O'reilly动物书

程序员的技术乐园

目 录

jQuery Cookbook

jQuery Cookbook

中文版

[美] *jQuery*社区专家组 著

姚军 孙博 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS

目 录

版权信息

书名：jQuery Cookbook中文版

ISBN：978-7-115-25590-7

- 著　　[美] jQuery社区专家组

  译　　姚　军　孙　博

  责任编辑　汪　振
- 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号

  邮编　100061　电子邮件　315@ptpress.com.cn

  网址　http://www.ptpress.com.cn
- 读者服务热线：(010)67132692

  反盗版热线：(010)67171154

版权声明

内容提要

本书旨在向读者展示业界领先的前端开发人员在日常项目中使用jQuery的方式和方法。全书共分18章，分别由不同的作者撰写。这些业界精英将带领读者经历由简到繁的各种问题的解决过程，不管是jQuery新手还是老练的JavaScript开发人员，都能通过本书对jQuery的能力有全新的认识。

本书从基础知识和通用最佳实践的介绍开始，内容包括在页面中包含jQuery、做出选择、遍历和操纵；接着转向实际用例，带领读者寻找常见问题的解决方案，包括事件、特效、尺寸、表单和用户界面元素等；最后研究jQuery应用程序的测试以及将jQuery集成到复杂网站的方法。

无论是为jQuery前端开发人员，还是编写客户端代码的服务器端开发人员，都可以从本书中找到自己感兴趣的内容，并提高自己的开发本领。

撰稿人

各章节作者

Jonathan Sharp从1986年起开始热心于互联网和Web开发，在以后的岁月中，他曾为多家创业公司和财富500强公司工作过。Jonathan在内布拉斯加州的大奥马哈（Omaha）市创立了Out West Media有限责任公司，提供以定制XHTML、CSS和jQuery开发为中心的前端工程和架构服务。Jonathan是jQuery核心团队成员，在编码之余也是一位作家和演说家。Jonathan非常感激他的妻子Erin、女儿Noel、两只狗和两匹马。

Rob Burns供职于A Mountain Top有限责任公司，从事交互式Web应用程序开发。在过去的12年中，他采用多种工具和技术探索网站开发。在业余时间，他喜欢自然语言处理和开放源码软件项目中带来的众多机会。

Rebecca Murphey是一位独立前端架构顾问，开发充当服务器端和浏览器端之间胶水的定制前端解决方案。她还提供以jQuery程序库为重点的前端开发的培训。她和她的伙伴——两只狗和两只猫一起生活在北卡罗来纳州的达勒姆（Durham）。

Ariel Flesler是一位Web开发人员和视频游戏程序员。他从2007年1月起开始投入jQuery的开发，并在2008年5月加入核心团队。他出生于阿根廷布利诺斯艾利斯，曾就学于国家科技大学（阿根廷）。他最初的工作是ASP.NET（C#）程序员，后转向XHTML网站和Ajax应用的客户端开发。他目前在QB9工作，负责基于AS3的休闲游戏和MMO。

Cody Lindley是一位基督徒、丈夫、儿子、父亲、户外活动爱好者和专业的客户端工程师。从1997年起，他热衷于HTML、CSS、JavaScript、Flash、界面设计和HCI。他因创建模态/对话框解决方案ThickBox而享誉于jQuery社区。2008年，他正式加入jQuery团队，成为一位倡导者。他目前关注的方向是客户端优化技术，以及有关jQuery的写作和演讲。Cody的网站是*http://www.codylindley.com*。

Remy Sharp是一位开发人员、作家、演说家和博主。Remy于1999年开始了他的Web开发生涯，当时他是一个财经网站的唯一开发人员，在“网络繁荣”之中和之后的很长一段时间，他经历了网站运营各个方面的工作。现在，他在英国布莱顿（Brighton）运营自己的开发公司Left Logic，编写JavaScript、jQuery、HTML 5、CSS、PHP、Perl和其他自己熟悉的程序。

Mike Hostetler是一位发明家、企业家、程序员和自豪的父亲。从20世纪90年代中期，Mike就开始使用Web技术，在PHP和JavaScript Web应用程序开发方面有着丰富的经验。目前，Mike掌管位于科罗拉多州费城的Web技术咨询公司A Mountain Top有限责任公司。Mike频繁参与开源项目，是jQuery核心团队成员，负责QCubed PHP5 Framework项目并参与Drupal项目。当他不在计算机前时，喜欢徒步旅行、飞钓、滑雪以及和家人共享时光。

Ralph Whitbeck毕业于罗彻斯特（Rochester）理工学院，目前是位于纽约罗彻斯特的BrandLogic公司的高级开发人员。他在BrandLogic的工作包括界面设计、可用性测试、Web和应用程序开发。Ralph能够用ASP.NET、C#和 SQL Server开发复杂的Web应用系统，也能使用XHTML、CSS和JavaScript/jQuery等客户端技术实现客户认可的设计。2009年10月，Ralph加入jQuery团队，成为一位倡导者。Ralph喜欢和妻子Hope及三个儿子Brandon、Jordan和 Ralphie共度时光。你可以在他的个人博客（*http://ralphwhitbeck.com*）上了解更多有关他的情况。

Nathan Smith是从20世纪末就开始构建网站的一个古怪的家伙。他喜欢手动编写HTML、CSS和JavaScript，对设计和信息架构也有涉猎。他曾为Adobe Developer Center、Digital Web和.NET Magazine等在线及纸质出版物撰稿，在Adobe MAX、BibleTech、Drupal Camp、Echo Conference、Ministry 2.0、Refresh Dallas和Webmaster Jam Session等会议上发表过演讲。Nathan在FellowshipTech.com任UX开发人员，拥有阿斯伯里（Asbury）神学院的神学硕士学位。他启动了Godbit.com，这是一个旨在帮助教堂和神职人员更好使用Web的公共资源。他还创建了960 Grid System（*http://www.960.gs*）——一个页面布局草拟、设计和编码的框架。

Brian Cherne是一位软件开发人员，在规划和构建基于Web的应用程序、信息站和高流量电子商务网站方面有超过10年的经验。他也

是hoverIntent jQuery插件的作者。在不摆弄代码的时候，Brian会跳交谊舞、练习武术或者学习俄罗斯语言文化。

Jörn Zaefferer是来自德国科隆的专业软件开发人员。他创建用于Web和桌面应用程序的应用编程接口（API）、图形用户界面（GUI）、软件架构和数据库。他的工作以Java平台为中心，同时围绕jQuery开发客户端脚本。2006年中期，他开始为jQuery做出贡献，目前已经共同创建和维护jQuery的单元测试框架QUnit；发布和维护了6个非常流行的jQuery插件，并且以作者和技术审核者身份参与了jQuery书籍的创作，还是jQuery UI的首席开发人员。

James Padolsey是一位热心的Web开发人员和博主，来自英国伦敦。自从第一次看到jQuery，他就为之疯狂；他编写了jQuery的教程、讨论文章和博客，并为社区贡献了丰富的插件。Jame将来的计划包括获得肯特（Kent）大学的计算机科学学位，以及不断开拓新领域的职业生涯。他的网站是*http://james.padolsey.com*。

Scott González是生活在北卡罗来纳罗利（Raleigh）的一位Web应用开发人员，他喜欢构建高动态性的系统和灵活的可伸缩框架。他从2007年开始为jQuery做出贡献，目前是jQuery的官方用户界面库jQuery UI的开发主力。Scott还在nemikor.com上编写了关于jQuery和jQuery UI的教程，并在许多会议上发表有关jQuery的演讲。

Michael Geary从电传打字机穿孔纸带的时代就开始了软件开发，当时“兼容标准”的含义只不过是遵循穿孔纸带数据交换的ECMA-10标准。现在，Mike是一位Web和Android开发人员，对编写快速、清晰和简洁的代码有着特别的兴趣，还喜欢在jQuery邮件列表上帮助其他开发人员。他的网站是*http://mg.to*。

Maggie Wachs、Scott Jehl、Todd Parker和 Patty Toland是Filament小组成员。他们一起为面向消费者和公司的网站、无线设备、安装的和基于Web的应用程序设计和开发实用的用户界面，他们的宗旨是提供直观易用的体验同时兼具广泛的可访问性。他们是jQuery UI团队的发起人和设计主力，设计和开发了ThemeRoller.com，并积极地投身于正式的jQuery UI程序库和CSS框架的持续开发。

Richard D. Worth是一位Web UI开发人员，jQuery UI的发布经理和服务时间最长的开发人员之一。他是Dialog、Progressbar、

Selectable和Slider插件的作者或者共同作者。Richard还喜欢在全球进行有关jQuery和jQuery UI的演讲和咨询工作。他和他可爱的妻子Nancy在弗吉尼亚州北部（华盛顿郊区）一起抚养儿女。他们已经有了三个漂亮的孩子：Naomi、Asher和Isaiah。Richard的网站是http://rdworth.org/。

技术编辑

Karl Swedberg，在4年前开始Web开发人员职业生涯之前，他曾经在中学教过英语，在广告公司担任文稿编辑并且拥有一间咖啡屋。现在，Karl为密歇根州大急流城（Grand Rapids）的Fusionary Media公司工作，专攻客户端脚本和交互设计。Karl是jQuery项目团队成员，《Learning jQuery 1.3》和《jQuery Reference Guide》（均由Packt出版）的合著者。你可以在*http://www.learningjquery.com*找到他编写的提示和教程。

Dave Methvin是PCPitstop.com首席技术官和该公司的创始股东之一。他从2006年开始使用jQuery，活跃于jQuery帮助小组，并且贡献了包括Corner和Splitter在内的多个流行jQuery插件。在加入PC Pitstop之前，Dave曾在《PC Tech Journal》和《Windows Magazine》担任执行编辑，是JavaScript方面的专栏作者。他持续地为多家PC相关网站（包括InformationWeek）撰稿。Dave持有弗吉尼亚大学计算机科学学士和硕士学位。

David Serduke是一位前端程序员，最近在服务器端投入了许多时间。在多年的编程工作之后，他于2007年底开始使用jQuery并很快加入了jQuery核心团队。David目前为金融机构创建网站，并将jQuery的优点带入ASP.NET企业应用程序。David住在加州北部，拥有加州大学伯克利分校电子工程学士学位和圣玛丽（St. Mary）学院的MBA学位。

Scott Mark是Medtronic公司的企业应用架构师。他致力于开发基于Web的个性化信息门户和事务型应用程序，关注可控环境中高可用性的维护。目前，他最感兴趣的领域是富互联网应用程序和多点触摸用户界面技术。Scott和他可爱的妻子、两个儿子和一条黑色的拉布拉多犬一起住在明尼苏达州。他的技术博客参见*http://scottmark.wordpress.com*。

序

当我在2005年开始构建jQuery的时候，心中有一个简单的目标：我希望能够编写一个Web应用程序，使其能在所有主流浏览器上正常工作——不需要进一步的修补和bug修复。在我建立一组足以完成个人目标的实用程序前几个月，我认为自己已经接近目标了，完全没有意识到我的工作才刚刚开始。

从这些简单的成果开始，随着新用户将这些程序库用于自己的项目，jQuery已经逐渐成长和发展起来。已经证明它是JavaScript程序库开发中最具挑战性的部分；虽然为个人或者具体的应用程序构建一个程序库相当简单，但是开发用于尽可能多环境（旧的浏览器、遗留网页和大量陌生的标记）的程序库却非常困难。令人惊讶的是，尽管jQuery已经为处理更多用例而做了修改，但是大部分原始API却完好无损地保留了下来。

我感到特别有趣的一件事情是了解开发人员如何使用jQuery达到自己的目的。作为具有计算机科技背景的人，我对有这么多设计人员和非程序员感受到jQuery的吸引力而吃惊。看到他们使用这个程序库的方式，使我对简洁的API设计有了更好的理解。此外，许多高级程序员采用jQuery开发复杂的大型程序，对我也有很大的启发。不过，最令我感觉良好的是能够从使用程序库的每一个人那里学到知识。

使用jQuery的另一个好处是可扩展的插件结构。在刚刚开发jQuery的时候，我确实为开发人员提供了扩展API的一些简单方法。这些扩展已经发展成了许多形形色色的插件社区，这些插件组成了应用程序、开发人员和用例这一完整的生态系统。这些插件社区加速了jQuery的成长——没有它们，这个程序库就没有今天的成就，所以我很高兴在本书中加入一些专门的章节，讲解一些最有趣的插件以及它们的用途。扩展对jQuery用途的感性认识的最佳途径之一，就是学习和使用来自jQuery插件社区的代码。

上述原因使这本“食谱”变得如此有趣：它为你带来了开发人员在日复一日的编码中完成的杰作和学习到的技巧，并加以提炼和总结，供以后使用。从个人的角度，我认为“食谱”类型的书籍是挑战

我对语言或者程序库感性认识的最佳途径之一。我很乐意看到我所熟知的API被人们以新颖而有趣的方式利用。希望本书能够很好地为读者服务，将这些新颖而有趣的jQuery使用方法传授给大家。

——John Resig

jQuery创始人、首席开发者

前言

jQuery程序库给前端开发带来了一场风暴。它极其简单的语法使曾经很复杂的任务变得轻松愉快。许多开发人员很快就为它的优雅和清晰而着迷。如果你已经开始使用这个程序库，你就已经将丰富而具有交互性的体验加入到你的项目中。

jQuery的入门非常容易，但是和许多用于开发网站的工具一样，完全体会到它的广度和深度需要花费几个月甚至几年的时间。这个程序库充满了你从未想象过的特性。一旦你了解了这些特性，这些特性就能够戏剧性地改变你解决问题的方法。

本书旨在向亲爱的读者展示业界领先的前端开发人员在日常项目中使用jQuery的方式和方法。在18章中，这些业界精英将带你经历由简到繁的各种问题的解决过程。不管你是jQuery新手还是老练的JavaScript开发人员，都能够对jQuery创建引人注目、健壮和高性能的用户界面的能力有全新的认识。

本书读者对象

你可能是为jQuery提供的交互性而着迷的设计人员，也可能是希望了解其他人如何完成常见任务的jQuery前端开发人员。你还可能是常常应要求编写客户端代码的服务器端开发人员。

坦白说，这本“食谱”对于任何使用jQuery的人（或者希望使用jQuery的人）都有价值。如果你刚刚开始使用这一程序库，可以考虑配套阅读Packt出版的《Learning jQuery 1.3》或者Manning出版的《jQuery in Action》。如果你已经在项目中使用jQuery，本书能够增强你对jQuery的功能、隐藏的精华和特色的了解。

你将会学习到的内容

我们从基础知识和通用最佳实践的介绍开始——在页面中包含jQuery、做出选择、遍历和操纵。即使常用jQuery的用户也能从中学

到一两个技巧。由此，我们转向实际的用例，带你经历对常见问题的可靠（并且经过测试）的解决方案，这些问题包括事件、特效、尺寸、表单和用户界面元素（可能需要或者不需要jQuery UI的帮助）。最后，我们将研究jQuery应用程序的测试以及将jQuery集成到复杂网站的方法。

在学习的过程中，你将学习到利用jQuery解决高级问题的策略。我们将研究如何最大限度地利用jQuery的事件管理系统，包括自定义事件和自定义事件数据，如何渐进增强表单，如何在页面上定位和重定位元素，如何从头开始创建选项卡、折叠控件和模态等用户界面元素，如何制作具备易读性和可维护性的代码，如何优化代码以简化测试、消除瓶颈和确保最高性能等。

因为这是一本“食谱”而非手册，你当然可以选择阅读最适合自己的“菜谱”[1]；书中的每一个单独的秘诀都物有所值。但是，本书自始自终提供的都是jQuery社区中一些难得的绝妙解题方法。因此，我们希望你能至少从头到尾浏览一遍——你永远不会知道，哪一行代码能够让你茅塞顿开，使你的技巧更上一层楼。

jQuery风格和惯例

jQuery非常强调链式（chaining）语法——依次调用选择元素的方法，确信每个方法都能返回继续工作所用的选择元素。这种模式将在第1章中深入说明——如果你还不熟悉这个程序库，就应该理解这个概念，因为在后续的章节中将会频繁地用到它。

对jQuery的功能做了一些简单的分类：核心功能、选择、操纵、遍历、CSS、属性、事件、特效、Ajax和工具。对这些分类和对应方法的学习，将极大地加强对书中内容的理解。

本书介绍的最佳实践之一是在变量中存储元素，而不是重复地进行相同的选择。当选择的元素存储在变量中时，该变量一般以$字符开头，表明它是一个 jQuery对象。这样的语法使代码更容易阅读和维护，但是应该理解，以$字符开头的变量名称并不是惯例，和PHP之类的语言不同，它在jQuery中没有特殊的意义。

一般来说，本书中的代码示例重视清晰性和易读性而不是简洁性，所以示例比起绝对必需的代码来说稍嫌冗长。如果你发现可以优化的地方，可以毫不犹豫地进行。同时，你也可以在自己的代码中坚持清晰性和易读性，然后使用代码精简工具准备用于生产环境的代码。

其他选择

如果你想寻找其他jQuery资源，下面是我推荐的一些书籍：

- Jonathan Chaffer、Karl Swedberg和John Resig编著的《Learning jQuery 1.3》（Packt）；
- Bear Bibeault、Yehuda Katz和John Resig编著的《jQuery in Action》（Manning)；
- Dan Wellman编著的《jQuery UI 1.6: The User Interface Library for jQuery》（Packt)。

如果运行示例时遇到问题

在检查其他问题之前，确保页面上加载了jQuery程序库——你将会很惊奇地发现，很多时候这就是“代码不工作”问题的解决方案。如果你同时使用jQuery和另一个JavaScript程序库，可能需要使用`jQuery.noConflict()`来确保和其他程序库的协同工作。如果有的脚本需要jQuery，一定要在已经加载jQuery程序库之后再加载这些脚本。

本书中的许多代码都要求文档在JavaScript与之交互之前处于“就绪”状态。如果你在文档的头部包含了代码，确保这些代码包含在`$(document).ready(function() { ... })`之内，这样它就能知道，要等到文档就绪才开始交互。本书中讨论的一些功能只存在于jQuery 1.3和更高版本中。

如果你从旧版本的jQuery升级，确保升级使用的所有插件——过时的插件可能导致不可预知的表现。如果你发现示例在现有的应用程序中难以正常运行，在将其集成到现有代码之前先确保它能够正常运行。如果可以，Firefox浏览器的Firebug等工具能帮助你找出错误的根源。

如果你包含了jQuery的精简版本而问题出现在jQuery程序库本身，可以考虑在调试时切换到完整版本的jQuery。这样，你就更容易定位导致问题的代码行，这常常能将你引向解决问题的正确方向。

如果仍然不能解决问题，可以考虑将问题张贴到Google上的jQuery用户组。本书的多位作者都是该用户组的常客，该用户组的成员往往能提供有用的建议。Freenode上的#jquery IRC频道是解决问题的另一个有价值的资源。

如果上述方法都无效，可能是我们的错误。我们认真地测试和审查了书中的所有代码，但是仍然有可能出错。检查勘误表（在下一小节中说明）并下载更新后的样板代码，其中可能已经改正了我们发现的问题。

如果你喜欢（或者不喜欢）本书

如果你喜欢——或者不喜欢——这本书，无论如何要让大家知道。Amazon reviews（亚马逊书评）是分享你的快乐（或者不快乐）的流行方法，你也可以在本书的网站上留下你的意见：

http://oreilly.com/catalog/9780596159771/

网站上也有指向勘误表的链接。读者可以通过勘误表让我们知道本书中出现的印刷问题、错误和其他问题。勘误在网站上立刻可以看到，我们将进行检查和确认。O’Reilly也会在本书未来的印刷中以及Safari网站上改正这些错误，尽快地为读者带来更好的体验。我们希望在未来版本的jQuery发行后更新本书，在以后的版本中采纳读者的建议和批评。

本书约定

本书使用下列排版约定：

斜体

表示互联网地址，例如，域名和URL，以及新术语的定义。

等宽字体

表示应该逐字输入的命令行和选项；程序中的名称和关键字，包括方法名称、变量名称和类名；HTML元素标记、开关、属性（attribute）、键值、函数、类型、命名空间、模块、属性（property）、参数、值、对象、事件、事件处理程序、宏、文件内容或者命令输出。

加粗等宽字体

表示程序代码行中需要强调的部分。

斜体等宽字体

表示文本应该由用户提供的值代替。

> **注意**
>
> 这个图标用来表示一个提示、建议或一般的说明。

> **警告**
>
> 这个图标用来说明一个警告或注意事项。

代码示例的使用

本书的目的是帮助你完成自己的工作，一般来说，你可以将本书的代码用在自己的程序和文档中。除非复制大部分代码，否则你没有必要得到我们的许可。例如，编写一个使用本书中多处代码的程序不需要许可，销售或者分发O’Reilly书籍中示例的CD-ROM则需要许可。引用本书中的文字和示例代码不需要许可，但是将本书中的大量示例代码用于你的产品文档需要许可。

我们重视但并不要求归属权。归属权通常包括书名、作者、出版社和ISBN编码。例如“*jQuery Cookbook*, by Cody Lindley. Copyright 2010 Cody Lindley, 978-0-596-15977-1”。如果你感觉代码示例的使用超出了正当使用或者上述授权的范围，可以与我们联系（*permissions@oreilly.com*）。

Safari　联机丛书

Safari联机丛书是一个按需的数字图书馆，你可以搜索超过7500种技术和创意参考书以及视频，快速地查找你所需要的答案。

订阅之后，你可以在线阅读图书馆中的任意页面，观看任何视频，在你的手机和移动设备上阅读书籍。在新书付印之前先睹为快，独家查阅编写中的手稿以及作者的反馈。你还可以复制和粘贴代码样例、组织收藏夹、下载章节、将关键部分记入书签、评论、打印页面，另外还能从许多省时的特性中获益。

O’Reilly Media已经将本书上传到Safari联机丛书服务。在*http://my.safaribooksonline.com*上免费注册，就可以访问本书及其O’Reilly和其他出版社类似主题书籍的数码版本。

如何联络我们

读者可以信件形式向出版商提出有关本书的评论和问题，地址及电话如下：

O’Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

关于本书的评论或者技术问题，可以发送电子邮件到如下邮箱：

bookquestions@oreilly.com

有关我们的书籍、课程、会议和新闻的更多信息可以参见我们的网站（http://www.oreilly.com）。

——Rebecca Murphey和Cody Lindley

---

[1] 译注：recipe在英文中既有“菜谱”之意（与书名相符），又有“秘诀”之意。

第1章　jQuery基础

Cody Lindley

1.0 导言

既然你已经选择了一本有关jQuery的“食谱”，本书作者基本就可以假定你对jQuery的定义和功能有了大致的认识。坦白说，“食谱”通常是为寻求加强已有知识基础的读者所编写的。因此，本书使用了问题-解决方案-讨论的编排方式，快速地介绍常见问题的解决方案。但是，如果你是一位jQuery新手，不要把本书抛诸脑后，认为第1章是老生常谈，这一章就是专为新手所写的。

如果你需要复习，或者只有很少或者完全没有jQuery的知识，第1章将帮助你学习jQuery的概要知识（其他章节假定你已经了解了这些基础知识）。现在，从实际出发，如果你对JavaScript和DOM完全没有了解，可能应该退后一步，问问自己：在对JavaScript核心语言及其与DOM之间的关系没有基本理解的情况下，学习jQuery是否可行。我的建议是在接触jQuery之前，先认真学习DOM和JavaScript的核心知识。我强烈建议将David Flanagan编著的《JavaScript: The Definitive Guide》（http://oreilly.com/catalog/9780596000486，O’Reilly出版）一书作为阅读本书之前的入门读物。但是，如果你试图在学习DOM和JavaScript之前就开始jQuery的学习，也不要让我的一家之言阻挡了脚步。许多人都通过jQuery学到了这些技术的有用知识。虽然这样做不理想，但是只要我们勇敢面对，仍然可以实现学习的目的。

说了这么多，现在我们来看看jQuery的正式定义和功能的简单描述：

jQuery是一个开放源码的JavaScript程序库，简化了HTML文档（更准确地说是文档对象模型（Document Object Model，DOM））与JavaScript之间的交互。

用通俗的话说，也为了让守旧的JavaScript黑客明白，jQuery将动态HTML（DHTML）变得极其简单。具体地说，jQuery简化了HTML文档遍历和操纵、浏览器事件处理、DOM动画、Ajax交互和跨浏览器JavaScript开发。

理解了jQuery的正式含义之后，我们接下来研究选择使用jQuery的原因。

1.0.1 为什么使用jQuery

在“食谱”中谈论jQuery的优点似乎有点傻，尤其是在你已选择阅读这本“食谱”，很可能已经意识到这些优点的情况下。

所以，虽然这么做就像在唱诗班面前传道，但是我们仍然要简单地看看开发人员选择使用jQuery的原因。通过在研究“怎么做”之前先解释“为什么”，能够促进你对jQuery基础知识的掌握。

在jQuery案例的构造中，我不打算将jQuery与其竞争者作比较来提高jQuery的重要性。这是因为，我相信这方面还没有真正的直接竞争者。而且，我相信jQuery是当今唯一同时满足设计师和程序员需求的程序库。从这一方面说，jQuery是独一无二的。

市场上充斥着声名狼藉的JavaScript程序库和框架，但是我绝对相信，每个产品都有自己合适的用途和价值。进行广泛的比较很愚蠢，但是人们总是这么做，连我自己也不能免俗。所有的程序库都有价值，哪一个更胜一筹取决于谁使用它以及如何使用它，而不是它实际上能做什么。而且，根据我的观察，考虑到JavaScript开发的目标广泛，各种JavaScript程序库之间的微小差别根本不值一提。所以，我们不再进一步进行哲学方面的探讨了，而是列出能够支持选择jQuery的一组特性：

- jQuery是开放源码的程序库，该项目在MIT和GNU通用公共授权（General Public License，GPL）下授权使用。在很多方面，它都是免费的！
- jQuery很小（精简后只有18KB），用GZIP压缩（解压后为114KB）。
- jQuery的流行程度令人难以置信，也就是说，有着大规模的用户社区，许多贡献者以开发者和传道者的身份参与该项目。
- jQuery规范了Web浏览器之间的差异，这样你就不需要为此费心。
- jQuery有意地设计为轻量级的程序库，具有简单而又智能的插件架构。
- jQuery的插件库（http://plugins.jquery.com）规模很大，而且从jQuery发布之后就稳步增长。

- jQuery的API有完整的文档，包括内联的代码示例，这在JavaScript程序库中可以称得上豪华了。多年以来，任何的文档都是奢侈品。
- jQuery很友好，提供了一些方法帮助用户避免与其他JavaScript程序库的冲突。
- jQuery的社区支持相当实用，包括了多个邮件列表、IRC频道和来自jQuery社区的大量教程、文章、博客文章。
- jQuery的开发是开放式的，任何人都可以提交缺陷修复、改进和开发帮助。
- jQuery的开发是稳定一致的，也就是说，开发团队并不担忧更新的发布。
- 大型机构（如Microsoft、Dell、Bank of America、Digg、CBS、Netflix）的采用已经并将持续地提高jQuery的生命力和稳定性。
- jQuery先于浏览器吸收了来自W3C的规范。例如，jQuery支持大部分CSS3选择器。
- jQuery目前已经为流行浏览器（Chrome 1、Chrome Nightly、IE 6、IE 7、IE 8、Opera 9.6、Safari 3.2、WebKit Nightly、Firefox 2、Firefox 3、Firefox Nightly）上的开发进行了测试和优化。
- jQuery在设计师的手里和程序员手里一样强大，对两类用户一视同仁。
- jQuery优雅、讲求方法以及改变JavaScript书写方式的观念正在成为标准。只要想想有多少其他解决方案借用了选择器和链接（chainning）模式就能明白这一点。
- jQuery无法解释的副作用——良好的编程感觉具有感染力，令人无法抗拒；甚至连批评家都深深地为jQuery的特性所着迷。
- jQuery的文档有许多使用路径（例如，API浏览器、仪表板应用、“小抄”），包括一个离线API浏览器（AIR应用程序）。
- jQuery旨在倾向于简化无干扰JavaScript方法。
- jQuery的核心仍然是一个JavaScript程序库（与框架相反），同时又提供用于用户界面部件和应用程序开发的姐妹项目（jQuery UI）。
- 由于jQuery建立在大部分开发人员和设计师已经理解的概念（例如，CSS和HTML）之上，因此它的学习曲线很平滑。

我认为，使jQuery不同于其他解决方案的是上述特性的结合，而不是单一特性。作为JavaScript工具，整个jQuery程序包是无以匹敌的。

1.0.2 jQuery原则

jQuery的原则是“用更少的代码做更多的事”。这一原则可以进一步分为三个概念：

- （通过CSS选择器）寻找一些元素，（通过jQuery方法）对其进行某些处理。
- 链接一组元素上的多个jQuery方法。
- 使用jQuery包装器和隐式迭代。

详细了解这三个概念是编写你自己的jQuery代码和扩展本书中学到的秘诀的基础。下面详细地解释这些概念。

1．寻找一些元素并对其进行某些处理

更具体地讲，这条原则是指在DOM中找到一组元素，然后对这组元素进行某种处理。例如，研究一下这样的场景：你想要对用户隐藏一个 元素，在隐含的 中加载一些新的文本内容，修改 的属性，最后让隐藏的 再次可见。

上面的最后一句话转换成的jQuery代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
</head>
<body>
<div>old content</div>
<script>
//隐藏页面上的所有div
```

```
jQuery('div').hide();

//更新所有div中包含的文本
jQuery('div').text('new content');

//为所有div添加值为updatedContent的class属性
jQuery('div').addClass("updatedContent");

//显示页面上的所有div
jQuery('div').show();

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

我们逐条查看这4条jQuery语句：

- 隐藏页面上的 元素，使用户无法看到它。
- 用新的文本（new content）替换隐藏 元素中的文本。
- 用新的属性（class）和值（updatedContent）更新 元素。
- 在页面上显示 元素，使用户又能看到它。

如果现在这些jQuery代码还让你觉得深奥，也没有关系。我们将在本章的第一个秘诀中介绍这些基础知识。另外，从这个代码示例中，你需要了解的是jQuery“找到一些元素并对其进行某些处理”的概念。在读例子中，用jQuery函数（jQuery()）找出HTML页面中所有的 元素，然后用jQuery方法对它们进行了一些处理（例如，hide()、text()、addClass()、show()）。

2．链

jQuery的构造方式允许jQuery方法链。例如，为什么不在找到元素之后，将该元素上的操作链接起来呢？上一个代码示例阐述了“找到一些元素并对其进行某些处理”的概念，它可以用链改写为一条JavaScript命令。

利用链，下面的代码原来如下：

```
//隐藏页面上的所有div
jQuery('div').hide();

//更新所有div中包含的文本
jQuery('div').text('new content');

//为所有div添加值为updatedContent的class属性
jQuery('div').addClass("updatedContent");

//显示页面上的所有div
jQuery('div').show();
```

更改后的代码如下：

```
jQuery('div').hide().text('new content').addClass("updatedContent").show();
```

或者加上缩进和换行，如下所示：

```
jQuery('div')
    .hide()
    .text('new content')
    .addClass("updatedContent")
    .show();
```

简而言之，链允许你在目前用jQuery函数选择（当前用jQuery功能包装起来）的元素上应用无限的jQuery方法链。在后台，每当应用jQuery方法之前，总是返回以前选择的元素，使链能够继续下去。在未来的秘诀中你将会看到，因为插件也以这种方式构造（返回包装的元素），所以使用插件也不会破坏这一链条。

虽然并非显而易见，但是根据对代码的研究，通过一次性选择一组DOM元素，由jQuery方法以链的方式进行多次操作，能够减少处理开销。避免不必要的DOM遍历是网页性能改进的关键部分，一定要尽可能重用或者缓存选中的DOM元素集。

3. jQuery包装器集

大部分时候，如果jQuery很复杂，你会使用所谓的“包装器”。换句话说，你会从一个HTML页面上选择一组用jQuery功能包装的DOM元素。我个人常常将其称为“包装器集”或者“包装集”，因为它是一组由jQuery功能包装的元素。有时候这种包装器集包含单个DOM元素，其他时候则包含多个元素，甚至还有包装器集没有包含任何元素的情况。在这种情况下，jQuery提供的方法/属性在空包装器集中将会“无提示”地失败，这就可以避免不必要的if语句。

现在，以我们用于解释“寻找一些元素并对其进行某些处理”的代码为基础，如果在网页上添加几个 元素，你认为会发生什么情况呢？在下面这段更新过的代码示例中，添加了三个 元素，这样网页上一共有4个 元素：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<script type="text/JavaScript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.0/jquery.min.js"></
<body>
<div>old content</div>
<div>old content</div>
<div>old content</div>
<div>old content</div>
<script>
//隐藏页面上的所有div
jQuery('div').hide().text('new content').addClass("updatedContent").show();

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

你在这里没有显式编写任何循环代码，但是猜猜看会怎么样？jQuery扫描网页，将所有 元素放在包装器集中，这样我所使用的jQuery方法在集合中的每个DOM元素上执行（亦称隐式迭代）。例如，.hide()方法实际上应用到集合中的每个元素。所以，如果再次查看代码，就会发现每个方法都应用到页面上的每个 元素，就像你编写了一个循环在每个DOM元素上调用各个jQuery方法一样。更新后的代码示例将导致页面中的所有 被隐藏，更新文本内容，指定一个新的类值，然后再次显示。

对包装器集和默认的循环系统（隐式迭代）的理解对于围绕循环的高级概念是至关重要的。你只要记住，在真正进行更多的循环（例如，jQuery('div').each(function(){})）之前，已经发生了简单的循环。你也可以这样看：包装器中的每个元素一般都会被所调用的jQuery方法所改变。

还要记住一点，在以后的章节中你将会学习到，某些情况下只有第一个元素（而不是包装器集中的所有元素）受到jQuery方法（例如，attr()）的影响。

1.0.3 jQuery API的组织方式

毫无疑问，在我开始接触jQuery时，选择它作为我的JavaSript程序库的主要原因只是因为它的文档很好（以及大量的插件）。后来，我意识到自己爱上jQuery的另一个原因是：它的API按照合乎逻辑的分类进行组织。只要查看API的组织方式，我就能缩小所需功能的范围。

在你真正开始使用jQuery之前，我建议你访问在线文档（http://docs.jquery.com/Main-Page），简单地了解一下API的组织方式。理解了API的组织方式，你就能更快地在文档中找到你所需要的信息，考虑到编写一个jQuery解决方案有许多不同的方法，这实在是一个巨大的优势。由于一个问题有许多解决方案，因此健全的API组织方式能帮助你更全心全意地投入实现之中。这里将重申API的组织方式，建议你记住API的大纲，至少记住第一级分类。

- jQuery核心
  - jQuery函数

    - jQuery对象访问器
    - 数据
    - 插件
    - 互操作性
- 选择器
    - 基础
    - 层次结构
    - 基本过滤器
    - 内容过滤器
    - 可见性过滤器
    - 属性过滤器
    - 子元素过滤器
    - 表单
    - 表单过滤器
- 属性
    - 属性
    - 类
    - HTML
    - 文本
    - 值
- 遍历
    - 过滤
    - 查找
    - 链
- 操纵
    - 修改内容
    - 内部插入
    - 外部插入
    - 周围插入
    - 替换
    - 删除
    - 复制
- CSS
    - CSS
    - 定位
    - 高度和宽度
- 事件
    - 页面加载
    - 事件处理
    - Live事件
    - 交互助手
    - 事件助手
- 特效
    - 基础
    - 滑行
    - 淡入/淡出
    - 自定义
    - 设置
- Ajax
    - Ajax请求
    - Ajax事件
    - 杂项
- 工具
    - 浏览器和特性检测
    - 数组和对象操作
    - 测试操作

