有通过生成引用，它才被命名

Gecko浏览器和函数名称

基于Gecko布局引擎的浏览器，如Firefox和Camino，利用顶层语法把已定义函数的名称存储在函数实例的、名为name的非标准属性里。虽然可能对于一般的公共开发而言，这用处不大，尤其是它受限于基于Gecko的浏览器，但是对于浏览器插件和调试器的编写者来说，这是非常有价值的。

请记住：在HTML页面上创建顶层变量时，变量被创建为window实例的属性。因此，下面的语句是完全等效的：

```
function hello(){ alert('Hi there!'); }

hello = function(){ alert('Hi there!'); }

window.hello = function(){ alert('Hi there!'); }
```

虽然看起来像语法“魔术”，但理解这一点很重要：正如其他对象类型的实例那样，Function实例是值，因此可以指派给变量、属性或参数。并且就像其他那些对象类型，无实体的无名实例毫无用处，除非它们被指派给变量、属性或参数。只有通过变量、属性或参数，它们才能被引用。

我们已经看到指派函数给变量和属性的示例，不过传递函数作为参数如何呢？下面看为什么要那样做以及怎样做。

328

A.2.2 作为回调函数

在代码跟随井然有序的同步流时，顶层函数当然好。但HTML页面的本质是一旦加载之后，就远不是同步的。不管处理事件、创建计时器，或发起Ajax请求，网页代码的本质是异步的。在异步编程中最为流行的概念是回调函数。

举个计时器的示例。可以使计时器触发（假定5秒钟之后），通过传递适当的持续时间值给window.setTimeout()方法。但是，那个方法如何让我们知道何时计时器已经过期，让我们不用空等而可以做任何其他的事情？它通过调用我们所提供的函数来达到目的。

思考下面的代码：

```
function hello() { alert('Hi there!'); }

setTimeout(hello,5000);
```

我们定义名为hello的函数并且设置计时器在5秒钟（第二个参数表示5 000毫秒）之后触发。我们传递一个函数引用作为setTimeout()方法的第一个参数。传递一个函数作为参数与传递任何其他值没有分别，就像我们传递一个Number作为第二个参数。

在计时器过期时hello函数被调用。因为setTimeout()方法在代码中往回调用一个函数，所以那个函数被称为回调函数。

这个代码示例会被最高级的JavaScript程序员认为是幼稚的，因为hello这个名称的创建是不必要的。除非在页面的别处函数将被调用，否则没有必要创建作为window属性的hello用来暂时地存储Function实例，以便传递hello作为回调参数。

编写这个片段的更加简洁的方式是：

```
setTimeout(function() { alert('Hi there!'); },5000);
```

这个语句在参数列表中直接地表达函数字面量，不产生不需要的名称。这是在jQuery代码中经常看到的习惯用法（在不必把函数实例指派给顶层属性的时候）。

迄今为止我们在示例中所创建的函数要么是顶层函数，要么在函数调用里被指派给参数（我们知道顶层函数本质上是顶层window属性）。我们也可以指派Function实例给对象的属性，在那里事情真正地变得有趣起来。请继续阅读。

9

A.2.3 this 到底是什么

面向对象语言在方法内自动提供办法来引用对象的当前实例。在Java和C++那样的语言里，名为this的变量指向对象的当前实例。在JavaScript里，存在类似的概念，甚至使用相同关键字this，同样提供办法以便存取与函数相关联的对象。但是面向对象程序员要小心！在微妙却意义重大的多个方面，this的JavaScript实现不同于面向对象语言里的this。

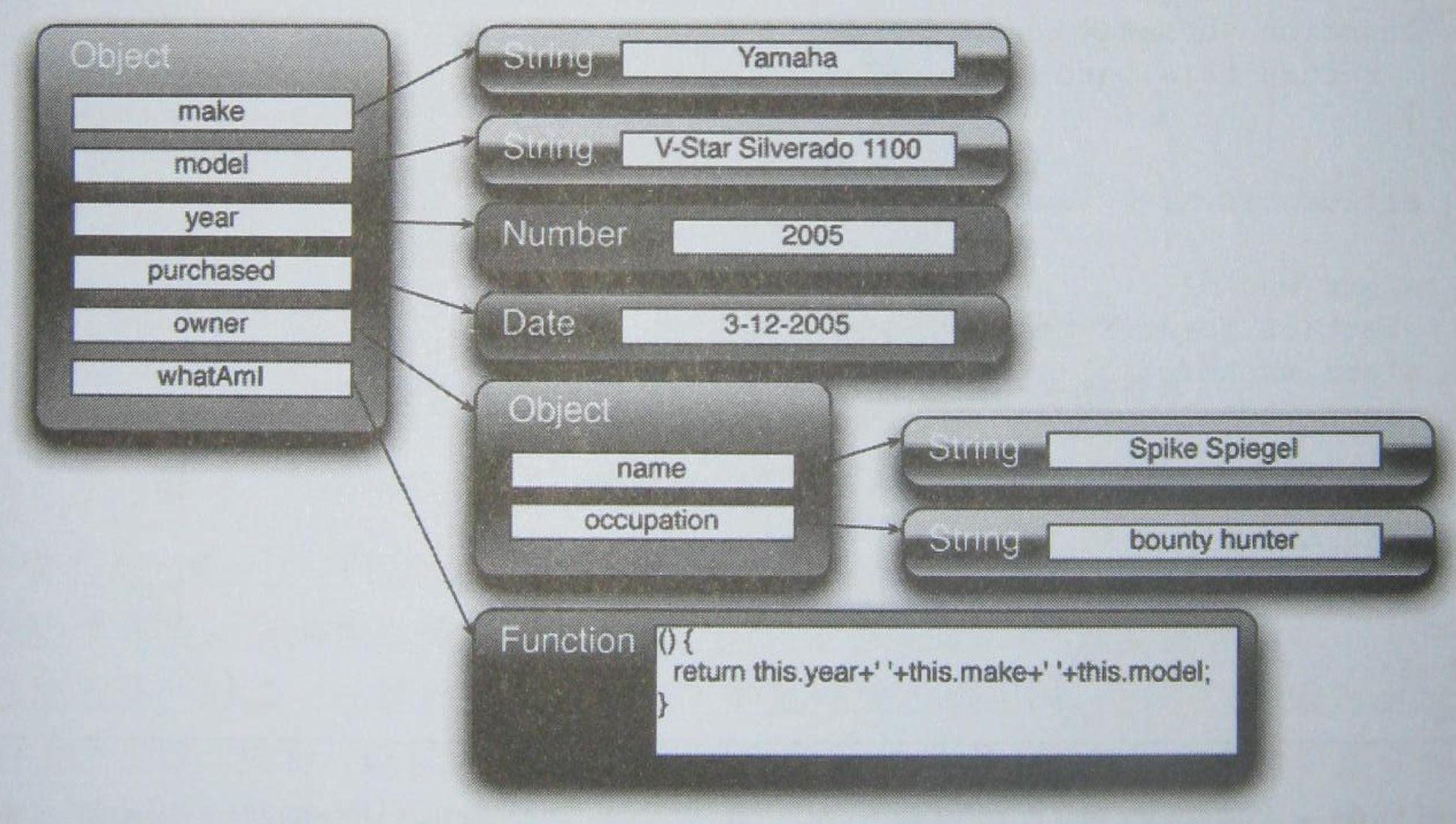
在基于类的面向对象语言里，this指针通常引用类的实例（方法在类中被声明）。在JavaScript中，函数是一等对象（不被声明为任何东西的一部分），而this所引用的对象被称为函数上下文，它不是由如何声明函数，而是由如何调用函数来决定。