- 字符串操作
- URL

在我们开始学习一系列基本的jQuery秘诀之前，我想提醒一下：本章介绍的秘诀是相互依赖的。也就是说，从第一个秘诀到最后一个的顺序遵循合乎逻辑的知识结构，对于首次接触这些秘诀的读者，我建议从1.1节到1.17节顺序阅读。

1.1 在HTML页面中包含jQuery程序库代码

1.1.1 问题

你打算在一个网页上使用jQuery JavaScript程序库。

1.1.2 解决方案

目前，在网页中嵌入jQuery程序库有两种理想的解决方案：

- 使用Google托管的内容分发网络（Content Delivery Network，CDN）来包含某个版本的jQuery（本章采用这种方式）。
- 从jQuery.com上下载你自己的jQuery版本，将其安装在你自己的服务器或者本地文件系统上。

1.1.3 讨论

包含jQuery JavaScript程序库和包含其他外部JavaScript文件没有什么不同。你只要使用HTML <script>元素并提供src=""属性的值（URL或者目录路径），你所链接的外部文件就将包含在网页中。例如，下面的模板包含了jQuery程序库，可以用后者启动任何jQuery项目：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
</head>
<body>
<script type="text/JavaScript">
    alert('jQuery ' + jQuery.fn.jquery);
</script>
</body>
</html>
```

注意，我将使用Google托管的jQuery精简版，并强烈建议公开网页使用这一方法。但是，精简代码中JavaScript错误的调试不理想。在代码开发期间，或者在生产网站上，使用来自Google的非精简版本进行调试可能更容易发现JavaScript错误。关于Google托管的jQuery版本的更多信息，可以访问Ajax程序库API网站：http://code.google.com/apis/ajaxlibs/。

你当然也可以自己安装一个jQuery代码副本。但是，在大部分情况下这很愚蠢，因为Google已经为你托管了一个很好的版本。使用Google托管的jQuery，你可以得益于一个可靠、高速且在全球都能访问的jQuery版本。而且，你还能够受益于降低的延迟、获得更高的并行性和更好的缓存。当然，没有Google的解决方案，你也能实现这一点，但是很可能要支付一点钱。

现在，不管出于什么原因，你可能不愿意使用Google托管的jQuery版本，而想要jQuery的自定义版本，也可能你的使用方式不需要（或者没有）互联网连接。或者，你可能认为Google是“统治者”，因为自己的控制欲和阴谋论而不愿意听命于它。对于不需要或者不愿意使用Google托管的jQuery代码的人们，可以从jQuery.com（http://docs.jquery.com/Downloading-jQuery）下载jQuery，将其安装到你自己的服务器或者本地文件系统上。按照我在本秘诀中提供的模板，你只要用指向你下载的jQuery JavaScript文件位置的URL或者目录路径替换src属性值就可以了。

1.2 在DOM加载之后、整个页面加载之前执行jQuery/JavaScript代码

1.2.1 问题

采用无干扰式JavaScript方法论的现代JavaScript应用程序通常只在DOM完全加载之后才执行JavaScript。实际情况是，任何DOM遍历和操纵都要求在操作之前必须加载DOM。需要一种手段来确定客户端（最常见的是Web浏览器）何时完成DOM的加载（这时图片和SWF文件等资源可能还没有完全加载）。如果在这种情况下使用window.onload事件，包括所有资源的整个文档完全加载之后才能触发onload事件，这对大部分Web冲浪者来说太费时间。需要一个事件，告诉我们何时可以遍历和操纵DOM。

1.2.2 解决方案

jQuery提供ready()方法，这是一个定制的事件处理程序，通常与DOM的文档对象绑定。ready()方法的参数是一个函数，后者包含在DOM可以遍历和操纵时执行的JavaScript代码。下面是一个简单的例子，在DOM就绪而页面还未完全加载时打开一个alert()窗口：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    jQuery(document).ready(function(){//DOM not loaded, must use ready event
        alert(jQuery('p').text());
    });
</script>
</head>
<body>
<p>The DOM is ready!</p>
</body>
</html>
```

1.2.3 讨论

jQuery用ready()事件处理程序方法来代替JavaScript核心的window.onload事件。可以根据需要多次使用它。使用这个定制事件时，建议将它放在样式表声明和包含文件之后，这样能够确保ready()事件执行任何jQuery或JavaScript代码之前，所有元素属性都已经正确定义。

此外，jQuery函数本身提供使用jQuery定制的ready事件的快捷方式。使用这个快捷方式，下面的alert()示例可以改写为：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    jQuery(function(){ //DOM未加载,必须使用就绪事件
        alert(jQuery('p').text());
    });
</script>
</head>
<body>
<p>The DOM is ready!</p>
```

```
</body>
</html>
```

这个定制的jQuery事件只有在JavaScript必须嵌入到页面顶端的文档流并封装在<head>元素里时才有必要。我只需将所有JavaScript文件包含和内联代码放在<body>结束元素之前，就能避免使用ready()事件，这么做有两个原因。

首先，现代优化技术已经断言，当JavaScript放在页面解析的最后由浏览器加载时，页面的加载就会变得更快。换句话说，如果你将JavaScript放在网页的最后，浏览器将先加载之前的所有内容，然后才加载JavaScript，这是一件好事，因为大部分浏览器通常会暂停其他加载活动，等待JavaScipt引擎编译网页中包含的JavaScipt。从某种程度上说，将JavaScript放在网页文档的开头会形成瓶颈。我知道某些情况下将JavaScript放在<head>元素中更加简单，但坦诚说，我从未发现绝对必须这么做的情况。我在开发中因为将JavaScript放在页面最后而造成的所有困难都很容易克服，比起得到的优化效果，这些努力也都是值得的。

其次，如果提高网页的速度是我们的目标，为什么要为简单地将代码放到页面最后就能解决的问题而加入更多的功能呢？如果让我在较多代码和较少代码之间做出选择，我选择使用较少的代码。不使用ready()事件减少了代码量，而代码越少，网页运行得总是越快。

基于这些理由，在下面这个例子中，alert()代码没有使用ready()事件：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>The DOM is ready!</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
     alert(jQuery('p').text());//提示DOM已经加载
</script>
</body>
</html>
```

注意，我已经将所有的JavaScript放在<body>结束元素之前。HTML文档中的任何其他标记应该都在JavaScript之前。

1.3 用选择器和jQuery函数选择DOM元素

1.3.1 问题

你需要选择一个DOM元素或者一组DOM元素，以便用jQuery方法作用于这些元素。

1.3.2 解决方案

当你需要从DOM中选择元素时，jQuery提供两种备选方案。这两种选项都要求使用jQuery函数（jQuery()或其别名$()）。第一种选项使用CSS选择器和自定义选择器，这是最常用和最清晰的解决方案。通过向jQuery函数传递一个包含选择器表达式的字符串参数，该函数将遍历DOM并查找表达式定义的DOM节点。下面的代码是一个例子，选择HTML文档中的所有<a>元素：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
     //用警告框显示页面上有6个元素
     alert('Page contains ' + jQuery('a').length + ' <a> elements!');
</script>
</body>
</html>
```

如果你在Web浏览器中运行这个HTML页面，就会看到这段代码执行一个浏览器alert()方法，告诉我们该页面包含6个<a>元素。我首先选择所有的<a>元素，然后用length属性返回jQuery包装器集中元素的数量，并将其传递给alert()方法。

你应该知道，这里使用的jQuery函数的第一个参数能够接受多个表达式，只要在传递给jQuery函数的第一个字符串参数中用逗号分隔多个选择器就行了。下面是一个例子：

```
jQuery('selector1, selector2, selector3').length;
```

选择DOM元素的第二种选项是向jQuery函数传递DOM元素的实际JavaScript引用，这种选项不如第一种常用。举个例子，下面的代码将选择HTML文档中的所有<a>元素。注意，我传递给jQuery函数的是一个用getElementsByTagName DOM方法收集到的<a>元素数组。这个例子的结果和前一个代码示例完全相同：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body bgcolor="yellow"> <!-- yes the attribute is depreciated, I know, roll
with it -->
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
```

```
<a href='#'>link</a>
<a href='#'>link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
     //用警告框提示页面上有6个元素
     alert('Page contains ' + jQuery(document.getElementsByTagName('a')).length +
' <a> Elements, And has a '
     + jQuery(document.body).attr('bgcolor') + ' background');
</script>
</body>
</html>
```

1.3.3 讨论

众所周知，jQuery能够胜任繁重的工作，这在某种程度上是因为选择器引擎Sizzle（http://sizzlejs.com/），该引擎能够从HTML文档中选择DOM元素。虽然向jQuery函数传递DOM引用在必要的时候是个很好的选择，当它并不是jQuery进入众人视野的原因。选择器拥有的许多强大的选项才是jQuery与众不同之处。

在本书余下的部分中，你将会学习到各种强大和健壮的选择器，对这些选择器的功能都要彻底地理解。这些知识在未来的jQuery编程工作中将使你获益匪浅。

1.4 在指定上下文中选择DOM元素

1.4.1 问题

你需要引用在另一个DOM元素或者文档上下文中的单个DOM元素或者一组DOM元素，以便用jQuery方法操作这些元素。

1.4.2 解决方案

当传递CSS表达式时，jQuery函数还有第二个参数，这个参数告诉jQuery函数应该在那个上下文中根据表达式搜索DOM元素。在这种情况下，第二个参数可以是一个DOM引用、jQuery包装器或者文档。在下面的代码中有12个<input>元素。注意我是如何根据<form>元素确定具体的上下文，仅选择特定的<input>元素的：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>

<form>
<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />
</form>

<form>
<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />
</form>

<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />

<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">

//使用上下文包装器在所有表单元素中搜索,警告框中显示"selected 8 inputs"
alert('selected ' + jQuery('input',$('form')).length + ' inputs');

//用DOM引用作为上下文,在第一个表单元素中搜索,警告框中显示"selected 4 inputs"
alert('selected ' + jQuery('input',document.forms[0]).length + ' inputs');
//用表达式搜索body元素中的所有输入元素,警告框中显示"selected 12 inputs"
alert('selected ' + jQuery('input','body').length + ' inputs');
</script>
</body>
</html>
```

1.4.3 讨论

正如1.4.2节所提到的，也可以将文档作为搜索的上下文。例如，可以在一个XHR请求（Ajax）发回的XML文档的上下文中搜索。你可以在第16章中看到更多相关的细节。

1.5 过滤DOM元素包装器集

1.5.1 问题

在jQuery包装器集中有一组选中的DOM元素，但是打算从集合中删除不匹配新指定表达式的元素，以创建一个新的操作元素集合。

1.5.2 解决方案

jQuery过滤器方法用于DOM元素的jQuery包装器集，可以排除不符合指定表达式的元素。简言之，可以用`filter()`方法过滤当前元素集，这是过滤器方法与jQuery查找方法的重要区别，查找方法通过寻找（使用新的选择器变量）新元素（包括当前包装器集的子元素）来缩小DOM元素的包装器集。

为了理解过滤器方法，我们来看看下面的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a href="#" class="external">link</a>
<a href="#" class="external">link</a>
<a href="#"></a>
<a href="#" class="external">link</a>
<a href="#" class="external">link</a>
<a href="#"></a></li>
<a href="#">link</a>
<a href="#">link</a>
<a href="#">link</a>
<a href="#">link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    //在警告框中显示集合中还有4个元素
        alert(jQuery('a').filter('.external').length + ' external links');
</script>
</body>
</html>
```

上述代码示例中的HTML页面包含一个有10个<a>元素的网页，其中的外部链接指定了类名`external`。使用jQuery函数选择页面上的所有<a>元素。然后，利用过滤器方法删除原始集合中所有`class`属性值不为`external`的元素。在`filter()`方法修改初始DOM元素集之后，调用`length`属性，该属性会告诉我在应用过滤器之后，新的集合中有多少个元素。

1.5.3 讨论

向`filter()`方法传递一个用于过滤包装器集的函数也是可行的。前一个代码示例中`filter()`方法的参数是一个字符串表达式，现在用一个函数来代替它：

```
alert(
   jQuery('a')
      .filter(function(index){ return $(this).hasClass('external');})
      .length + ' external links'
);
```

注意，现在传递给`filter()`方法的是一个匿名函数。调用这个函数的上下文与当前元素相同，也就是说当在函数中使用`$(this)`时，实际应用的是包装器集中的每个DOM元素。在函数中，我将检查包装器集中每个<a>元素的类值（`hasClass()`）是否为`external`。如果是，返回逻辑真值，该元素保留在集

合中；否则（返回逻辑假值），从集合中删除元素。也可以这么理解：如果函数返回假值，则删除该元素。如果函数返回其他值，该元素就会留在包装器集中。

你可能已经注意到：这里向函数传递了一个名为index的参数，但是并不打算使用它。在必要的时候，这个参数可用来以数字形式指出jQuery包装器集中元素的索引。

1.6 查找当前选择包装器集中的后代元素

1.6.1 问题

你选择了一组（或者一个）DOM元素，希望在当前选中元素的上下文中找到后代（子）元素。

1.6.2 解决方案

使用.find()方法，根据当前集合及其后代的上下文创建一个新的元素包装器集。例如，假设你有一个包含多个段落的网页，这些段落中封装的是需要强调（以斜体显示）的单词。如果你只想选择<p>元素中包含的元素，可以使用如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>Ut ad videntur facilisis <em>elit</em> cum. Nibh insitam erat facit
<em>saepius</em> magna. Nam ex liber iriure et imperdiet. Et mirum eros
iis te habent. </p>
<p>Claram claritatem eu amet dignissim magna. Dignissim quam elit facer eros
illum. Et qui ex esse <em>tincidunt</em> anteposuerit. Nulla nam odio ii
vulputate feugait.</p>
<p>In quis <em>laoreet</em> te legunt euismod. Claritatem <em>consuetudium</em>
wisi sit velit facilisi.</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
 //在警告框中显示<p>元素中找到的斜体单词总数
       alert('The three paragraphs in all contain ' +
       jQuery('p').find('em').length + '
italic elements');
</script>
</body>
</html>
```

记住，也可以改写上述代码，将上下文引用作为jQuery函数的第二个参数：

```
alert('The three paragraphs in all contain ' + jQuery('em',$('p')).length +
' italic elements');
```

此外，值得一提的是，最后这两个代码示例只是为了说明的目的而编写的。使用CSS选择器表达式选择包含在<p>元素（祖先）中的斜体元素（后代）即使不实用，也是更符合逻辑的。

```
alert('The three paragraphs in all contain ' + jQuery('p em').length +
' italic elements');
```

1.6.3 讨论

jQuery .find()方法可以用于根据当前DOM元素集及其子元素的上下文创建新的元素集。人们往往混淆filter()和.find()方法的用法。记住两者差异的最简手段是记住.find()将操作/选择当前集合的子元素，而.filter()只操作当前元素集。换句话说，如果你想将当前包装器集当作上下文，进一步选择集合中元素的子元素，从而改变当前的包装器集，就应该使用.find()。如果你只想过滤当前包装集，获得集合中当前DOM元素的一个新子集，则使用.filter()。归纳起来就是：find()返回子元素，而filter()只过滤当前包装集里的元素。

1.7 返回破坏性修改之前的选择

1.7.1 问题

需要删除用于一组元素的破坏性jQuery方法（例如，filter()或find()），以便将集合恢复到破坏性方法使用之前的状态，就像破坏性方法从来没有调用过一样。

1.7.2 解决方案

jQuery提供end()方法，可以用它返回使用破坏性方法之前选择的一组DOM元素。为了理解end()方法，我们来看看下面的HTML。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>text</p>
<p class="middle">Middle <span>text</span></p>
<p>text</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    alert(jQuery('p').filter('.middle').length); //alerts 1
    alert(jQuery('p').filter('.middle').end().length); //alerts 3
    alert(jQuery('p').filter('.middle').find('span')
.end().end().length); //alerts 3
</script>
</body>
</html>
```

代码中的第一条alert()语句包含的jQuery语句搜索文档中所有<p>元素，然后对选中的<p>元素应用filter()方法，仅选择类为middle的元素。length属性报告了集合中剩下元素的数量：

```
alert(jQuery('p').filter('.middle').length); //提示1
```

下一条alert()语句使用了end()方法。这里所做的操作和前一条语句相同，唯一例外的是撤消了filter()方法，返回filter()方法使用前包装器集包含的元素：

```
alert(jQuery('p').filter('.middle').end().length); //提示3
```

最后一条alert()语句示范了如何两次使用end()方法移除filter()和find()破坏性修改，使包装器集返回其原始构成的方法：

```
alert(jQuery('p').filter('.middle').find('span').end().end().length); //提示3
```

1.7.3 讨论

如果使用end()方法之前没有执行破坏性操作，将会返回一个空集。破坏性操作指的是任何改变匹配jQuery元素集合的操作，也就是返回jQuery对象的任何遍历或者操纵方法，包括add()、andSelf()、children()、closes()、filter()、find()、map()、next()、nextAll()、not()、parent()、parents()、prev()、prevAll()、siblings()、slice()、clone()、appendTo()、prependTo()、insertBefore()、insertAfter()和replaceAll()。

1.8 将前一个选择集包含到当前选择集

1.8.1 问题

你刚刚对一组元素进行操纵，获得新的元素集。但是，你想同时操作前一个元素集和当前元素集。

1.8.2 解决方案

可以用andSelf()方法合并前一个DOM元素选择集和当前选择集。例如，在下面的代码中，首先选择页面上的所有 元素。接下来，操纵这组元素，寻找 元素中的所有<p>元素。现在，为了同时操作 和 中找到的<p>元素，可以用andSelf()方法将 包含到当前集合。如果省略andSelf()，边框颜色将只应用到<p>元素：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<div>
<p>Paragraph</p>
<p>Paragraph</p>
</div>
<script type="text/JavaScript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></
<script type="text/JavaScript">
    jQuery('div').find('p').andSelf().css('border','1px solid #993300');
</script>
</body>
</html>
```

1.8.3 讨论

记住，当使用andSelf()方法时，它只向当前操作集合中添加前一个集合，而不是以前选择的所有集合。

1.9 根据当前上下文遍历DOM获得新的DOM元素集

1.9.1 问题

你已经选择了一组DOM元素，根据选择集在DOM结构树中的位置，你打算遍历DOM获得一个新的元素集以供操作。

1.9.2 解决方案

jQuery提供一组方法，可以根据当前选择的DOM元素的上下文遍历DOM。

例如，查看如下的HTML片段：

```
<div>
<ul>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
</ul>
</div>
```

现在，用:eq()索引自定义选择器选择第2个<li>元素：

```
//根据索引选择<li>集合中的第2个元素,索引从0开始
jQuery('li:eq(1)');
```

现在有一个上下文——HTML结构中的一个出发点。出发点是第2个<li>元素。从这里开始，可以到达任何位置——对，几乎任何位置。

让我们来看看，使用几个jQuery提供的DOM遍历方法能够到达哪里。代码中的注释就能说明问题：

```
jQuery('li:eq(1)').next() //选择第三个<li>
jQuery('li:eq(1)').prev() //选择第一个<li>
jQuery('li:eq(1)').parent() //选择<ul>
jQuery('li:eq(1)').parent().children() //选择所有<li>
jQuery('li:eq(1)').nextAll() //选择第二个<li>之后的所有<li>
jQuery('li:eq(1)').prevAll() //选择第二个<li>之前的所有<li>
```

记住，这些遍历方法产生新的包装器集，使用end()可以返回前一个包装器集。

1.9.3 讨论

目前为止介绍的遍历方法展示了简单遍历。要了解遍历，还必须知道另外两个重要的概念。

第一个概念可能显而易见——遍历方法可以链接。我们再来看看前面出现过的jQuery语句：

```
jQuery('li:eq(1)').parent().children() //选择所有<li>
```

注意，我已经从第二个<li>元素遍历到父元素<ui>，然后再从父元素选择<ui>的所有子元素。jQuery包装器集现在包含的是<ui>中的所有<li>元素。当然，这只是为了说明遍历方法而设计的例子。如果我们想要的是一个只有<li>元素的包装器集，从一开始就选择所有<li>元素要简单得多（例如，jQuery('li')）。

处理遍历方法时需要牢记的第二个概念是许多方法都接受一个可选的参数，用于过滤选择集。我们仍然用链接的示例来说明这一点，看看如何修改代码，以便只选择最后一个<li>元素。别忘了，这个例子

仅仅用来说明遍历方法如何传递用于选择特定元素的表达式：

```
jQuery('li:eq(1)').parent().children(':last') //选择最后一个 <li>
```

jQuery还提供了其他遍历方法，在这里不作介绍。完整的列表和文档可以参阅 http://docs.jquery.com/Traversing。在本书里你将会看到这些遍历方法。

1.10 创建、操作和插入DOM元素

1.10.1 问题

你打算创建一个或者多个新的DOM元素，立刻选中这些元素加以操作，然后把它们注入到DOM中。

1.10.2 解决方案

你可能还不清楚，jQuery函数是多功能的，根据你发送的不同参数结构，一个函数能以不同的方式运行。如果以原始HTML文本字符串为参数调用函数，它将立刻创建这些元素。例如，下列语句将创建一个包装在<p>元素中的<a>元素，在<p>和<a>元素中还封装了一个文本节点：

```
jQuery('<p><a>jQuery</a></p>');
```

创建了元素之后，还可以使用jQuery方法对它进行进一步的操作，就像一开始就从现有的HTML文档中选择了<p>元素似的。例如，可以用.find()方法选择<a>元素,并设置它的一个属性。在下面的代码中，将其href属性设置为http://www.jquery.com：

```
jQuery('<p><a>jQuery</a></p>').find('a').attr('href','http://www.jquery.com');
```

这很棒，对吗？目前为止，所做的只不过是在运行中创建元素并在代码中进行操纵，实际上还可以做得更好。可以说，实际上还没有真正改变当前加载的DOM。想做到这一点，就必须使用jQuery提供的操纵方法。下面是在HTML文档上下文中的代码。在这段代码中将创建元素、在这些元素上进行操作，然后用操纵方法appendTo()将它们插入DOM中：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
jQuery('<p><a>jQuery</a></p>').find('a').attr('href','http://www.jquery.com')
  .end().appendTo('body');
</script>
</body>
</html>
```

请注意这里是如何使用end()方法撤消find()方法的，这样当调用appendTo()方法时，它将在DOM中附加包含在初始包装器集中的元素。

1.10.3 讨论

在本秘诀中，向jQuery函数传递原始HTML字符串，这一参数被方法用来在运行中创建DOM元素。 还可以简单地向jQuery函数传递一个由DOM方法createElement()创建的DOM对象：

```
jQuery(document.createElement('p')).appendTo('body'); //在页面中添加一个空白的p元素
```

当然，如果用一个包含多个元素的HTML字符串就能正常工作，这么做可能就显得麻烦了，如何选择取决于具体的用法。

值得一提的是，这里只是用appendTo()方法简单地介绍了操纵方法的基础。除了appendTo()方法之外，还有如下操纵方法：

- append()
- prepend()
- prependTo()
- after()
- before()
- insertAfter()
- insertBefore()
- wrap()
- wrapAll()
- wrapInner()

1.11 删除DOM元素

1.11.1 问题

你想从DOM中删除元素。

1.11.2 解决方案

remove()方法可以用于从DOM中删除选中的元素集及其子元素。请看如下代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<h3>Anchors</h3>
<a href='#'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('a').remove();
</script>
</body>
</html>
```

当把上述代码加载到浏览器中时，锚元素将留在页面中，直到JavaScript执行。一旦使用remove()方法从DOM中删除所有锚元素，该页面在外观上只包含一个<h3>元素。也可以向方法传递一个表达式，过滤需要删除的元素集。例如，代码可以进行如下改写，只删除具有特殊类的锚：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<h3>Anchors</h3>
<a href='#' class='remove'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<a href='#' class="remove">Anchor Element</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('a').remove('.remove');
</script>
</body>
</html>
```

1.11.3 讨论

当使用jQuery方法时，必须记住两点：

- 在使用remove()从DOM中删除选择的元素时，它们并没有从jQuery包装器集中删除。这意味着，从理论上说，可以继续操作它们，甚至可以在必要的时候将它们重新添加到DOM中。
- 这种方法不仅从DOM中删除元素，而且删除被删除元素包含的所有事件处理程序和内部缓存数据。

1.12 替换DOM元素

1.12.1 问题

你需要用新的DOM节点代替DOM中的现有节点。

1.12.2 解决方案

使用replaceWith()方法，可以选择一组DOM元素进行替换。在下面的代码示例中，使用replaceWith()方法，将class属性为remove的所有<li>元素替换为新的DOM结构：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
</ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('li.remove').replaceWith('<li>removed</li>');
</script>
</body>
</html>
```

添加到DOM中的新DOM结构作为一个字符串参数传递给relaceWith()方法。在例子中，所有
元素（包括子元素）都被新的结构
remove
替代。

1.12.3 讨论

jQuery提供一个方向相反的方法replaceAll()，该方法的功能与relaceWith()方法相同，但是参数位置颠倒。例如，可以将本秘诀中的代码改写成：

```
jQuery('<li>removed</li>').replaceAll('li.remove');
```

这里将HTML字符串传递给jQuery函数，然后使用replaceAll()方法选择想要删除和替换的DOM节点及其子节点。

1.13 克隆DOM元素

1.13.1 问题

你需要克隆/复制DOM的一部分。

1.13.2 解决方案

jQuery提供clone()方法复制DOM元素。它的用法很简单，只要用jQuery函数选择DOM元素，然后在选择的元素集上调用clone()方法就可以了。结果是返回用于链接的DOM结构的一个副本，而不是原来选中的DOM元素。在下面的代码中，将克隆一个<ul>元素，然后用插入方法appendTo()将这个副本附加到DOM中。实际上，在页面上插入了与现有的<ul>完全相同的一个结构：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul>
<li>list</li>
<li>list</li>
<li>list</li>
<li>list</li>
</ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('ul').clone().appendTo('body');
</script>
</body>
</html>
```

1.13.3 讨论

克隆方法对在DOM中移动DOM片段非常方便，尤其是在你打算复制和移动的不仅是DOM元素而且包括附加到克隆的DOM元素中的事件时。认真看看下面的HTML和jQuery：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul id="a">
<li>list</li>
<li>list</li>
<li>list</li>
<li>list</li>
</ul>
<ul id="b"></ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
 jQuery('ul#a li')
     .click(function(){alert('List Item Clicked')})
     .parent()
         .clone(true)
             .find('li')
             .appendTo('#b')
         .end()
     .end()
```

```
    .remove();
</script>
</body>
</html>
```

如果在浏览器中运行这段代码，它将克隆页面上附加了单击事件的<li>元素，将新克隆的元素（包括时间）插入空的<ul>中，然后删除克隆的<ul>元素。

这对于新的jQuery开发人员来说可能有点不可思议，所以我们一起来逐步观察这段代码，以便解释这个链式方法：

1. jQuery('ul#a li')=选择id属性为a的<ui>元素，然后选择该<ui>元素里的所有<li>元素。

2. .click(function(){alert('List Item Clicked')})=为每个<li>添加一个单击事件。

3. .parent()=将选择集改为<ul>元素，遍历DOM。

4. .clone(true)=克隆<ui>元素和所有子元素，包括附加到克隆元素中的任何事件。这通过传递给clone()方法一个布尔值true完成。

5. .find('li')=现在，在克隆元素中，将元素集改为仅包含在克隆<ui>元素中的<li>元素。

6. .appendTo('#b')=获取选中的克隆<li>元素，并将它们放置在id属性为b的<ui>元素中。

7. .end()=返回前一个元素选择集，该集即克隆的<ui>元素。

8. .end()= 返回前一个元素选择集——克隆的原始<ul>元素。

9. .remove() =删除原始的<ul>元素。

对于复杂的jQuery语句来说，理解如何操纵选中的元素集或者返回前一个选择集都是至关重要的。

1.14 获取、设置和删除DOM元素属性

1.14.1 问题

你已经用jQuery函数选择了一个DOM元素，需要获取或者设置该元素的属性值。

1.14.2 解决方案

jQuery提供attr()方法以获取和设置属性值。在下面的代码中，将设置<a>元素的href属性值，然后获取该值：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a>jquery.com</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
//提示jQuery主页URL
alert(
    jQuery('a').attr('href','http://www.jquery.com').attr('href')
);
</script>
</body>
</html>
```

在上述代码示例中你可以看到，选择HTML元素中仅有的<a>元素，设置它的href属性，然后用单一的属性名作为参数调用相同的attr()方法获取属性值。如果文档中有多个<a>元素，attr()方法将访问第一个匹配的元素。当代码加载到浏览器时，将用alert()方法显示设置的href属性值。

现在，因为大部分元素都有多个属性，所以也可以用单个attr()方法设置多个属性。例如，可以用一个对象代替两个字符串参数作为attr()方法的参数，设置前一个例子中的title属性：

```
jQuery('a').attr({'href':'http://www.jquery.com','title':'jquery.com'}).attr('href')
```

和添加属性功能相伴的是删除属性及其值的功能。removeAttr()方法可以用来从HTML元素中删除属性。要使用这个方法，只要传递代表要删除的属性值的字符串即可（例如，jQuery('a').removeAttr('title')）。

1.14.3 讨论

除了attr()方法之外，jQuery为使用HTML元素class属性提供了一组很特殊的方法。因为class属性可能包含多个值（例如，class="class1 class2 class3"），所以可以使用这些独特的属性方法管理这些类值。

下面列出的就是这些jQuery方法：

addClass()

用新的类/值更新class属性值，包括任何已经设置的类。

hasClass()

检查特定类的class属性值。

removeClass()

从class属性中删除特定的类，同时保持已经设置的任何值。

toggleClass()

如果特定类不存在则添加，如果存在则删除该类。

1.15 获取和设置HTML内容

1.15.1 问题

你需要获取或者设置当前网页中的一块HTML内容。

1.15.2 解决方案

jQuery提供html()方法，用于获取和设置HTML元素块（或者DOM结构）。在下面的代码中，使用这个方法设置HTML元素中找到的<p>元素的HTML值，然后获取该值：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p></p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
jQuery('p').html('<strong>Hello World</strong>, I am a <em>&lt; p&gt; </em> element.');
alert(jQuery('p').html());
</script>
</body>
</html>
```

在浏览器中运行这段代码将造成浏览器修改<p>元素中包含的HTML内容，这些内容是使用html()方法设置的，然后用html()方法获取相同内容。

1.15.3 讨论

这个方法使用DOM的innerHTML属性获取和设置HTML块。你还应该知道，html()在XML文档中不可用（但是可用于XHTML文档）。

1.16　获取和设置文本内容

1.16.1　问题

你需要获取或者设置包含在一个或多个HTML元素中的文本。

1.16.2　解决方案

jQuery提供text()方法，用于获取和设置元素的文本内容。在下面的代码中，使用这个方法设置HTML文档中<p>元素的文本值，然后获取该值：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p></p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
    jQuery('p').text('Hello World, I am a <p> element.');
    alert(jQuery('p').text());
</script>
</body>
</html>
```

在浏览器中运行上述代码将造成浏览器修改<p>元素的内容，用text()方法设置这些内容，然后用text()方法读取。

1.16.3　讨论

重要的一点是，要记住text()方法与html()方法没有什么不同，唯一的例外是text()方法将对HTML进行转义（将<和>替换为HTML实体）。这意味着，如果在text()方法的字符串参数中放入标记，该方法会将这些标记转换为HTML实体（<和>）。

1.17　在不造成全局冲突的情况下使用$别名

1.17.1　问题

你希望使用快捷方式$别名代替全局命名空间名称（jQuery）的输入，而又不用担心全局冲突。

1.17.2　解决方案

这里提供的解决方案是创建一个匿名的自调用函数，将jQuery对象传递给这个函数，然后将$字符当作指向jQuery对象的一个参数。

例如，所有jQuery代码可以封装在如下的自调用函数中：

```
(function($){ //用$参数创建私有作用域的函数
    //私有作用域和$的使用无须担心冲突
})(jQuery); //调用无名函数并将其传递给jQuery对象
```

1.17.3　讨论

实际上，这里所做的就是将对jQuery的全局引用传递给一个创建私有作用域的函数。如果没有这么做，而是直接在全局作用域中使用简写的$别名，就必须假定包含在HTML文档中的其他脚本（或者未来包含的脚本）都没有使用$字符，这是有一定风险的。当你能够创建自己的私有作用域时，何必去冒险呢？

这样做的另一个好处是包含在匿名的自调用函数中的代码将运行于自己的私有作用域中。可以确信，在该函数中放置的任何内容都决不会和全局作用域中编写的任何其他JavaScript代码发生冲突。同样，为什么要冒编程冲突的风险？你要做的只不过是创建自己的私有作用域。

第2章　用jQuery　选择元素

James Padolsey

2.0 导言

jQuery最核心的部分是选择器引擎，它允许根据名称、属性、状态等选择任何文档中的元素。因为CSS非常流行，所以采用它的选择器语法简化jQuery中的元素选择就很有意义了。除了支持CSS1-3规范中的大部分选择器之外，jQuery还添加了一些自定义选择器，这些选择器可以用于根据特殊状态和特征选择元素。此外，可以创建自己的自定义选择器！本章讨论用jQuery选择元素时较常遇见的一些问题。

在第一个秘诀之前，先讨论几个基本原理。

在文档中选择一个特定的元素或者一组元素的最简方法是在jQuery包装器函数中使用CSS选择器，如：

```
jQuery('#content p a');
    //选择#content中所有段落里的所有锚元素
```

既然已经选择了所需的元素，就可以在集合上运行任何jQuery方法了。例如，向所有链接添加selected类很简单：

```
jQuery('#content p a').addClass('selected');
```

jQuery提供了许多有助于元素选择过程的DOM遍历方法，例如，next()、prev()和parent()。这些方法和其他方法都以一个选择器表达式作为唯一的参数，相应地过滤返回的结果。所以，可以在许多地方使用CSS选择器，而不是仅仅在jQuery(...)中。

在构造选择器时，有一个通用的优化原则：只追求必要的确定性。要记住重要的一点：选择器越复杂，jQuery处理字符串的时间就越长。jQuery用原生的DOM方法读取你搜索的元素。之所以能够使用选择器，只是因为在jQuery中很好地应用了抽象；这本身没有什么问题，但是理解你所编写的选择器造成的后果非常重要。下面是不必要的复杂选择器的一个典型例子；

```
jQuery('body div#wrapper div#content');
```

越高的确定性并不一定越快。前一个选择器可以改写成：

```
jQuery('#content');
```

这个选择器的效果相同，但是能够减少前一个版本的开销。还要注意，有时候可以为选择器指定一个上下文，对其进一步的优化；本章稍后将讨论这个主题（见秘诀2.11）。

2.1 仅选择子元素

2.1.1 问题

你需要选择特定元素的一个或者多个直接子元素。

2.1.2 解决方案

使用直接后代组合符（>）。这个组合符的两边各有一个选择器表达式。例如，如果你想要选择直接在列表项下的所有锚元素，可以使用选择器li>a。这个选择器选择作为列表项子元素的所有锚；换句话说，选择的是所有直接处于列表项目之下的所有锚。下面是一个例子：

```
<a href="/category">Category</a>
<ul id="nav">
    <li><a href="#B25590_anchor1">Anchor 1</a></li>
    <li><a href="#B25590_anchor2">Anchor 2</a></li>
    <li><span><a href="#B25590_anchor3">Anchor 3</a></span></li>
</ul>
```

现在，可以调用如下的jQuery，选择每个列表项目下的锚：

```
jQuery('#nav li > a');
 //正如预期,上述代码选择两个元素
```

#nav列表中的第三个锚没有选中，因为它不是列表项的子元素，而是元素的子元素。

2.1.3 讨论

区分子元素和后代元素很重要。后代（descendant）是存在于另一个元素中的任何元素，而子元素是直接后代；用父母和子女的类推对此很有帮助，因为DOM的层次结构大体与此类似。

值得注意的是，像>这样的组合符在使用的时候，如果已经指定了上下文，左边的表达式可以省略：

```
jQuery('> p', '#content');
    //本质上和jQuery('#content > p')一样
```