这意味着，根据函数如何被调用，同一个函数可以拥有不同的函数上下文。刚开始可能觉得奇怪，但这相当有用。

在默认情况下，函数调用的上下文（this）是对象，其属性包含可以调用该函数的引用。为了演示，我们回顾摩托车示例，修改对象的创建如下（添加之处以粗体来突出显示）：

```
var ride = {
  make: 'Yamaha',
  model: 'V-Star Silverado 1100',
  year: 2005,
  purchased: new Date(2005,3,12),
  owner: {name: 'Spike Spiegel',occupation: 'bounty hunter'},
  whatAmI: function() {
    return this.year+' '+this.make+' '+this.model;
  }
};
```

我们给原始示例代码添加名为whatAmI的属性（这个属性引用Function实例）。新的对象层次结构（其中Function实例被指派给名为whatAmI的属性）如图A-3所示。



图A-3　这个模型清晰地显示函数不是Object的一部分，只是被名为whatAmI的Object属性所引用

当函数通过属性引用而被调用时，如下所示

```
var bike = ride.whatAmI();
```

函数上下文（this引用）被设置为ride所指向的对象实例。作为结果，变量bike被设置为字符串2005 Yamaha V-Star Silverado 1100，因为该函数经由this进行调用并获得对象的属性。

330

对于顶层函数来说也是如此。请记住：顶层函数是window的属性，因此在作为顶层函数而被调用时，其函数上下文是window对象。

虽然以上是常见的隐式行为，但JavaScript提供办法来显式地控制什么被用作函数上下文。通过Function方法call()或apply()来调用函数，可以把函数上下文设置为所想的任何东西。

对，作为一等对象，函数甚至拥有定义为Function构造器的几个方法。

用call()方法来调用函数（这个函数作为第一个参数），指定对象作为函数上下文，而其余参数成为被调用函数的参数，也就是说，call()的第二个参数成为被调用函数的第一个实参，以此类推。apply()方法的工作方式与call()相似，除了第二个参数要求是对象数组以外（这些对象将成为被调用函数的实参）。

弄糊涂了？该是介绍更为综合的示例的时间了。考虑代码清单A-1的代码（在可下载的代码里找到appendixA/function.context.html）。

331

代码清单A-1　演示函数上下文的值依赖于函数如何被调用

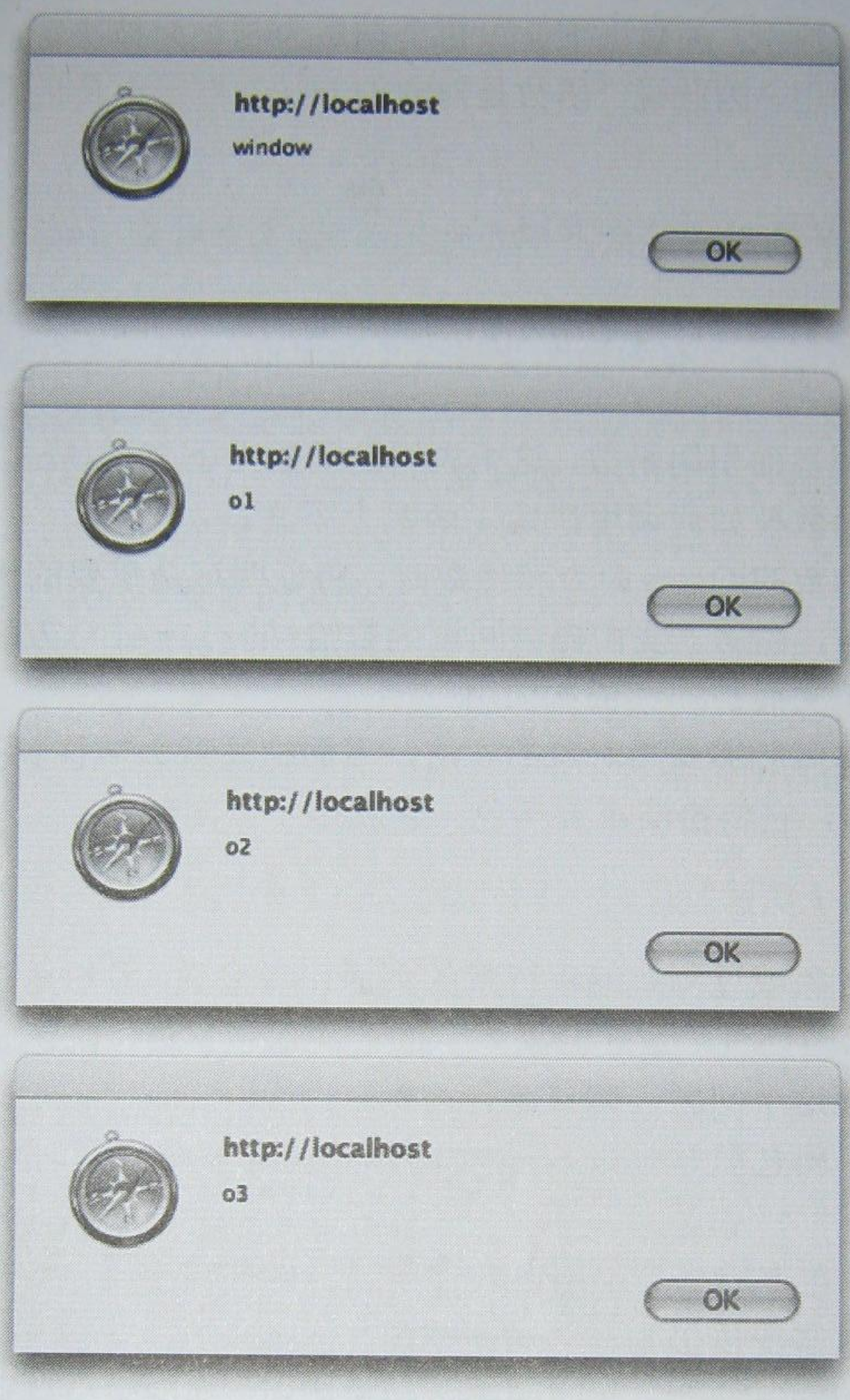
```
<html>
  <head>
    <title>Function Context Example</title>
    <script>
      var o1 = {handle:'o1'};
      var o2 = {handle:'o2'};
      var o3 = {handle:'o3'};
      window.handle = 'window';

      function whoAmI() {
        return this.handle;
      }

      o1.identifyMe = whoAmI;

      alert(whoAmI());
      alert(o1.identifyMe());
      alert(whoAmI.call(o2));
      alert(whoAmI.apply(o3));

    </script>
  </head>

  <body>
  </body>
</html>
```

在这个示例中，我们定义3个简单的对象，每个对象都带有handle属性，使得易于标识对象（给定一个引用）❶。我们也给window实例添加handle属性，因此window实例也易于标识。

然后定义顶层函数，无论什么对象作为其函数上下文，都返回handle属性值❷。把同一函数实例指派给o1对象的名为identifyMe的属性❸。我们可以说，这会在o1对象上创建名为identifyMe的方法，而注意到这一点很重要：这个函数独立于o1对象而被声明。

最后，我们弹出4个警告消息框，每个警告消息框都使用不同的机制来调用相同的函数实例。在用浏览器加载这个页面时，4个警告消息框的序列如图A-4所示。

332



图A-4　作为函数上下文的对象随着函数被调用的方式不同而发生变化

这个警告消息框的序列说明了以下几点。

- 在函数作为顶层函数而被直接调用时，函数上下文是window实例❹。
- 在函数作为对象（在这种情况下是o1）的属性而被调用时，那个对象成为函数调用的函数上下文❺。我们可以说，函数就像那个对象的*方法*而（在面向对象语言里类似）进行操作。请不要对这个类比感到厌烦。如果你不小心谨慎，就会误入歧途，就像这个示例其余部分所显示的那样。
- 采用Function的call()方法导致函数上下文被设置为作为第一个参数而传递给call()的任何对象。在这种情况下是o2❻。在这个示例中，函数表现得像是o2的方法，即使这个函数与o2没有任何关联（连o2的属性都不是）。
- 就像对待call()那样，利用Function的apply()方法来把函数上下文设置为作为第一个参数而传入的任何对象❼。在传递更多的参数给函数时（为了简洁，在这个示例中没有那样做），这两个方法之间的差别才变得显著起来。