在更为编程化的环境中，选择子元素应该用jQuery的children()方法完成，在该方法中可以用一个选择器过滤返回的元素。下面的代码选择#content元素的所有直接子元素：

```
jQuery('#content').children();
```

上述代码基本上和jQuery('#content > *')相同，但是有一个重要的差异；它运行的速度更快。jQuery不解析选择器，而是立即知道目的。但是，运行得更快这一差异并没有什么实用性。而且，速度的差别取决于浏览器和选择的内容，在某些情况下，这种差别几乎无法察觉。当处理存储在变量中的jQuery对象时，children()方法特别有用。例如：

```
var anchors = jQuery('a');
// 获取所有锚元素的直接子元素可以用三种方式实现：
// #1
anchors.children();
// #2
jQuery('> *', anchors);
// #3
anchors.find('> *');
```

实际上，还有一种实现方式！在这种情况下，第一种方法最快。前面已经提到，可以向children()方法传递一个选择器表达式以过滤结果：

```
jQuery('#content').children('p');
```

只有作为#content直接子元素的段落元素才会返回。

2.2 选择特定的兄弟元素

2.2.1 问题

你只需要选择特定元素的一组兄弟元素。

2.2.2 解决方案

如果你想寻找的是特定元素相邻的兄弟元素，那么可以使用相邻兄弟元素组合符（+）。和子元素组合符（>）类似，兄弟元素组合符的两边各有一个选择器表达式。右边的表达式是选择器的主题，而左边的表达式是你希望匹配的兄弟元素。

下面是一些HTML标记示例：

```
<div id="content">
    <h1>Main title</h1>
    <h2>Section title</h2>
    <p>Some content...</p>
    <h2>Section title</h2>
    <p>More content...</p>
</div>
```

如果你只想选择紧跟在<h1>元素之后的<h2>元素，可以使用如下选择器：

```
jQuery('h1 + h2');
//选择与H1元素紧邻的所有H2元素
```

在上述例子中，只有一个<h2>元素将被选中（第一个）。没有选中第二个<h2>元素是因为，虽然它是<h1>元素的兄弟元素，但它不是相邻的兄弟元素。

另一方面，如果你想选择和过滤元素的所有兄弟元素（不管相邻与否），那么可以使用jQuery的siblings()方法，还可以传递一个可选的选择器表达式过滤选择结果：

```
jQuery('h1').siblings('h2,h3,p');
    //选择作为H1元素兄弟的所有H2、H3和P元素
```

有时候，你希望根据和其他元素的相对位置选择兄弟元素；例如，下面是一些典型的HTML标记：

```
<ul>
    <li>First item</li>
    <li class="selected">Second item</li>
    <li>Third item</li>
    <li>Fourth item</li>
```

```
    <li>Fifth item</li>
</ul>
```

可以使用如下方法选择第二个列表项之后（在li.selected之后）的所有列表项：

```
jQuery('li.selected').nextAll('li');
```

正如siblings(),nextAll()方法接受一个选择器表达式，在选择集返回之前进行过滤。如果省略这个选择器,nextAll()将返回存在于主题元素之后主题元素的所有兄弟元素，尽管不会返回前面的元素。

对于前一个例子，也可以使用另一个CSS组合符选择第二个列表项之后的所有列表项。在CSS3中加入了普通兄弟元素组合符（～），因此你可能无法在实际的样式单中使用它，幸运的是，可以在jQuery中使用它而不用考虑支持的问题。该组合符的工作方式与相邻兄弟元素组合符（+）完全相同，但是它选择的是后续（不仅是相邻的那一个）的兄弟元素。使用前面的标记，可以用如下的选择器选择li.selected之后的所有列表项：

```
jQuery('li.selected~li');
```

2.2.3 讨论

相邻兄弟元素组合符从概念上可能难以使用，因为它不遵循其他大部分选择器表达式所采用的自顶向下层次方法。但是，它仍然是值得研究的，也无疑是选择所需元素的一种实用而简单的方法。

不使用选择器也能实现相同的功能，按照以下方式：

```
jQuery('h1').next('h2');
```

next()方法是选择器语法的一个很好的备用方案，特别是在编程环境中将jQuery对象当作变量处理的时候，如：

```
var topHeaders = jQuery('h1');
topHeaders.next('h2').css('margin','0');
```

2.3 按照索引顺序选择元素

2.3.1 问题

你需要根据元素在其他元素中的顺序来选择元素。

2.3.2 解决方案

根据你想要选择的元素，可以随意使用如下的过滤器。这些过滤器看上去很像CSS的伪类，但是在jQuery中它们称作过滤器。

`:first`

匹配选中的第一个元素。

`:last`

匹配选中的最后一个元素。

`:even`

匹配索引为偶数的元素（索引从0开始）。

`:odd`

匹配索引为奇数的元素（索引从0开始）。

`:eq(`*n*`)`

按照索引（*n*）匹配单个元素。

`:lt(`*n*`)`

匹配索引小于*n*的所有元素。

`:gt(`*n*`)`

匹配索引大于*n*的所有元素。

假设下面HTML标记：

```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
```

```
    <li>Fourth item</li>
</ol>
```

列表中的第一项可以许多不同的方式选择：

```
jQuery('ol li:first');
jQuery('ol li:eq(0)');
jQuery('ol li:lt(1)');
```

注意eq()和lt()过滤器都接受一个数字参数；因为索引从0开始，所以第一项为0，第二项为1，以此类推。

表格行中的交替样式是常见的需求，可以用:even和:odd过滤器实现：

```
<table>
    <tr><td>0</td><td>even</td></tr>
    <tr><td>1</td><td>odd</td></tr>
    <tr><td>2</td><td>even</td></tr>
    <tr><td>3</td><td>odd</td></tr>
    <tr><td>4</td><td>even</td></tr>
</table>
```

可以根据每个表格行的索引应用不同的类：

```
jQuery('tr:even').addClass('even');
```

在CSS样式表中必须指定对应的类（even）：

```
table tr.even {
    background: #CCC;
}
```

这段代码产生的效果如图2-1所示。

0	even
1	odd
2	even
3	odd
4	even

图2-1　偶数行为暗色的表格

2.3.3　讨论

前面已经提到，元素的索引从0开始，所以第一个元素的索引为0。除了这一事实之外，上面提到的过滤器使用都非常简单。还要注意一点，这些过滤器需要一个

匹配集合；指定初始集合之后，索引才能确定。所以，下面的选择器不能正常工作：

```
jQuery(':even');
```

警告

实际上，上述选择器可以正常工作，但这只是因为jQuery在后台对选择器进行了一些更正性的预处理。如果没有指定初始集合，jQuery假定你所选择的是文档中的所有元素。所以，选择器实际上可以工作，因为它与jQuery('*:even')相同。

过滤器的左边需要一个初始集合，也就是说，需要一个旨在应用过滤器的目标。这个集合可以在已经实例化的jQuery对象中，如下所示：

```
jQuery('ul li').filter(':first');
```

这个过滤器方法将运行在已经实例化的jQuery对象（包含列表项）上。

2.4 选择当前动画元素

2.4.1 问题

你需要根据元素是否正在连续变化来选择元素。

2.4.2 解决方案

jQuery为这个极端的目的提供了一个方便的过滤器。:animated过滤器将只匹配正在连续变化的元素：

```
jQuery('div:animated');
```

这个选择器选择当前正在连续变化的所有 元素。实际上，jQuery选择的是动画队列不为空的所有元素。

2.4.3 讨论

当需要向当前连续变化的所有元素应用一个总括函数时，这个过滤器特别有用。例如，为了使所有还没有连续变化的 元素开始连续变化，可以采用如下简单的代码：

```
jQuery('div:not(div:animated)').animate({height:100});
```

有时候你可能希望检查元素是否连续变化，这可以用jQuery的userful()方法完成：

```
var myElem = jQuery('#elem');
if( myElem.is(':animated') ) {
    //进行某些处理
}
```

2.5 根据包含的内容选择元素

2.5.1 问题

你需要根据包含的内容选择元素。

2.5.2 解决方案

在这方面，通常只查询两种内容：文本内容和元素内容（其他元素）。对于前者，可以使用:contains()过滤器：

```
<!-- HTML -->
<span>Hello Bob!</span>
// 选择包含"Bob"的所有SPAN元素
jQuery('span:contains("Bob")');
```

注意，这个选择器是区分大小写的，所以搜索bob什么也不会找到（使用了一个小写的b）。而且，在所有情况下都不需要引号，但是加上引号是个好习惯，能够预防遇到必须使用引号的情况（例如，在打算使用圆括号时）。

可以使用:has()过滤器测试嵌套的元素。可以向这个过滤器传递任何有效选择器：

```
jQuery('div:has(p a)');
```

这个选择器将匹配在<p>元素（段落）中封装<a>元素（锚）的所有 元素。

2.5.3 讨论

:contains()过滤器可能无法满足你的要求。你可能需要更多地控制允许和不允许的文本。如果需要这样的控制，我建议使用正则表达式并测试元素的文本，如下所示：

```
jQuery('p').filter(function(){
    return /(^|\s)(apple|orange|lemon)(\s|$)/.test(jQuery(this).text());
});
```

这将选择包含apple、orange或者lemon的所有段落。jQuery的filter()方法更多的相关信息参见秘诀2.10。

2.6 选择不匹配的元素

2.6.1 问题

你需要选择一些不匹配特定选择器的元素。

2.6.2 解决方案

对此，jQuery给出了:not过滤器，使用方法如下：

```
jQuery('div:not(#content)'); //选择#content之外的所有DIV元素
```

这个过滤器将从当前选择集中删除任何匹配选择器的元素。该选择器的复杂度不限；它不一定是简单的表达式，如：

```
jQuery('a:not(div.important a,a.nav)');
//选择不在"div.important"中或者类不为"nav"的锚元素
```

警告

向:not过滤器传递复杂的选择器只在jQuery版本1.3以及更高的版本中才可行。在较低的版本中，只能接受简单的选择器表达式。

2.6.3 讨论

除了前面提到的:not过滤器之外，jQuery还提供了功能相近的一个方法。这个方法的参数包括选择器和DOM集合/节点。下面是一个例子：

```
var $anchors = jQuery('a');
$anchors.click(function(){
    $anchors.not(this).addClass('not-clicked');
});
```

根据上述代码，在单击一个链接锚时，其他的所有锚都将添加类“not-clicked”。this关键字指代单击的元素。

not()方法也接受选择器作为参数：

```
$('#nav a').not('a.active');
```

上述代码选择处于#nav中类不为active的所有锚元素。

2.7 根据可见性选择元素

2.7.1 问题

你需要根据是否可见选择元素。

2.7.2 解决方案

在必要时可以使用:hidden或者:visible过滤器：

```
jQuery('div:hidden');
```

下面是其他用法的例子：

```
if (jQuery('#elem').is(':hidden')) {
    // 有条件地进行一些处理
}
jQuery('p:visible').hide(); //只隐藏目前可见的元素
```

2.7.3 讨论

警告

> 从jQuery 1.3.2开始，这些过滤器有了显著的变化。在1.3.2之前，两个过滤器的反应类似于你对CSS visibility属性所期望的功能，但是现在已经不再考虑这些。相反，jQuery测试正在讨论的元素的高度和宽度（与其offsetParent相对）。如果任何一个维度为0，就认为该元素是隐藏的；否则，该元素是可见的。

如果需要更多的控制，可以始终使用jQuery的filter()方法，该方法允许以任何方式测试元素。例如，你可能想要选择设置为display:none的所有元素，而不是那些设置为visibility:hidden的元素。使用:hidden过滤器是没有效果的，因为它匹配具有这几种特征之一的元素（1.3.2以前版本）或者完全不考虑这两个属性（1.3.2及以后版本）：

```
jQuery('*').filter(function(){
    return jQuery(this).css('display') === 'none'
            && jQuery(this).css('visibility') !== 'hidden';
});
```

上述的代码应该留给你一组设置了display:none但是没有设置visibility:hidden的元素。注意，这样的选择通常没有必要——:hidden过滤器适用于大部分情况。

2.8 根据属性选择元素

2.8.1 问题

你需要根据属性和属性值选择元素。

2.8.2 解决方案

使用属性选择器匹配特定的属性和对应的属性值：

```
jQuery('a[href="http://google.com"]');
```

上述选择器将选择href属性等于指定值（http://google.com）的所有锚元素。

下面是使用属性选择器的一些方法：

[attr]

匹配具有指定属性的元素。

[attr=val]

匹配指定属性为某个值的元素。

[attr!=val]

匹配没有指定属性或者属性值的元素。

[attr^=val]

匹配指定属性值以特定值开始的元素。

[attr$=val]

匹配具有指定属性，且属性值以某个值作为结束的元素。

[attr~=val]

匹配包含指定值和两侧空格的元素（也就是说，car匹配car但是不匹配cart）。

警告

> 在jQuery1.2之前的版本，必须使用XPath语法（即，在属性名称之前加上@符号），现在这种语法已经弃用。

也可以组合多个属性选择器：

```
// 选择具有TITLE和HREF属性的所有元素
jQuery('*[title][href]');
```

2.8.3 讨论

和往常一样，对于特殊的要求，使用filter()方法更确切地指出所要查找的元素更为合适：

```
jQuery('a').filter(function(){
    return (new RegExp('http:\/\/(?!' + location.hostname + ')')).test(this.href);
});
```

在这个过滤器中，使用一个正则表达式测试每个锚的href属性。它选择任何页面中的所有外部链接。

属性选择器对于根据略有变化的属性选择元素来说特别实用。例如，如果有如下的HTML：

```
<div id="content-sec-1">...</div>
<div id="content-sec-2">...</div>
<div id="content-sec-3">...</div>
<div id="content-sec-4">...</div>
```

可以使用如下的选择器匹配所有 元素：

```
jQuery('div[id^="content-sec-"]');
```

2.9 按照类型选择表单元素

2.9.1 问题

你需要根据类型选择表单元素（hidden、text、checkbox等)。

2.9.2 解决方案

jQuery提供了一组用于这个极端目的的实用过滤器，如表2-1所示。

表2-1 jQuery表单过滤器

jQuery选择器语法	选择什么
:text	<input type="text" />
:password	<input type="password" />
:radio	<input type="radio" />
:checkbox	<input type="checkbox" />
:submit	<input type="submit" />
:image	<input type="image" />
:reset	<input type="reset" />
:button	<input type="button" />
:file	<input type="file" />
:hidden	<input type="hidden" />

举个例子，如果需要选择所有文本输入控件，可以简单地使用下列代码：

```
jQuery（':text'）；`
```

还有一个:input过滤器，可以选择所有的input、textarea、button和select元素。

2.9.3 讨论

注意，前面讨论过的:hidden过滤器没有测试hidden类型；它检查的是经过计算的元素高度。这对于hidden类型的输入元素是有效的，因为和其他隐藏元素一样，它们的offsetHeight为0。

和所有选择器一样，可以按照需要混合匹配：

```
jQuery(':input:not(:hidden)');
    //选择除了隐藏的输入元素之外的所有输入元素
```

这些过滤器也可以和常规的CSS语法一同使用。例如，选择所有文本输入元素加上所有的<textarea>元素可以用如下代码完成：

```
jQuery(':text, textarea');
```

2.10 选择有具体特性的元素

2.10.1 问题

你不仅需要根据和其他元素的关系或者简单的属性值，而且要根据各种特征（如选择器表达式无法表达的编程状态）选择元素。

2.10.2 解决方案

如果搜索的元素具有非常特殊的性质，选择器表达式可能不是最好的工具。使用jQuery的DOM过滤方法（`filter()`），可以根据函数中表达的任何条件选择元素。

jQuery中的过滤器方法允许传递一个字符串（也就是，选择器表达式）或者一个函数作为参数。如果传递的是一个函数，它的返回值将定义某个元素是否被选择。传递的函数将对当前选择集中的每个元素运行；每当函数返回假值时，对应的元素就从选择集中删除，每当返回真值时，对应的元素不受影响（即留在集合中）。

```
jQuery('*').filter(function(){
    return !!jQuery(this).css('backgroundImage');
});
```

上述代码选择所有具有背景图片的元素。

初始集合是所有元素（*）；然后以一个函数为参数调用`filter()`方法。这个函数在讨论的元素指定了`backgroundImage`时返回真值。你看到的`!!`是JavaScript中将任何类型值转换为布尔表达方式的快速方法。被当作假值的包括空字符串、数值0、数值`undefined`、空类型，当然还有布尔值`false`。如果查询`backgroundImage`返回的是上述值，该函数将返回`false`，从而删除集合中所有没有背景图片的元素。刚刚讲解的大部分内容并不是jQuery特有的，只是JavaScript的基础知识。

实际上，`!!`没有必要，因为jQuery将把返回值转换为布尔值，但是保留它仍是一个好主意；任何查看你的代码的人都能绝对肯定你的意图（有助于可读性）。

在传递给`filter()`的函数中，可以通过`this`关键字引用当前元素。将它包装在jQuery函数中就可以变成一个jQuery对象（这样就可以访问和执行jQuery方法）：

```
this; // 常规的元素对象
jQuery(this); // jQuery对象
```

下面是其他一些激发你的想象力的过滤示例：

```
//选择所有宽度在100-200像素之间的DIV元素:
jQuery('div').filter(function(){
    var width = jQuery(this).width();
    return width > 100 && width < 200;
});
//选择所有带有常用图像文件扩展名的图像:
jQuery('img').filter(function(){
    return /\.(jpe?g|png|bmp|gif)(\?.+)?$/.test(this.src);
});
//选择所有具备10个或者20个子元素的元素:
jQuery('*').filter(function(){
    var children = jQuery(this).children().length;
    return children === 10 || children === 20;
});
```

2.10.3 讨论

做一件事情总是有多种不同的方法，用jQuery选择元素也不例外。关键的差别通常是速度；有些方法快一些，而其他方法慢一些。为使用复杂的选择器时，应该考虑jQuery在后台必须进行的处理有多少。越长和越复杂的选择器返回结果所需的时间也越长。Jquery的原生方法有时候比单个选择器快得多，而且还有可读性方面的额外好处。下面的例子比较这两种技术：

```
jQuery('div a:not([href^=http://]), p a:not([href^=http://])');
jQuery('div, p').find('a').not('[href^=http://]');
```

第二种技术比第一种更简短，可读性也好得多。 在Firefox（V3）和Safari（V4）中进行测试表明，它也比第一种技术更快。

2.11 使用上下文参数

2.11.1 问题

你已经听说过上下文参数，但是还没有碰到适用的场合。

2.11.2 解决方案

在向jQuery()或者$()传递选择器表达式时，可以传递第二个参数，指定上下文。jQuery将在这个上下文中搜索匹配选择器表达式的元素。

上下文参数可能是利用最不充分的jQuery功能。其用法特别简单：传递一个选择器表达式、一个jQuery对象、一个DOM集合或者一个DOM节点给上下文参数，jQuery将仅在这个上下文中搜索元素。

下面是一个例子：用户想在表单提交之前选择所有输入字段：

```
jQuery('form').bind('submit', function(){
    var allInputs = jQuery('input', this);
    //现在你准备对"所有输入"进行处理
});
```

注意，this作为第二个参数传递，在刚才看到的处理程序中，this指的是表单元素。因为它被设置为上下文，jQuery只返回表单中的input元素。如果我们不包含第二个参数，文档的所有input元素将被选中——这不是我们想要的。

前面已经提到，也可以传递常规的选择器作为上下文：

```
jQuery('p', '#content');
```

上述代码返回和如下选择器相同的集合：

```
jQuery('#content p');
```

指定一个上下文有助于可读性和速度，这是一个很有用的功能。

2.11.3 讨论

jQuery使用的默认上下文是document，也就是DOM层次结构中最顶部的元素。只有在上下文不同于这个默认值时才需要指定。使用上下文可以按照如下方式表达：

```
jQuery( context ).find( selector );
```

实际上，这就是jQuery在后台所做的事情。

考虑到这一点，如果你已经有了对上下文的引用，那么你应该传递它而不是选择器——让jQuery再次经历选择过程没有道理。

2.12 创建一个子定义过滤器选择器

2.12.1 问题

你需要一个可重用的过滤器，根据特征选择特定的元素。你想要的是简洁而且可以包含在选择器表达式中的过滤器。

2.12.2 解决方案

可以在jQuery.expr[':']对象下扩展jQuery的选择器表达式，这是Sizzle.selectors.filters的别名。每个新的过滤器表达式都定义为这个对象的属性，如：

```
jQuery.expr[':'].newFilter = function(elem, index, match){
    return true; //像filter()方法一样返回true/false
};
```

该函数将在当前集合的所有元素上运行，必须返回真（在集合中保留元素）或者假（从集合中删除元素）。传递给函数的信息有三个部分：问题中的元素、元素在整个集合中的索引，以及从包含用于更复杂表达式的重要信息的正则表达式匹配返回的匹配数组。

例如，你打算选择有某个属性的所有元素。下列过滤器匹配内联显示的所有元素：

```
jQuery.expr[':'].inline = function(elem) {
    return jQuery(elem).css('display') === 'inline';
};
```

既然已经创建了一个自定义选择器，就可以将其用在任何选择器表达式中：

```
// E.g. #1
jQuery('div a:inline').css('color', 'red');
// E.g. #2
jQuery('span').filter(':not(:inline)').css('color', 'blue')
```

jQuery的自定义选择器（:radio、:hidden等）就是这样创建的。

2.12.3 讨论

前面已经提到过，过滤器函数的第三个参数是一个从jQuery在选择器字符串上进行的正则表达式匹配返回的数组。如果你打算创建接受参数的过滤器表达式，这个匹配就特别有用。假定我们打算创建一个查询jQuery保留数据的选择器：

```
jQuery('span').data('something', 123);
//我们希望能够做到:
jQuery('*:data(something,123)');
```

这个选择器的目的是选择通过jQuery的data()方法附加了数据的所有元素——具体选择的是数据键值为something，数值为123的元素。

上面提出的过滤器（:data）可以这样创建：

```
jQuery.expr[':'].data = function(elem, index, m) {
    //从匹配的元素中删除" :data(" and the trailing ")",因为这些部分不必要:
    m[0] = m[0].replace(/:data\(|\)$/g, '');
    var regex = new RegExp('([\'"]?)((?:\\\\\\1|.)+?)\\1(,|$)', 'g'),
        //读取数据键值:
        key = regex.exec( m[0] )[2],
        //读取测试的数据值:
        val = regex.exec( m[0] );
    if (val) {
        val = val[2];
    }
    //如果传递了一个值,对其进行测试;否则,测试键值是否为真:
    return val ? jQuery(elem).data(key) == val : !!jQuery(elem).data(key);
};
```

使用如此复杂的正则表达式是因为我们希望使其尽量灵活。新的选择器能够以许多不同的方法使用：

```
//和我们最初的想法一样(同上):
jQuery('span:data("something",123)');
//检查"something"是不是"真"值:
jQuery('span:data(something)');
//带不带(内部的)引号:
jQuery('span:data(something, "something else")');
```

现在，我们有了一种查询jQuery在元素中保留数据的全新方法。

如果你想要同时添加超过一个新的选择器，最好使用jQuery的extend()方法：

```
jQuery.extend(jQuery.expr[':'], {
    newFilter1 : function(elem, index, match){
        //返回真或假
    },
    newFilter2 : function(elem, index, match){
        // 返回真或假
    },
    newFilter3 : function(elem, index, match){
        //返回真或假
    }
});
```

第3章　超越基础

Ralph Whitbeck

3.0 导言

jQuery是非常轻量级的程序库，有助于进行页面上DOM元素的简单选择。在第1章中你看到了这些简单的用法。本章将探索如何使用jQuery操纵、遍历，以及扩展jQuery实现无限的可能性。jQuery的轻量级特征使得它健壮而且可扩展。

3.1 循环读取选择结果集合

3.1.1 问题

你需要从选择的DOM元素集中创建一个列表，但是在选择集上执行任何的操作都将集合看做一个整体。为了能够用每个单独元素创建列表，你需要在选择集的每个元素上执行单独的操作。

3.1.2 解决方案

假定你希望建立某个DOM元素中所有链接的一个列表（可能这是个有很多用户提供内容的网站，你希望快速地查看用户提交的链接）。可以首先创建自己的jQuery选择$("div#post a[href]")，这将选择id为post的 中具有href属性的所有链接。然后，我们希望循环读取每个匹配的元素，将其附加到一个数组中，参见如下的代码示例：

```
var urls = [];
 $("div#post a[href]").each(function(i) {
    urls[i] = $(this).attr('href');
});
alert(urls.join(","));
```

使用$().each();方法，就能循环读取jQuery对象中的每个元素，建立一个数组。可以访问单独的元素，并对这些元素执行jQuery方法，因为在jQuery包装器$()中包装了this变量，从而使它变成了一个jQuery对象。

3.1.3 讨论

jQuery提供了一个核心方法，可以用它来循环读取选择的DOM元素集。$().each()就是jQuery的for循环，它将在集合中循环，并为集合中的每个元素提供一个单独的函数作用域。$().each()专用于jQuery对象中的循环。

> **警告**
>
> $().each();不同于jQuery工具方法jQuery.each(object, callback);。jQuery.each方法是更通用化的循环方法，能够在对象和数组中循环。关于jQuery.each()的更多信息参见jQuery的在线文档：
> http://docs.jquery.com/Utilities/jQuery.each。

在每次循环中，获取主选择集中当前元素的href属性。可以通过this关键字获取当前的DOM元素，然后将其包装在jQuery对象$(this)中，这样就可以对它执行jQuery方法/操作——在例子中是从DOM元素中提取href属性。最后的操作是将href属性赋值给一个全局数组urls。

我们已经看到，URL数组用逗号相互连接并在一个警告框中显示。还可以将这个列表添加到一个无序列表DOM元素显示给用户。更实用的方法是，可以将URL列表格式化为JSON格式，发送到服务器，以便在数据库中处理。

下面我们来看看另一个使用$().each();的例子。这个例子可能是$().each();最显而易见的用法。假定有一个无序的名称列表，希望突出显示每个名称。实现方法是为所有列表项设置交替的背景颜色：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 1 - Looping through a set of selected results</title>
     <style type="text/css">
         .even { background-color: #ffffff; }
         .odd { background-color: #cccccc; }
```

```
    </style>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js "
type="text/javascript"></script>
    <script type="text/javascript">
         (function($){
            $(document).ready(function() {
                $("ul > li").each(function(i) {
                    if (i % 2 == 1)
                    {
                        $(this).addClass("odd");
                    }
                    else
                    {
                        $(this).addClass("even");
                    }
                });
            });
        })(jQuery);
    </script>
</head>
<body>
    <h2>Family Members</h2>
    <ul>
        <li>Ralph</li>
        <li>Hope</li>
        <li>Brandon</li>
        <li>Jordan</li>
        <li>Ralphie</li>
    </ul>
</body>
</html>
```

图3-1展示了上述代码的输出效果。

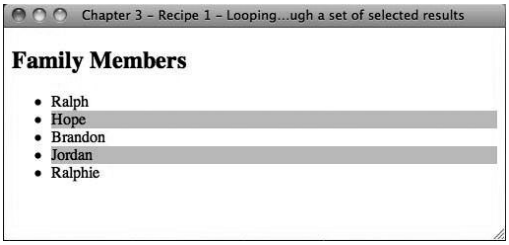


图3-1　代码输出

在循环读取每个<li>元素时，测试当前索引（作为执行函数时的唯一参数）除以2的余数是否为1，根据这个条件设置CSS类（.odd或者.even）。

注意

尽管这是$().each()最显而易见的用法，但是并不是处理交替背景颜色的最有效手段。可以用一行代码实现这一功能：

```
$("ul > li:odd").addClass("odd");
```

所需要做的是在CSS中将所有<li>元素设置为.even类，这样就可以用jQuery将索引为奇数的<li>元素设置为.odd类。

$.each()：的基本功能是通过索引的引用循环读取匹配集合中的每个元素，执行某种操作，然后转到匹配集中的下一个元素，直到集合中所有元素都处理完。

3.2 将选择集缩减为某个特定项

3.2.1 问题

jQuery选择器很广泛，根据查询选择页面上的所有元素。在需要根据单个元素的位置选择它时，如果不对代码进行编辑，就没有一种简单的方法能够选择该元素。

3.2.2 解决方案

使用jQuery选择元素之后，可以链接.eq()方法并传入你想要处理的选择集索引。

> **注意**
>
> 选择集的索引值从0开始，选择集中的第一个项是$().eq(0)；这里的0代表选择集中的第1项。$ ().eq(4)；代表的是第5项。

我们以美国全国冰球联赛（NHL）季末排名为例，说明显示哪些球队进入季后赛，哪些球队被淘汰的方法。我们需要做的是按照赛季结束时的积分列出每个联盟中的所有球队。因为每个联盟的前8名有资格进入季后赛，所以只需要在每个列表中找到8个列表项，然后画上一条线：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 2 - Reducing the selection set to specified item</title>
     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
     <script type="text/javascript">
          (function($){
             $(document).ready(function(){
                 $("ol#east > li").eq(7).css("border-bottom", "1px solid #000000");
                 $("ol#west > li").eq(7).css("border-bottom", "1px solid #000000");
             });
         })(jQuery);
     </script>
</head>
<body>
     <h2>Eastern Conference</h2>
     <ol id="east">
         <li>Boston Bruins</li>
         <li>Washington Capitals</li>
         <li>New Jersey Devils</li>
         <li>Pittsburgh Penguins</li>
         <li>Philadelphia Flyers</li>
         <li>Carolina Hurricanes</li>
         <li>New York Rangers</li>
         <li>Montreal Canadians</li>
         <li>Florida Panthers</li>
         <li>Buffalo Sabres</li>
         <li>Ottawa Senators</li>
         <li>Toronto Maple Leafs</li>
         <li>Atlanta Thrashers</li>
         <li>Tampa Bay Lightning</li>
         <li>New York Islanders</li>
     </ol>
     <h2>Western Conference</h2>
     <ol id="west">
         <li>San Jose Sharks</li>
         <li>Detroit Red Wings</li>
         <li>Vancouver Canucks</li>
         <li>Chicago Blackhawks</li>
         <li>Calgary Flames</li>
         <li>St. Louis Blues</li>
         <li>Columbus Blue Jackets</li>
         <li>Anaheim Ducks</li>
```

```
        <li>Minnesota Wild</li>
        <li>Nashville Predators</li>
        <li>Edmonton Oilers</li>
        <li>Dallas Stars</li>
        <li>Phoenix Coyotes</li>
        <li>Los Angeles Kings</li>
        <li>Colorado Avalanche</li>
    </ol>
</body>
</html>
```

图3-2展示了代码的输出效果。

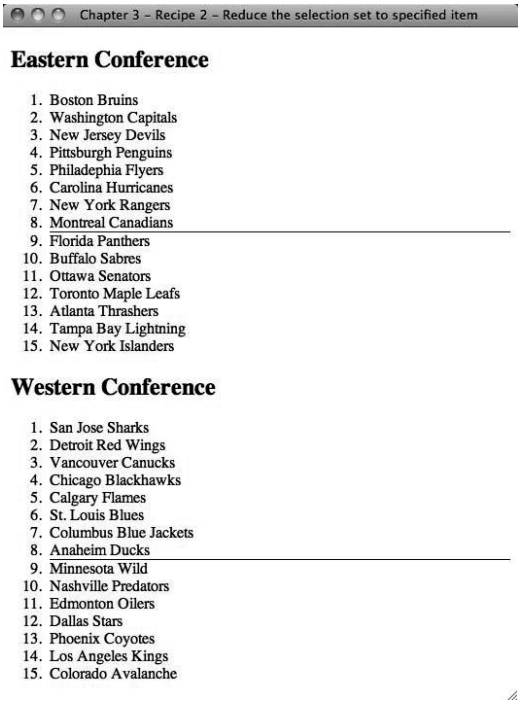


图3-2　代码输出

正如你所看到的，我们只使用了一个排序列表按照球队的顺位列出它们，然后使用jQuery在每个列表中的第8个项上添加一个下边框。如果使用`$("li").eq(7);`，就只会选择第一个列表中的对应元素，因为这个查询将页面上的所有`<li>`元素一起计数。

3.2.3　讨论

`.eq()`方法用于将一个选择集缩减为该集合中的一个元素，方法的参数是该元素的索引。索引从0开始，到长度−1结束。如果该参数是个无效索引，方法将返回一个空的元素集合，而不是null。

`.eq()`方法类似于在选择集中使用`$(":eq()");`，但是`.eq()`方法可以链接到选择集上，进一步微调结果，例如：

```
$("li").css("background-color","#CCCCCC").eq(0).css("background-color","#ff0000");
```

上述代码改变所有`<li>`元素的背景，然后选择第一个元素，用不同的颜色表示它可能是个标题项。

3.3 将选中的jQuery对象转换为原始DOM对象

3.3.1 问题

用jQuery在一个页面上选择元素返回的集合是一个jQuery对象而不是一个原始的DOM对象。因为它是jQuery对象，所以只能对选择集运行jQuery方法。要在选择集合上运行DOM方法和属性，该集合必须转换为一个原始的DOM对象。

3.3.2 解决方案

jQuery提供一个核心方法get()，将所有匹配的jQuery对象转换为一个DOM对象集合。此外，还可以传递一个索引值作为get()的参数，这样该方法以DOM对象（如$.get(1);）的形式返回匹配集中给定索引的元素。现在，虽然可以通过$.get(index)获得单个元素的DOM对象，它的存在也只是因为历史的原因；“最佳实践”是使用[] notation, $("div")[1];。

> **警告**
>
> 我们讨论的是将jQuery对象转换为DOM数组的核心.get()方法，而不是Ajax get方法——该方法用HTTP GET请求加载一个远程页面。

因为get()方法返回一个数组，所以可以遍历数组获得每个DOM元素。一旦获得DOM元素，就可以对它调用传统的DOM属性和方法。我们来研究一个从元素中取得innerHTML属性的简单例子：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 3 - Converting a selected jQuery object into a
raw DOM object</title>
     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
     <script type="text/javascript">
          (function($){
             $(document).ready(function(){
                  var inner = $("div")[0].innerHTML;
                  alert(inner);
             });
         })(jQuery);
     </script>
</head>
<body>
     <div>
         <p>
             jQuery, the write less, do more JavaScript library. Saving the day
for web developers since 2006.
         </p>
     </div>
</body>
</html>
```

图3-3展示了输出的效果。

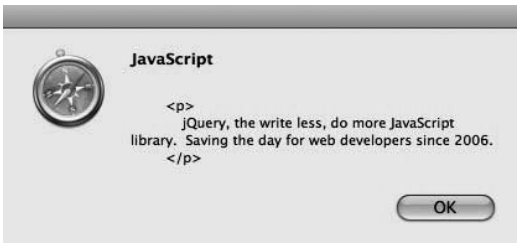


图3-3　代码输出

从选择页面上的所有 元素并调用[0]开始。传入想要处理的选择元素索引；因为页面上只有一个 ，所以传入的索引为0。最后，调用一个属性（这里是innerHTML）读取原始的DOM元素。

3.3.3　讨论

核心的get()方法可能非常有用，因为可以利用一些非JavaScript方法。假定有一个列表，需要以逆序显示。因为get()返回一个数组，所以可以使用原生的数组方法反向排序列表，然后重新显示：

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 3 - Converting a selected jQuery object into a raw DOM
object</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
    <!--
         (function($){
            $(document).ready(function(){
                var lis = $("ol li").get().reverse();
                $("ol").empty();
                $.each(lis, function(i){
                    $("ol").append("<li>" + lis[i].innerHTML + "</li>");
                });
            });
        })(jQuery);
    //-->
    </script>
</head>
<body>
    <h2>New York Yankees - Batting Line-up</h2>
    <ol>
        <li>Jeter</li>
        <li>Damon</li>
        <li>Teixeira</li>
        <li>Posada</li>
        <li>Swisher</li>
        <li>Cano</li>
        <li>Cabrera</li>
        <li>Molina</li>
        <li>Ransom</li>
    </ol>
</body>
</html>
```