这个示例页面清楚地演示：函数上下文以每个调用为基础而被决定，并且一个函数能以任何对象作为其上下文而被调用。因此说“函数是对象的方法”肯定不是正确的。下面的陈述则准确得多：

在对象o充当函数F的调用的函数上下文时，函数F充当对象o的方法。

为了更进一步说明这个概念，考虑给示例添加下面的语句所带来的效果：

```
alert(o1.identifyMe.call(o3));
```

即使我们作为o1的属性而引用函数，这个调用的函数上下文还是o3。这就进一步强调：不是函数被如何声明，而是函数被如何调用决定了函数上下文。

在使用采取了回调函数的jQuery命令和函数时，将证明这是重要的概念。我们早在2.3.3节就看到这个概念的实际操作，在那里我们提供回调函数给$的filter()方法，而那个函数以包装集的各个元素作为函数上下文而被连续依次调用。

既然我们理解函数如何能够充当对象的方法，下面把注意力转移到另一个高级的函数专题，在jQuery的有效利用方面，它将扮演重要角色，它就是——闭包。

A.2.4　闭包

对于拥有传统面向对象或过程式编程背景的页面作者来说，闭包是让人领会起来常常觉得奇怪的概念。然而，对于那些拥有函数式编程背景的页面作者来说，闭包是令人感到熟悉和惬意的概念。首先给初学者回答这个问题：闭包是什么？

尽可能简单地表述，闭包就是Function实例，外加对于Function实例的执行来说必需的、来自环境的本地变量。

在声明函数时，可以在声明之处引用在其作用域内的任何变量。甚至在声明之处已经超出作用域而关闭声明之后，这些变量仍为该函数所支持。

对于编写有效的JavaScript来说，在回调函数被声明时，回调函数能够引用有效本地变量是不可或缺的关键。再一次利用计时器，请看代码清单A-2里的说明示例（找到文件appendixA/closure.html）。

代码清单A-2　通过闭包获得函数声明的环境的存取

```
<html>
  <head>
    <title>Closure Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.js"></script>
    <script>
      $(function(){
        var local = 1;                    ❶
        window.setInterval(function(){    ❷
          $('#display')
            .append('<div>At '+new Date()+' local='+local+'</div>');
          local++;                        ❸
        },3000);
      });
```

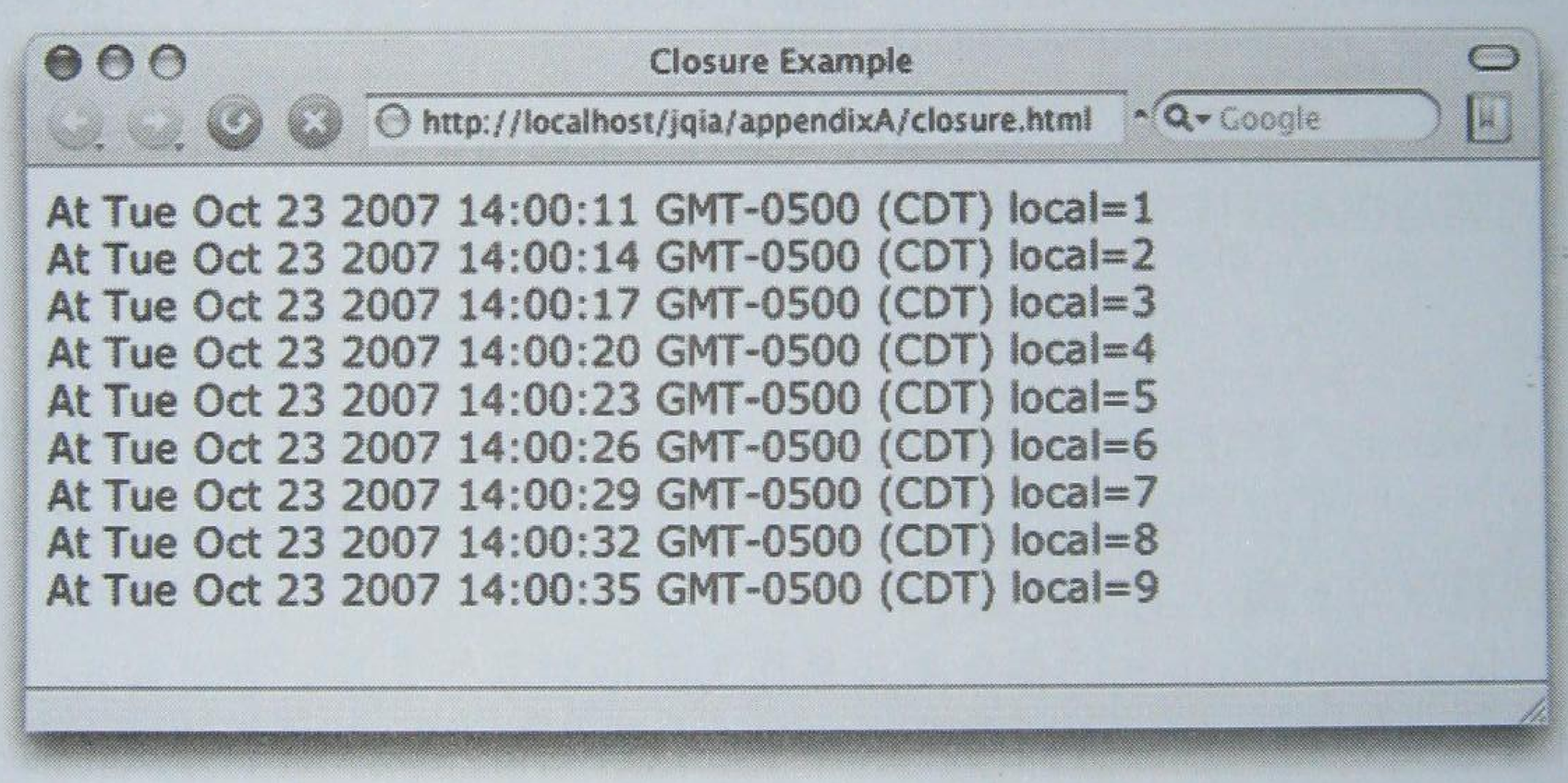
```
    </script>
  </head>

  <body>
    <div id="display"></div>    ◁—❹
  </body>
</html>
```

在这个示例中，我们定义在DOM加载之后触发的就绪处理程序。在这个处理程序里，我们定义名为local的本地变量❶并且给它指派数字值1。然后用window.setInterval()方法来建立每3秒就触发一次的计时器❷。作为计时器的回调函数，我们指定内联函数，引用local变量并且显示当前时间以及local的值（通过把 元素写入在页体内定义的名为display的元素）❹。作为回调函数的一部分，local变量的值也被增加了❸。

335

在运行这个示例之前，如果我们不熟悉闭包，可能查看这段代码并发现一些问题。我们可能猜测：因为回调函数将在页面加载之后3秒内触发（在就绪处理程序执行完毕之后很久），所以在回调函数执行期间，local的值是未定义的。毕竟，local声明所在的块在就绪处理程序结束时超出了作用域，不是吗？

但是在加载页面之后，让其运行短短的一段时间，我们看到图A-5所示的画面。



图A-5　闭包允许回调函数访问其环境，即使那个环境已经超出作用域

它工作起来了！但是如何工作的呢？

当就绪处理程序退出时，虽然local声明所在的块确实超出了作用域，但是函数声明所创建的闭包（包括local），在该函数的生命期内保持在作用域内。

注意　你可能已经注意到：闭包，JavaScript里所有的闭包，被隐式地创建，而不像其他支持闭包的语言那样，需要显式语法。这是一把双刃剑，一方面使得易于创建闭包（不管你有意或无意），但另一方面使得在代码中难以发现闭包。

无意的闭包可能带来意外的后果。例如，循环引用可以导致内存泄漏。内存泄漏的典型示例是创建向后引用闭包变量的DOM元素，阻止了那些变量的回收。

闭包的另一重要功能是，函数上下文决不被包含为闭包的一部分。例如，执行下列代码的结果会出乎我们所料：

```
...
this.id = 'someID';
$('*').each(function(){
   alert(this.id);
});
```

请记住：每个函数调用都有各自的函数上下文，因此，在上面的代码中，给each()传递的函数上下文在回调函数内是来自jQuery包装集的元素，而不是被设置为'someID'的外部函数属性。每个回调函数的调用依次弹出警告消息框，显示包装集的各个元素的id。

如果需要访问在外部函数里作为函数上下文的对象，可以采取普通的习惯用法：在本地变量里创建this引用的副本，这个副本将被包含在闭包里。考虑示例的下列变化：

```
this.id = 'someID';
var outer = this;
$('*').each(function(){
   alert(outer.id);
});
```

本地变量outer被指派外部函数的函数上下文的引用，成为闭包的一部分并且可以在回调函数中被存取。改动后的代码现在弹出警告消息框，显示字符串'someID'，包装集里有多少元素就弹出多少次警告消息框。

在使用jQuery命令（这些命令利用异步回调函数）来创建典型的代码时，我们将发现闭包是绝对必要的，尤其是在Ajax请求和事件处理的领域。

A.3 小结

JavaScript是Web里广泛使用的语言，不过编写JavaScript的页面作者多半常常用得不是那么有深度。在本章里，我们介绍了这门语言的较深的一些方面。为了在页面上有效地使用jQuery，我们必须理解这些较深的方面。

我们看到，JavaScript Object的存在主要是作为其他对象的容器。考虑到Object实例是“名称/值对”的无序集合，如果你拥有面向对象背景，就会觉得与原来所想的对象大不相同。但即使是在编写不太复杂的JavaScript时，把握这个概念也是很重要的。

337

在JavaScript里函数是“一等公民”，能以类似于其他对象类型的方法来声明和引用。可以用字面量表示法来声明函数，把函数存储在变量里和对象属性里，甚至可以充当回调函数，把函数作为参数而传递给其他函数。

术语“函数上下文”是指在一个函数的调用期间，被this指针所引用的对象。虽然通过把对象设置为函数上下文，函数可以像对象的方法那样进行操作，但是函数并没有被声明为任何一个对象的方法。调用方式（可由调用者显式地控制）决定调用的函数上下文。

最后，我们看到函数声明及其环境如何形成闭包。在后续调用时，闭包允许该函数访问成为闭包的一部分的那些本地变量。

牢牢地把握这些概念，我们准备直面挑战，直面在页面上利用jQuery来编写有效的JavaScript时遇到的挑战。

338

索　引

索引中的页码为英文原书的页码，与书中边栏的页码一致。

符号

$

as function namespace（作为函数命名空间），154
as namespace prefix（作为命名空间前缀），9
avoiding name collisions（避免命名冲突），189
defining locally（局部定义），164
in plugins（在插件里），189
in ready handler （在就绪处理程序里），165
naming conflicts（命名冲突），14
sharing with other libraries（与其他库共享），164

`$()`，6
adding wrapper methods（添加包装器方法），216
for element creation（为了创建元素），11

`$.ajax()`，249, 261, 278
`$.ajaxSetup()`，252
`$.browser` 157, 247
`$.each()`，169
`$.extend()`，177, 191, 210, 261
`$.fn` 199, 246, 259
`$.get()`，236, 252
`$.getJSON()`，237, 241
`$.getScript()`，180
`$.grep()`，170
`$.inArray()`，175
`$.livequery.run()`，294
`$.makeArray()`，175
`$.map()`，172
`$.noConflict()`，164, 189, 192
`$.post()`，248, 252
`$.styleFloat` 163
`$.trim()`，9, 168
`$.unique()`，176

A

`abbr` element（`<abbr>`元素），262
ActiveX control（ActiveX控件），218
Adaptive Path（Adaptive Path公司），218
`add()`，36, 247
`addClass()`，59
adding a wrapper method（添加包装器方法），199-201
adding select options（添加`<select>`的选项`<option>`），159
`after()`，73
Ajax（异步JavaScript和XML），218-226
comprehensive（综合），258-266
comprehensive jQuery API（综合jQuery API），249-252
diagrammed（图解），223
Form Plugin, using（使用表单插件），276-284
form submission via（表单提交经由）276-284
GET（GET请求），235-236
Global Callback Info object（全局回调信息对象）253
global functions（全局函数），254-258
growth catalyst（成长催化剂），2
HTTP requests（HTTP请求），233-234
iframes, using（使用`<iframe>`），218
initilizing（初始化），219
loading content（加载内容），226
loading scripts（加载脚本），181
POST requests（POST请求），248-249
ready state handler（就绪状态处理程序），221
request life cycle（请求生命周期），223
request parameters（请求参数）221
responses（响应），223-224
responseText，223
responseXML，223
setting defaults（设置默认值），252-253
uploading files（上传文件），284
XML, not using（不使用XML），223
ajaxForm Lab（ajaxForm实验室），282, 284
`ajaxForm()`，282
`ajaxFormUnbind()`，282
ajaxSubmit Lab（ajaxSubmit实验室），279-280
`ajaxSubmit()`，277, 279
alpha filters（alpha滤镜），62
`andSelf()`，47
`animate()`，145, 148
animations（动画）

CSS properties（CSS属性），146
custom drop（自定义坠落动画），148-150
custom fading（自定义褪色动画），147
custom puff（自定义消散动画），150-151
custom scale（自定义放大动画），148
easing （曲线缓和），147
stopping（停止），145
using Flash（使用Flash），127
anonymous event handler（匿名事件处理程序）86
anonymous functions（匿名函数），165
anonymous listeners（匿名监听器），86
API. 参见jQuery API
`append()`，70
appending content （追加内容），70, 72
`appendTo()`，71
appetizer menu（开胃菜菜单），113-124
Application Programming Interface（应用编程接口）参见 jQuery API
arrays, filtering（数组，筛选），170-172
assigning properties（指派属性），51
Asynchronous JavaScript and XML（异步JavaScript和XML），参见Ajax
attr()，52, 54-55
attribute selectors（特性选择器），22
attributes（特性），49
applying（应用），56-58
diagrammed（图解），50
Internet Explorer limitations（IE的限制），56
normalized values（规范化值），53
removing（删除），56
setting（设置），54-56
values, fetching（获取值），52-54
augmenting wrapped sets （增大包装集）38

B

Basic Event Model（基本事件模型），84
`before()`，73
behavior, separating from structure（行为从结构分离）4
`bind()`，98, 293
black box（黑盒），225
`blur()`，102, 107
Boot Closet page（靴子储藏室页面），264-266
box model （方框模型）161, 286
diagrammed（图解），162
browser detection（浏览器检测），155-161
alternatives（其他选择），156-157
jQuery flags（jQuery标志），157-163
bubble phase （冒泡阶段），94
bubbling（冒泡），88-90
stopping（停止），91