图3-4显示了输出的效果。

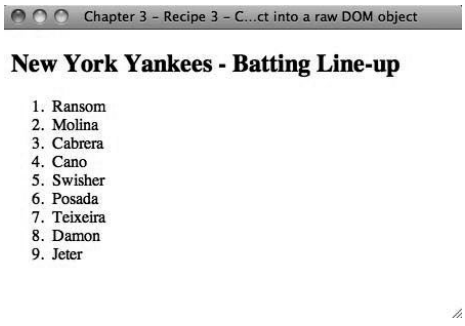


图3-4　代码输出

3.4 获得选择集中某个元素的索引

3.4.1 问题

为页面上选择的许多元素绑定事件时，你需要知道选择集合中单击的到底是哪一个元素，以便“个性化”绑定事件的操作。

3.4.2 解决方案

当单击一个元素时，可以使用核心方法index()搜索整个选择集，了解该元素的索引：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 4 - Getting the index of an item in a selection</title>
     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
     <script type="text/javascript">
     <!--
          (function($){
             $(document).ready(function(){
                 $("div").click(function() {
                     alert("You clicked on div with an index of " +
$("div").index(this));
                     });
                 });
             })(jQuery);
         //-->
     </script>
</head>
<body>
     <div>click me</div>
     <div class="test">test</div>
     <div>click me</div>
</body>
</html>
```

从绑定所有 元素的单击事件开始。当单击 时，可以搜索相同选择集中的元素找出单击的是哪一个 ：$("div").index(this)；，其中的this是单击的 。

3.4.3 讨论

可以使用核心方法index()获得所查找的DOM元素在jQuery集合中的索引。从jQuery 1.2.6开始，还能够以所要搜索的jQuery集合作为index()方法的参数。该方法将返回它所找到的第一个元素的索引：

```
var test = $("div.test");
$("div").each(function(i){
     if ($(this).index(test) >= 0)
     {
         //进行一些处理
     }
     else
     {
         //进行另一些处理
     }
});
```

将检查循环中的 是否匹配保存在变量test中的集合，如果匹配，将在匹配的集合上执行自定义操作。

注意

如果索引方法找不到传入的对象，将返回−1。

3.5 从现有数组中建立独特的数组

3.5.1 问题

在你的网页上有一个有序列表。你用jQuery选择了列表中的所有<li>元素；现在你需要将这个列表转换为另一个列表。

3.5.2 解决方案

假定在一个有序列表中列出一组人名。我们希望将这个有序列表中的前三个人名作为一个句子显示：

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 5 - Making a unique array of values from an existing
array</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
    <!--
         (function($){
            $(document).ready(function(){
                var arr = $.map($("LI"), function(item, index){
                    while (index < 3)
                    {
                        return $(item).html();
                    }
                    return null;
                });
                $(document.body).append("<span>The first three authors are: " +
arr.join(", ") + "</span>");
            });
        })(jQuery);
    //-->
    </script>
</head>
<body>
    <h1>jQuery Cookbook Authors</h1>
    <ol>
        <li>John Resig</li>
        <li>Cody Lindley</li>
        <li>James Padolsey</li>
        <li>Ralph Whitbeck</li>
        <li>Jonathan Sharp</li>
        <li>Michael Geary</li>
        <li>Scott González</li>
        <li>Rebecca Murphey</li>
        <li>Remy Sharp</li>
        <li>Ariel Flesler</li>
        <li>Brian Cherne</li>
        <li>Jörn Zaefferer</li>
        <li>Mike Hostetler</li>
        <li>Nathan Smith</li>
        <li>Richard D. Worth</li>
        <li>Maggie Wachs</li>
        <li>Scott Jehl</li>
        <li>Todd Parker</li>
        <li>Patty Toland</li>
        <li>Rob Burns</li>
    </ol>
</body>
</html>
```

图3-5展示了输出的效果。

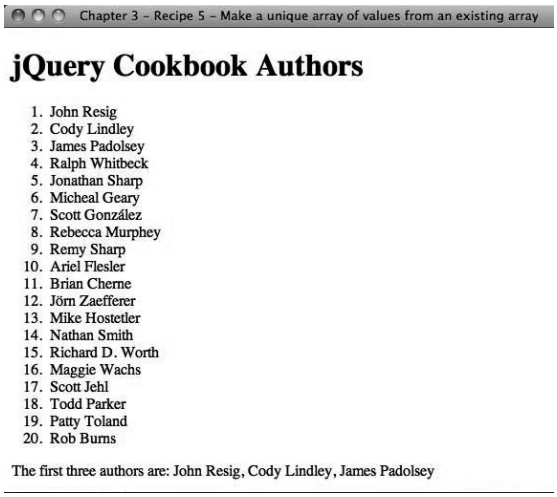


图3-5　代码输出

先从有序列表中建立一个<li>元素数组。将使用一个jQuery选择器选择页面上的所有<li>元素，并将其作为jQuery工具方法$.map()的参数，该方法将一个现有的数组“映射”到另一个数组。第二个参数是一个函数，循环读取数据、执行转换，并返回存储在新数组中的新值。

在上述的例子中，循环读取自己制作的数组，仅返回前三个列表元素的html()值，并将这些值映射到新的数组中。然后，用连接方法将该数组转换为一个字符串，并将其注入文档的尾部。

3.5.3　讨论

在这个解决方案中，将使用jQuery工具方法$.map()，该方法的功能是将现有数组转换为另一个数组。$.map()有两个参数——一个数组和一个回调函数：

```
$.map([1,2,3], function(n,i) { return n+i;});
//输出: [1,3,5]
```

$.map()将遍历原始数组的所有元素，传入待转换的元素以及数组的当前位置索引。该方法应该返回一个值，该值将被插入新的数组。

注意

如果返回的是null值，则新数组中不保存任何值。返回null基本上将该元素从新数组中删除。

3.6 在选择集合的子集上执行某项操作

3.6.1 问题

你需要在一个标记集合上执行某种操作，但是没有办法将这些标记与jQuery选择集中的其他页面标记隔离。

3.6.2 解决方案

可以使用slice()方法过滤选择集合，形成一个子集。可以向该方法传递起始索引值和结束索引值，然后再链接操作：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 6 - Performing an action on a subset of the selected
set</title>
     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
     <script type="text/javascript">
     <!--
          (function($){
             $(document).ready(function(){
$("p").slice(1,3).wrap("<i></i>");
             });
         })(jQuery);
     //-->
     </script>
</head>
<body>
     <p>
         Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin eget nibh ut
tortor egestas pharetra. Nullam a hendrerit urna. Aenean augue arcu, vestibulum eget
faucibus nec, auctor vel velit. Fusce eget velit non nunc auctor rutrum id et ante.
Donec nec malesuada arcu. Suspendisse eu nibh nulla, congue aliquet metus. Integer
porta dignissim magna, eu facilisis magna luctus ac. Aliquam convallis condimentum
purus, at lacinia nisi semper volutpat. Nulla non risus justo. In ac elit vitae elit
posuere adipiscing.
     </p>
     <p>
         Aliquam gravida metus sit amet orci facilisis eu ultricies risus iaculis. Nunc
tempus tristique magna, molestie adipiscing nibh bibendum vel. Donec sed nisi luctus
sapien scelerisque pretium id eu augue. Mauris ipsum arcu, feugiat non tempor
tincidunt, tincidunt sit amet turpis. Vestibulum scelerisque rutrum luctus. Curabitur
eu ornare nisl. Cras in sem ut eros consequat fringilla nec vitae felis. Nulla
facilisi. Mauris suscipit feugiat odio, a condimentum felis luctus in. Nulla interdum
dictum risus, accumsan dignissim tortor ultricies in. Duis justo mauris, posuere vel
convallis ut, auctor non libero. Ut a diam magna, ut egestas dolor. Nulla convallis,
orci in sodales blandit, lorem augue feugiat nulla, vitae dapibus mi ligula quis
ligula. Aenean mattis pulvinar est quis bibendum.
     </p>
     <p>
         Donec posuere pulvinar ligula, nec sagittis lacus pharetra ac. Cras nec
tortor mi. Pellentesque et magna vel erat consequat commodo a id nunc. Donec velit
elit, vulputate nec tristique vitae, scelerisque ac sem. Proin blandit quam ut magna ultrices porttitor
     </p>
     <p>
         Suspendisse facilisis molestie hendrerit. Aenean congue congue sapien, ac
luctus nulla rutrum vel. Fusce vitae dui urna. Fusce iaculis mattis justo sit amet
varius. Duis velit massa, varius in congue ut, tristique sit amet lorem. Curabitur
porta, mauris non pretium ultrices, justo elit tristique enim, et elementum tellus
enim sit amet felis. Sed sollicitudin rutrum libero sit amet malesuada. Duis vitae
gravida purus. Proin in nunc at ligula bibendum pharetra sit amet sit amet felis.
Integer ut justo at massa ullamcorper sagittis. Mauris blandit tortor lacus,
convallis iaculis libero. Etiam non pellentesque dolor. Fusce ac facilisis ipsum.
```

```
Suspendisse eget ornare ligula. Aliquam erat volutpat. Aliquam in porttitor purus.
    </p>
    <p>
        Suspendisse facilisis euismod purus in dictum. Vivamus ac neque ut sapien
fermentum placerat. Sed malesuada pellentesque tempor. Aenean cursus, metus a
lacinia scelerisque, nulla mi malesuada nisi, eget laoreet massa risus eu felis.
Vivamus imperdiet rutrum convallis. Proin porta, nunc a interdum facilisis, nunc dui aliquet sapien, nc
    </p>
</body>
</html>
```

图3-6展示了输出的效果。

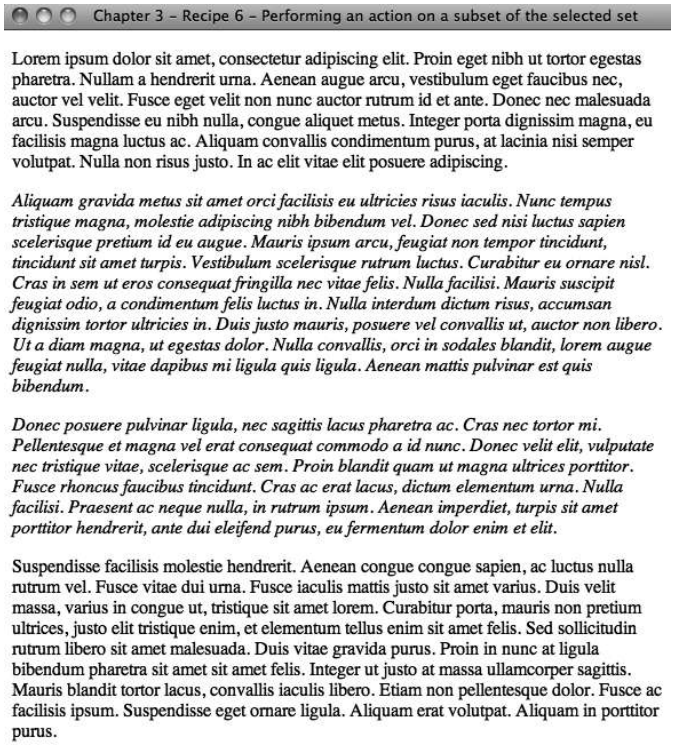


图3-6　代码输出

上述例子选择从索引1开始，在索引3之前结束的子集，然后将子集包装在一个斜体标记中。

3.6.3　讨论

jQuery方法`slice()`有两个选项；第一个是起始索引位置，第二个参数是可选的结束索引位置。假定你想要除了第一个之外的`<P>`标记，可以使用`$("p").slice(1)`，它将从jQuery选择集的第二个元素开始，选择余下的所有元素。

`slice()`的参数也可以是负数。如果参数是负数，将从选择集的结束位置开始计数。所以，`$("p").slice(-1)`；选择的是选择集中的最后一个元素。此外，`$("p").slice(1, -2)`；将从第二个位置开始选择，直到倒数第二个元素。

3.7 配置jQuery，避免与其他程序库冲突

3.7.1 问题

如果jQuery和另一个JavaScript程序库在同一个网页中加载，两个程序库可能都实现了$变量，这会导致这些方法中只有一个能够正常工作。

3.7.2 解决方案

假定你继承一个需要更新的网页，以前的程序员使用了另一个JavaScript程序库（如Prototype），而你仍然打算使用jQuery。这会导致冲突，根据两个程序库在页面头部排列的先后，其中一个程序库无法正常工作。

如果在同一个页面声明jQuery和Prototype，如：

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/prototype/1.6.0.3/prototype.js"></script>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
```

这会导致一个JavaScript错误：*element.dispatchEvent is not a function inprototype.js*.（element.dispatchEvent不是prototype.js中的函数）。幸好，jQuery提供了jQuery.noConflict()方法来解决这一问题：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 7 - Configuring jQuery to free up a conflict with
another library</title>

     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/prototype/1.6.0.3/prototype.js"></script>
     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
     <script type="text/javascript">
     <!--
          jQuery.noConflict();

          // Use jQuery via jQuery(...)
          jQuery(document).ready(function(){
             jQuery("div#jQuery").css("font-weight","bold");
          });

          // Use Prototype with $(...), etc.
          document.observe("dom:loaded", function() {
               $('prototype').setStyle({
                    fontSize: '10px'
               });
          });
     //-->
     </script>

</head>
<body>
     <div id="jQuery">Hello, I am a jQuery div</div>
     <div id="prototype">Hello, I am a Prototype div</div>
</body>
</html>
```

图3-7展示了输出效果。

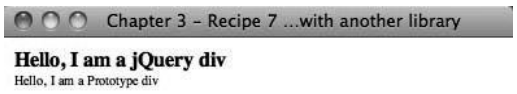


图3-7　代码输出

当调用jQuery.noConflict()时，它会将$变量的控制权交还给首先实现它的程序库。在释放$变量之后，就只能用jQuery变量访问jQuery。例如，在过去一直使用$("div p")的地方，现在要使用jQuery("div p")。

3.7.3　讨论

jQuery程序库及其所有插件都受到jQuery命名空间的约束。不应该与jQuery程序库和其他任何程序库（也就是Prototype、YUI等）冲突。而jQuery使用$作为jQuery对象的快捷方式，这个快捷方式定义与其他也使用$变量的程序库冲突。正如我们在这个解决方案中所看到的，可以释放jQuery的$快捷方式，回到jQuery对象上。

还有另一种选项。如果你希望确保jQuery不与另一个程序库冲突，而且仍然有简短名称的好处，可以调用jQuery.noConflict()并将其复制给一个变量：

```
var j = jQuery.noConflict();

j(document).ready(function(){
  j("div#jQuery").css("font-weight","bold");
});
```

可以选择jQuery.noConflict()赋值的变量名，定义自己的简写名称。

最后一种选择是在闭包中封装jQuery代码：

```
jQuery.noConflict();

(function($){
    $("div#jQuery").css("font-weight","bold");
})(jQuery);
```

通过闭包的使用，在函数中运行时，暂时使$变量可用于jQuery对象。一旦函数结束，$变量将归还给最早具有控制权的程序库。

警告

如果你使用这种技术，就不能在封装函数内部使用其他程序库中需要$的方法。

3.8 用插件增加功能

3.8.1 问题

jQuery程序库是一个小型、灵活而强大的JavaScript程序库，但是它并没有预先加载所需要的所有功能。

3.8.2 解决方案

jQuery在构建时考虑了可扩展性。如果jQuery核心程序库无法满足你的需要，可能会有jQuery插件制作者编写了应对这一需求的插件，它可能只需要一行代码。

要在网页包含一个插件，只需要下载插件的.js文件，在页面上包含jQuery程序库，并立刻在页面中包含插件。然后，通常需要在另一个.js文件或者页面上的一个脚本块中调用插件，并提供必要的选项。

下面是使用Mike Alsup开发的jQuery cycle插件（*http://jquery-cookbook.com/go/plugin-cycle*）的例子：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 8 - Adding Functionality with Plugins</title>
     <style type="text/css">
          .pics {
               height: 232px;
               width:  232px;
               padding: 0;
               margin:  0;
          }

          .pics img {
               padding: 15px;
               border: 1px solid #ccc;
               background-color: #eee;
               width:  200px;
               height: 200px;
               top:  0;
               left:  0
          }
     </style>
     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
     <!--Now include your plugin declarations after you've declared jQuery on the page-->
      <script type="text/javascript" src="scripts/2.8/jquery.cycle.all.min.js?
v2.60"></script>
     <script type="text/javascript">
     <!--
          (function($){
               $(document).ready(function(){
                    $('.pics').cycle('fade');
               });
          })(jQuery);
     //-->
     </script>
</head>
<body>
     <div class="pics">
          <img src="images/2.8/beach1.jpg" width="200" height="200" alt="Beach 1" />
          <img src="images/2.8/beach2.jpg" width="200" height="200" alt="Beach 2" />
          <img src="images/2.8/beach3.jpg" width="200" height="200" alt="Beach 3" />
     </div>
</body>
</html>
```

图3-8展示了输出效果。

图3-8　代码输出（一幅图像谈入另一幅图像中）

只用一行代码，就能完成一个幻灯片效果，一次显示一幅图像，然后自动淡入下一幅图像。Cycle插件的编写方式也是可以扩展的，开发人员可以提供不同的选项，产生不同的过渡效果和布局。

3.8.3　讨论

jQuery有JavaScript程序库中最大的开发人员社区之一。大的社区能够贡献大量可在网上找到的插件和教程。jQuery托管着一个插件库，作者编写插件之后提交到*http://plugins.jquery.com*。在这个插件库中目前已经有了超过1600个插件，可以在许多不同的分类中找到插件。插件作者受邀提交插件并给出描述、指向插件的链接和指向插件文档的链接。这个插件库帮助开发人员更轻松地搜索他们所需要的特定功能。

作为开发人员，你很可能最终找到符合需求的插件。但是万一这样的插件不存在，自己动手创建一个也很容易。下面是需要牢记的一些要点：

- 文件取名为*jquery.[插件名称].js*，如*jquery.debug.js*。
- 所有新的方法都附属于*jQuery.fn*对象；所有函数附属于jQuery对象。
- 在方法中，*this*是对当前jQuery对象的引用。
- 所附加的任何方法或者函数最后必须有一个分号（；），否则代码在压缩的时候将被破坏。
- 除非明确指出，否则方法必须返回jQuery对象。
- 应该使用`this.each`循环读取当前匹配元素集——这样制作出来的代码清晰且有很好的兼容性。
- 在插件代码中始终使用jQuery代替$——这样用户可以在单个位置修改jQuery的别名。

创建插件的更多信息和例子可以参考*http://docs.jquery.com/Plugins/Authoring*，或者跳到第12章，Mike Hosteler将在该章中更详细地介绍这方面的知识。

3.9 确定使用的到底是哪一个查询

3.9.1 问题

在编写扩展jQuery的插件或者方法时，你需要知道调用方法时所使用的确切选择集和上下文，以便重新调用方法。

3.9.2 解决方案

可以结合使用核心属性.selector和m.context，重新创建原始查询。必须将两个核心结合使用，因为函数或者插件中的所有查询并不都在默认文档上下文中：

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
     <title>Chapter 3 - Recipe 9 - Determining the exact query that was used</title>
     <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
     <script type="text/javascript">
     <!--
          (function($){
             $.fn.ShowQuery = function(i) {
                 alert("$(\""+ $(this).selector + "\", " + $(this).context +")");
                 if (i < 3)
                 {
                      $($(this).selector, $(this).context).ShowQuery(i+1);
                 }
             };
             $("div").ShowQuery(1);
         })(jQuery);
     //-->
     </script>
</head>
<body>
     <div>
         This is a div.
     </div>
</body>
</html>
```

图3-9展示了输出效果。

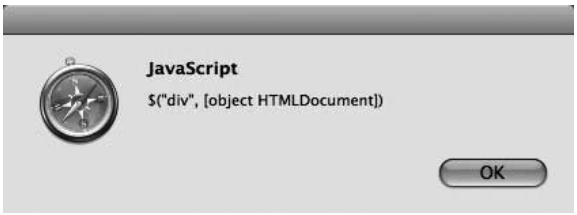


图3-9 代码输出（警告框）

3.9.3 讨论

在上述的示例中，定义了一个可以从jQuery选择集中调用的方法——ShowQuery。在该方法中，用警告框显示了传入的查询，然后用相同的jQuery选择器再次调用ShowQuery方法。if语句用于避免进入递归循环。

核心属性.selector和.context在2009年1月发行的jQuery1.3中引入。这些方法更多地是为需要在传入的原始查询上进行操作的插件开发人员所准备的，它们可能的用例是返回选择查询或者检查某个元素是否在选择集中。

.selector以字符串形式返回用于匹配指定元素的实际选择器。如果使用一个选择器得到选择集，然后用find()方法缩小范围而破坏该选择集，.selector将返回原始的选择器：

```
$("div").find("a").selector;

//返回:"div a"
```

.context返回传入jQuery()中的原始DOM节点。如果选择器中没有设置上下文，默认的上下文将是文档。

第4章　jQuery工具

Jonathan Sharp

4.0 导言

在考虑和谈论jQuery时，首先想到的主要概念往往是DOM和样式操纵及行为（事件）。但是，开发人员还可以从许多“核心”功能和工具函数中得到很大的好处。本章的中心就是揭示、说明和解释这些不那么常用的jQuery工具方法。

4.1 用jQuery.suppoert检测功能

4.1.1 问题

你需要为所有仅仅有当前页面hash的锚标记附加特殊的click处理程序，而且不希望因为浏览器支持问题而使其损坏。

4.1.2 解决方案

```
(function($) {
    $(document).ready(function() {
     $('a')
        .filter(function() {
            var href = $(this).attr('href');
            //规范化URL
            if ( !jQuery.support.hrefNormalized ) {
                var loc = window.location;
                href = href.replace( loc.protocol + '//' + loc.host + loc.pathname,
'');
            }
            // 这个锚标记的形式为<a href="#hash">
            return ( href.substr(0, 1) == '#' );
        })
        .click(function() {
            // 特殊单击处理程序代码
        });
    });
})(jQuery);
```

4.1.3 讨论

jQuery.support对象在版本1.3中加入，包含一些布尔标志，帮助开发人员编写使用浏览器功能检测的代码。在例子中，Internet Explorer（IE）处理href属性时和其他浏览器的表现不同。IE将返回完整的URL，而不是href属性的内容。使用hrefNormalized属性，就拥有了应对IE更高版本改变这一表现的预防措施。否则，就需要包含具体浏览器版本的条件语句。尽管这种功能很有诱惑力，但是最好避免采用这种方法，因为它需要在浏览器发行新版本时进行维护工作。避免针对特定浏览器的另一个原因是，客户端可能有意无意地报告不正确的用户代理串。除了hrefNormalized属性，还存在另外一些属性：

boxModel

如果浏览器呈现器按照W3C CSS“盒子模型”规范显示，返回真值。

cssFloat

如果使用style.cssFloat获得当前CSS浮动值，返回真值。

hrefNormalized

如果浏览器不改变getAttribute('href')的结果，返回真值。

htmlSerialize

如果浏览器正常序列化带有innerHTML属性的链接元素，返回真值。

leadingWhitespace

如果浏览器使用innerHTML时保留前导空格，返回真值。

noCloneEvent

如果浏览器克隆元素时不克隆事件处理程序，返回真值。

objectAll

如果元素上的getElementsByTagName('*')返回所有后代元素，返回真值。

opacity

如果浏览器能够解释CSS透明度样式，返回真值。

scriptEval

如果对<script>使用appendChild将执行脚本，返回真值。

style

如果getAttribute('style')能返回元素指定的嵌入样式，返回真值。

tbody

如果浏览器允许没有<tbody>元素的<table>元素，返回真值。

4.2 用jQuery.each循环读取数组和对象

4.2.1 问题

你需要循环读取数组中的每个元素或者对象的各个属性。

4.2.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var months = [ 'January', 'February', 'March', 'April', 'May',
                       'June', 'July', 'August', 'September', 'October',
                       'November', 'December'];
        $.each(months, function(index, value) {
            $('#months').append('<li>' + value + '</li>');
        });
        var days = {     Sunday: 0, Monday: 1, Tuesday: 2, Wednesday: 3,
                       Thursday: 4, Friday: 5, Saturday: 6 };
        $.each(days, function(key, value) {
            $('#days').append('<li>' + key + ' (' + value + ')</li>');
        });
    });
})(jQuery);
```

4.2.3 讨论

在本秘诀中，用$.each()循环读取一个数组和一个对象，该方法为常见循环任务提供了简洁的接口。$.each()的第一个参数是需要循环读取的数组或者对象，第二个参数是在每个元素上执行的回调方法（注意，这和jQuery的集合方法$('div').each()有所不同，它的第一个参数是回调函数）。

当执行开发人员定义的回调函数时，this变量被设置为目前循环读取到的元素值。因此，前一个秘诀中的代码可以改写为：

```
(function($) {
    $(document).ready(function() {
        var months = [ 'January', 'February', 'March', 'April', 'May',
                       'June', 'July', 'August', 'September', 'October',
                       'November', 'December'];
        $.each(months, function() {
            $('#months').append('<li>' + this + '</li>');
        });
        var days = {    Sunday: 0, Monday: 1, Tuesday: 2, Wednesday: 3,
                       Thursday: 4, Friday: 5, Saturday: 6 };
        $.each(days, function(key) {
            $('#days').append('<li>' + key + ' (' + this + ')</li>');
        });
    });
})(jQuery);
```

4.3 用jQuery.grep过滤数组

4.3.1 问题

你需要过滤和删除一个数组中的元素。

4.3.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var months = [ 'January', 'February', 'March', 'April', 'May',
                       'June', 'July', 'August', 'September', 'October',
                       'November', 'December'];
        months = $.grep(months, function(value, i) {
            return ( value.indexOf('J') == 0 );
        });
        $('#months').html( '<li>' + months.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

4.3.3 讨论

本秘诀使用$.grep()方法过滤months数组，使其只包含以大写字母J开始的条目。$.grep方法返回过滤后的数组。开发人员定义的回调方法有两个参数，返回布尔值true表示保留元素，返回false则删除元素。第一个参数是数组元素（本例中是月份）的值，第二个参数是$.grep()方法已经循环的次数。所以，如果你想每隔一个月份删除相应元素，可以测试条件(i % 2)== 0，该条件返回i/2的余数。（%是取模操作符，返回除法操作的余数。所以当i=4时，除于2的余数为0）。

```
(function($) {
    $(document).ready(function() {
        var months = [ 'January', 'February', 'March', 'April', 'May',
                       'June', 'July', 'August', 'September', 'October',
                       'November', 'December'];
        months = $.grep(months, function(value, i) {
            return ( i % 2 ) == 0;
        });
        $('#months').html( '<li>' + months.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

4.4 用jQuery.map循环修改数组元素

4.4.1 问题

你需要循环读取数组中的每个元素，修改其值。

4.4.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var months = [ 'January', 'February', 'March', 'April', 'May',
                       'June', 'July', 'August', 'September', 'October',
                       'November', 'December'];
        months = $.map(months, function(value, i) {
            return value.substr(0, 3);
        });
        $('#months').html( '<li>' + months.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

4.4.3 讨论

在本秘诀中，$.map()循环读取months数组，返回月份的缩写（前三个字母）。$.map()方法的参数为数组和一个回调方法，对数组的每个元素执行开发人员定义的回调方法。数组元素将用回调方法的返回值更新。

4.5 用jQuery.merge合并两个数组

4.5.1 问题

你有两个数组，需要合并或者连接它们。

4.5.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var horseBreeds = ['Quarter Horse', 'Thoroughbred', 'Arabian'];
        var draftBreeds = ['Belgian', 'Percheron'];

        var breeds = $.merge( horseBreeds, draftBreeds );
        $('#horses').html( '<li>' + breeds.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

4.5.3 讨论

在这个例子中，有两个包含马种列表的数组。按照1+2的顺序合并这两个数组，最后的breeds数组如下：

```
['Quarter Horse', 'Thoroughbred', 'Arabian', 'Belgian', 'Percheron']
```

4.6 用jQuery.unique过滤重复的数组元素

4.6.1 问题

你有两个jQuery DOM集合，需要删除重复的元素：

```
(function($) {
    $(document).ready(function() {
        var animals = $('li.animals').get();
        var horses = $('li.horses').get();
        $('#animals')
            .append( $(animals).clone() )
            .append( $(horses).clone() );
    });
})(jQuery);
```

4.6.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var animals = $('li.animals').get();
        var horses = $('li.horses').get();
        var tmp = $.merge( animals, horses );
        tmp = $.unique( tmp );
        $('#animals').append( $(tmp).clone() );
    });
})(jQuery);
```

4.6.3 讨论

jQuery的$.unique()函数从数组或者集合中删除重复的DOM元素。在前一个秘诀中，用$.merge()合并了animals和horses数组。jQuery在大部分核心和内部函数（如.find()和.add()）中使用了$.unique()。因此，这个方法的最常见用例是在不是由jQuery构造的数组上进行的操作。

4.7 用jQuery.isFunction测试回调函数

4.7.1 问题

你编写了一个插件，需要测试其中的一个设置是不是有效的回调函数。

4.7.2 解决方案

```
(function($) {
    $.fn.myPlugin = function(settings) {
        return this.each(function() {
            settings = $.extend({ onShow: null }, settings);
            $(this).show();
            if ( $.isFunction( settings.onShow ) ) {
                settings.onShow.call(this);
            }
        });
    };
    $(document).ready(function() {
        $('div').myPlugin({
            onShow: function() {
                alert('My callback!');
            }
        });
    });
})(jQuery);
```

4.7.3 讨论

虽然JavaScript语言提供了typeof操作符，但是必须考虑不同Web浏览器产生的不一致结果和边界情况。jQuery提供.isFunction()方法来简化开发人员的工作。值得一提的是，从版本1.3开始，这个方法可用于用户定义的函数，但是对内置的语言函数返回不一致的结果：

```
jQuery.isFunction( document.getElementById );
```

在Internet Explorer各个版本中，上述函数返回假值。

4.8 用jQuery.trim从字符串或者表单值中删除空格

4.8.1 问题

你有一个输入表单，需要删除用户在字符串开始或者结尾处输入的空格。

4.8.2 解决方案

```
<input type="text" name="first_name" class="cleanup" />
<input type="text" name="last_name" class="cleanup" />

(function($) {
    $(document).ready(function() {
        $('input.cleanup').blur(function() {
            var value = $.trim( $(this).val() );
            $(this).val( value );
        });
    });
})(jQuery);
```

4.8.3 讨论

在用户的焦点离开一个字段时，用$(this).val()读取用户输入的值，将其传递给$.trim()方法，删除字符串开始和结尾的空白字符（空格、制表符和换行符）。把修剪后的字符串重新设置为输入字段的值。

4.9 用jQuery.data将对象和数据附加到DOM中

4.9.1 问题

假定有如下的DOM代码：

```
var node = document.getElementById('myId');
node.onclick = function() {
    // 单击事件处理程序
};
node.myObject = {
    label: document.getElementById('myLabel')
};
```

为了方便引用，为DOM元素关联了元数据。因为有些Web浏览器的垃圾收集机制实现有缺陷，所以上述代码可能造成内存泄漏。

4.9.2 解决方案

由于某些Web浏览器的垃圾收集机制实现有缺陷，因此运行时添加到对象或者DOM节点中的属性（称作expandos）可能造成许多问题。jQuery为开发人员提供了直观而简练的方法.data()，帮助开发人员完全避免内存泄漏问题。

```
$('#myId').data('myObject', {
    label: $('#myLabel')[0]
});

var myObject = $('#myId').data('myObject');
myObject.label;
```

4.9.3 讨论

在本秘诀中，使用.data()方法管理对数据的访问，提供数据和标记的清晰分离。

使用date()方法的其他好处之一是隐式触发目标元素上的getData和setData事件，假设有如下HTML：

```
<div id="time" class="updateTime"></div>
```

可以附加一个setData事件处理程序分离不同的关注点（模式和视图），这个处理器方法有三个参数（event对象，数据key（键值）和数据value（值））：

```
// 监听新数据
$(document).bind('setData', function(evt, key, value) {
    if ( key == 'clock' ) {
```

```
        $('.updateTime').html( value );
    }
});
```

之后，每当在文档元素上调用.date()都会触发setData事件：

```
//在类为'updateTime'的所有元素上更新'time'数据
setInterval(function() {
    $(document).data('clock', (new Date()).toString() );
}, 1000);
```

这样，在前一个秘诀中，每隔1秒（1000毫秒）更新document对象上的clock数据属性，触发绑定到document的setData事件，更新当前时间的显示。

4.10 用jQuery.extend扩展对象

4.10.1 问题

你已经开发了一个插件，需要提供允许用户重写的默认选项。

4.10.2 解决方案

```
(function($) {
    $.fn.myPlugin = function(options) {
        options = $.extend({
            message: 'Hello world',
            css: {
                color: 'red'
            }
        }, options);
        return this.each(function() {
            $(this).css(options.css).html(options.message);
        });
    };
})(jQuery);
```

4.10.3 讨论

在本秘诀中，使用jQuery提供的$.extend()方法。$.extend()返回对传入的第一个对象的引用，稍后的对象将重写它们所定义的任何属性。下列代码演示了实际的工作方式：

```
var obj = { hello: 'world' };
obj = $.extend(obj, { hello: 'big world' }, { foo: 'bar' });

alert( obj.hello ); //显示警告'big world'
alert( obj.foo ); // 显示警告'bar';
```

这使得本秘诀中的myPlugin()可以接受一个options对象，由它覆盖默认设置。下列代码展示了最终用户覆盖默认的CSS color设置的方法：

```
$('div').myPlugin({ css: { color: 'blue' } });
```

$.extend()方法的一个特殊情况是，当参数是一个对象时，它将扩展基本jQuery对象。因此，可以这样定义自己的插件，以扩展jQuery核心：

```
$.fn.extend({
    myPlugin: function() {
        options = $.extend({
            message: 'Hello world',
            css: {
                color: 'red'
```

```
            }
        }, options);
        return this.each(function() {
            $(this).css(options.css).html(options.message);
        });
    }
});
```

$.extend()还提供了一个深（递归）拷贝的机制。这通过传递布尔值true作为第一个参数来实现。下面是深拷贝的一个例子：

```
var obj1 = { foo: { bar: '123', baz: '456' }, hello: 'world' };
var obj2 = { foo: { car: '789' } };

var obj3 = $.extend( obj1, obj2 );
```

当没有传递true参数时，obj3的内容如下：

```
{ foo: { car: '789 }, hello: 'world' }
```

如果指定深拷贝，obj3在递归复制所有属性之后的内容如下：

```
var obj3 = $.extend( true, obj1, obj2 );
// obj3
{ foo: { bar: '123', baz: '456', car: '789 }, hello: 'world' }
```

第5章　更快、更简单、更有趣

*Michael Geary*和*Scott González*

5.0 引言

几乎每天都有人在jQuery Google用户组上询问如何使代码更简单或者更快，或者如何调试一段无法正常工作的代码。

本章将帮助你简化自己的jQuery代码，使其更加易读，工作方式更有趣。我们还将分享一些寻找和修复此类缺陷的技巧。

我们还要帮助你使代码运行得更快，同样重要的是，找出代码中需要加速的部分。这样，网站的访问者在使用反应敏捷的网页时就能得到更多的乐趣。

这就是我们所说的双赢局面，编码快乐！

5.1 这不是jQuery，而是JavaScript

5.1.1 问题

你是一位初涉jQuery的Web设计人员，在`if/else`语句的语法上碰到了麻烦。你知道这肯定是一个简单的问题，在jQuery邮件列表上提问之前，你也做了功课：搜索jQuery文档，可是什么也没找到。在Web上搜索“jquery if else语句”之类的词语也没有得到什么帮助。