C

callback functions（回调函数），329-330
Camino，155-160, 328
reloading（重新加载），306
capability detection（功能检测），156
capture phase（捕获阶段），94
cascading dropdowns（层叠下拉列表），237-245
Cascading Style Sheets（层叠样式表），2
class names（类名称），58, 67
comma operator（逗号操作符），38
chain（链），7
managing（管理），45-47
chaining（链接），45-47
`change()`，102, 231
child selector（子选择器），21
`children()`，43
class names（类名称）
adding, removing（添加、删除），58-60
fetching（获取），67
Clear and Reset Lab（清除和复位实验室），274
`clearForm()`，275
`click()`，102, 107
`clone()`，45, 78
closures（闭包），84, 90, 211, 334-337
code consistency, reusability（代码一致性、可重用性），186
collapsible list, implementing（实现可折叠列表），128-134
collecting properties（收集属性），52
collisions, naming（命名冲突），187-189
command chain（命令链），7
managing（管理），45-47
commands（命令），17
commerce（商业），229
container selector（容器选择器），23
`contains()`，44
content scrolling（内容滚动），287
`contents()`，43
copying（复制）
address information（地址信息），202-206
elements（元素）78
creating DOM elements（创建DOM元素），11
creating utility functions（创建实用工具函数），194,196-199
cross-browser Ajax（跨浏览器Ajax），219
CSS
absolute positioning（绝对定位），150
hiding elements（隐藏元素），128

inline vs block（内联与块），128
offset parent（偏移父元素），289
opacity（不透明度），142, 147
overflow（溢出），289
relative positioning（相对定位），149
styling lists（设置列表的样式），132
也请参见Cascading Style Sheet（层叠样式表）
`css()`，61-62, 132, 200
CSS3，6
custom animations（自定义动画），145-151
custom attributes（自定义特性），52-53
custom properties（自定义属性），211
custom selectors（自定义选择器），27-30

D

data conversion（数据变换），173
data translation（数据转换），172-176
Date（JavaScript的Date类型），195
date formatting（日期格式化），195-199
`dblclick()`，102
defining functions（定义函数），193
dependent dropdowns（从属下拉列表），237
DHTML，49
dimensions（尺寸），286-289
Dimensions Plugin（尺寸插件）285-291
`height()`，285
`innerHeight()`，286
`innerWidth()`，286
`offset()`，290
`offsetParent()`，290
`outerHeight()`，286
`outerWidth()`，286
`position()`，290
`scrollLeft()`，287
`scrollTop()`，287
`width()`，285
Dimensions Plugin Scrolling Lab（尺寸插件滚动实验室），287
exercises（练习），289
disabling form elements（禁止表单元素），13
DOCTYPE，161
DOM（Document Object Model，文档对象模型），10, 17, 49-58
cloning elements（克隆元素），78
copying elements（复制元素），70, 78-80
creating new elements（创建新元素），11
event bubbling（事件冒泡），88-90
form elements（表单元素），79-80
generating elements（生成元素），31-32
manipulation（操作），68
moving elements（移动元素），70-80
NodeList（节点列表），50
removing elements（删除元素），76-77
replacing elements（替换元素），77
setting content（设置内容），68
wrapping elements（包裹元素），75
document ready handler（文档就绪处理程序），9
document scrolling（文档滚动），288
DOM elements, positioning（DOM元素，定位），285-291
DOM Level 0（DOM第0层），85-87
DOM Level 1（DOM第1层），84
DOM Level 2（DOM第2层）91-96
DOM manipulation（DOM操作），258
DOM，参见Document Object Model (DOM)，
Double Submit Problem（双重提交问题），57
drag-and-drop（拖放），300
`draggable()`，300, 304
`draggableDestroy()`，303
`draggableDisable()`，304
`draggableEnable()`，304
draggables（可拖动元素），300-308
axis of movement（移动轴），307
constraining（约束），307
containment（包含），307
grid（网格），308
required files（必需的文件），301
suitability concept（适用性概念），309
dropdowns（下拉列表），237
`droppable()`，308
`droppableDestroy()`，314
`droppableDisable()`，315
`droppableEnable()`，315
droppables（可放置元素），308-315
drop event（放置事件），309
options（选项），309
required files（必需的文件），308
state transition diagram（状态转换图），309
states（状态），309
dynamically loading scripts（动态加载脚本），180-183

E

`each()`，13, 51, 90, 148, 200, 211, 246, 337
easing（曲线缓和），147
Easing Plugin（曲线缓和插件），147
effects（效果）127
custom drop（自定义坠落效果），148

custom fading（自定义褪色效果），147
custom puff（自定义消散效果），150
custom scale（自定义放大效果），148
fading（褪色效果），140
hide（隐藏效果），127
show（显示效果），127
sliding（滑动效果），143-144
elements（元素）
abbr（<abbr>元素），262
animating（动画），135-145
attributes（特性），49-58
cloning（克隆），78
content（内容），68-70
copying（复制），70, 78-80
event handlers（事件处理程序），85
fading（褪色），140
form elements（表单元素），79-80
listeners（监听器），85
moving（移动），70-80
properties（属性），49-58
removing（删除），76-77
replacing（替换），77
selecting（选择），17
setting content（设置内容），68
showing and hiding（显示和隐藏），127-130
sliding（滑动），143-144
span（元素），262
styling（设置样式），58-68
title attribute（标题特性），258
toggling（切换），134
wrapping（包裹），75
empty()，77
emptying select elements（清空<select>元素），246
encodeURIComponent()，221
end()，46
error callback（错误回调），253
error()，102
event handlers（事件处理程序），83
anonymous（匿名），86
as attributes（作为特性），86
removing（删除），103
toggling（切换），108-110
event handling（事件处理）
hovering（悬停），110-112
proactive（主动），292-293
Event instance（Event实例），87, 104
cancelBubble（取消冒泡），91
normalizing（规范化），104
preventDefault()，106
stopPropagation()，91, 106
target（目标），107
event models（事件模型）
Basic（基本），85-87
DOM Level 0（DOM第0层），85-87
DOM Level 2（DOM第2层），91-96
Internet Explorer（IE浏览器），97
jQuery 98-124
Netscape 85-87
event propagation（事件传播），106
diagrammed（图解），95
event, toggling（事件，切换），108
event-driven interfaces（事件驱动界面），83
events（事件），112-124
addEventListener()，92
attachEvent()，97
binding（绑定），98
bubble phase（冒泡阶段），94
bubbling（冒泡），88-90
cancelling propagation（取消传播），91
capture phase（捕获阶段），94
establishing multiple（建立多个事件），92-94
Event instance（Event实例），88
key codes（键代码），105
modifier keys（修改键），105
propagation（传播），94-96
semantic actions（语义操作），106
srcElement，88
stopping propagation（停止传播），106
target element（目标元素），88
triggering（触发），106-107
expando（自定义属性），211
expire()，295
extending jQuery（扩展jQuery），13, 186-216
comprehensive examples（综合示例），245
defining wrapper methods（定义包装器方法），199-216
emptying select elements（清空<select>元素），246
implementation functions（实现函数），212-214
in $ namespace（在$命名空间内），192-193
loading select elements（加载<select>元素），246
motivations（动机），186-187
naming files（给文件命名），188
The Termifier，260-264
utility functions（实用工具函数），192-193
extending objects（扩展对象），176-180