你需要将一个电子邮件地址在@符号处分为两部分。你已经听说有一个分割字符串的函数，但是在jQuery文档中似乎也没有相关的信息。

jQuery的文档真的这么差吗？

5.1.2 解决方案

`if/else`语句和用于字符串的`.spli()`方法是JavaScript的一部分，而不是jQuery的一部分。

所以，下面的这些Web搜索能够找到更有用的结果：

*javascript if else*语句

*javascript*字符串分割

5.1.3 讨论

JavaScript专家，请不要打击新手。

新手在遇到这问题的时候也不要觉得难过。如果你是JavaScript老手，可能会嘲笑这些问题。但是它们常常出现在jQuery邮件列表上，是完全可以理解的。jQuery的设计使简单的JavaScript编码变得非常轻松，甚至之前没有编程经历的人也能够理解基本概念，为页面添加有用的效果，根本无须学习“真正”的编程语言。

但是jQuery就是JavaScript。jQuery本身是100%纯粹的JavaScript代码，而你所写的每一行jQuery代码也是JavaScript的代码。

即使没有真正理解jQuery与JavaScript之间的关系，你也确实能够用jQuery完成简单的任务，但是对底层语言的了解越多，你的jQuery工作也就更有效率——挫败感也会越少。

5.2 $（this）出了什么问题

5.2.1 问题

你的一个事件处理程序为DOM元素添加类，用setTimeout()等待一秒钟，然后删除该类：

```
$(document).ready( function() {
    $('.clicky').click( function() {
        $(this).addClass('clicked');
        setTimeout( function() {
            $(this).removeClass('clicked');
        }, 1000 );
    });
});
```

当你单击时添加该类，但是却无法删除它。你已经确认正在调用setTimeout()中的代码，但是似乎什么也没做。 你以前使用过.removeClass()，代码看上去也是正确的。在两个地方你都使用了$(this)，但是在setTimeout()调用中它似乎不起作用。

5.2.2 解决方案

在调用setTimeout()之前将this`保存在一个变量中：

```
$(document).ready( function() {
    $('.clicky').click( function() {
        var element = this;
        $(element).addClass('clicked');
        setTimeout( function() {
            $(element).removeClass('clicked');
        }, 1000 );
    });
});
```

更好的做法是，因为将在两个位置调用$()，按照秘诀5.3中的建议，将$(this)复制到一个变量中，代替this：

```
$(document).ready( function() {
    $('.clicky').click( function() {
        var $element = $(this);
        $element.addClass('clicked');
        setTimeout( function() {
            $element.removeClass('clicked');
        }, 1000 );
    });
});
```

5.2.3 讨论

$(this)到底是什么？为什么总是不起作用呢？如果将它分成两部分——$()和this,就更容易理解。

$()看上去很神秘，实际上并非如此：它只是一个函数调用。$是对jQuery函数的引用，只是jQuery()较为简短的写法而已。它只是返回一个对象的普通JavaScript函数调用。

> **注意**
>
> 如果你要使用另一个重新定义$的JavaScript程序库，就是另外一回事了——之后你就不能在jQuery代码中使用$()，而应该使用jQuery()或者自定义的别名。

this是jQuery中较容易混淆的特性之一，因为它用在许多不同的方面。在面向对象的JavaScript编程中，this用于在对象方法中引用该对象，和Python或者Ruby中的self类似：

```
function Foo( value ) {
    this.value = value;
}
Foo.prototype.alert = function() {
    alert( this.value );
};
var foo = new Foo( 'bar' );
foo.alert(); // 'bar'
```

在传统onevent属性的代码中，this引用接收事件的元素——但是只在属性中，而不在从属性调用的函数中：

```
<a href="#" id="test" onclick="clicked(this);">Test</a>

function clicked( it ) {
    alert( it.id );    // 'test'
    alert( this.id );   // undefined
    alert( this === window ); // true (什么？)
}
```

从第三个alert()中可以看到，this实际上是函数中的window对象。由于历史原因，当直接调用函数(也就是说，不是作为对象方法调用)时,window是this的“默认”含义。

在jQuery事件处理程序中，this是处理事件的DOM元素，所以$(this)是用于该DOM元素的一个jQuery包装器。这就是为什么在“问题”代码中$(this).addClass()能够按照预期工作的原因。

但是，代码接着调用setTimeout()，setTimeout()以直接函数调用的方式工作：this指的是window对象。所以当代码调用$(this).removeClass()时，实际上试图从window对象中删除类！

为什么将this或者$(this)复制到一个局部变量中能够解决这个问题？JavaScript为参数和函数的局部变量创建了一个闭包。

闭包初看似乎很神秘，但实际上可以归纳为三条简单的原则：

- 可以在JavaScript函数中嵌套另一个函数，嵌套可以为多级。
- 函数不仅能读写自己的参数和局部变量，而且能读写嵌套函数中的变量。
- 前一条原则始终有效，即使外部函数已经返回之后再调用内部函数也是如此（例如，时间处理程序或者setTimeout()回调）。

不管函数是命名的还是匿名的，上述原则对所有函数都适用。但是，this不是函数参数或者局部变量——它是特殊的JavaScript关键字——所以这些原则不适用于它。通过将this的值复制到一个局部变量中，就能利用闭包使该值在任何嵌套函数中可用。

5.3 删除多余的重复

5.3.1 问题

你需要在页面加载时隐藏、显示或者操纵一些DOM元素，之后在响应一些不同事件时也需要采取同样的措施：

```
$(document).ready( function() {
    // 在启动时设置可见性
    $('#state').toggle( $('#country').val() == 'US' );
    $('#province').toggle( $('#country').val() == 'CA' );

    // 通过鼠标修改国家选择器时更新可见性
    $('#country').change( function() {
        $('#state').toggle( $(this).val() == 'US' );
        $('#province').toggle( $(this).val() == 'CA' );
    });

    //通过键盘修改国家选择器时也要更新
    $('#country').keyup( function() {
        $('#state').toggle( $(this).val() == 'US' );
        $('#province').toggle( $(this).val() == 'CA' );
    });
});
```

上述代码可以正常工作，但是你希望简化它，省略过多的重复代码。

注意

为什么既要处理change事件又要处理keyup事件？许多网站只处理选择列表的change事件，如果用鼠标进行选择，这就能够正常工作，但是如果单击选择列表，然后用上下光标键选择选项，就什么也不会发生：在列表中按键不会触发change事件。如果也处理keyup事件，选择列表就能响应光标键，为键盘用户提供更好的体验。

5.3.2 解决方案1

将重复的代码移入一个函数中，在加载的时候和响应事件的时候都调用该函数。使用jQuery的bind()方法同时连接两个事件处理程序，并将多次使用的数据保存在变量中：

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

var $country = $('#country');

function setVisibility() {
    var value = $country.val();
    $('#state').toggle( value == 'US' );
        $('#province').toggle( value == 'CA' );
    }
    setVisibility();
    $country.bind( 'change keyup', setVisibility );
});
```

5.3.3 解决方案2

在绑定事件处理程序之后立即用jQuery的事件触发机制触发事件，同时采用解决方案中的.bind()技术和局部变量：

```
$(document).ready( function() {
    $('#country')
        .bind( 'change keyup', function() {
            var value = $(this).val();
            $('#state').toggle( value == 'US' );
            $('#province').toggle( value == 'CA' );
```

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

5.3.4 讨论

将重复的代码移入可以从多处调用的函数是任何语言中的标准编程方法。解决方案1遵循了这种方法：将设置可见性的代码放在`setVisibility()`函数中，而不是一再重复。然后，代码在启动时直接调用该函数，而在`change`事件触发时间接调用该函数。

解决方案2也对于两种情况使用公共的函数。但是方案中的代码没有命名函数以便在启动时直接调用它，而是将函数设置为`change`事件的事件处理程序，然后用`trigger()`方法触发相同的事件——从而间接调用函数。

这些方法在某种程度上可以互换使用，使用哪一种只是个人喜好的问题。

5.4 格式化jQuery链

5.4.1 问题

你有一个冗长的jQuery链，包含.children()和.end()之类的方法，处理多组相关的元素。很难看出哪个操作应用到哪些元素上：

```
$('#box').addClass('contentBox').children(':header')
    .addClass('contentTitle').click(function() {
        $(this).siblings('.contentBody').toggle();
    }).end().children(':not(.contentTitle)')
    .addClass('contentBody').end()
    .append('<div class="contentFooter"></div>')
    .children('.contentFooter').text('generated content');
```

5.4.2 解决方案

将链中的每个方法调用自成一行，在每行的开始放置.操作符，然后将链接的每一部分缩进，指出你将要切换到不同元素集的地方。

在使用.children()或.siblings()之类的方法选择不同元素时增加缩进，在调用.end()返回前一个jQuery选择集时减少缩进。

如果你是jQuery新手，应该阅读第1章中有关基本链和.end()的秘诀：

```
$('#box')
    addClass('contentBox')
    .children(':header')
        .addClass('contentTitle')
        .click(function() {
            $(this).siblings('.contentBody').toggle();
        })
    .end()
    .children(':not(.contentTitle)')
        .addClass('contentBody')
    .end()
    .append('<div class="contentFooter"></div>')
    .children('.contentFooter')
        .text('generated content');
```

5.4.3 讨论

将每个调用自成一行，可以非常轻松地浏览代码和观察发生的情况。使用缩进指出修改元素集的时机，能够更好地跟踪破坏性操作发生以及用.end()撤销的时间。

缩进风格使得指定元素集的每个调用都对齐，即使没有破坏性操作也是如此。例如，尽管包装器 操作之间有对其他元素的操作，但是仍然能很清晰地看到 前后各添加了一个元素。

在每行的开头（而不是结尾）加上.操作符是点睛之笔：它更好地从视觉上提醒我们，这是方法调用而不是普通的函数调用。

注意

链是jQuery发明的吗？不。jQuery确实很好地使用了方法链，但是在JavaScript刚出现的时候，方法链就已经存在。

例如，下面这个就是字符串对象链的一个为人熟知的用法：

```
function htmlEscape(text){
    return text
        .replace('&', '&amp;')
        .replace('<', '&lt;')
```

```
        .replace('>', '&gt;');
}
```

5.5 从其他程序库借用代码

5.5.1 问题

你在另一个JavaScript程序库中找到一个有用的函数，希望在jQuery代码中使用相同的技术。在本例中，要使用来自Ext Core程序库的.radioClass()方法，该方法能为匹配的元素添加一个类并从匹配元素的所有兄弟元素中删除同一个类。

> **注意**
>
> .radioClass()这一名称来自于Web应用和桌面应用中单选按钮的表现，单击选中一个按钮之后，会删除同一组中的其他按钮。这种输入元素命名为单选按钮（radio button）是源自旧的汽车收音机上的选台按钮——当这种机械按钮按下之后，其他的所有按钮都会弹起。

假定有如下HTML：

```
<div>
    <div id="one" class="hilite">One</div>
    <div id="two">Two</div>
    <div id="three">Three</div>
    <div id="four">Four</div>
</div>
```

你想要这样运行代码：

```
// 为div#three添加"hilite"类,从其兄弟元素中删除该类(如div#one)

// $('#three').radioClass('hilite');
```

你甚至可能想要允许“多选”按钮类：

```
// 为div#three和div#four添加"hilite"类,从其兄弟元素中删除该类(div#one和div#three)

$('#two,#four').radioClass('hilite');
```

5.5.2 解决方案

编写一个简单的插件，将.radioClass()方法添加到jQuery中：

```
//从选中元素的所有兄弟元素中删除指定类,然后将该类添加到选中元素。按照这个顺序可以选中多个兄弟元素
//感谢Ext Core提供的思路
jQuery.fn.radioClass = function( cls ) {
    return this.siblings().removeClass(cls).end().addClass(cls);
};
```

这个函数很简短，所以写在一行中也不难理解，但是按照秘诀5.4所描述的那样缩进代码，能够很清晰地看出其工作方式：

```
jQuery.fn.radioClass = function( cls ) {
    return this                  // 开始链,返回其结果
        .siblings()              // 选择选中元素的所有兄弟元素
            .removeClass(cls)    // 从兄弟元素中删除该类
        .end()                   //回到原始选择集
        .addClass(cls);          // 为选中元素添加类
};
```

5.5.3 讨论

据报道，作曲家伊戈尔·斯特拉文斯基曾经说过：“好的作曲家借用别人的想法，伟大的音乐家则将其窃为己有。”这句话明显剽窃了T.S. 艾略特的名言，艾略特曾经写过：“不朽的诗人模仿；成熟的诗人剽窃。”

好主意来自许多地方，其他JavaScript程序库充满了好的代码和思路。如果其他开放源码程序库中有你可以使用的代码，或者可以转移到jQuery中的代码，你完全可以使用它们——但是你要尊重其他作者的版权和许可。

> **注意**
>
> 关于开放源码和免费软件的信息可以参加如下网站：
>
> - *http://www.opensource.org/*
> - *http://www.fsf.org/*

本例中的实现非常简单，缺少的只是一个“radio类”方法的思路，在这种情况下你甚至不需要实际的代码。尽管不是必需的，但是对思路的来源表示感谢还是一个很有礼貌的做法。

不管思路是来源于其他地方还是你自己的思考，在许多情况下，你都可以花费一行或者少数几行代码编写有用的jQuery插件。

什么是jQuery.fn？为什么jQuery插件要使用它？

jQuery.fn与jQuery.prototype引用同一个对象。当在jQuery.fn对象中添加一个函数时，实际上就把函数添加到jQuery.prototype中。

当用jQuery()或$()创建一个jQuery对象时，实际上调用的是new jQuery()。（jQuery代码自动为你添加new操作。）和其他JavaScript构造器一样，jQuery.prototype为每个new jQuery()调用返回的对象提供方法和默认属性。所以，当编写jQuery.fn插件时，实际上进行的是传统的面向对象JavaScript编程，用构造器的原型向对象添加一个方法。

那么为什么存在jQuery？而不像其他面向对象JavaScript代码那样使用jQuery.prototype？这并不只是为了少输入几个字符。

jQuery最初的版本（在1.0发行之前很长的时间）不使用JavaScript的prototype功能为jQuery对象提供方法。它通过循环读取对象将jQuery.fn（后称$.fn）中的每个属性和方法的引用复制到jQuery对象中。

因为对象中可能有数百个方法，每次调用$()都进行复制可能相当缓慢。所以，把这段代码改成使用JavaScript原型，消除了所有复制工作。为了避免破坏已经使用$.fn的插件，将它作为$.prototype的别名：

```
$.fn = $.prototype ;
```

这就是为什么jQuery.fn到今天还存在的原因——插件早在2006年初期就使用了$.fn!

5.6 编写自定义迭代器

5.6.1 问题

你在jQuery对象中选择了多个元素，需要循环读取这些元素（例如，一个接一个地显示这些元素），在每次循环之间有一个暂停：

```
<span class="reveal">Ready? </span>
<span class="reveal">On your mark! </span>
<span class="reveal">Get set! </span>
<span class="reveal">Go!</span>
```

你尝试了each()，但是它一次性地显示所有元素：

```
$('.reveal').each( function() {
    $(this).show();
});
//上面的代码不如下面这个更简单的版本：
$('.reveal').show();
```

5.6.2 解决方案

编写一个自定义迭代器，使用setTimeout()延迟回调：

```
// jQuery.each() 回调在一个数组(通常是一个jQuery对象,但是可以是任意数组)中循环,为每个元素调用回调函数,在每次回调之间插
jQuery.slowEach = function( array, interval, callback ) {
    if( ! array.length ) return;
    var i = 0;
    next();

    function next() {
        if( callback.call( array[i], i, array[i] ) !== false )
            if( ++i < array.length )
                setTimeout( next, interval );
    }
    return array;
};
//在"this"(jQuery对象)中循环并为每个元素调用一个回掉函数,在回调之间插入延时
//回调函数接收的参数与普通的jQuery(...).each()相同
jQuery.fn.slowEach = function( interval, callback ) {
    return jQuery.slowEach( this, interval, callback );
};
```

然后，简单地将.each()代码改为使用slowEach()并定义一个超时值就可以了：

```
//每隔半秒显示一个元素
$('.reveal').slowEach( 500, function() {
    $(this).show();
});
```

5.6.3 讨论

jQuery的.each()方法不是什么高技术。实际上，如果截取jQuery 1.3.2的实现在大部分典型使用（在jQuery对象中循环）中的实际代码，就会发现它只是一个相当简单的循环：

```
jQuery.each = function( object, callback ) {
    var value, i = 0, length = object.length;
    for(
        value = object[0];
        i < length && callback.call( value, i, value ) !== false;
        value = object[++i]
```

```
    ) {}
    return object;
};
```

each()函数还可以用更熟悉的方法编写：

```
jQuery.each = function( object, callback ) {
    for(
        var i = 0, length = object.length;
        i < length;
        ++i
    ) {
        var value = object[i];
        if( callback.call( value, i, value ) === false )
            break;
        }

        return object;
};
```

我们可以编写类似的函数，以其他实用的方式循环读取数组或者jQuery对象。比.slowEach()更简单的一个例子是以逆序循环读取jQuery对象：

```
//以相反的顺序循环读取数组或者jQuery对象
jQuery.reverseEach = function( object, callback ) {
    for( var value, i = object.length; --i >= 0; ) {
        var value = object[i];
        console.log( i, value );
        if( callback.call( value, i, value ) === false )
            break;
        }
};
//以相反的顺序循环读取"this"(一个jQuery对象)
jQuery.fn.reverseEach = function( callback ) {
    jQuery.reverseEach( this, callback );
    return this;
};
```

上面的代码并不试图处理.each()所处理的所有情况，只适用于典型jQuery代码的常规情况。

有趣的是，自定义迭代器可以完全不使用循环。.reverseEach()和标准的.each()都使用比较常规的循环，但是在.slowEach()中没有明确的JavaScript循环。为什么会这样？如何在不使用循环的情况下读取所有元素？

Web浏览器中的JavaScript没有在许多语言中都能找到的sleep()函数，所以不能这样暂停脚本的执行：

```
doSomething();
sleep( 1000 );
doSomethingLater();
```

作为替代，和JavaScript中的任何异步活动一样，setTimeout()函数有一个回调参数，每隔一段时间就调用回调函数。.slowEach()方法在setTimeout()回调函数中递增“循环”变量i，利用闭包在两次“循环”之间保留变量值。（闭包的讨论参见秘诀5.2。）

和.each()一样，.slowEach()直接在jQuery对象或者数组上操作，所以完成循环之前对数组的任何更改都会影响循环。和.each()不同的是，.slowEach()是异步的(对回调函数的调用发生在.slowEach()返回之后)，所以如果在.slowEach()返回之后、回调之前改变jQuery对象或其元素，这也会影响到循环。

5.7 切换属性

5.7.1 问题

你需要切换一组复选框中所有选择标志的手段。每个复选框的切换应该独立于其他复选框。

5.7.2 解决方案

编写一个.toggleCheck()插件，工作方式与jQuery核心中的.toggle()和.toggleClass()方法类似，可以用它设置、清除或者切换一个（或者一组）复选框：

```
//选中或者反选jQuery对象中选择的所有复选框元素,如果check参数省略,切换每个元素的选中状态

jQuery.fn.toggleCheck = function( check ) {
    return this.toggleAttr( 'checked', true, false, check );
};
```

然后可以这样启用一组按钮：

```
$('.toggleme').toggleCheck( true );
```

或者禁用它们：

```
$('.toggleme').toggleCheck( false );
```

也可以切换所有按钮的状态，每个都独立于其余按钮：

```
$('.toggleme').toggleCheck();
```

这个.toggleCheck()方法是在更通用的.toggleAttr()方法（可用于任何属性）的基础上构建的：

```
// 对于这个jQudery对象中选择的每个元素
//根据on的值将'name'属性设置为'onValue' 或 'offValue'
//如果'on'省略,独立地在'onValue' 和 'offValue'之间切换该属性
//如果选中的值('onValue' 或'offValue')为null或者undefined,删除该属性
jQuery.fn.toggleAttr = function( name, onValue, offValue, on ) {

    function set( $element, on ) {
        var value = on ? onValue : offValue;
        return value == null ?
            $element.removeAttr( name ) :
            $element.attr( name, value );
    }
    return on !== undefined ?
        set( this, on ) :
        this.each( function( i, element ) {
            var $element = $(element);
            set( $element, $element.attr(name) !== onValue );
        });
};
```

为什么要花这么多精力去构建这么通用的函数？现在我们可以几乎毫不费力地为其他属性编写相似的切换器. 假定你需要和.toggleCheck()相同的功能，但是现在你所要禁用和启用的是输入控件。可以用一行代码写出一个.toggleEnable()：

```
//启用或者禁用这个jQuery对象选择的所有输入元素
//如果省略enable参数,切换所有元素的启用状态
```

```
jQuery.fn.toggleEnable = function( enable ) {
    return this.toggleAttr( 'disabled', false, true, enable );
};
```

请注意是如何按照onValue和offValue参数切换true和false属性值的，这种用法更容易说明元素的“启用”，代替disabled属性提供的不太直观的“禁用”功能。

另举一个例子，假定需要切换一个foo属性，该属性的“开启”状态是字符串值bar，“关闭”状态是删除该属性。这仍然可以用一行代码完成：

```
//添加或者删除属性foo="bar"
//如果省略add参数,切换该属性的存在

jQuery.fn.toggleFoo = function( add ) {
    return this.toggleAttr( 'foo', 'bar', null, add );
};
```

5.7.3 讨论

知道feeping creaturism（也就是creeping featurism，可怕的特征主义）总是一件好事。如果所需要的就只是切换复选框状态，可以这样编码：

```
jQuery.fn.toggleCheck = function( on ) {
    return on !== undefined ?
        this.attr( 'checked', on ) :
        this.each( function( i, element ) {
            var $element = $(element);
            $element.attr( 'checked', ! $element.attr('checked') );
        });
};
```

这比.toggleAttr()方法简单一些，但是只能用于checked属性，不能用于其他。如果以后需要.toggleEnable()方法怎么办？复制整个方法改变几个名称？

在.toggleAttr()中所作的额外工作给我们带来了许多灵活性：现在我们能够编写一整组属性切换器，每个都只需要一行代码。

注意

在编写这类新的工具方法之前，检查你正在使用的jQuery版本文档，在jQuery的未来版本中有可能添加类似的方法，你也许不再需要花费力气编写自己的方法。

5.8 寻找瓶颈

5.8.1 问题

你的网站加载太慢，或者对单击和其他用户交互的反应太慢，而你不知道原因。哪一部分代码花费了过多的时间？

5.8.2 解决方案

使用某种分析器（profiler），可以在许多可用工具中选择一个，也可以自己编写一个简单的分析器。

5.8.3 讨论

分析器是寻找代码中最耗时的部分的一种手段。你可能已经有了至少一种好的JavaScript分析器。Firebug有一个分析器，IE 8和Safari 4也自带了这类工具。这些分析器都是全功能的：可以启动分析，与页面交互，停止分析，然后获得一个报告，说明每个函数中花费的时间。这可能已经足以告诉你所需要加速的代码。

也可以在网上搜索“jquery profiler”，寻找专用于jQuery的分析器。这些分析器可以剖析选择器性能，更深入地了解jQuery函数性能。

为了真正细致地分析，需要分析比函数更小的代码独立段，这时可以用几行代码编写一个简单的分析器。你可能已经用这种临时性的风格编写过代码：

```
var t1 = +new Date;
// ... do stuff ...
var t2 = +new Date;
alert( ( t2 - t1 ) + ' milliseconds' );
```

注意

代码中的+new Date只是较为熟悉的new Date().getTime():的简单方式，返回以毫秒表示的当前时间。

为什么这个语句可以正常工作？newDate部分是一样的；它返回一个代表当前时间的Date对象（因为没有参数，所以()是可选的）。+运算符将对象转换为一个数值。JavaScript将对象转换为一个数值的方法是调用对象的.valueOf()方法。Date对象的.valueOf()方法正好和.getTime()方法相同，给出以毫秒表示的时间。

可以仅用15行代码就建立更通用和更易用的函数：

```
(function() {

    var log = [], first, last;

    time = function( message, since ) {
        var now = +new Date;
        var seconds = ( now - ( since || last ) ) / 1000;
        log.push( seconds.toFixed(3) + ': ' + message + '<br />' );
        return last = +new Date;
    };

    time.done = function( selector ) {
        time( 'total', first );
        $(selector).html( log.join('') );
    };

    first = last = +new Date;
})();
```

现在有一个time()函数，可以再想要记录自最后一个time()调用后经过的时间（或者，可以选择记录从指定的某个时间后经过的时间）。当我们做好准备报告结果时，调用time.done()。下面是一个例子：

```
// do stuff
time( 'first' );
// do more stuff
time( 'second' );
// and more
time( 'third' );
time.done( '#log' );
```

上述JavaScript代码需要在页面上添加如下HTML代码：

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

代码运行后， 中将填写如下列表：

```
0.102 first
1.044 second
0.089 third
1.235 total
```

我们可以看到time('first')和time('second')调用之间花费的时间最长。

警告

小心Firebug!如果在计时的页面上启用Firebug，它可能对结果有很大的影响。JavaScript的eval()函数被jQuery 1.3.2及更早的版本用来解析下载的JSON数据，它所受到的影响达到了极端的程度：按照秘诀5.11中的格式，10 000个姓名/地址的数组在Firefox中正常花费2.2秒，而启用Firebug的脚本（Script）窗格之后花费了55秒。后来的jQuery使用Function()，不会受到Firebug的影响。

如果Firebug严重影响你的页面，而你无法找到解决方法，可以检测Firebug并显示一个警告：

```
<div id="firebugWarning" style="display:none;">
    Your warning here
</div>
$(document).ready(function() {
    if(window.console && console.firebug)
        $('#firebugWarning').show();
});
```

对于许多优化应用，这样的代码可能足够了。但是如果需要测试的代码在循环中会怎么样呢？

```
for( var i = 0; i < 10; ++i ) {
    // do stuff
    time( 'first' );
    // do more stuff
    time( 'second' );
    // and more
    time( 'third' );
}
time.done( '#log' );
```

现在，小分析器将显示第一个、第二个和第三个项各10次！这不难修复——当多次调用它时，只需要累加每个具体消息标签花费的时间就可以了：

```
(function() {

    var log = [], index = {}, first, last;

    //累计指定消息所花费的时间
    //每个消息字符串都有自己的总计秒数
    function add( message, seconds ) {
        var i = index[message];
        if( i == null ) {
            i = log.length;
            index[message] = i;
```

```
            log[i] = { message:message, seconds:0 };
        }
        log[i].seconds += seconds;
    }

    time = function( message, since ) {
        var now = +new Date;
        add( message, ( now - ( since || last ) ) / 1000 );
        return last = +new Date;
    }

    time.done = function( sel ) {
        time( 'total', first );
        $(sel).html(
            $.map( log, function( item ) {
                return(
                    item.seconds.toFixed(3) +
                    ': ' +
                    item.message + '<br />'
                );
            }).join('')
        );
    };
    first = last = +new Date;
})();
```

经过这样的修改，从循环中可以得到有用的结果：

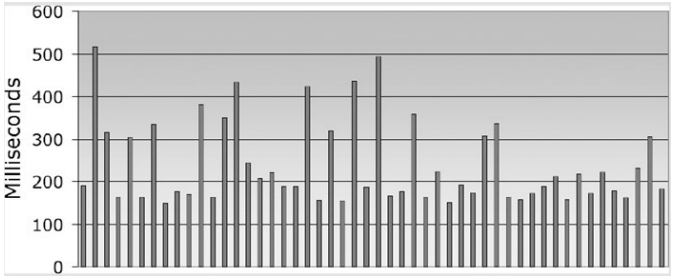
```
0.973 first
9.719 second
0.804 third
11.496 total
```

当计时测试的结果变化时

当在网页上运行计时测试的时候，不会每次都得到相同的结果。实际上，如果重新加载页面或者多次重新运行测试，计时的结果可能有很大不同。

你应该怎么做才能获得“真实”的数字？取结果的平均值吗？

可能不行。下面是秘诀5.11中的fillTable()连续运行50次的统计图表，每次运行之间间隔大约10秒：



这里有一个特别的模式：大部分运行时间介于150～200毫秒之间，一小部分零散的运行花费的时间更长一些。175毫秒左右似乎是真实的运行时间，而花费时间很多的运行可能是受到机器上的其他进程影响。

有些花费时间较长的运行结果可能是由于浏览器垃圾收集机制的影响。因为很难区分其他进程花费的时间，所以最实用的处理方法可能是不考虑这些异常值。

5.9 缓存jQuery对象

5.9.1 问题

你打算记录发生mousemove事件时event对象的各种属性，因为使用$('.classname')选择器寻找事件数据，并用它更新表格单元格，所以代码的执行滞后了。

页面包含如下用于记录的HTML代码：

```
<table id="log">
    <tr><td>Client X:</td><td class="clientX"></td></tr>
    <tr><td>Client Y:</td><td class="clientY"></td></tr>
    <tr><td>Page X:</td><td class="pageX"></td></tr>
    <tr><td>Page Y:</td><td class="pageY"></td></tr>
    <tr><td>Screen X:</td><td class="screenX"></td></tr>
    <tr><td>Screen Y:</td><td class="screenY"></td></tr>
</table>
```

还有如下的JavaScript代码：

```
$('html').mousemove( function( event ) {
    $('.clientX').html( event.clientX );
    $('.clientY').html( event.clientY );
    $('.pageX').html( event.pageX );
    $('.pageY').html( event.pageY );
    $('.screenX').html( event.screenX );
    $('.screenY').html( event.screenY );
});
```

该页面还包含大量（数千个！）其他DOM元素。在简单的测试页面中，上述代码运行得很好，但是它在这个复杂的页面中运行得太慢了。

5.9.2 解决方案

缓存$(...)调用返回的jQuery对象，这样DOM查询只需要运行一次：

```
var
    $clientX = $('.clientX'),
    $clientY = $('.clientY'),
    $pageX = $('.pageX'),
    $pageY = $('.pageY'),
    $screenX = $('.screenX'),
    $screenY = $('.screenY');
$('html').mousemove( function( event ) {
    $clientX.html( event.clientX );
    $clientY.html( event.clientY );
    $pageX.html( event.pageX );
    $pageY.html( event.pageY );
    $screenX.html( event.screenX );
    $screenY.html( event.screenY );
});
```

还可以显著地加速这些选择器；下一个秘诀将会介绍加速的方法。但是只调用一次，而不是重复进行可能就足以改进这种情况。

5.9.3 讨论

优化代码的经典方法之一是将重复的计算“提升”到循环之外，这样就只需要计算一次。在循环中未作改变的任何值都应该只在循环开始之前计算一次。如果这些计算的代价很高，循环就会因此而大大加速。

这种方法在“循环”是一系列频繁触发的事件（如mousemove）而且“计算”是一个jQuery选择器的时候也很合适。将选择器提升到事件处理程序之外，将使事件处理程序的响应更快。

当然，如果在一个循环中调用多个选择器，也同样能够从将它们移出循环中受益。

注意

为什么$clientX和其他变量名以$字符开头?

$在JavaScript中没有特别的含义——它被当作字母表中的一个字符。在jQuery代码中使用$前缀只是一个约定，提醒人们该变量包含对jQuery对象的引用，而不是DOM元素等，因为变量名$foobar从视觉上很类似jQuery操作$('#foobar')。

当需要同时使用jQuery对象和底层的DOM元素时，这种约定特别有用，例如：

```
var $foo = $('#foo'),foo = $foo[0];
//现在可以使用jQuery对象:
$foo.show();
//或者DOM元素:
var id = foo.id;
```

5.10 编写更快的选择器

5.10.1 问题

你的代码包含了大量$('.classname')选择器。你打算像前一个秘诀所描述的那样缓存它们，但是这些选择器仍然影响你的页面加载时间。你需要使它们更快一些。

5.10.2 解决方案

首先，确定你使用的是最新版本的jQuery（1.3.2或者更高），新版本能在浏览器中得到更高的选择器性能，尤其是对类选择器来说。

如果能够控制HTML页面内容，就可以修改页面，使用id属性和'#xyz'选择器代替class属性和".xyz"选择器：

```
<div class="foo"></div>
<div id="bar"></div>

$('.foo') // 更慢
$('#bar') // 更快
```

如果必须使用类名选择器，先看看能不能用更快的ID选择器找到双亲元素，然后向下寻找子元素。例如，使用前一个秘诀中的HTML：

```
<table id="log">
    <tr><td>Client X:</td><td class="clientX"></td></tr>
    ...
</table>
```

可以使用如下代码：

```
$('.clientX')          // 较慢
$('td.clientX')        // 可能快一些
$('#log .clientX')     // 可能快得多
$('#log td.clientX')   // 在某些浏览器中可能较快
```

注意

一定要注意无法反映你所使用的真实页面内容的选择器速度测试页面。在非常简单的页面中，简单的$('.clientX')的测试结果可能远远超过花哨的选择器（如$('#log td.clientX')），甚至在你认为类选择器应该运行得较慢的浏览器和jQuery版本中也是如此。

这只是因为更复杂的选择器的建立时间更长，在简单页面中的建立时间可能对性能起主要的作用。这个秘诀所用的测试页面有意地包含了非常多的元素，以便引发在大页面中才会出现的选择器性能问题。

当然，不管怎么测试都无法准确地说明在你的页面中选择器的性能。确定哪一个选择器在特定页面中最快的唯一方法是在页面中测试所有选择器。

5.10.3 讨论

看似没有问题的$('.clientX')等调用可能花费大量的时间，这一点很容易忽视。根据浏览器和jQuery的版本，这种选择器可能必须建立页面中每个DOM元素的列表，并且循环寻找指定的类。

1.3之前的jQuery版本在每种浏览器上都使用这种慢速方法。JQuery 1.3引入了Sizzle选择器引擎，该引擎利用较新的浏览器中更快的DOM API，如getElementsBy ClassName()和querySelectorAll()。

但是，对于大部分网站，你可能需要在一段时间内支持IE 7，而在IE 7中如果你的页面复杂，类选择器就运行得很慢。

如果你能够使用它，那么像$('#myid')那样按照ID选择在所有浏览器中通常都非常快，因为它只对getElementById() API进行一次调用。

减少需要搜索的元素数量也是有帮助的，要做到这一点，可以指定双亲元素、用标记名称建立更具体的类选择器，或者组合各种技巧。

5.11 更快地加载表格

5.11.1 问题

你打算加载一个具有1000个姓名和地址的JSON数据对象，并用jQuery以这些数据创建一个表格。在IE 7中创建这个表格花费5～10秒，这还不包括下载时间。

JSON数据格式如下：

```
{
    "names": [
        {
            "first": "Azzie",
            "last": "Zalenski",
            "street": "9554 Niemann Crest",
            "city": "Quinteros Divide",
            "state": "VA",
            "zip": "48786"
        },
        //重复1000个姓名
    ]
}
```

JavaScript代码如下：

```
//返回净化过的文本版本,对& < >等文本进行转义
function esc( text ) {
    return text
        .replace( '&', '&amp; ' )
        .replace( '<', '&lt; ' )
        .replace( '>', '&gt; ' );
}
$(document).ready( function() {

    function fillTable( names ) {
        $.each( names, function() {
            $('<tr>')
                .append( $('<td>').addClass('name').html(
                    esc(this.first) + ' ' + esc(this.last)
                ) )
                .append( $('<td>').addClass('address').html(
                    esc(this.street) + '<br />' +
                    esc(this.city) + ', ' +
                    esc(this.state) + ' ' + esc(this.zip)
                ) )
                .appendTo('#nameTable');
        });
    }
    $.getJSON( 'names/names-1000.json', function( json ) {
        fillTable( json.names );
    });
});
```

在文档中还有下列HTML代码：

```
<table id="nameTable">
</table>
```

这些代码工作得很好，浏览器的显示如图5-1所示。

Arica Hence	5473 Brallier Crossing Bejar Centers, MA 37967
Vicente Hofmann	4070 Cree Pike Gosier Cove, LA 67602
Renna Pastorius	4666 Moorer Tunnel Marmo Centers, PO 53702
Exie Duca	7139 Langenfeld Court Albrecht, RE 24425
Sean Pickerel	9956 Urquidez Rest Iveson Islands, EN 05425
Justa Ocasio	3463 Lair Terrace Stanislawski, SE 80277
Sommer Lafortune	9101 Palmateer Road

图5-1　姓名表格的浏览器输出

但是它运行得太慢了。

5.11.2　解决方案

组合多种优化方案：

- 插入一个<table>或者<tbody>代替多个<tr>元素。
- 使用.innerHTML或.html()代替DOM操纵。
- 用a[++i]和.join()构建数组，代替字符串连接。
- 使用基本的for循环代替$.each。
- 减少名称查找。

优化的结果是如下的新版本代码（使用和前面一样的esc()函数）：

```
$(document).ready( function() {
    function fillTable( names ) {
        //用局部函数名称减少名称查找
        var e = esc;
        //
        var html = [], h = -1;
        html[++h] = '<table id="nameTable">';
        html[++h] = '<tbody>';
        for( var name, i = -1; name = names[++i]; ) {
            html[++h] = '<tr><td class="name">';
            html[++h] = e(name.first);
            html[++h] = ' ';
            html[++h] = e(name.last);
            html[++h] = '</td><td class="address">';
            html[++h] = e(name.street);
            html[++h] = '<br />';
            html[++h] = e(name.city);
            html[++h] = ', ';
            html[++h] = e(name.state);
            html[++h] = ' ';
            html[++h] = e(name.zip);
            html[++h] = '</td></tr>';
        }
        html[++h] = '</tbody>';
        html[++h] = '</table>';

        $('#container')[0].innerHTML = html.join('');
    }
    $.getJSON( 'names/names-1000.json', function( json ) {
        fillTable( json.names );
    });
});
```

新代码要求文档中的HTML代码改为：

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

在IE 7中的一个测试系统上，新代码仅仅运行0.2秒，而原来的代码运行了7秒。快了34倍！

当然，新代码没有原来的代码那么清晰和简练，但是你的网站访问者永远也不知道或者在意这一回事。他们注意的将是页面加载的速度。

5.11.3 讨论

有时候你很幸运，能够发现一个明确地解决性能问题的优化方案。而有些时候，你需要像本秘诀里一样，使用多种技巧来得到想要的速度。

这段代码中对速度提升产生最大作用的是在一个DOM操作中插入一个<table>元素及其所有子元素，而不是一个接一个地插入冗长的<tr>元素。为了做到这一点，你需要生成整个表格的HTML代码。这意味着你需要粘贴大量用于构建HTML的字符串，这种操作可能非常快也可能非常慢，具体取决于你的做法。因为需要循环读取1000个项，所以找到最快的循环编写方法是值得一试的。

你可能感到奇怪，“这还是jQuery代码吗？这看上去就像简单旧式的JavaScript!”答案是肯定的。混合搭配jQuery和其他JavaScript代码完全没有问题。你可以在网站的大部分编码中使用简单的jQuery方法，在发现慢速的部分时，可以找到更快速的jQuery技巧或者在必要的时候用简单旧式的JavaScript来改进性能。

5.12 编写基本的循环代码

5.12.1 问题

你正在代码中调用$.each(array, fn)或者$(selector).each(fn)循环读取数千个项，你怀疑这些函数调用可能增加加载的时间：

```
$.each( array, function() {
    // 对this进行处理
});
```

或者：

```
$('.lotsOfElements').each( function() {
    // 对this或$(this)进行处理
});
```

5.12.2 解决方案

使用for循环代替.each()。在循环读取数组的方法中，下面的循环是难以战胜的：

```
for( var item, i = -1; item = array[++i] ) {
    // 处理item
}
```

但是这里有个例外：这种循环只能在数组没有“假”元素时才能正常工作，也就是说，数组里不能有值为undefined、null、false、0或者“”的元素。尽管有这样的限制，这种循环在许多常见情况下还是有用的，例如，在jQuery对象中的循环。只是，一定要将对象缓存在变量中：

```
var $items = $('.lotsOfElements');
for( var item, i = -1; item = $item[++i] ) {
    // 处理item (DOM节点)
}
```

像秘诀5.11那样，将对象数组包含在JSON数据中也是常见的：

```
{
    "names": [
        {
            // ...
            "zip": "48786"
        },
        //重复1000个名字
    ]
}
```

如果你知道组成names数组元素的对象都不为null，使用这种快速循环就很安全了。

为了编写一个更通用的、可以用于任何数组的循环，可以使用你在许多地方都能看到的经典循环：

```
for( var i = 0; i < array.length; i++ ) {
    var item = array[i];
    //处理item
}
```

还可以在三个方面改进这种循环：

- 缓存数组长度。

- 使用++i，在某些浏览器中它快于i++。
- 合并循环变量的测试和递增，减少一次名称查找。

结果如下：

```
for( var i = -1, n = array.length; ++i < n; ) {
    var item = array[i];
    // 处理item
}
```

注意

使用while或者do...while循环会更快吗？可能不会。可以将前一个循环改写为：

```
var i = -1,n = array.length;
while(++i < n){
    var item = array[i];
    // 处理item
}
```

或:

```
var i = 0,n = array.length;
if(i < n)do {
    var item = array[i];
    // 处理item
}
while(++i < n);
```

但是两种写法都不比更易读的for循环快。

可以使用for..in循环读取对象（不是数组）：

```
for( var key in object ) {
    var item = object[key];
    // 处理item
}
```

有关for..in循环的警告

决不要将for..in循环用于jQuery对象或者任何类型数组的循环读取中。如果数组有任何自定义属性或者方法，它们将和数字型数组元素一起循环读取。例如，下面的代码枚举一个DOM元素——文档主体（i=0）：

```
$('body').each(function(i){ console.log(i);});
```

下面的代码似乎也做同样的工作，但是除了[0]元素以外，它还会列举所有jQuery方法，如show和css：

```
for(var i in $('body'))console.log(i);// BAD
```

这时应该用前面列举的数组循环来完成。

如果你的页面上有任何代码修改了Object.prototype，用附加的方法或者属性扩展所有对象，那么即使是for..in循环的“安全”用法也可能遇到问题。除了你想要的内容之外，循环还会列举那些方法或者属性。

我们非常反对扩展Object.prototype，因为它会破坏很多代码。实际上，至少在jQuery 1.3.2中，它就因为导致each()列举添加的方法或者属性而破坏了jQuery本身。如果你的代码必须工作于这种环境，就必须对所有的循环多加注意，如测试每个对象属性的hasOwnProperty()方法。遗憾的是，因为这些额外的测试会降低代码运行速度，所以你必须在速度和健壮性之间做出选择。

5.12.3 讨论

`$(selector).each(fn)`是创建一个jQuery对象并循环读取它的常用方法，但是它并不是唯一的方法。jQuery对象是一个类似数组的对象，具有`.length`和`[0]`、`[1]`、`...`、`[length-1]`属性。因此，可以使用适合于任何其他数组的循环技术。而且，因为jQuery对象不会包含“假”元素，可以使用本解决方案一开始列举的最快速的`for`循环。

如果你使用秘诀5.2中的`time()`函数或者其他分析器来测量循环性能，一定要测试实际的代码，而不是仅仅运行循环而非整个循环体的简化测试用例。简化的测试会忽略`for`循环的一个潜在优势：因为较少的函数嵌套，造成的名称查找也较少。详情参见秘诀5.13。

5.13 减少名称查找

5.13.1 问题

你的代码有一个内部循环，位于好几级的嵌套函数内，运行几百甚至几千次。这个内循环调用多个全局函数，并引用一些定义在外部函数中的变量或者全局变量。

因为函数的嵌套，每次引用都会触发多个名称查找。这会降低代码的速度，但是分析器不能显示问题所在，从代码上也无法明显地看出问题！

5.13.2 解决方案

审视出现在最内部循环中出现的每个名称，计算需要的名称查找次数。通过局部缓存对象引用或者减少嵌套函数的使用减少名称查找的次数。

5.13.3 讨论

闭包是个极好的东西。它们能轻而易举地实现状态信息的捕捉，并将这些信息传递给事件处理程序或者定时器回调等异步函数。如果JavaScript没有闭包，所有异步回调都需要有传递状态的手段。而现在你只需使用嵌套函数。

JavaScript的动态特性也是很好的。可以在任何时候为任何对象添加属性和方法，JavaScript运行时在必要的时候能够轻松地找到这些引用。

综合上述技术，你可能需要许多名称查找。

> **注意**
>
> 最现代化的JavaScript解释程序在这一领域已经有了很大的改进。但是如果你想在最流行的浏览器（例如任何版本的IE）中快速运行代码，仍然需要注意名称查找的次数。

考虑如下的代码：

```
//获得局部作用域的典型函数包装器
(function() {
     //查找一组数值中最大的绝对值
     function maxMagnitude( array ) {
         var largest = -Infinity;
         $.each( array, function() {
              largest = Math.max( largest, Math.abs(this) );
         });
     return largest;
    }
     //这里放置在大数组上调用maxMagnitude的其他代码
})();
```

记住，JavaScript首先在局部作用域（函数）中查找，如果没有找到名称，就查找上一级嵌套函数，直至全局作用域。JavaScript不仅在每次使用名称时需要查找这些名称，而且在名称实际上定义于上级函数或者全局作用域中时也必须重复这些查找。

所以，如果这个代码块在全局作用域中，each()回调在每次循环中都要进行如下的名称查找：

```
1. 在局部作用域中查找largest[失败]
2. 在MaxMagnitude()中查找largest[成功]
3. 在局部作用域中查找Math[失败]
4. 在MaxMagnitude()中查找Math[失败]
5. 在匿名包装器函数中查找Math[失败]
6. 在全局作用域中查找Math[成功]
```

```
7. 在Math对象中查找.abs[成功]
8. 在局部作用域中查找Math[失败]
9. 在MaxMagnitude()中查找Math[失败]
10. 在匿名包装器函数中查找Math[失败]
11. 在全局作用域中查找Math[成功]
12. 在Math对象中查找max[成功]
13. 在局部作用域中查找largest[失败]
14. 在MaxMagnitude()中查找largest[成功]
```

现在将代码重写为：

```
//获得局部作用域的典型函数包装器
(function() {
    //查找一组数值中最大的绝对值
    function maxMagnitude( array ) {
        var abs = Math.abs, max = Math.max;
        var largest = -Infinity;
        for( var i = -1, n = array.length; ++i < n; ) {
            largest = max( largest, abs(array[i]) );
        }
        return largest;
    }
    //这里放置在大数组上调用maxMagnitude的其他代码
})();
```

新的代码不仅消除了每次循环中对回调函数的调用，而且将每次循环中的名称查找减少了至少10次。在这一版本的循环体中进行如下的名称查找：

```
1. 在局部作用域中查找largest[成功]
2. 在局部作用域中查找abs[成功]
3. 在局部作用域中查找max[成功]
4. 在局部作用域中查找largest[成功]
```

这比第一个版本的性能增强了70%以上。

如果这段代码更深地嵌套到另一个函数内，差异可能更大，因为每个嵌套函数都会增加一次对Math对象的查找。

注意

在上面的讨论中，省略了this和array[i]的查找，以及for循环本身的查找。这些查找在两个版本中基本相同。

在秘诀5.11中，单个名称查找的优化就能带来100毫秒的改进。这虽然不是巨大的差异，但是对于一行代码来说，能够减少1/10秒的页面加载时间，就已经很有价值了。

原来的代码在每次循环中调用esc()6次，在1000个名字的测试用例中一共要调用6000次。这些调用位于3个嵌套函数内部，而esc()是一个全局函数，所以仅为了解析函数名称，每次调用就要进行4次名称查找，一共就是24 000次！

改进的代码将函数嵌套减少了一级，从而将名称查找减少到18 000次（两级嵌套函数和全局作用域，每一级6000次），然后还在最内层的函数中使用了最后一种技巧：

```
function fillTable( names ) {
    var e = esc;
    //现在在内部循环调用e()代替esc()
}
```

现在，对e()的6000次调用，每次只需要进行一次名称查找，又减少了12 000次名称查找。难怪它减少了1/10秒的加载时间。

5.14 用.innerHTML更快地更新DOM

5.14.1 问题

你打算创建一个大的HTML代码块，并使用$('#mydiv').html(myhtml)；将其插入DOM。对代码的分析发现.html()方法花费的时间比预期的更长。

5.14.2 解决方案

使用$('#mydiv')[0].innerHTML = myhtml；加速DOM更新——条件是不需要.html()提供的特殊处理。

5.14.3 讨论

.html()方法使用.innerHTML属性将HTML内容插入DOM，但是该方法需要一些预处理。在大部分情况下，这不成问题，但是在性能至关重要的代码中，可以直接设置.innerHTML属性以节约时间。

实际进行这一处理的是jQuery内部方法.clean()。如果你阅读.clean()的源代码，就会发现它对HTML输入进行了一些清理工作。

> **注意**
>
> 找到jQuery源代码中大部分方法的最简方式是搜索方法名称后面加上一个：，例如，为了寻找.clean()方法，可以在无压缩的jQuery源代码中搜索clean:。

秘诀5.11中的代码与这种清理工作冲突。该秘诀的HTML代码包含大量的
标记。.clean()中有一个正则表达式，用来查找所有自结束标记（以/>结束，因而不需要结束标记），并检查这些标记是否确实是可以自结束的HTML标记集中的一员。如果不是，则将HTML转换为一个开始-结束标记。

例如，如果编写的代码是$('#test').html(' ');，这个无效的HTML会自动转换为$('#test').html(' ');，这种转换使编码更容易，但是如果HTML字符串很长，包含许多自结束标记，.clean()就必须全部检查——即使所有的标记都是有效的（如其他秘诀中的
标记）也不例外。

.html()方法替换所有现有的内容，并且删除所有被替换元素中通过jQuery附加的事件处理程序，以避免内存泄漏。如果被替换内容中有事件处理程序，你应该坚持使用.html()，如果只需要清理事件处理程序，不需要其他HTML清理，可以使用$('#test').empty() [0].innerHTML = myhtml;。

基本的标准是：如果你确定代码中不需要jQuery提供的事件清理或者HTML清理，那么你可以小心地直接使用.innerHTML。否则为了安全起见，坚持使用.html()。

5.15 分解方法链

5.15.1 问题

jQuery方法链中某处出现问题。HTML代码如下：

```
<div class="foo">
    before
        <span class="bar" style="display:none;">
            test
        </span>
    after
</div>
```

JavaScript代码（按钮单击事件处理程序的一部分）如下：

```
$('.foo').css({ fontsize: '18px' }).find('.bar').show();
```

当运行上述代码时，字体大小没有设置，隐藏的元素也没有显示出来。你有Firebug或者其他的JavaScript调试器，但是很难跟踪这段代码。你能说出这个链中有问题的地方吗？

5.15.2 解决方案

将链分解为单独的语句，在变量中存储各个jQuery对象：

```
// $('.foo').css({ fontsize: '18px' }).find('.bar').show();
var $foo = $('.foo');
$foo.css({ fontsize: '18px' });
var $bar = $foo.find('.bar');
$bar.show();
```

现在对于调试你有多种选择。其中之一是使用调试器的单步运行命令逐条运行这些语句，在每一步之后观察变量和页面的状态。

在这段代码中，应该在`$foo`和`$bar`赋值之后检查二者。每个变量的`.length`属性值是多少？这能够告诉你选中的DOM元素数量。每个对象是否包含预期的DOM元素？检查`[0]`、`[1]`、`[2]`等属性，查看这些DOM元素。

假如`$foo`包含正确的DOM元素，那么在调用`.css()`方法之后发生了什么呢？使用Firebug的CSS Inspector，你会发现CSS的`font-size`属性在方法调用之后没有变化。等一下！是`font-size`而不是`fontsize`?这就是问题所在。查阅文档，你发现编写这条语句的正确方式如下：

```
$foo.css({ fontSize: '18px' });

$foo.css({ 'font-size': '18px' });
```

这是一个问题，但是另一个问题是什么呢？在`$bar`赋值之后，如果查看`.length`属性，就会发现它的值为0。这告诉我们，选择元素的操作没有成功。查看HTML和JavaScript代码就能发现，类名拼写错误。

现在在原始的方法链上进行这两处修改：

```
$('.foo').css({ fontSize: '18px' }).find('.bar').show();
```

另一种方法是使用Firebug的日志（logging）语句：

```
// $('.foo').css({ fontsize: '18px' }).find('.bar').show();
var $foo = $('.foo');
console.log( $foo );
$foo.css({ fontsize: '18px' });
console.log( $foo.css('fontsize') );
var $bar = $foo.find('.bar');
console.log( $bar );
$bar.show();
```

这些console.log()调用将揭示$bar没有选中任何元素，但在试图记录字体大小时落入了陷阱：在console.log()调用中也拼错了fontSize!

这种情况下组合多种调试技术很有帮助：记录这些变量，使用Firebug的检查器，重新阅读源代码，并让其他人帮你查找问题。

5.15.3 讨论

jQuery的链接帮助你更轻松地写出简洁的代码，但是它可能妨碍调试，因为难以逐步运行方法链的每个单独步骤，观察它们的结果。将链分解为单独的语句，即使是在调试时临时为之，也能使调试任务更轻松一些。

5.16 这是jQuery的缺陷吗

5.16.1 问题

你打算调用一些jQuery代码显示隐藏的一个元素，并用setTimeout()在一段延时之后设置其HTML内容：

```
function delayLog( text ) {
    setTimeout( "$('#log').show().html(text)", 1000 );
}
//代码中的另一个地方...
delayLog( 'Hello' );
```

.show()调用成功，但是.html(text)调用失败了。Firebug控制台报告text变量未定义。在setTimeout()之外调用相同的jQuery代码也可以成功。在jQuery中使用setTimeout()会出现问题吗？

5.16.2 解决方案

确定jQuery是不是问题来源的方法之一是用不包含jQuery的其他JavaScript代码代替你的jQuery代码。在这个例子中，可以用简单的alert()替代jQuery代码：

```
function delayLog( text ) {
    setTimeout( "alert(text)", 1000 );
}
```

当尝试这个版本的代码时，发生了同样的问题：没有显示警告信息，Firebug同样报告text未定义。

这并没有确定问题所在，但是已经大大缩小了范围。很明显，这不是jQuery的问题（除非jQuery程序库的存在妨碍了你的页面，但是你可以在一个不包含jQuery简单的测试页面中运行上述代码，排除这种可能性）。所以，问题肯定出现在代码自身，最有可能与使用serTimeout()的方式有关。

确实，这里的问题是当setTimeout()接受一个字符串参数时，它执行于全局作用域，也就是说，它位于任何函数之外。最简单的修复手段就是使用一个局部函数作为回调，代替文本字符串：

```
function delayLog( text ) {
    setTimeout( function() {
        alert(text);
    }, 1000 );
}
```

和以字符串方式调用setTimeout()的代码不同，嵌套的函数完全能够访问外部函数的变量和参数。所以，这段代码能够按照预期显示文本警告。

最后，原来的jQuery代码修改如下：

```
function delayLog( text ) {
    setTimeout( function() {
        $('#log').show().html(text);
    }, 1000 );
}
```

5.16.3 讨论

调试的时候，如果你不确定问题的根源，那么找出不可能发生问题的地方也能够帮助你跟踪问题所在。本秘诀的目的不是帮助你排除setTimeout()的问题（毕竟，这是一本jQuery书籍，而不是通用的

JavaScript书籍），而是为了帮助你快速地排除（或者确认）jQuery是问题来源的可能性，从而调整调试工作的重点。

5.17 跟踪jQuery

5.17.1 问题1

你使用Firebug或者其他JavaScript调试器的单步进入（Step Into）功能，试图单步执行jQuery代码，查看调用jQuery时的实际操作。但是当单步进入jQuery代码时，所有的代码混合成一个冗长而难以理解的代码行，无法单步运行它：

```
(function(){var l=this,g,y=l.jQuery,p=l.$,o=l.jQuery=l.$=function(E,F)...
```

5.17.2 解决方案1

你正在使用的是精简版的jQuery，为了测试，可以在页面中加载未压缩版本的jQuery版本。

如果用<script>标记从Google Ajax程序库API通过<script>标记加载代码，将它改为：

```
<!—注释精简版jQuery -->
<!--
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js
"></script>
-->
<!—使用未压缩版本进行测试 -->
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js"></script>
```

如果使用的是Google的JavaScript加载程序（loader），将它改为：

```
// 注释精简版的jQuery
// google.load( 'jquery', '1.3.2' );
// 用未压缩版本进行测试
google.load( 'jquery', '1.3.2', { uncompressed:true });
```

现在就可以单步进入jQuery代码进行跟踪了。

5.17.3 问题2

修复上一个问题之后，你希望了解jQuery的.html()和.show()方法的工作原理。所以，你打算在调试器中跟踪如下代码：

```
$('#test').html( 'test' ).show();
```

但是当你使用单步进入命令时，调试器进入jQuery的构造器而不是你所感兴趣的两个方法之一。

5.17.4 解决方案2

前面的代码行包含3个函数调用：jQuery($)构造器调用，然后是对.html()and.show()方法的调用。单步进入命令进入第一个调用（也就是构造器）内部。

这时可以立刻使用单步退出（Step Out）命令，然后再进行单步进入。这能够让你离开jQuery构造器（从而回到原始代码行的中间），然后进入.html()方法。

要进入.show()方法，可以再配对使用单步退出和单步进入命令。每次这么做，就可以进入jQuery链的下一环。

如果你觉得这样做太乏味，可以像秘诀5.15描述的那样分解方法链，并在希望停止运行的地方添加debugger；语句。如果你希望跟踪.show()方法，可以对代码作如下修改：

```
var $test = $('#test');
$test.html( 'test' );
debugger;
$test.show();
```

现在，当代码在debugger；语句处暂停时，可以使用单步进入（两次，第一次进入$test.show()；语句，然后单步进入函数调用）。

注意

可以使用单步跳出（Step Over）从debugger；语句单步执行到下一行，毕竟你现在还不打算“进入”任何函数内，但是单击单步进入（或者在Windows中按F11键）两次更简单，效果也一样好。还可以在$test.show()行上设置断点代替debugger；语句，然后用一个单步进入命令可以进入.show()方法的代码。

5.17.5 讨论

jQuery的精简版本适合于在生产环境中使用，但是对于开发并不理想。它将所有代码组合为一两行，在调试器中几乎无法单步执行。而且，常用的方法链使得单步进入jQuery方法更加困难。使用本秘诀提供的技巧，可以轻松地在调试器中跟踪jQuery代码，寻找缺陷或者了解代码的工作原理。

注意

不要被从事测试驱动开发的朋友的言论所迷惑而不使用调试器！即使通过单元测试和其他方法发现了大部分的缺陷，在调试器中单步执行并研究变量和属性的变化仍然是了解一个代码块的最佳方法之一。

毕竟，在你阅读代码的时候，必须在头脑里单步执行，形成变量所包含内容的构思模型。为什么不让计算机来单步执行，然后为你显示变量的内容呢？

5.18 减少服务器请求的数量

5.18.1 问题

你的页面包含jQuery和许多插件。大量服务器请求会降低页面加载的速度：

```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" src="superfish.js"></script>
<script type="text/javascript" src="cycle.js"></script>
<script type="text/javascript" src="history.js"></script>
<script type="text/javascript" src="hoverintent.js"></script>
<script type="text/javascript" src="jcarousel.js"></script>
<script type="text/javascript" src="thickbox.js"></script>
<script type="text/javascript" src="validate.js"></script>
```

在页面加载之后，将用$.getJSON()下载一些JSON数据，从而增加一次服务器请求：

```
$(document).ready( function() {
    $.getJSON( 'myjson.php?q=test', function( json ) {
        $('#demo').html( json.foo );
    });
});
```

myjson.php是服务器上返回如下JSON数据的脚本：

```
{
    "foo": "bar"
}
```

5.18.2 解决方案

从Google的Ajax程序库中加载jQuery，将所有插件组合为单个文件：

```
<script type="text/javascript"

    src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>

<script type="text/javascript" src="plugins.js">
</script>
```

或者将最常用的JavaScript代码（jQuery、插件和自己的代码）组合为单个文件：

```
<script type="text/javascript" src="allmyscripts.js"></script>
```

不管采用哪一种方法，精简.js文件（移除注释和多余的空格），减小文件尺寸都是有帮助的。确保服务器对下载的文件使用gzip压缩。

对于JSON数据，因为页面由服务器应用程序生成，所以可以使用<script>标记，将JSON数据在生成时直接“烧制”到HTML页面中：

```
<script type="text/javascript">
    var myJson = {
        "foo": "bar"
     };
</script>
```

脚本标记中突出显示的部分和原始代码中myjson.php下载的JSON数据相同。在大部分服务器语言中，这样包含内容应该都很简单。

现在，使用JSON数据的jQuery代码很简单：

```
$(document).ready( function() {
    $('#demo').html( myJson.foo );
});
```

这又减少了一次服务器请求。

5.18.3 讨论

加速页面加载的关键之一是最大限度地减少HTTP请求的数量。请求不同的服务器也有所帮助。浏览器对于单个域（或者子域）只能有少量的并发下载，但是如果从不同域下载一些文件，浏览器可能并行下载它们。

> **注意**
>
> 将不同的<script>标记指向不同域可以并行下载标记中的内容，但是不会影响执行的顺序。<script>标记按照它们出现在HTML源代码中的顺序执行。

可以手动复制并粘贴到大文件来合并JavaScript文件。这对于开发来说不太方便，但是确实能加速下载。

对于各种服务器语言，有许多文件合并/精简程序。

Ruby on Rails:

- Bundle-fu（*http://jquery-cookbook.com/go/bundle-fu*）
- AssetPackager（*http://jquery-cookbook.com/go/asset-packager*）
- Rails 2.0内置的包装程序（packager）

PHP:

- Minify（*http://jquery-cookbook.com/go/minify*）

Python:

- JSCompile（*http://jquery-cookbook.com/go/js-compile*）

Java:

- YUI Compressor（*http://jquery-cookbook.com/go/yui-compressor*）

除了JavaScript代码，检查CSS是否有多个.css文件。上面列出的有些工具能够将.css文件组合为单个下载，就像它们对.js文件所做的那样。

> **注意**
>
> 以前，“包装”（packing）JavaScript曾经风靡一时。“包装”不仅删除注释和空格，还重写所有JavaScript代码，甚至使其不再成为JavaScript。包装需要在运行时有一个解包的步骤——在每次页面加载时，即使JavaScript代码已经缓存也必须完成这一步。因此，包装已经不再得到支持，而代之以“精简”代码（删除注释和空格）和gzip压缩的组合。包装的好处主要来自删除重复的字符串，而gzip也可以做到这一点。

5.19　编写无干扰式的JavaScript

5.19.1　问题

你的一个页面上有内联事件处理程序属性，为菜单创建悬停效果。

你的内容、表现（CSS）和行为（JavaScript）全都混在一起，难以单独维护，造成了JavaScript和样式设置的重复：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Menu Demo</title>

    <style type="text/css">
        .menu {
            background-color: #ccc;
            list-style: none;
            margin: 0;
            padding: 0;
            width: 10em;
        }
        .menu li {
            margin: 0;
            padding: 5px;
        }
        .menu a {
            color: #333;
        }
    </style>
</head>
<body>
<ul class="menu">
    <li onmouseover="this.style.backgroundColor='#999';"
        onmouseout="this.style.backgroundColor='transparent';">
        <a href="download.html">Download</a>
    </li>
    <li onmouseover="this.style.backgroundColor='#999';"
        onmouseout="this.style.backgroundColor='transparent';">
        <a href="documentation.html">Documentation</a>
    </li>
    <li onmouseover="this.style.backgroundColor='#999';"
        onmouseout="this.style.backgroundColor='transparent';">
        <a href="tutorials.html">Tutorials</a>
    </li>
</ul>
</body>
</html>
```

5.19.2　解决方案 用jQuery事件处理程序代替内联JavaScript，添加/删除类代替直接操纵`backgroundColor`样式：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Menu Demo</title>

    <style type="text/css">
        .menu {
            background-color: #ccc;
            list-style: none;
            margin: 0;
```

```
            padding: 0;
            width: 10em;
        }
        .menu li {
            margin: 0;
            padding: 5px;
        }
        .menu a {
            color: #333;
        }
        .menuHover {
            background-color: #999;
        }
    </style>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js">
    </script>

    <script type="text/javascript">

        $(document).ready( function() {
            $('li').hover(
                function() {
                    $(this).addClass('menuHover');
                },
                function() {
                    $(this).removeClass('menuHover');
                });
        });
    </script>
</head>
<body>

<ul class="menu">
    <li><a href="download.html">Download</a></li>
    <li><a href="documentation.html">Documentation</a></li>
    <li><a href="tutorials.html">Tutorials</a></li>
</ul>
</body>
</html>
```

我们已经删除了内联事件处理程序并用jQuery事件处理程序代替，隔离内容与行为。现在如果我们想添加更多的菜单项，就不必复制和粘贴同一批事件处理程序；这些事件处理程序会自动添加。

我们还将用于悬停效果的样式规则移到一个CSS类中，隔离行为和表现。如果我们以后想改变悬停效果的样式，只要更新样式表就可以了，没有必要修改标记。

5.19.3 讨论

尽管具有onevent属性的“合一版”HTML文件在简单的小型页面上工作得很好，但是它不具备很好的伸缩性。随着页面变得越来越复杂，隔离表现和行为能够生成更易于维护的代码。

我们不在这个简单的例子中进行上述工作，但是如果你有多个页面使用相同的JavaScript或者CSS代码，可以将这些代码移到公共的.js或者.css文件中。这样，就可以只把它下载到浏览器缓存中一次，而不用在每次页面加载时都重新发送。结果是，只要访问过一个页面，其他页面的加载都会变得更快。

5.20 将jQuery用于渐进增强

5.20.1 问题

你希望构建一个网站，用动画和Ajax提供具有极佳用户体验的简单任务管理功能，但是也希望支持禁用JavaScript的用户。

5.20.2 解决方案

你可以构建一个不具备浮华外观的网站，然后以无干扰的方式添加JavaScript功能：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Task List</title>

    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js">
    </script>
    <script type="text/javascript">

        $(document).ready( function() {
            var url = $('form').attr('action');
            $(':checkbox').click(function() {
                $.post(url, this.name + '=1');
                $(this).parent().slideUp(function() {
                    $(this).remove();
                });
            });
            $(':submit').hide();
        });
    </script>
</head>
<body>
<form method="post" action="tasklist.html">
    <ul>
        <li>
            <input type="checkbox" name="task1" id="task1" />
            <label for="task1">Learn jQuery</label>
        </li>
        <li>
            <input type="checkbox" name="task2" id="task2" />
            <label for="task2">Learn Progressive Enhancement</label>
        </li>
        <li>
            <input type="checkbox" name="task3" id="task3" />
            <label for="task3">Build Great Websites</label>
        </li>
    </ul>
    <input type="submit" value="Mark Complete" />
</form>
</body>
</html>
```

这个页面的输入表单不需要JavaScript>。用户登记已经完成的任务并提交表单，然后由服务器加载一个新页面，将完成的任务从列表中删除。

现在，可以用jQuery对该页面进行渐进增强：为复选框绑定一个事件处理程序，获得表单的提交URL并生成说明复选框选中的POST数据，模拟标准表单提交。然后，制作删除任务的动画为用户提供反馈。因为标记任务结束已经成为瞬时就能完成的任务，所以还隐藏了提交按钮。

5.20.3 讨论

尽管现在用户很少在不使用JavaScript的情况下浏览，但是构建可以在没有JavaScript的情况下正常工作的页面，然后用jQuery和JavaScript增强这些页面仍然是一个好的习惯。

> **注意**
>
> 别因为JavaScript增强而使用户的体验变差。非JavaScript版本的页面可能无法在你登记任务时立刻提出反馈，但是它确实给了你一个轻松改正错误的手段：在提交之前清除复选框或者完全不提交表单。
>
> 如果你在单击每个复选框的时候立刻提交，一定要为访问者提供撤销该操作的手段。如果任务项目在页面中消失，人们将会因为害怕单击错误的项目而不敢单击任何控件。你可以在页面中保留项目，但将其移动到“已完成”部分，或者添加一个明确的撤销选项。

5.21 使页面易于访问

5.21.1 问题

你打算构建一个具有复杂的窗口组件和许多Ajax功能的Web应用，但是又希望有残疾的访问者也能够使用它。

5.21.2 解决方案

在窗口组件中添加键盘可访问性和可访问富互联网应用程序（Accessible Rich Internet Applications，ARIA）语义。在下面的代码中，支持这些特性的更改用粗体表示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Dialog Demo</title>

    <style type="text/css">
        table {
            border-collapse: collapse;
            width: 500px;
        }
        th, td {
            border: 1px solid #000;
            padding: 2px 5px;
        }
        .dialog {
            position: absolute;
            background-color: #fff;
            border: 1px solid #000;
            width: 400px;
            padding: 10px;
        }
        .dialog h1 {
            margin: 0 0 10px;
        }
        .dialog .close {
            position: absolute;
            top: 10px;
            right: 10px;
        }
    </style>

    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js">
    </script>

<script type="text/javascript">

    $(document).ready( function() {
        function close() {
        dialog.hide();
        $('#add-user').focus();
     }

        var title = $('<h1>Add User</h1>')
                .attr('id', 'add-user-title'),

            closeButton = $('<button>close</button>')
                .addClass('close')
                .click(close)
                .appendTo(title),

            content = $('<div/>')
                .load('add.html'),
```

```
            dialog = $('<div/>')
                  .attr({
                       role: 'dialog',
                       'aria-labelledby': 'add-user-title'
                   })
                  .addClass('dialog')
                  .keypress(function(event) {
                     if (event.keyCode == 27) {
                        close();
                     }
                  })
                  .append(title)
                  .append(content)
                  .hide()
                  .appendTo('body');

        $('#add-user').click(function() {
             var height = dialog.height(),
                 width = dialog.width();
             dialog
                  .css({
                       top: ($(window).height() - height) / 2
                           + $(document).scrollTop(),
                       left: ($(window).width() - width) / 2
                           + $(document).scrollLeft()
                       })
                       .show();

                  dialog.find('#username').focus();

                  return false;
             });
        });
    </script>
</head>
<body>
<h1>Users</h1>
<a id="add-user" href="add.html">add a user</a>
<table>
<thead>
    <tr>
        <th>User</th>
        <th>First Name</th>
        <th>Last Name</th>
    </tr>
</thead>
<tbody>
    <tr>
        <td>jsmith</td>
        <td>John</td>
        <td>Smith</td>
    </tr>
    <tr>
        <td>mrobertson</td>
        <td>Mike</td>
        <td>Robertson</td>
    </tr>
    <tr>
        <td>arodriguez</td>
        <td>Angela</td>
        <td>Rodriguez</td>
    </tr>
    <tr>
        <td>lsamseil</td>
        <td>Lee</td>
        <td>Samseil</td>
    </tr>
    <tr>
        <td>lweick</td>
        <td>Lauren</td>
        <td>Weick</td>
    </tr>
</tbody>
</table>
```