F

fadeIn()，141

fadeOut()，141
fadeTo()，142
fieldSerialize()，272
fieldValue()，270-272
filesystem browsing（浏览文件系统），128
filter selectors（筛选选择器），29
filter()，41, 202
filtering data（筛选数据），170-172
filtering wrapped sets（筛选包装集），40
find selectors（查找选择器），29
find()，44
Firebug 321
Firefox 155, 160, 321, 328
  reload vs. refresh（重新加载vs刷新），140
  reloading（重新加载），306
fixed-width output（定宽输出），193-194
flags（标志），154
  $.boxModel，161
  $.browser，157
  $.styleFloat，163
Flash 127
float styles, browser differences（浮动样式，浏览器差异）163
flyout（工具提示），258
focus()，102, 107
form controls（表单控件）
  clearing（清除），274-276
  resetting（复位），274-276
  serializing（序列化），272-274
  submission order（提交顺序），273
  successful concept（成功概念），270
form elements（表单元素），79-80
Form Plugin（表单插件），80, 269-284
  ajaxForm Lab（ajaxForm实验室），282
  ajaxForm()，282
  ajaxFormUnbind()，282
  ajaxSubmit Lab（ajaxSubmit实验室），279
  ajaxSubmit()，277
  Clear and Reset Lab（清除和复位实验室），274
  clearForm()，275
  download location（下载地址），269
  fieldSerialize()，272
  fieldValue()，271
  formSerialize()，272
  Get Form Value Lab（获得表单值实验室），271
  resetForm()，275
  uploading files（上传文件），284
form submission（表单提交）
  hijacking（拦截），281-284
  semantic events（语义事件），282
  via Ajax（经由Ajax），281-284
  via focusable element（经由可聚焦元素），284
formatting fixed-width output（格式化定宽输出），194
forms, serializing（表单，序列化），227
formSerialize()，272
Function（函数）
  apply()，331
  as first-class object（作为一等对象），326
  call()，331
  function keyword operation（函数关键字操作），327
  naming（命名），327-328
function context（函数上下文），84, 200, 330-331
functional programming（函数式编程），334
functions（函数）
  as callbacks（作为回调），329-330
  as methods（作为方法），328, 334
  assigning context（指派上下文），331
  context（上下文），330
  function literal（字面函数），327
  top-level（顶层），327

G

Gecko，158, 328
GET，221, 227, 234, 261
Get Form Value(s)，Lab（获取表单值实验室），271, 273
get()，35
GIF，266
global Ajax functions（全局Ajax函数），253-258
Global Callback Info object（全局回调信息对象），253
global flags（全局标志），154
global namespace（全局命名空间），14
  polluting（污染），198
Google，218
GUI（Graphical User Interface，图形用户界面），83
  drag-and-drop（拖放），300
grep，170
GUI，参见Graphical User Interface (GUI)

H

halting form submission（中止表单提交），91
hasClass()，67
hash as parameter（散列对象作为参数），190-192
height()，63-64, 285
hide()，127, 135
hover()，111
hovering（悬停），110-112
HTML 5 Draft Specification（HTML 5规范草案），262

HTML Specification（HTML规范），270
html()，7, 68
HTML, generation（HTML，生成），31-32
HTTP. 参见Hypertext Transfer Protocol (HTTP)，
HTTP（Hypertext Transfer Protocol，超文本传输协议），
83, 248
methods（方法），221, 227
effects on caching（对缓存有影响），234
status codes（状态码），222

I

idempotent requests（幂等请求），234
iframe（<iframe>元素），284
implementation functions（实现函数），213
index()，36
information overload（信息超载），128, 130
inheritance（继承），176
innerHeight()，286
innerWidth()，286
insertAfter()，73
insertBefore()，73
inserting dynamic HTML（插入动态HTML），74
Internet Explorer（IE浏览器），155, 158, 160
box model（方框模型），161
event handling limitations（事件处理限制）96
Event Model（事件模型），97
inverting selectors（反转选择器），29-30
is()，45, 67, 132, 135
iterating（迭代），51
properties and collections（属性与集合）169-170

J

Java Swing，83
JavaScript
. operator（点操作符），323
adding select options（添加<select>元素的<option>选项），158
advanced example（高级示例），258
closure variables（闭包变量），335
closures（闭包），84, 211, 334-337
creating objects（创建对象），320
Date（JavaScript的Date类型），195
dot operator（点操作符），323
dynamic creation of properties（属性的动态创建），321
essential concepts（必要的概念），320
extending objects（扩展对象），176
for loop（for循环），169
for-in loop（for-in循环），169
Function（函数），84
function contexts（函数上下文），84, 200
function keyword（函数关键字），327
functions（函数），326
general reference operator（通用引用操作符），323
getAttribute()，53
isNaN()，173
libraries（库），14
libraries, using with jQuery（使用其他库与jQuery一起）
163-167
NaN 173
navigator（JavaScript的navigator对象），155
new operator（new操作符），320
nonstandard property names（非标准属性名称），323
Number（JavaScript的Number类型），174
Object（Object对象），84, 320-326
object hash（散列对象），191
Object literals（字面对象），324-325
object properties（对象属性），321-323
object property diagram（对象属性图），322
object-oriented（面向对象），176
property references（属性引用），322
prototype property（原型属性），177
regular expressions（正则表达式），172, 198
setAttribute()，53
String（String类型），168, 172, 174
top-level scope（顶层作用域），325
Unobtrusive JavaScript（不唐突的JavaScript），3-5
var keyword explained（解释var关键字），325
window properties（window属性），325
XML and（XML以及……），223
Jedi mind tricks（武士的精神控制），327
Jesse James Garrett，218
jQuery，2-5
$()，6
add()，36
browser detection flags（浏览器检测标志），157-163
chaining（链接），7, 186, 200, 261
chains（链），45-47
commands（命令），17
contains()，44
CSS implementation（CSS实现），20
custom selectors（自定义选择器），27-30
document ready handler（文档就绪处理程序），9
DOM manipulation（DOM操作），17
essential JavaScript concepts（必要的JavaScript概念），
319
Event Model（事件模型），99
extending（扩展），12-13, 186-216

flags（标志），154
global Ajax commands（全局Ajax命令），254
manipulating objects（操作对象），167-180
plugins（插件），268-317
selectors（选择器），17-30
translating data（转换数据），172-176
trimming strings（修整字符串），168
using with other libraries（与其他库一起使用），14, 163-167
utility functions（实用工具函数），8, 154
wrapper（包装器），6-7
jQuery API
$.ajax()，249
$.ajaxSetup，252
$.boxModel，161
$.browser，157
$.each()，169
$.extend()，177
$.get()，234-236
$.getJSON()，237
$.getScript()，180
$.grep()，170
$.inArray()，175
$.makeArray()，175
$.map()，172
$.noConflict，164
$.post()，248
$.styleFloat，163
$.trim()，9, 168
$.unique()，176
addClass()，59
after()，73
ajaxComplete()，254
ajaxError()，254
ajaxSend()，254
ajaxStart()，254
ajaxStop()，254
ajaxSuccess()，254
andSelf()，47
animate()，145
append()，70
appendTo()，71
attr()，52, 54-55
before()，73
bind()，98
blur()，102, 107
change()，102
children()，43
click()，102, 107
clone()，78
contents()，43
css()，61-62
dblclick()，102
each()，13, 51
empty()，77
end()，46
error()，102
fadeIn()，141
fadeOut()，141
fadeTo()，142
filter()，41
find()，44
focus()，102, 107
get()，35
hasClass()，67
height()，63-64
hide()，135
hover()，111
html()，7, 68
index()，36
insertAfter()，73
insertBefore()，73
is()，45
jQuery()，6
keydown()，102
keypress()，102
keyup()，102
load()，102, 226
mousedown()，102
mousemove()，102
mouseout()，102
mouseup()，102
next()，43
nextAll()，43
noConflict()，14
not()，39
one()，102
parent()，43
parents()，43
prepend()，73
prependTo()，73
prev()，43
prevAll()，43
ready()，10
removeAttr()，56
removeClass()，59
resize()，102
scroll()，102

select()，102, 107
serialize()，227
serializeArray()，228
show()，136
siblings()，43
size()，34
slice()，42
slideDown()，143
slideToggle()，144
slideUp()，144
stop()，145
submit()，102, 107
toggle()，108, 137
toggleClass()，59
trigger()，106
unbind()，103
unload()，102
val()，79-80
width()，63-64
wrap()，75
jQuery Effects Lab（jQuery效果实验室），138-144
JSON（JavaScript Object Notation，JavaScript对象表示法），236, 241, 263, 277, 324
array example（数组示例），325
array notation（数组表示法），241
as Ajax response（作为Ajax响应），223
JavaScript Dates and object example（JavaScript日期和对象示例），324
JSP（Java Server Page，Java服务器页面），225

K

keydown()，102
keypress()，102
keyup()，102
Konqueror，161

L

Laboratory pages（实验室页面）
ajaxForm Lab（ajaxForm实验室），282
ajaxSubmit Lab（ajaxSubmit实验室），279
Clear and Reset Lab（清除和复位实验室），274
Dimensions Plugin Scrolling Lab（尺寸插件滚动实验室），287
Get Form Value Lab（获取表单值实验室），271
Get Form Values（获取表单值），271
Live Query Lab（实时查询实验室），296
Move and Copy Lab（移动和复制实验室），72
Selectors Lab（选择器实验室），17
UI Draggables Lab（UI可拖动元素实验室），304
UI Droppables Lab（UI可放置元素实验室），311
Wrapped Set Lab（包装集实验室），33
leveraging jQuery（利用jQuery），186
libraries, using with jQuery（利用其他库与jQuery一起），14
listeners（监听器），83
match and mismatch（匹配和不匹配），294
proactive（主动），292-293
Live Query Lab（实时查询实验室），296
Live Query Plugin（实时查询插件），292-299
$.livequery.run()，294
capabilities（功能），292
expire()，295
expiring listeners（使监听器过期），295
forcing evaluation（强制求值），294
livequery()，293-294
load()，102, 226, 231, 252, 263
loading content（加载内容）
Ajax，224-226
dynamic data（动态数据），229-233
jQuery，226-228
select elements（<select>元素），246

M

match handler（匹配处理程序），298
match listeners（匹配监听器），294
merging objects（合并对象），179
merging options（合并选项），210
method（方法），334
Microsoft（微软公司），218
MIME type in Ajax response（Ajax响应的MIME类型），223
mismatch handler（不匹配处理程序），298
mismatch listeners（不匹配监听器），294
mousedown()，102
mousemove()，102
mouseout()，102
mouseup()，102
Move and Copy Lab（移动和复制实验室），72
-moz-opacity，62
multi-browser support（多浏览器支持），159
multipart forms（多部分表单），284
multiple class names（多个类名称），58

N

name collisions（命名冲突），187-189
namespace, global（命名空间，全局），198
NaN，174

navigator object（navigator对象），155
nesting selectors（嵌套选择器），23
.NET framework（.NET框架），83
Netscape Communications Corporation（网景通信公司），84
Netscape Event Model（Netscape事件模型），84
Netscape Navigator，84
next()，43
nextAll()，43
noConflict()，14
NodeList（节点列表），50, 175
non-idempotent requests（非幂等请求），234
normalizing event targets（规范化事件目标），88
not()，39
Number.NaN，174

O

Object
　literals（字面量），324-325
　properties（属性），321-323
object detection（对象检测），156, 247
　feasibility（可行性），158
　for Ajax（用于Ajax），219
object hash（散列对象），190-192
object orientation（面向对象），176
object-oriented JavaScript（面向对象JavaScript），176
objects, extending（对象，扩展），176
Observable pattern（观察者模式），92
offset parent（偏移父元素），289
offset()，290
offsetParent()，290
offsets（偏移），289-291
OmniWeb 155, 157, 160
one()，102
online retailers（在线零售商），229
onload handler, alternative to（onload处理程序，替代……），9
onresize，65
Opera，155, 157, 160, 247
operator（操作符），323
options hash（选项的散列对象），190-192, 261
　extensive example（广泛示例），208
　used for wrapper method（用于包装器方法），261
OS X, dragging images from browser（OS X，从浏览器拖动图像），305
outerHeight()，286
outerWidth()，286
Outlook Web Access，218
overflow（溢出），289

P

page rendering（页面呈现），161
parameters（参数），190-192
parent()，43
parents()，43
parsing ZIP codes（解析ZIP编码），172
patterns（模式）
　necessity in Rich Internet Applications（在富因特网应用里使用jQuery的必要性），5
　Observable（可观察），92
　options hash（选项的散列对象），192
　progressive disclosure（渐进式公开），113
　repetitiveness（反复性），186
　responsibilities and（责任以及……），5
　structure and（结构以及……），4
Photomatic jQuery extension（针对照片的jQuery扩展），206-216
PHP，225
plain text response（普通文本响应），223
plugins（插件），268-317
　creating（创建），186-216
　Dimensions（尺寸），149
　Dimensions Plugin（尺寸插件），285-291
　Easing（曲线缓和），147
　Form Plugin（表单插件），269-284
　Live Query Plugin（实时查询插件），292-299
　on the web（在互联网上），269
　strategy（策略），269
　UI Plugin（UI插件），299-316
PNG files（PNG文件），266
position()，290
positional selectors（位置选择器），24-27
positioning（定位），289
positions（位置），289-291
POST，221, 227, 234, 248-249
prepend()，73
prependTo()，73
prev()，43
prevAll()，43
proactive event handling（主动事件处理），292
product description page（产品说明页面），231-233
progressive disclosure（渐进式公开），113, 128
propagation of events（事件的传播），94-96
properties（属性），49
　diagrammed（图解），50
　of JavaScript objects（JavaScript对象的属性），321-323

property referencing（属性引用），322
Prototype，192
  using with jQuery（与jQuery一起使用），14, 163-167
prototype（原型），177, 199

Q

query string（查询字符串），221, 227, 274
quirks mode（兼容模式），161

R

read-only status, applying（只读状态，应用），201-206
ready handler（就绪处理程序），165
ready state handler（就绪状态处理程序），222-223
`ready()`，10
real-time data（实时数据），229
regular expression（正则表达式），170
relational selectors（关系选择器），20
`removeAttr()`，56
`removeClass()`，59
removing elements（删除元素），77
removing wrapped set elements（删除包装集元素），39
rendering（呈现）
  quirks mode（兼容模式），161
  strict mode（严格模式），161
rendering engine（呈现引擎），158
replacing elements（替换元素），77
reporting Ajax errors（报告Ajax错误），253
request header（请求标头），155
request parameters（请求参数），221, 227, 236, 264
requests（请求）
  idempotent（幂等），234
  non-idempotent（非幂等），234, 248
`resetForm()`，275
`resize()`，102
resources（资源）
  Form Plugin download（表单插件下载），269
  jQuery plugins（jQuery插件），188
  plugins（插件），269
  quirksmode.org，161
  UI Plugin URL（UI插件URL），300
response（响应），223-224
JSON，236
responseText，220, 223, 226
responseXML，223
retail web sites（零售网站），229
reusability（可重用性），186
reusable components（可重用组件），167
RIA，参见Rich Internet Application
Rich Internet Application（富因特网应用）（RIA），2, 17, 92, 218, 237

S

Safari，155, 160, 247
  problems loading scripts（加载脚本时的问题），181
scripts, dynamic loading（动态加载脚本），180-183
`scroll()`，102
scrollbars（滚动条），288
scrolling dimensions（滚动尺寸），287-289
`scrollLeft()`，287
`scrollTop()`，287
`select()`，102, 107
selecting check boxes（选中复选框），27
selectors（选择器），3, 6, 17-30, 258
  attribute（特性选择器），20, 22-24
  basic（基本选择器），19-20
  child（子选择器），20-24
  container（容器选择器），20, 23-24
  CSS syntax（CSS语法选择器），17
  custom（自定义选择器），27-30
  filter（筛选选择器），29
  find（查找选择器），29
  form-related（表单相关选择器），29
  inverting（反转选择器），29-30
  nesting（嵌套选择器），23
  positional（位置选择器），24-27
  regular expression syntax（正则表达式语法选择器），22
  relational（关系选择器），20-24
  XPath plugin（XPath插件），30
Selectors Lab（选择器实验室），17
self-disabling form（自禁用表单），57
semantic actions（语义操作），83, 106
`serialize()`，227
`serializeArray()`，228
`serializeForm()`，276
server setup（Web服务器设置），225
server-side（服务器端）
  resources（资源），225
  state（状态），248
  templating（模板化），230
servlet（服务器端小程序），225
setting width（设置宽度），63
`show()`，127, 136
`siblings()`，43
`size()`，34
`slice()`，42
`slideDown()`，143

slideshows（幻灯片），206-209
slideToggle()，144
slideUp()，144
sniffing（探查），155-161
spoofing（欺骗），155
stop()，145
strict mode（严格模式），161
string trimming（字符串修整），168
styles, setting（样式，设置），61-67
styling（样式设置），58-68
submit()，102, 107
submitting forms（提交表单）
　Ajax，276-284
subsettting wrapped sets（获取包装集的子集），42-43