```
</body>
</html>
```

用少量附加代码添加几个有用的功能：

- 添加了ARIA语义（role和aria-labelledby），这样屏幕阅读器之类的辅助技术设备能够知道 是一个对话框，而不仅仅是页面上的附加内容。
- 在打开对话框时将键盘焦点放在第一个输入字段上。这对于所有访问者都有帮助，不管他们的视力正常还是失明。
- 当对话框关闭时，将键盘焦点移回Add Users链接。
- 可以用Escape键撤销对话框。

5.21.3 讨论

因为ARIA还在开发之中，所以浏览器和屏幕阅读器支持仍然有限。但是现在添加它们，你就可以更好地准备迎接可以使用它的访问者。改进的键盘访问则能为所有访问者带来益处。

> **注意**
>
> 关于ARIA的更多信息参见如下网站：
>
> - WAI-ARIA概述（*http://jquery-cookbook.com/go/aria-overview*）
> - DHTML样式指南（*http://jquery-cookbook.com/go/dhtml-style-guide*）

不要被过时的DHTML这一名称所迷惑；DHTML样式指南是适用于所有最新窗口组件的键盘可访问性参考。

第6章　尺寸

Rebecca Murphey

6.0 引言

尺寸是为网站添加高级行为的核心部分。一旦知道如何操纵元素的尺寸和在页面上的位置，你对用户界面的控制水平就上了一个新台阶，能够在应用程序中提供类似桌面应用程序的行为和交互。

6.1 求取窗口和文档的尺寸

6.1.1 问题

你想要获得以像素表示的窗口和文档宽度和高度。

6.1.2 解决方案

jQuery的width和height方法提供对窗口或者文档基本尺寸的简单访问：

```
jQuery(document).ready(function() {
    alert('Window height: ' + jQuery(window).height()); // 返回视区高度
    alert('Window width: ' + jQuery(window).width()); // 返回视区宽度

    alert('Document height: ' + jQuery(document).height()); // 返回文档高度
    alert('Document width: ' + jQuery(document).width()); //返回文档宽度
});
```

6.1.3 讨论

理解文档宽度和高度可能（而且很有可能）与窗口的宽度和高度不同这一点很重要。窗口的尺寸指的是视区（浏览器可用于显示文档的部分）的大小。在大部分情况下，文档的高度大于窗口的高度。文档的宽度总是至少等于窗口的宽度，但也可以大于窗口宽度。在图6-1中，`jQuery('body').width() < jQuery(document).width()`，`jQuery(document).width() == jQuery(window).width()`。如果文档主体宽于窗口，文档宽度相应增加。

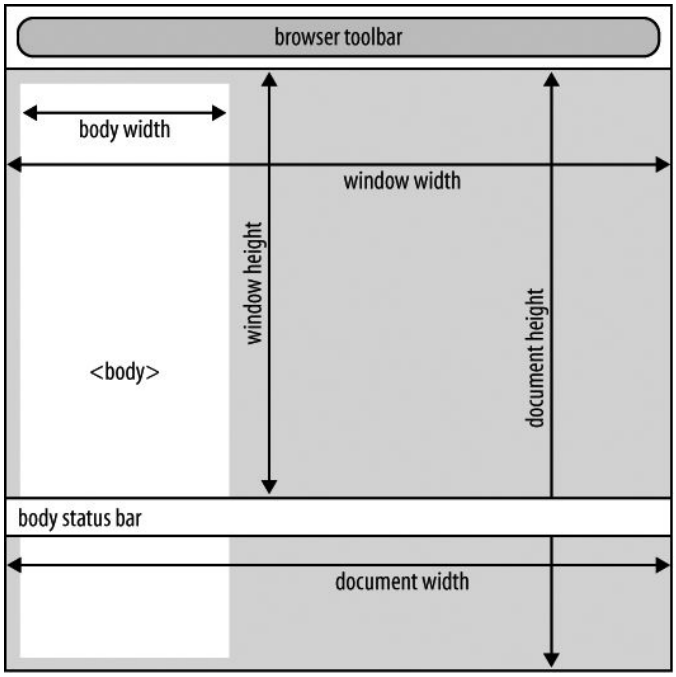


图6-1　文档大小和窗口大小往往不同

如果你想要设置元素的尺寸，`width`和`height`方法也可以接受参数。参数可以是整数——这时它被看做像素数，也可以是字符串——这是它被看做类似CSS的度量方式（也就是`$('#foo').width('300px')`这样的写法）。

6.2 求取元素的尺寸

6.2.1 问题

你希望确定元素占据的空间。

6.2.2 解决方案

width和height方法可以应用到任何元素，它们都能用于确定元素的宽度或者高度。但是如果需要确定元素在屏幕上实际占据的空间，它们就力不能及了。除了width和height之外，jQuery还提供了如下用于确定元素更具体尺寸的方法：

innerWidth

返回不包含边框但包含内边距的宽度。

innerHeight

返回不包含边框但包含内边距的高度。

outerWidth

返回包括边框和内边距的宽度。

outerHeight

返回包含边框和内边距的高度。

图6-2是视觉参考。

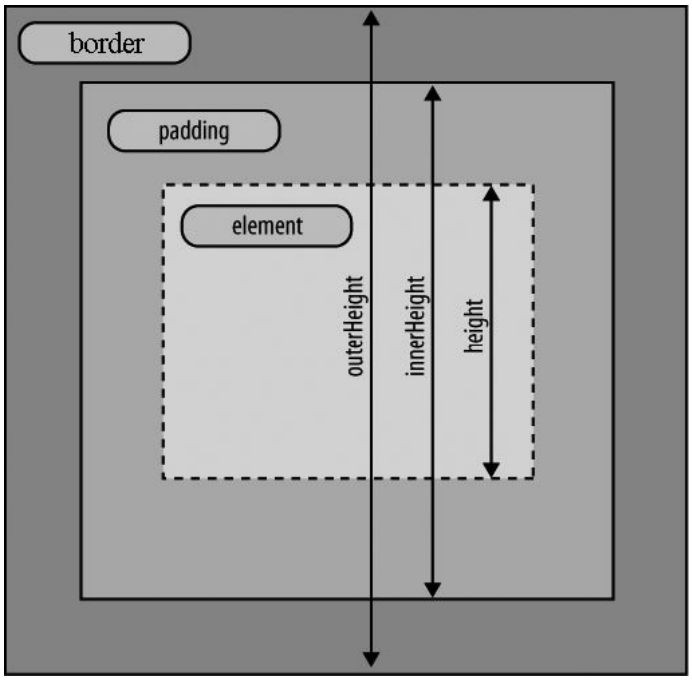


图6-2　元素height、innerHeight和outerHeight的图解

假定有如下HTML：

```
<div id="results"></div>
<div id="myDiv">Some text.</div>
```

和下面的CSS：

```
#myDiv {
    width:100px;
    height:30px;
    padding:10px;
    border:1px;
}
```

你可以预期下面的代码：

```
jQuery(document).ready(function() {
    var $myDiv = jQuery('#myDiv');
    var $results = jQuery('#results');

    jQuery('<p>Computed width: ' + $myDiv.width() + '</p>')
        .appendTo($results); // 100
    jQuery('<p>Computed height: ' + $myDiv.height() + '</p>')
        .appendTo($results); // 30
    jQuery('<p>Inner width: ' + $myDiv.innerWidth() + '</p>')
        .appendTo($results); // 120
    jQuery('<p>Inner height: ' + $myDiv.innerHeight() + '</p>')
        .appendTo($results); // 50
    jQuery('<p>Outer width: ' + $myDiv.outerWidth() + '</p>')
        .appendTo($results); // 122
    jQuery('<p>Outer height: ' + $myDiv.outerHeight() + '</p>')
        .appendTo($results); // 52
```

```
    jQuery('<p>Document outer height: ' + jQuery(document).outerHeight() + '</p>')
        .appendTo($results); // NaN
    jQuery('<p>Document inner height: ' + jQuery(document).innerHeight() + '</p>')
        .appendTo($results); // NaN
    jQuery('<p>Window outer height: ' + jQuery(window).outerHeight() + '</p>')
        .appendTo($results); // NaN
    jQuery('<p>Window inner height: ' + jQuery(window).innerHeight() + '</p>')
        .appendTo($results); // NaN
});
```

6.2.3 讨论

innerWidth/innerHeight和outerWidth/outerHeight方法是确定你想要的实际尺寸的有用工具——当你试图测量具有边框和内边距的元素在屏幕上占据的实际空间大小时，基本的宽度和高度方法用途有限。

注意，在jQuery(document)或jQuery(window)对象上使用innerWidth、innerHeight、outerWidth或outerHeight方法将返回NaN。

6.3 求取元素的偏移量

6.3.1 问题

你想要确定文档中某个元素的位置。

6.3.2 解决方案

jQuery提供三个确定元素位置的实用方法：

offset

返回一个对象，其中包含元素左上角与文档左上角的相对位置。

position

返回一个对象，其中包含元素左上角与该元素第一个定位的双亲元素（offsetParen）左上角的相对位置。

offsetParent

返回一个jQuery对象，其中包含元素的offsetParent。

offset方法可用于确定元素在页面上的位置——例如，在你打算滚动窗口到某个元素所在位置的时候。position方法用于改变元素位置，或者寻找元素在滚动容器中的位置。上述两个任务在后续几节中将讨论；本节的目标是概述定位方法。

假定有如下HTML，其中的<body>元素外边距为0，内边距为10个像素：

```
<body id="the_offset_parent">
    <h1>Finding the Offset of an Element</h1>
    <div id="foo">
        <div id="bar">Some text inside #bar, which is inside #foo</div>
    </div>

    <div id="results"></div>
</body>
```

可以使用如下代码确定两个DIV的位置、偏移量和offsetParent元素：

```
jQuery(document).ready(function() {
    var $foo = jQuery('#foo');
    var $bar = jQuery('#bar');

    var $results = jQuery('#results');
    var fooPosition = $foo.position();
    var barPosition = $bar.position();
    var fooOffset = $foo.offset();
    var barOffset = $bar.offset();
```

```
    var $fooOffsetParent = $foo.offsetParent();
    var $barOffsetParent = $bar.offsetParent();

    $results
        .append('<p>#foo position.top: ' + fooPosition.top + '</p>') // 10
        .append('<p>#foo position.left: ' + fooPosition.left + '</p>') // 10
        .append('<p>#bar position.top: ' + barPosition.top + '</p>') // 10
        .append('<p>#bar position.left: ' + barPosition.left + '</p>') // 10

        .append('<p>#foo offset.top: ' + fooOffset.top + '</p>') // 10
        .append('<p>#foo offset.left: ' + fooOffset.left + '</p>') // 10
        .append('<p>#bar offset.top: ' + barOffset.top + '</p>') // 10
        .append('<p>#bar offset.left: ' + barOffset.left + '</p>') // 10

        .append('<p>ID of #foo offsetParent: '
            + $fooOffsetParent.attr('id')) // the_offset_parent
        .append('<p>ID of #bar offsetParent: '
            + $barOffsetParent.attr('id')); // the_offset_parent
});
```

在这个例子中，两个元素的位置相同，也有相同的offsetParent（文档的<body>元素）。

但是如果用CSS定位#foo：

```
<body id="the_offset_parent">
    <div id="foo" style="position:absolute; top:10px; left:10px;">
        <div id="bar">Some text inside #bar, which is inside the
absolutely-positioned #foo</div>
    </div>

    <div id="results" style="position:absolute; top:60px; left:10px;"></div>
</body>
```

结果就改变了。#foo DIV实际上没有移动，它的offsetParent也没有变，所以它的位置和偏移量保持不变；#bar DIV也没有移动，但是因为它的offsetParent变化，所以位置也随之变化——记住，元素的位置是相对于其offsetParent的。

```
jQuery(document).ready(function() {
    var $foo = jQuery('#foo');
    var $bar = jQuery('#bar');

    var $results = jQuery('#results');
    var fooPosition = $foo.position();
    var barPosition = $bar.position();
    var fooOffset = $foo.offset();
    var barOffset = $bar.offset();

    var $fooOffsetParent = $foo.offsetParent();
    var $barOffsetParent = $bar.offsetParent();

    $results
        .append('<p>#foo position.top: ' + fooPosition.top + '</p>') // 10
        .append('<p>#foo position.left: ' + fooPosition.left + '</p>') // 10
        .append('<p>#bar position.top: ' + barPosition.top + '</p>') // 0
        .append('<p>#bar position.left: ' + barPosition.left + '</p>') // 0

    .append('<p>#foo offset.top: ' + fooOffset.top + '</p>') // 10
    .append('<p>#foo offset.left: ' + fooOffset.left + '</p>') // 10
```

```
    .append('<p>#bar offset.top: ' + barOffset.top + '</p>') // 10
    .append('<p>#bar offset.left: ' + barOffset.left + '</p>') // 10

    .append('<p>ID of #foo offsetParent: '
        + $fooOffsetParent.attr('id')) // the_offset_parent
    .append('<p>ID of #bar offsetParent: '
        + $barOffsetParent.attr('id')); // foo
});
```

6.3.3 讨论

需要记住的一点是：offset方法给出的总是元素相对于文档的位置。position方法的返回值可能是元素相对于文档的位置，这取决于元素是否有一个offsetParent。如果有（也就是说，有一个应用了定位的父元素），那么position方法提供的位置信息将是相对于offsetParent而不是相对于文档的。

> **注意**
>
> jQuery的offsetParent方法提供了标准JavaScript的DOM属性offsetParent的一个替代品。在某些情况下——例如，元素有固定位置时——有些浏览器请求元素的offsetParent属性时会返回null。

6.4 滚动元素使其可见

6.4.1 问题

你希望滚动文档或者元素，使另一个元素可见。

6.4.2 解决方案：滚动整个窗口

如果需要滚动整个窗口，可以使用offset方法确定目标元素与文档的相对位置，然后用scrollTop方法滚动文档使该元素可见。

例如，假定希望在用户单击#bar元素时滚动到#foo元素：

```
jQuery('#bar').click(function() {
    var fooOffset = jQuery('#foo').offset(),
        destination = fooOffset.top;
    jQuery(document).scrollTop(destination);
});
```

6.4.3 解决方案：在一个元素中滚动

如果目标元素在一个滚动容器中，可以使用positon方法确定目标元素与容器的相对位置，将其添加到容器的当前滚动位置，然后使用scrollTop方法滚动容器，使元素可见。注意，滚动容器必须使用position: relative、position: absolute或者position: fixed定位，这种方法才能正常工作。

例如，考虑如下标记样式，#foo的大小不足以同时显示两个段落。

```
<head>
    <style>
    #foo {
    width:300px;
    padding:10px;
    height:20px;
    border:1px solid black;
    overflow:auto;
    position:relative;
    }
    </style>
</head>
<body>
    <input type="button" id="bar" value="Click to scroll to last paragraph" />
    <input type="button" id="bam" value="Click to scroll to last paragraph with
animation" />
    <div id="foo">
    <p>This is the first paragraph. Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
```

```
id est laborum.</p>
    <p>This is the second paragraph. Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
id est laborum.</p>
    <!-- several more paragraphs -->
    </div>
</body>
```

滚动#foo显示最后一个段落很简单：

```
var $foo = jQuery('#foo');
$('#bar').click(function() {
    var lastParagraphPosition = jQuery('#foo p:last').position();
    var scrollPosition = $foo.scrollTop() + lastParagraphPosition.top;
    $foo.scrollTop(scrollPosition);
});
```

在上述两个例子中，滚动都是立即发生的——这很有效，但是不一定具有吸引力。动画方法能动态改变元素的scrollTop属性，因此可以很轻松地创建过渡动画。下面的代码为滚动容器添加动画效果：

```
var $foo = jQuery('#foo');
$('#bam').click(function() {
    var lastParagraphPosition = jQuery('#foo p:last').position();
    var scrollPosition = $foo.scrollTop() + lastParagraphPosition.top;
    jQuery('#foo').animate({scrollTop: scrollPosition}, 300);
});
```

注意

jQuery还包含一个scrollLeft方法，它的表现与scrollTop类似。

6.5 确定元素是否在视区内

6.5.1 问题

你希望确定元素是否在视区内；还希望进一步确定元素可见的百分比，如果可见比例小于50%则进行滚动。

6.5.2 解决方案

使用本章前几节讨论的几个方法。

这一过程有如下几个步骤：

1. 确定视区大小。

2. 确定文档的滚动位置。

3. 计算元素可见时左上角位置的最小和最大值。

4. 用计算出来的这些值与元素位置比较。

```
jQuery(document).ready(function() {
    var viewportWidth = jQuery(window).width(),
        viewportHeight = jQuery(window).height(),

        documentScrollTop = jQuery(document).scrollTop(),
        documentScrollLeft = jQuery(document).scrollLeft(),

        minTop = documentScrollTop,
        maxTop = documentScrollTop + viewportHeight,
        minLeft = documentScrollLeft,
        maxLeft = documentScrollLeft + viewportWidth,
        $myElement = jQuery('#myElement'),
        elementOffset = $myElement.offset();
    if (
         (elementOffset.top > minTop && elementOffset.top < maxTop) &&
         (elementOffset.left > minLeft &&elementOffset.left < maxLeft)
    ) {
        alert('element is visible');
    } else {
        alert('element is not visible');
    }
});
```

利用这个解决方案，我们知道元素的顶部在视区中是否可见；更好的解决方案应该测试整个元素是否包含在视区中：

```
jQuery(document).ready(function() {
    var viewportWidth = jQuery(window).width(),
        viewportHeight = jQuery(window).height(),
        documentScrollTop = jQuery(document).scrollTop(),
        documentScrollLeft = jQuery(document).scrollLeft(),
```

```
        $myElement = jQuery('#myElement'),

        elementOffset = $myElement.offset(),
        elementHeight = $myElement.height(),
        elementWidth = $myElement.width(),

        minTop = documentScrollTop,
        maxTop = documentScrollTop + viewportHeight,
        minLeft = documentScrollLeft,
        maxLeft = documentScrollLeft + viewportWidth;

    if (
         (elementOffset.top > minTop && elementOffset.top + elementHeight < maxTop) &&
         (elementOffset.left > minLeft && elementOffset.left + elementWidth < maxLeft)
    ) {
        alert('entire element is visible');
    } else {
        alert('entire element is not visible');
    }
});
```

作为替代，可以查看元素可见的部分有多少——如果它小于某个数值，就滚动到该元素：

```
jQuery(document).ready(function() {

var viewportWidth = jQuery(window).width(),
    viewportHeight = jQuery(window).height(),

    documentScrollTop = jQuery(document).scrollTop(),
    documentScrollLeft = jQuery(document).scrollLeft(),

    $myElement = jQuery('#myElement'),

    verticalVisible, horizontalVisible,

    elementOffset = $myElement.offset(),
    elementHeight = $myElement.height(),
    elementWidth = $myElement.width(),

    minTop = documentScrollTop,
    maxTop = documentScrollTop + viewportHeight,
    minLeft = documentScrollLeft,
    maxLeft = documentScrollLeft + viewportWidth;

function scrollToPosition(position) {
    jQuery('html,body').animate({
        scrollTop : position.top,
        scrollLeft : position.left
    }, 300);
}
if (
     ((elementOffset.top > minTop && elementOffset.top < maxTop) ||
     (elementOffset.top + elementHeight > minTop && elementOffset.top +
elementHeight < maxTop))
&&
     ((elementOffset.left > minLeft && elementOffset.left < maxLeft) ||
     (elementOffset.left + elementWidth > minLeft && elementOffset.left +
elementWidth < maxLeft))
) {
    alert('some portion of the element is visible');
```

```
    if (elementOffset.top >= minTop && elementOffset.top + elementHeight
<= maxTop) {
        verticalVisible = elementHeight;
    } else if (elementOffset.top < minTop) {
        verticalVisible = elementHeight - (minTop - elementOffset.top);
    } else {
        verticalVisible = maxTop - elementOffset.top;
    }
    if (elementOffset.left >= minLeft && elementOffset.left + elementWidth
<= maxLeft) {
        horizontalVisible = elementWidth;
    } else if (elementOffset.left < minLeft) {
        horizontalVisible = elementWidth - (minLeft - elementOffset.left);
    } else {
        horizontalVisible = maxLeft - elementOffset.left;
    }

    var percentVerticalVisible = (verticalVisible / elementHeight) * 100;
    var percentHorizontalVisible = (horizontalVisible / elementWidth) * 100;

    if (percentVerticalVisible < 50 || percentHorizontalVisible < 50) {
        alert('less than 50% of element visible; scrolling');
        scrollToPosition(elementOffset);
    } else {
        alert('enough of the element is visible that there is no need to scroll');
    }
} else {
    // 元素不可见;滚动
    alert('element is not visible; scrolling');
    scrollToPosition(elementOffset);
}
});
```

注意

Ariel Flesler开发的scrollTo插件（*http://jquery-cookbook.com/go/plugin-scrollto* ）提供了对许多这类方法的快捷访问方法，只要编写`$.scrollTo('#myElement');`这样的代码就能确定目标元素的位置。

6.6 将元素放在视区的中央

6.6.1 元素

你希望滚动窗口，将某个元素放在视区的中央。

6.6.2 解决方案

获取视区的尺寸；确定元素的宽度、高度和偏移量；利用一些算术运算将元素放在视区的中央：

```
jQuery(document).ready(function() {
    jQuery('#bar').click(function() {
        var viewportWidth = jQuery(window).width(),
            viewportHeight = jQuery(window).height(),

            $foo = jQuery('#foo'),
            elWidth = $foo.width(),
            elHeight = $foo.height(),
            elOffset = $foo.offset();
        jQuery(window)
            .scrollTop(elOffset.top + (elHeight/2) - (viewportHeight/2))
            .scrollLeft(elOffset.left + (elWidth/2) - (viewportWidth/2));
    });
});
```

在最后一行中，将元素顶部偏移量加上元素高度的一半，确定元素的垂直中心。然后，减去视区高度的一半，确定窗口滚动的位置。最后，进行类似的计算，在水平方向上使视区居中。

6.7 在当前位置绝对定位一个元素

6.7.1 问题

你希望将一个静态或者相对定位的元素改成绝对定位的。

6.7.2 解决方案

要实现这一目标，只要获得元素的位置，然后相应设置CSS属性即可：

```
var $myElement = jQuery('#foo p').eq(0),
             elPosition = $myElement.position();

         $myElement.css({
             position : 'absolute',
             top : elPosition.top,
             left : elPosition.left
         });
```

也可以简单地相对当前位置重新定位一个元素：

```
var $myElement = jQuery('#foo p').eq(1),
             elPosition = $myElement.position();

         $myElement.css({
             position : 'absolute',
             top : elPosition.top + 20,
             left : elPosition.left + 20
         });
```

6.8 按照与另一个元素的相对位置定位元素

6.8.1 问题

你打算相对现有元素定位一个新元素。

6.8.2 解决方案

获取现有元素的宽度、高度和偏移量，用这些值相应地定位新元素。

假定有如下HTML：

```
<style>
#foo {
    width: 300px;
    height: 100px;
    border: 1px solid red;
    padding: 5px;
}
#tooltip {
    border: 1px solid black;
    padding: 5px;
    background-color: #fff;
</style>

<div id="foo">An existing element</div>
```

下面的代码将为现有的元素添加一个兄弟元素，但是定位在元素“内部”，顶部距离现有元素顶部10像素，左侧距离现有元素左侧10像素，宽度比现有元素短20像素：

```
jQuery(document).ready(function() {
    var $foo = jQuery('#foo'),
        fooPosition = $foo.position(),
        $tooltip = $('<div id="tooltip">A new element</div>').insertAfter($foo);

    $tooltip.css({
        position : 'absolute',
        top : fooPosition.top + 10,
        left : fooPosition.left + 10,
        width : $foo.width() - 20
    });
});
```

如果你希望在页面的其他地方添加一个新元素——也就是说，如果你不希望它是现有元素的兄弟元素——可以调整代码查看原始元素的偏移量而非位置：

```
jQuery(document).ready(function() {
    var $foo = jQuery('#foo'),
        fooOffset = $foo.offset(),
        $tooltip = $('<div id="tooltip">A new element</div>').appendTo('body');

    $tooltip.css({
```

```
        position : 'absolute',
        top : fooOffset.top + 10,
        left : fooOffset.left + ($foo.width() / 2),
        width : $foo.width() - 20
    });
});
```

6.9 根据浏览器宽度切换样式表

6.9.1 问题

你希望根据浏览器的宽度改变文档的CSS。

6.9.2 解决方案

对这个问题有几种解决方案。第一种是修改正文元素的class属性，第二种是改变所要修改的样式表的href属性，最后一种是在页面上包含所有与尺寸相关的样式表，但是在同一时间内只启用一个。

在每一种情况下，都将创建一个函数检查浏览器的宽度并将其绑定到文档的ready事件和窗口的resize事件。然后，checkWidth函数调用setSize函数，将根据采用的不同方法定义这个函数：

```
var checkWidth = function() {
    var browserWidth = $(window).width();
    if (browserWidth < 960) {
        setSize('small');
    } else {
        setSize('large');
        }
    };
jQuery(document).ready(function() {
    checkWidth();
    $(window).resize(checkWidth);
});
```

setSize函数定义取决于切换样式的方法。

6.9.3 解决方案1：修改正文元素的类

```
var setSize = function(size) {
    var $body = jQuery('body');
      $body.removeClass('large small').addClass(size);
};
```

6.9.4 解决方案2：修改负责设置与尺寸相关样式的样式表的href属性

假定在文档中有如下与尺寸相关的样式表：

```
<link rel="stylesheet" type="text/css" id="css_size" href="size-small.css" />
```

在这种情况下，可以定义如下setSize函数：

```
var setSize = function(size) {
    var $css = jQuery('#css_size');
    $css.attr('href', 'size-' + size + '.css');
};
```

注意，在这种情况下，从服务器请求新的CSS文件，可能导致在样式变化发生时有短暂的延时。因此，这可能是最不受欢迎的方法。

6.9.5 解决方案3：在页面中包含所有与尺寸相关的样式表，但一次只启用一个

```
<link rel="stylesheet" type="text/css" class="css_size small" href="size-small.css" />
<link rel="alternate stylesheet" type="text/css" class="css_size large"
    href="size-large.css" disabled=true/>
```

在这种情况下，定义如下setSize函数：

```
var setSize = function(size) {

    jQuery('link.css_size').each(function() {
        var $this = $(this);
        if ($this.hasClass(size)) {
            $this
                .removeAttr('disabled')
                .attr('rel', 'stylesheet');
        } else {
            $this
                .attr('disabled', true)
                .attr('rel', 'alternate stylesheet');
        }
    });
};
```

在这种方法中，所有样式表都在页面加载时载入，切换不同样式表时不读取任何新的内容，这减少了解决方案2导致的延迟，但是如果用户不需要备选的样式表，这种方案也可能造成不必要的HTTP请求。

6.9.6 讨论

这三种样式切换方法没有绝对的好坏之分。当选择某种方法时，你应该考虑用户需要不同样式表的可能性，尺寸相关样式表的大小，以及对管理尺寸相关样式表的偏好。在许多情况下，第一种解决方案既能满足要求，在其他方面也较为合理。

第7章　特效

Remy Sharp

7.0 导言

jQuery自带了许多预设的特效和健壮的低级动画方法，可以用它们来创建自己的特效。

这些预设的特效包括：

- 以切换方式隐藏和显示元素；
- 缩放同时淡入/淡出元素；
- 上下滑动和切换；
- 淡入淡出为特定的不透明度。

所有这些预设特效都支持速度和完成时触发的回调函数。

除了预定义特效之外，还有一些工具有助于更好地控制动画：

- 评估元素是否处于动画过程中的:animated选择器；
- 关闭和开启全部特效的功能；
- 在动画队列中添加自定义函数的功能；
- 修改整个动画队列的函数。

> **警告**
>
> 值得一提的是，经过封装的动画方法hide（持续一段时间）和slideup在元素高度接近0的时候会减小内外边距。这可能影响到页面代码和特效所用CSS的编写方法。还要注意，jQuery对于怪癖（Quirks）模式中的文档特效还不正式支持。

7.0.1 动画方法

使用animate方法，可以完全控制动画，实现预定的特效。animate方法能够完成如下功能：

- 控制CSS属性（仅限于数值型的属性）；
- 控制DOM属性scrollTop和scrollLeft（如果元素有overflow属性）；
- 使用任何CSS计量单位，如像素、em、英寸或者百分比来表示终点值；
- 以固定值或者元素当前状态的相对值指定特效的终点；
- 使用toggle值表示状态之间的切换，例如，opacity: toggle；
- 指定完成动画的缓动（easing）方法；
- 在动画的各个阶段设置回调：在动画的各个步骤和结束时；
- 指定多个动画是否应该排队，还是立即同时显示动画。

> **警告**
>
> 当指定动画属性时，它们必须以驼峰式大小写方式编写，例如，marginLeft而不是margin-left。如果不这么做，什么动画也不会显示！

7.0.2 动画速度

speed参数可以用毫秒或者几种预定义字符串指定：

- slow表示600毫秒。
- fast表示200毫秒。

如果动画函数中没有显式指定速度值，动画将以400毫秒的默认速度运行。

> **注意**
>
> 如果明确地传递速度值0，动画将和.css()函数一样运行，但是在jQuery 1.3版本中，该方法调用将同步运行，而不采用其他所有动画的异步方式。

7.0.3 特效模板

在本秘诀中，除非另作说明，否则将对所有示例都采用如下模板，为每个解决方案应用不同的jQuery片段：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
   <title>Chapter 6</title>
   <link rel="stylesheet" href="chapter6.css" type="text/css" />
   <script src="jquery-latest.js" type="text/javascript"></script>
</head>
<body id="single">
   <input type="button" id="animate" value="animate" />
   <div class="box">
   <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
     eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
   </div>
</body>
</html>
```

所有示例都可以从*http://jquery-cookbook.com/examples/06/* 在线获得，包括各个秘诀组合而成的完整版本。

7.1 滑动和淡入/淡出元素

7.1.1 问题

我们想要在视图中显示或者切换一块内容。这可能是由用户单击某个元素或者其他事件触发。

简单地显示/隐藏在视觉上可能不和谐，我们希望创建一个渐变特效来显示这些内容。

在下面的解决方案中，假定我们打算允许用户切换特效。

7.1.2 解决方案

作为参考，如果只是要显示元素，代码如下：

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').show();
   });
};
```

如果我们想要切换方框的显示/隐藏，可以使用下面的语句代替`.show()`：

```
$('.box').toggle();
```

但是，解决方案要比简单地切换显示属性更吸引人。所以，我们来看看滑动和淡入/淡出方法的使用：

Slide

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').slideToggle('slow');
   });
});
```

Fade

因为没有不透明度的切换函数，所以可以使用`fadeIn`和`fadeOut`的组合：

```
$(document).ready(function () {
   $('#animate').click(function () {
      var $box = $('.box');
      if ($box.is(':visible')) {
         $box.fadeOut('slow');
      } else {
         $box.fadeIn('slow');
      }
```

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

也可以用fadeTo方法创建自己的淡入/淡出切换动画：

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').fadeTo('slow', 'toggle');
   });
});
```

但是，我认为使用animate方法对未来的维护更好：

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').animate({ opacity : 'toggle' }, 'slow');
   });
});
```

同时切换

如果我们想同时切换高度和不透明度，可以重用前面的解决方案，同时切换高度。这使得方框在淡出的同时向上滑动：

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').animate({
         opacity : 'toggle',
         height: 'toggle'
      }, 'slow');
   });
});
```

7.1.3 讨论

正如在前面的解决方案中所看到的那样，滑动和淡入/淡出方法比起直接显示（和隐藏）以及切换方法前进了一步。滑动方法有如下几种：

- slideUp
- slideDown
- slideToggle

淡入/淡出方法没有明确的切换功能，但是也可以实现该功能，淡入/淡出方法有如下几个：

- fadeIn
- fadeOut
- fadeTo

除了fadeTo之外，其余方法都以speed作为第一个参数，第二个参数则是一个回调函数，这两个参数都是可选的。回调函数在动画完成时调用，把上下文设置为动画应用

的元素；也就是说，this变量指的是当前元素。

我选择animate代替fadeTo切换不透明度的原因是fadeTo的参数很容易误解。如果新的开发人员查看这段代码，使用动画函数几乎就和普通的英语没什么两样，因而更容易阅读和理解代码中所发生的操作。

还要再补充一点，如果在show（或者hide）方法中使用速度值，该方法将在一次动画中变动高度、宽度、不透明度、外边距和内边距，如图7-1所示。

图7-1　向show方法传递速度参数，
在动画中变动高度、宽度、内边距、外边距和不透明度

7.2 通过向上滑动使元素可见

7.2.1 问题

你打算将一块内容滑动到可视区域，但是UI设计指定这块内容必须向上滑动才能显示。slideUp方法可以隐藏元素，从其顶部位置减去它的高度。

为了向上滑动，需要使用CSS定位元素，然后考虑所要显示的内容。

7.2.2 解决方案

1. HTML

我们需要绝对定位打算以动画显示的元素，使其保持在底部位置，这样在显示的时候它就会向上滑动。

为此，需要将动画元素包装在另一个 (或者适合设计的其他元素)中，设置position:relative样式。(也可以是position: absolute。只需要一个事先定义的位置，以便触发#revealUp上的position: absolute样式，进行相对定位；但是，因为我们希望文档流程正常，所以使用了position:relative。）

```
<div class="box">
   <div id="revealUp">
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
      eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
   </div>
</div>
```

2. CSS

现在，需要为box元素设置一个相对位置，这样才能相对于它绝对定位#revealUp:

```
.box {
   position: relative;
}

#revealUp {
   position: absolute;
   overflow: hidden;
   display: none;
   bottom: 0;
   background-color: #c00;
   height: 0;
}
```

3. jQuery

可以根据元素的高度切换#revealUp。我们打算花费更长的时间显示增加高度的动画（通过检查当前高度）而不只是使用slideToggle()——我们将在7.2.3节中提及这么做的原因：

```
$(document).ready(function () {
   $('#animate').click(function () {
      var $box = $('#revealUp');

      if ($box.height() > 0) {
         $box.animate({ height : 0 });
      } else {
         $box.animate({ height : '100%' });
      }
   });
});
```

7.2.3 讨论

这个解决方案要求我们检查方框的高度，然后确定下一步的行动。

注意，不使用slideToggle，该方法和.animate({ height: 'toggle' })即使不是完全一样，也已经非常接近。

不使用切换方法的原因是：为了使切换正常工作，必须捕捉实际的高度。因为元素的高度从0开始，jQuery无法计算出完整的高度。如果使用slideToggle，#revealUp元素在短暂的时间里以1像素的高度出现，然后消失。这是因为没有可以用来显示动画的实际高度。

相反，我们确定高度是否大于0，然后相应地变动高度显示动画。因为元素嵌套在另一个绝对定位的元素中，所以可以将高度定为100%，这样它就会逐步变大直到充满整个空间。

警告

在本秘诀中，已经使用了overflow:hidden。但是，如果用户增大字体，我的示例会隐藏某些内容。在实际解决方案中，一定要测试内容在字体增大时仍然可用，并考虑确保显示用的方框足够大，或者在#revealUp元素上使用overflow: auto样式。

7.3 创建水平折叠特效

7.3.1 问题

jQuery UI程序库提供了现成的垂直折叠特效支持，实际上，可以用一些简单的代码段创建基本的折叠特效。但是，水平方向的折叠需要特殊的CSS，在jQuery代码上要采取稍有不同的措施。

在这个解决方案中，不使用模板，因为模板中的标记与用于水平折叠的标记不同。

7.3.2 解决方案

1. HTML

```
<div id="accordionWrapper">
   <h3 class="red"><a href="#red">Red</a></h3>
   <div id="red" class="box"><p>Lorem ipsum dolor sit amet, consectetur
adipisicing.</p></div>

   <h3 class="green"><a href="#green">Green</a></h3>
   <div id="green" class="box"><p>Lorem ipsum dolor sit amet, consectetur
adipisicing.</p></div>

   <h3 class="blue"><a href="#blue">Blue</a></h3>
   <div id="blue" class="box"><p>Lorem ipsum dolor sit amet, consectetur
adipisicing.</p></div>
</div>
```

2. CSS

```
#accordionWrapper {
   margin: 10px;
}

#accordionWrapper h3 a {
   text-indent: -9999px;
   height: 150px;
   width: 50px;
   float: left;
}

#accordionWrapper .red {
   background: #c00 url(images/red.png) no-repeat;
}

#accordionWrapper .green {
   background: #0c0 url(images/green.png) no-repeat;
}