T

Termifier, The，259
test-driven development（测试驱动开发），208
text()，69
this，330-334
thumbnail images（缩略图），206
timers（计时器），83
title attribute（标题特性），258, 261
toggle()，108, 134-137
toggleClass()，59
toggling display state（切换显示状态），134
Tomcat web server（Tomcat Web服务器），225
tooltip（工具提示），258
top-level flags and functions（顶层标志和函数），154
translating data（转换数据），173
translation（转换），172-176
Trash（回收站），300
trigger()，106
trimming（修整），168

U

UI Draggables Lab（UI可拖动元素实验室），304-305
UI Droppables Lab（UI可放置元素实验室），311-313
UI Plugin（UI插件），299-316
　accordian（可折叠小部件），316
　calendar（日历），316
　dialog（对话框），316
　download location（下载地址），300
　draggable()，300
　draggableDestroy()，303
　draggableDisable()，304
　draggableEnable()，304
　draggables（可拖动元素），300-308
　drop shadow（投影），316
　drop zones（拖放区域），312
　droppable()，308
　droppableDestroy()，314
　droppableDisable()，315
　droppableEnable()，315
　droppables（可放置元素），308-315
　Droppables Lab（可放置元素实验室），311
　magnify（放大），316
　mouse interactions（鼠标交互），300-315
　required files（必须的文件），301
　resizables（可调整大小），315
　selectables（可选中），315
　slider（滑块），316
　sortables（可排序），315
　table（表格），316
　tabs（选项卡），316
　UI Draggables Lab（UI可拖动元素实验室），304
　widgets（小部件），316
UI principles（UI原则）
　gradual transition（逐渐过渡），135
　progrssive disclosure（渐进式公开），128
unbind()，103, 295
United States Postal Codes（美国邮政编码），172
unload()，102
Unobtrusive JavaScript（不唐突的JavaScript），3, 87,115, 230, 292
　practical application（实际应用），209
unsuccessful form elements（不成功的表单元素），79
uploading via Ajax（经由Ajax上传），284
URI（Universal Resource Identifier，通用资源标识符），
　encoding（URI编码），221, 227
URL（Uniform Resource Locator，统一资源定位符），221, 227
　encoding（编码），272
user agent detection（用户代理检测），155-161
user interface, annoyances（用户界面，困扰），49
utility functions（实用工具函数），8, 154

V

val()，79-80, 231, 270
variables as part of closure（变量作为闭包的一部分），335
viewport scrolling（视区滚动），287

W

W3C 159
　box model（方框模型），161
W3C DOM Specification（W3C DOM规范），84

Wastebasket（废纸篓），300
web server（Web服务器），225
width and height（宽度和高度），64
width()，63-64, 285
wiki（维基网站），57
window
  origin（原点），289
  properties（属性），325
  scrolling（滚动），288
window.event，87, 97, 104
window.setInterval()，335
window.setTimeout()，329
wrap()，75
wrapped set（包装集），7, 32, 43-44
  adding elements（添加元素），36-39
  as array（作为数组）34
  augmenting（增大），36-43
  determining size（确定大小），34
  filtering（筛选），40
  iterating over（迭代），51
  manipulation（操作），32
  obtaining elements from（从……获取元素），34-36
  removing elements（删除元素），39-42
  subsetting（获取子集），42-43
Wrapped Set Lab（包装集实验室），33
wrapper（包装器），7
wrapper methods（包装器方法）
  applying multiple operations（应用多个操作），201-206
  defining（定义），199-216
  implementation functions（实现函数），212-214
  mainaining state（维持状态），210

X

X11，83
XHR，参见XMLHttpRequest（XHR）
XHTML，53
XML，236, 277
XML DOM，223
XMLHTTP，218
XMLHttpRequest（XHR），218
  instance creation（实例创建），219
  instantiating（实例化），219
  making the request（生成请求），221-222
  methods and properties（方法和属性）219
  ready state handler（就绪状态处理程序），221
  responses（响应），223-224
  responseText，223
  responseXML，223
  status（状态），222
XPath selectors plugin（XPath选择器插件），30

Z

zebra-striping（斑马条纹），2
ZIP Codes（ZIP编码），172

"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。" ——John Resig，jQuery之父，《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

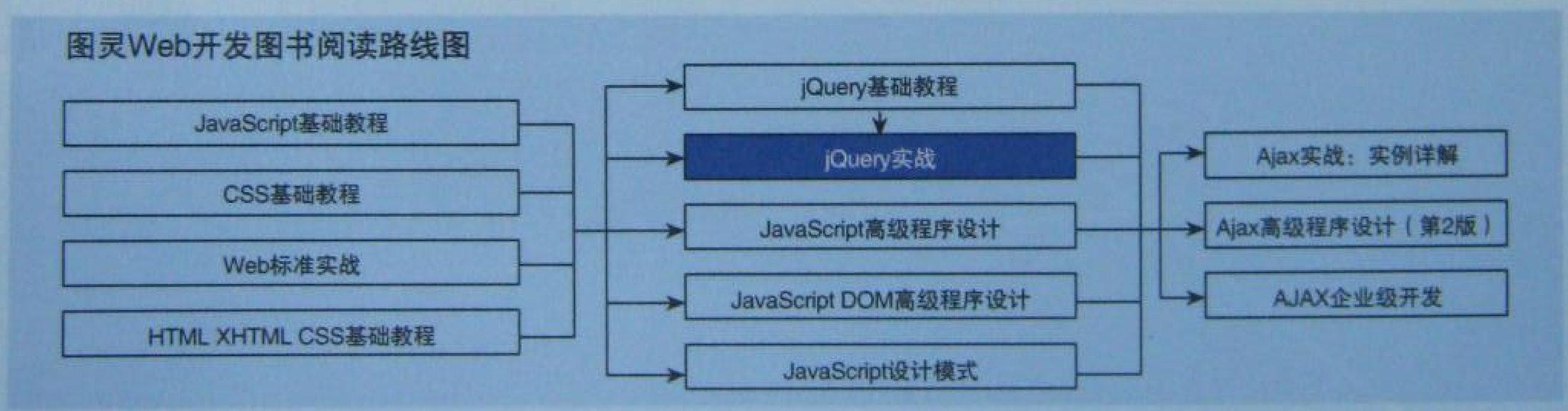
jQuery in Action
jQuery实战

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种策略和理论，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，只需要使用层叠样式表、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。

Bear Bibeault 著名Web技术专家，有30多年编程经验，也是技术社区JavaRanch的核心人物之一。除本书外，他还和其他世界级Web专家联袂打造了圣经级巨著《Ajax实战：实例详解》和《Ajax实战：Prototype与Scriptaculous篇》（均由人民邮电出版社出版）。

Yehuda Katz 著名Web技术专家，jQuery插件开发团队领导人，Merb等开源项目核心开发人员。他还维护着热门网站VisualjQuery.com。



MANNING

本书相关信息请访问：图灵网站 http://www.turingbook.com
读者/作者热线：(010)88593802
反馈/投稿/推荐信箱：contact@turingbook.com

上架建议 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 978-7-115-19599-9

ISBN 978-7-115-19599-9/TP

定价：49.00 元