#accordionWrapper .blue {
   background: #00c url(images/blue.png) no-repeat;
}

#accordionWrapper div.box {
```

```
   float: left;
   height: 150px;
   width: 150px;
   border: 0;
   margin: 0;

   /* to cancel the image from .red, etc */
   background-image: none;
}
```

3. jQuery

```
$.fn.horizontalAccordion = function (speed) {
   return this.each(function () {
      var $accordionHeaders = $(this).find('h3'),
         $open = $accordionHeaders.next().filter(':first'),
         width = $open.outerWidth();

      //初始化显示
      $accordionHeaders.next().filter(':not(:first)').css({ display : 'none', width : 0
});

      $accordionHeaders.click(function () {
         if ($open.prev().get(0) == this) {
            return;
         }
         $open.animate({ width: 0 }, { duration : speed });
         $open = $(this).next().animate({ width : width }, { duration : speed });
      });
   });
};

$(document).ready(function () {
   $('#accordionWrapper').horizontalAccordion(200);
});
```

7.3.3 讨论

在HTML和CSS中设计了折叠特效的布局，其中的元素都向左浮动。如果将这样的布局用在一个网页上，可能应该在折叠之后直接添加一个清除元素，使后面的内容能够有正常的流程。

通过向左浮动元素，折叠特效以h3>a作为内容面板的标题。

如果禁用CSS和JavaScript，内容流程正常，而且可以被Google等搜索引擎理解。

如果启用CSS而禁用JavaScript，默认的视图可以看到所有内容面板。

使用jQuery，通过隐藏除了第一个面板之外的所有面板初始化显示，并将单击处理程序绑定到标题，将内容滑入/滑出可视区域。

水平折叠特效已经被制作成一个jQuery插件，特别要说明的是，不需要在折叠特效中硬编码任何变量。只要将持续速度变量传递给插件，就能确定特效的持续时间。可以简单地升级该插件，使其能够进行缓动工作或者回调。

要注意的是，在整段代码中，所有单击处理和DOM浏览都围绕<h3>元素而非<a>元素发生。这仍然习以为常工作，保持代码相对简单（不需要从<a>元素向上和向下浏览以得到父元素<h3>和邻近的 元素），但是更重要的是，因为<a>元素能够用Tab键进入和通过键盘触发，所以这些代码能够提供键盘可访问性。不必为<a>元素绑定单击事件处理程序，因为当<a>元素触发单击事件（通过单击或者键盘）时，事件会通过DOM向上冒泡，从而被<h3>元素上的单击处理程序捕获。

该插件首先收集折叠特效的必要部分：可单击的标题、第一个可见的面板和面板的宽度（注意，这个版本的插件只能用于相同尺寸的面板）：

```
var $accordionHeaders = $(this).find('h3'),
```

this是当前的折叠包装器元素，通常是一个 。

从折叠包装器中，代码收集所有<h3>元素。注意，将使用next()和prev()将DOM集合从<h3>改为DOM树中的下一个节点，特别是各个折叠内容面板：

```
$open = $accordionHeaders.next().filter(':first'),
```

$open是一个临时变量，指向当前可见的面板。不能使用.is(':visible')，因为实际上减少了宽度，而面板仍然有display: block属性。所以，通过这个$open变量跟踪当前面板：

```
width = $open.outerWidth();
```

在初始化的最后，捕捉打开的面板的宽度，以便正确地显示变动面板宽度的动画。

现在还剩下两件工作：

- 初始化面板视图，仅显示第一个面板；
- 绑定单击处理程序以显示/隐藏面板。

当初始化视图时，隐藏第一个之外的所有面板。还必须将宽度设置为0，使动画函数能够逐渐增加宽度，而不是在显示的时候“弹出”面板。

使用来自$open变量的反向过滤器来实现这一功能，特别是:not(:first)：

```
$accordionHeaders.next().filter(':not(:first)').css({ display : 'none',width : 0 });
```

一旦选择了除了第一个之外的其他面板，就修改CSS属性对其进行初始化。

最后，附加单击处理程序。

记住，$accordionHeaders包含h3元素，所做的第一件事情就是：如果单击的<h3>就是当前打开的面板，那么什么也不做。

因为$open变量就是面板，所以使用.prev()浏览前一个<h3>元素，测试它与单击元素的当前上下文是否匹配。如果单击的元素不是当前打开的面板，就将$open面板的宽度`逐渐减小为0，而将当前单击的面板宽度逐步增大为捕捉到的宽度。

注意单击事件处理程序的最后一行：

```
$open = $(this).next().animate({ width : width }, { duration : speed });
```

因为jQuery通常返回jQuery（除了获取一个值的时候），而且将要以动画显示的是现在要打开的面板，所以可以同时捕捉$open变量中的面板，用最新的面板覆盖它。

7.4 同时滑动和淡入/淡出元素

当页面的某些部分隐藏，仅在特定操作时显示给用户，有时候简单的显示/隐藏就足够了。我们打算为访问者创建更赏心悦目的特效。

根据页面的布局，即时的显/隐特效不足以让访问者清楚所显示的内容。这就是将元素滑入视野的另一个好处，因为这样能给访问者一个视觉提示，让他看清页面布局的变化。

可以使用jQuery的内置显示方法，传递一个持续时间，它几乎能完成所需要的任务，但是这还不够，因为它也会变动元素宽度来显示动画，就像图7-1中显示的那样。前面你就应该已经注意到，show方法将会变动元素周围的内外边距，为了解决这个问题，将使用动画函数创建一个自定义特效。

7.4.1 解决方案

使用动画方法同时切换高度和不透明度：

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').animate({ opacity: 'toggle', height: 'toggle' });
      return false;
   });
});
```

7.4.2 讨论

使用animate方法使我们能够明确指定使用哪个CSS属性来创建动画特效。

还使用toggle作为终点值。这样，animate方法将当前高度作为初始状态，并在0和初始状态的100%之间切换。

在例子中，方框的初始状态是可见的。如果我们希望它滑动并淡入可视区域，只需要将样式表中的显示属性设置为none。

警告

没有必要在样式中将高度设置为0；实际上，这样做意味着动画不会扩展到正确的高度，因为它将在0高度（来自CSS）和0高度之间切换，从而不会显示任何内容（slideUp终点）。

7.5 应用连续的特效

7.5.1 问题

你希望在一组元素上显示特效之后，在另一组元素上应用不同的特效。如果你只有一组其他的特效需要执行，这就很容易解决，但是如果你希望一个接一个地为任何数量的元素应用特效，代码可能很难维护。

7.5.2 解决方案

本解决方案使用本章开篇时概述的标准模板，例外之处是在页面上有div.box元素的多个副本。这一解决方案设计为可以处理任何数量（从一个到许多个）的div.box元素，因为自动序列解决方案能够应对所有元素。

1. 手动回调

应用连续特效的基本方法是使用回调。如果下一个特效与第一个特效不同也可以采用这种方法：

```
$(document).ready(function () {
   var $boxes = $('.box').hide();

   $('#animate').click(function () {
      $boxes.eq(0).fadeIn('slow', function () {
         $boxes.eq(1).slideDown('slow');
      });
   });
});
```

2. 自动序列

另一种方法是根据Dave Methvie的解决方案，可以在任何数量的元素上按照顺序重复执行特效：

```
$(document).ready(function () {
   var $boxes = $('.box').hide(),
      div = 0;

   $('#animate').click(function () {
      $($boxes[div++] || []).fadeIn('slow', arguments.callee);
   });
});
```

7.5.3 讨论

简单的解决方案使用了回调功能，然后进入序列中的下一个动画。使用的选择器以第一个div.box为目标；但是，这样的方法不具备伸缩性，因为它预计有两个（且只有两个）动画元素。如果元素少于两个，代码就会中断，多于两个则会丢失一些元素。

如果动画序列中有更多（甚至未知个数）的元素，那么Dave Methvin的解决方案是完美的。

代码采用了两个技巧，第一个是使用空数组进行故障恢复：

```
$($boxes[div++] || [])
```

这段代码递增索引计数，如果元素不存在，则向jQuery传递一个空数组。

当jQuery结果集为空，运行一个没有任何效果的动画。因为结果为空，所以jQuery不向链式调用传递任何DOM元素，链式方法的任何回调也就不会调用。

例如，如果运行如下代码，警告框永远不会出现——这是本秘诀正常工作的关键因素：

```
$('made-up-element').show(function () {
   alert('will never appear');
});
```

本秘诀的第二个技巧是回调参数：

```
arguments.callee
```

arguments是JavaScript的一个关键字，它指代所有函数都能访问的一个局部变量。arguments对象类似于任何数组，但是没有任何数组方法（如slice）或者除了length以外的属性。

arguments还在arguments.callee属性中包含了当前执行函数的一个引用。这可用于递归函数调用，这正是在本解决方案中使用这个属性的方式。

这个解决方案在$boxes jQuery中持续递增，在动画完成时，递归执行函数。这一动作持续到 的索引超出$boxes jQuery集合的长度($boxes.length)，这时jQuery集合是一个空数组，因此不执行回调，代码也随之结束运行。

7.6 确定元素目前是否处于动画中

7.6.1 问题

当动画正在进行时，应该在动画结束前阻止用户触发动画再次运行。

这种情况的例子之一是用户单击按钮触发某些动画的情况。动画可能用于显示某些信息。所举的例子是有意编造的，当用户单击按钮时，将来回摇晃方框。

如果用户连续单击按钮，就不应该连续在队列中加入动画，所以需要测试动画是否已经在运行中，如果已经运行，则忽略动画请求。

7.6.2 解决方案

在这个解决方案中，我打算包含一些调试信息，所以我包含了一个ID为debug的 元素，并将在这个元素上附加日志信息，帮助我们查看发生的情况。

将使用自定义jQuery选择器:animated测试动画是否运行：

```
$(document).ready(function () {
   var speed = 100;

   $('#animate').click(function () {
      $('.box')
         .filter(':not(:animated)')
         .animate({ marginLeft: -10 }, speed, function () {
            $('#debug').append('<p>Starting animation.<p>');
         })
         .animate({ marginLeft: 10 }, speed)
         .animate({ marginLeft: -10}, speed)
         .animate({ marginLeft: 10 }, speed)
         .animate({ marginLeft: -10}, speed)
         .animate({ marginLeft: 10 }, speed)
         .animate({ marginLeft: 0}, speed, function () {
            $('#debug').append('<p>Finished animation.</p>');
         }); //长方法链结束
      });
   });
```

7.6.3 讨论

在这个有意编造的例子中，多次调用animate方法使方框来回晃动（但是如果在实际中需要这个效果，使用跳跃效果的缓动函数更好！）。

这个动画在用户单击动画按钮时触发。

我已经包含了两个回调函数以显示动画何时开始和结束。注意，即使有多行代码，因为方法链的工作方式，实际上它就是从“$('.box')”开始到“}); //长方法链结束”为止的一行JavaScript代码。

下面的jQuery代码行从集合中过滤当前正在动画中的任何div.box元素，只在剩下的元素上运行后续的动画：

```
.filter(':not(:animated)')
```

因为在例子中有一个div.box元素，所以动画只在该元素还没有显示动画的时候才能运行。

7.7 停止和复位动画

7.7.1 问题

如果动画正在运行，就可能必须在中途停止它。常见的一个问题是在使用mouseover和mouseout事件触发动画显示和隐藏特定内容时发生的。

如果鼠标在触发区域进进出出，就会不断触发动画；例如，内容块不断地上下滑动，直到完成触发的次数。

解决的方法之一是使用:animated选择器过滤动画元素。但是，你可能希望在用户把鼠标移出触发区域时将元素淡出而不是完成动画。这可以用stop()方法解决。

7.7.2 解决方案

已经在div.box元素中添加了一个CSS，将不透明度设置为0。

不在用户单击按钮时触发特效，而是在光标悬停于按钮之上时运行动画。这只是为了说明没有stop()调用，动画就会失控：

```
$(document).ready(function () {
   $('#animate').hover(function () {
      $('.box').stop().fadeTo(200, 1);
   }, function () {
      $('.box').stop().fadeTo(200, 0);
   });
});
```

7.7.3 讨论

上述问题通常可以组合fadeIn()和fadeOut()来解决。但是，如果使用这些方法而不首先使用stop()，特效每当光标悬停于按钮之上时都会重复运行。

为了避免这种情况，在下一个动画加入队列之前插入stop()命令。这样做的最大好处是能够在中途停止动画，意味着如果元素的不透明度为0.5（在IE中为50），下一个动画的出发点就为0.5。

因为现在在不透明度变动的中途停止动画，所以也就意味着不能正常地使用fadeIn()和fadeOut()。我们必须明确地指出希望淡出的位置。所以，现在将使用fadeTo()，以持续时间和目标不透明度作为参数。

现在，当用户在按钮上来回移动光标时，动画不会重复，而是在流畅的过渡中淡入和淡出。

7.8 为特效使用自定义的缓动方法

7.8.1 问题

jQuery只自带了两个内置的缓动函数：swing和linear。默认的缓动函数是swing。如果我们想让动画更有趣一些，可以使用不同的缓动函数——它可以带来一个跳跃或者有灵活性的动画，也可能只是减缓结束进程的一个动画。可以手动添加缓动函数，也可以用jquery.easing插件包含一组预定义的缓动函数，从 *http://jquery-cookbook.com/go/easing/*可以下载这个插件。

7.8.2 解决方案

在包含jQuery程序库之后立刻包含jquery.easing.1.3.js，就可以使用31个新的缓动函数：

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').animate({ scrollTop: '+=100' },
         { duration: 400, easing: 'easeOutElastic' });
   });
});
```

7.8.3 讨论

通过包含缓动程序库，可以在options参数的easing属性中指定大范围的值。animate方法还支持第三个参数easing，所以前一个解决方案可以写成如下形式：

```
$('.box').animate({ scrollTop: '+=100' }, 400, 'easeOutElastic');
```

为了创建自己的自定义缓动函数，可以使用如下代码扩展easing：

```
jQuery.extend(jQuery.easing, {
   customEasing: function(x, t, b, c, d) {
      return c*(t/=d)*t + b;
   },
});
```

上述例子和easeInQuad缓动效果相同。所有缓动函数都有5个参数：

fraction

动画的当前位置，以0（动画开始处）～1（动画结束处）之间的时间表示。

elapsed

自动画开始以来的毫秒数（很少使用）。

attrStart

动画变动的CSS属性开始值。

attrDelta

动画变动的CSS属性开始和结束值之间的差别。

duration

动画持续的总毫秒数（很少使用）。

7.9 禁用所有特效

7.9.1 问题

你的用户或者Web应用可能需要禁用所有动画，但是仍然需要显示信息或者滚动（或者任何动画类型）的动画。

这可能是个人偏好，用户可能正在使用低分辨率的设备，或者可能是因为用户发现动画给他们的浏览造成了问题。

jQuery能够从一个访问点禁用所有动画，但是仍然支持animate方法和它的最终值。

7.9.2 解决方案

```
$.fx.off = true;

$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').animate({ width: '+=100', height: '+=100' });
   });
});
```

7.9.3 讨论

用下列代码行将fx设置为off，所有的动画调用将与直接调用css()等效：

```
$.fx.off = true;
```

上述代码行可以在任何时点设置，它将会禁用所有动画，这意味着它可以作为用户首选项提供。只要将该标志设置为false就能够再次启用动画：

```
$.fx.off = false;
```

7.10　将jQuery UI用于高级特效

7.10.1　问题

如果你想要创建更复杂的特效，当然可以使用animate方法。这种特效可能用于需要变动元素上整组CSS属性的应用程序，或者在对话框关闭时用一种特殊的方式使其消失——例如，从屏幕上爆炸后消失（见图7-2）。

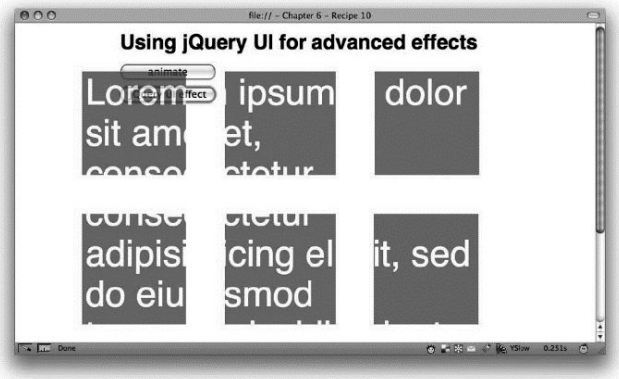


图7-2　运行于div.box元素的爆炸特效

7.10.2　解决方案

从*http://jquery-cookbook.com/go/jqueryui-download*下载jQuery UI程序库。该程序库可以在jQuery包含语句之后包含，以使用新的插件。

在这个解决方案中，添加一个额外的按钮显示两种特效，并且在CSS中添加一个新类。

1. CSS

```
.big {
   font-size: 400%;
   width: 500px;
   height: 500px;
   line-height: 100%;
}
```

2. jQuery

```
$(document).ready(function () {
   $('#animate').click(function () {
      $('.box').toggleClass('big', 2000);
   });

   $('#effect').click(function () {
      $('.box').effect('explode', null, 2000);
```

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

7.10.3 讨论

jQuery UI特效库还修改了addClass、removeClass和toggleClass的工作方式；特别是，可以将持续时间作为第二个参数，这将用新类的所有属性产生从当前状态到新类的动画过渡。

所以，第一个例子添加big类并将动画运行时间设置为2秒。来自big类的所有CSS属性都在div.box元素上产生动画效果。因为toggleClass方法在jQuery UI中已经作了修改，所以可以在当前状态和原始状态之间切换。

其次，将使用effect()方法，该函数是jQuery UI程序库的预设方法。此方法提供了一组显示和隐藏函数。

> **注意**
>
> effect()方法要求传递第二个变量——选项对象；这个对象可以为null或者空对象，但是必须提供它才能传递持续时间参数。

用字符串explode作为参数，div.box将分解为9块并淡出页面，如图7-2所示。

> **警告**
>
> 在本书编写期间，有一两种特效在Safari 4上有轻微的副作用。它们在其他所有A级浏览器中都能正常工作，请参阅Yahoo!的概述：*http://developer.yahoo.com/yui/articles/gbs/*。

可以访问*http://jquery-cookbook.com/go/jqueryui-effects*并尝试所有交互式演示，了解各种不同的特效。

第8章　事件

Ariel Flesler

8.0 导言

事件是用户和网站或者Web应用程序之间通信的主要方法。大部分JavaScript/jQuery编码都用来响应各种用户和浏览器事件。

这里所说的用户事件，基本是指键盘和鼠标交互，如click、mousedown、keypress等事件。浏览器事件指的主要是DOM事件，如document.ready、window.onload和许多与DOM元素相关的事件。

在Ajax应用程序编码中，还有自定义的jQuery Ajax事件，如ajaxSend、ajaxComplete、ajaxError等，这些事件在Ajax请求过程中调度。

jQuery的API一致性很好，尤其是对于事件。附加任何类型的事件处理程序的代码结构都相同：

```
jQuery( listener).bind( 'eventName', handlerFunction);
```

这一语法也适用于前面已经提到过的4类事件。jQuery的事件系统可以用于事件驱动编程。[1]可以创建和常规事件绑定与触发方式相同的自定义事件。

jQuery还为最常见的浏览器和Ajax事件提供了一个快捷方法。使用快捷方法的典型调用如下：

```
jQuery( listener).eventName( handlerFunction);
```

当使用bind()方法时，eventName是放在单引号或者双引号中的一个字符串，而在使用快捷方法时，可以将事件名称当作jQuery方法的名称。

下面是使用和不使用快捷方式绑定单击事件处理程序的例子：

```
// 使用bind()
 jQuery('div').bind('click',function(e){...});
 // 使用快捷方法
 jQuery('div').click(function(e){...});
```

在本章中，将在可以使用快捷方式的地方使用它，因为在我看来，快捷方式更简短易读。两者的效果相同，除了清晰和简洁之外，使用快捷方式没有其他的好处，使用哪一种方法只是爱好问题。

假定你已经阅读过第1章，该章已经详细地解释了document.ready事件（秘诀1.2）。如果你对该事件的用法有任何疑问，可以参考秘诀1.2。

我还要澄清一点，当使用插件这一术语时，大部分时候的意思是“插件、窗口组件或者就是代码块。”大部分jQuery用户倾向于将代码组织成类似插件的结构，通常在jQuery的命名空间中添加名称。

最后，jQuery的事件模块在jQuery 1.3中进行了很大的修改。我将根据使用的jQuery版本，提到代码中需要以不同方式处理的部分。

8.1 将一个事件处理程序用于许多事件

8.1.1 问题

在许多常见的情况下，开发人员需要为多个事件（在同一个元素上）绑定相同的处理程序函数。你可以这样做：

```
jQuery('div').click(function(e){
          alert('event');
        })
         .keydown(function(e){
           alert('event');
        });
```

如果函数很短，上述代码不会成为一个问题，但是对于较长的代码块，一次又一次地重复毫无意义，也绝对不是最佳方法。

8.1.2 解决方案

对这个简单而常见的问题，解决方案不止一种。

无需太多重复就解决这个问题的方法之一如下：

```
function handler(e){
     alert('event');
 }
jQuery('div').click(handler)
         .keydown(handler);
```

定义一个函数，然后多次引用它是一个不错的方法，但是jQuery还提供了一个更简单的方法。

bind()接受一个用空格分隔的事件列表。这意味着，可以这样解决上述问题：

```
jQuery('div').bind'(click keydown', function(e){
    alert('event');
});
```

8.1.3 讨论

还可以将这个行为应用到unbind()和one()。

要解除某个函数的绑定，需要得到对它的引用，所以即使你使用多事件功能，也仍然需要保存对处理程序的引用。如果没有将函数传递给unbind()，绑定到该事件的其他事件处理程序也会被删除：

```
function handler(e){
    alert('event');
}
jQuery('div').bind('click keydown', handler);
// ...
jQuery('div').unbind('click keydown', handler);
```

8.2 对不同的数据重用处理程序函数

8.2.1 问题

你遇到一种情况，其中有许多绑定，而处理程序函数看上去相当类似。这些绑定是否应用到不同的元素/事件组合并不重要。关键是，你不希望多次重复这些代码（谁会希望这样呢？）

下面是一个例子：

```
jQuery('#button1').click(function(e){
    jQuery('div.panel').hide();
    jQuery('#panel1').show();
    jQuery('#desc').text('You clicked the red button');
});
jQuery('#button2').click(function(e){
    jQuery('div.panel').hide();
    jQuery('#panel2').show();
    jQuery('#desc').text('You clicked the blue button');
});

jQuery('#button3').click(function(e){
    jQuery('div.panel').hide();
    jQuery('#panel3').show();
    jQuery('#desc').text('You clicked the green button');
});
```

正如你所看到的，每个处理器中唯一的不同是颜色和显示的面板。代码量随着按钮数量的增加，或者每次处理器函数的增大而变大。

8.2.2 解决方案

bind()接受一个可选的data参数，与每个特定的处理器函数绑定。数据值可以从这个函数中通过访问event.data[2]来存取，这里的event是jQuery提供的事件对象参数。

注意，这个值可以是任何类型：一个数组，一个字符串，一个数字或者一个对象。这是以文字方式传递对象的常见方法，即使只传递一个值，代码也会变得更加易读。这样，为对象中的这个属性所命名的名称将使代码的含义更明显。

8.2.3 讨论

event.data用于为函数提供预先计算的值，这意味着传递给bind()的值在绑定时必须已经可用。为了处理更加“动态”的值，可以使用秘诀8.5中将要学习的另一种方法。

前一个问题的解决方案如下：

```
function buttonClicked(e){
    jQuery('div.panel').hide();
    jQuery('#panel'+e.data.panel).show();
    jQuery('#desc').text('You clicked the '+e.data.color+' button');
}

jQuery('#button1').bind('click',{panel:1, color:'red'}, buttonClicked);
jQuery('#button2').bind('click',{panel:2, color:'blue'}, buttonClicked);
jQuery('#button3').bind('click',{panel:3, color:'green'}, buttonClicked);
```

当然，也可以使用循环使这段代码更短。这种方法被某些编码人员称作“宏”（Macro），对于jQuery代码来说很常见。

这些宏确实能够缩短代码长度，有时候能够改进代码的可读性。但在另一些情况下，它们也会使代码完全不可读，所以使用它们要小心。

下面是可以采用的做法：

```
jQuery.each(['red','blue','green'], function(num, color){
    num++; // it's 0-index based
    jQuery('#button'+num).bind('click',function(e){
        jQuery('div.panel').hide();
        jQuery('#panel'+num).show();
        jQuery('#desc').text('You clicked the '+color+' button');
    });
})
```

正如你所看到的，这里不使用数据参数，因为实际上没有必要。代码现在更加简短了，但是改进不大，而且它没有更好的可读性。

结论是，对这种情况来说，两种方法都可以使用。根据不同的问题，某种方法可能优于（更短、更好理解或者更容易维护）另一种方法。

8.3 删除整组事件处理程序

8.3.1 问题

你已经编写了一个类似插件的代码块，将许多事件处理程序绑定到某个DOM元素。

稍后，你打算清除所有的事件处理程序，以便完全卸载这个插件。

如果添加许多处理程序，这可能有些冗长。你甚至可能无法访问绑定的事件处理程序，因为它们属于另一个局部作用域。

你不能解除某个事件（或者任何现有的事件）的所有处理程序的绑定，因为你可能删除其他不在考虑之列的处理程序。

8.3.2 解决方案

为创建的每个插件使用单独的命名空间。在这个插件中绑定的任何处理程序必须添加到这个命名空间里。

以后在清理时，只需要“解除整个命名空间的绑定”，用一行代码就能清除所有相关的事件处理程序。

8.3.3 讨论

1. 如何绑定命名空间

为了在事件类型中添加命名空间，只要添加一个“.”，后面跟上命名空间名称就可以了。

从jQuery1.3开始，就可以为每个事件添加多个命名空间。

下面是为click和mousedown函数绑定命名空间的方法：

```
jQuery.fn.myPlugin = function(){
    return this
        .bind('click.myPlugin', function(){
            // [code]
        })
```

```
        .bind('mousedown.myPlugin', function(){
            // [code]
        });
};
```

2．如何清除插件

要解除上面的绑定，可以这样做：

```
jQuery.fn.disposeMyPlugin = function(){
    return this.unbind('.myPlugin');
};
```

8.4 触发特定事件处理程序

8.4.1 问题

你需要在某个元素（或者多个元素）上触发一个事件。这个元素属于一个或者多个插件，所以它可能有绑定到这个事件处理程序。

问题是，这个事件是公用的，如click或者mousedown。简单地触发事件可能运行其他在预期之外的事件处理程序。

8.4.2 解决方案

和前一个秘诀的原则相同，命名空间也可以用于触发事件。在绑定的时候，必须确保为每组事件处理程序添加唯一的命名空间。

这也可以用于相反的情况；如果需要触发除了命名空间里的事件之外的任何事件，可以使用!运算符。8.4.3节将展示一个示例。

8.4.3 讨论

1．如何用某个命名空间触发处理程序

现在，假定你想要以编程方式触发插件myPlugin中绑定的单击事件。你可以简单地触发单击事件，但是这是一个不好的方法，因为绑定到相同事件的所有其他处理程序也会启动。

下面是正确进行这一操作的方法：

```
jQuery.fn.runMyPlugin = function(){
    return this.trigger('click.myPlugin');
};
```

2．如何触发没有命名空间的处理程序

相反，你可能需要触发一个单击事件（或者任何其他事件），但是目标元素属于一个或者多个插件。触发事件可能运行不受欢迎的事件处理程序，并且可能导致难以调试的问题。

所以，假定所有插件都使用了命名空间，下面是安全触发一个单击事件的方法：

```
jQuery('div.panels').trigger('click!');
```

8.5 向事件处理程序传递动态数据

8.5.1 问题

你打算向事件处理程序传递某些数值，但是在“绑定时”这些数值未知，且在每次调用处理程序时都会变化。

8.5.2 解决方案

这一问题有两种解决方法：

- 向trigger()传递额外的参数；
- 向trigger()传递一个自定义事件对象。

这两种方法都能正常工作，也没有明显的好坏之分。第二种方法在jQuery 1.3之前比较难以使用。从这个版本起，该方法变得相当简单，问题也较少。8.5.3节将详细解释每种选择。

将数据传递给处理程序而不是让函数从某处获取数据（全局变量、jQuery命名空间等），使代码更容易维护，因为这样使处理程序函数更简单且与环境无关。

通过这样做，还能在许多情况下重用相同的处理程序。

8.5.3 讨论

1. 传递额外的参数

trigger()能够接受一个或者多个值，这些值将传递给所触发的处理程序。

这些值可以是任何类型，任意个数。当有超过一个参数值时，需要用数组包装它们：

```
jQuery('form').trigger('submit', ['John','Doe', 28, {gender:'M'}]);
```

上述情况中绑定的函数如下：

```
jQuery('form').bind('submit', function(e, name, surname, age, extra){
    //用这些参数进行某些处理
});
```

这种方法简单且易于理解。问题是当需要接受许多参数时它看上去很糟糕；我个人认为参数的个数不能超过4～5个。

如果读者习惯于常见的function(e){}函数，这种方法也容易造成误导。

你开始觉得奇怪，其他这些参数是从哪里来的？

在编程事件中使用的其他例子：

```
jQuery('#slideshow').bind('add-image', function(e, src){
    var $img = jQuery('<img />').attr('src', src);
    jQuery(this).append($img);
});
jQuery('#slideshow').trigger('add-image', 'img/dogs4.jpg');
```

用在实际事件的例子：

```
jQuery('#button').bind('click', function(e, submit){
    if( submit )
        // 进行某些处理
    else
        // 进行其他的处理
});
jQuery('#button').trigger('click', true);
```

2．传递自定义事件对象

如果选择传递自定义事件对象，所传递的每个值可以当作处理程序接受的事件对象的属性来访问。

这意味着，无论传递多少数据，处理程序都只有一个参数——事件对象。

这比起第一个方法来已经是个优势，因为它使函数声明不那么冗长。

前面已经提到过，这种方法在jQuery 1.3之后变得更好。下面是使用自定义对象的第一个例子：

```
jQuery('form').bind('submit', function(e){
    //用e.name、e.surname等进行某些操作
});
jQuery('form').trigger({
    type:'submit',
    name:'John',
    surname:'Doe',
    age: 28,
    gender:'M'
});
```

传递一个对象实际上是创建一个jQuery.Event实例的快捷方式。[3] 下面是另一种方法：

```
var e = jQuery.Event('submit'); // new运算符可以省略
e.name = 'John';
e.surname = 'Doe';
e.age = 28;
e.gender = 'M';
jQuery('form').trigger(e);
```

当然可以使用jQuery.extend代替一次设置一个属性的做法。

如果你计划在调用trigger()之后从这个对象中获取数据，就需要自行创建一个事件对象。顺便说一句，这是从处理程序向调用者传递信息的巧妙技术（下一章将介绍这方面的知识）。

3．这与event.data有何区别

使用event.data对于函数绑定时可以访问的静态值来说是有用的。当需要传递的数据必须在以后（或者每次）求值时，event.data就无能为力了。

8.6 尽早访问元素（在document.ready之前）

8.6.1 问题

你需要尽快获得对某个DOM元素的访问。

使用document.ready不够快；你实际上希望在页面渲染完成之前控制这个元素。

这样的问题在大型网页上特别引人注目，在这种网页上document.ready事件要花费更长的时间才能触发。

8.6.2 解决方案

这是一个很常见和普通的问题，可以用许多不同的方法解决。

有一种方法适用于所有DOM元素，但是需要轮询DOM，因此它给页面渲染过程增加了开销（这是我们绝对不希望的！）。

可以依赖轮询解决的常见问题有：

- 在元素显示（或者另一个样式操作）之前立刻隐藏它。
- 尽快为元素ASAP绑定一个事件处理程序，使其快速发挥作用。
- 任何其他情况。

8.6.3节将讨论每种情况下的更好方法。

8.6.3 讨论

1. 马上隐藏元素（或者其他样式操作）

你的问题和样式直接相关，你希望为元素应用一个条件样式，这个条件必须由JavaScript求值。

正确的方法是为很快可以访问的元素（如<html>元素）添加一个特定的CSS类，然后相应地设置元素的样式。

代码如下：

```
<!DOCTYPE html>
<html>
<head>
 <style type="text/css">
    html.no-message #message{ display:none; }
</style>
<script src="assets/jquery-latest.js"></script>
<script type="text/javascript">
    //不好的做法
    jQuery(document).ready(function($){
        $('#message').hide();
    });
     // 正确的做法
     jQuery('html').addClass('no-message');
     //也可以这么做
     document.documentElement.className = 'no-message';
 </script>
</head>
<body>
    <p id="message">I should not be visible</p>
    <!--
      Many more html elements
    -->
</body>
</html>
```

2．尽快为元素ASAP绑定一个事件处理程序

我们很经常会有一个大型的页面，上面有按钮和链接之类的交互式元素。

你不希望这些元素在页面加载的时候只是挂在那里，而没有任何附加的功能。

幸运的是，“事件委托”这一杰出概念能够解决这个问题。事件委托可以用多个jQuery插件轻松实现，而且从jQuery 1.3版本开始，不再需要插件，因为这个功能已经添加到核心jQuery文件中。

现在，可以使用`live()`方法为还不存在的元素绑定实践处理器[4]。这样，不需要等到元素就绪之后才能添加事件。

事件委托的更多知识参阅秘诀8.10。

3．任何其他情况

你的问题与样式或者事件无关。那么我的朋友，你遇到了最糟糕的情况。

但是不要惊慌！如果你关心性能，有比轮询更好的解决方案。我将在最后加以说明。

轮询。轮询可以通过在一个简单的时间间隔（setInterval）内检查元素来实现，一旦发现元素，就运行某个函数并清除时间间隔。

有两个插件可以帮助你完成轮询。其中之一是LiveQuery，[5]它有一个选项，能够注册一个对匹配选择器的新发现元素运行的函数。这种方法相当慢，但是支持整个选择器集。

另一个插件名为ElementReady#，[6]该插件也能正常地处理这种情况。

ElementReady#让你注册一对id/function，然后它轮询DOM。一旦发现id，就调用该function，并从队列中删除id。

这个插件实现的可能是检测元素的最快方法，也就是使用document.getElementById，这个插件运行速度相当快，但是只支持id。

自定义位置的脚本。文档就绪的完整概念是“HTML解析之后”。这意味着浏览器已经到达文档正文的结束标记</body>。

换句话说，可以将脚本放在</body>之后，代替使用document.ready。

你可以对DOM的其他部分应用相同的原则：你可以在想要访问的元素后面添加一个<script>标记，无疑，这样你就能知道已经可以访问该元素。

下面是一个例子：

```
<!DOCTYPE html>
<html>
<head>
 <script src="assets/jquery-latest.js"></script>
</head>
<body>
    <p>The time is <span id="time"> </span></p>
```

```
    <script type="text/javascript">
    jQuery('#time').text( new Date().toString() );
    </script>
    <!-- Many more html elements -->
</body>
</html>
```

正如你所看到的，这种情况下不需要轮询。如果你不需要过多使用或者打算给页面添加很多脚本，这是个可行的解决方案。

8.7 停止处理程序执行循环

8.7.1 问题

你在同一个元素/事件组合中绑定了多个处理程序。

你打算从一个处理程序中阻止其余处理程序调用，就像`event.stopPropagation()`[7]所做的那样。问题是，`event.stopPropagation()`只能用于DOM层次结构中低于当前元素的元素。

8.7.2 解决方案

从jQuery 1.3起，传递给处理程序的事件对象有一个新方法`stopImmediatePropagation()`[8]。这个方法可以完成上述的功能，当前事件不会再通知任何后续的事件处理程序。它也和`stopPropagation()`一样阻止事件的传播。

这个方法取自ECMAScript的DOM level 3事件规范。[9]

如果你查询事件对象以了解这个方法是否调用，可以调用`event.isImmediatePropagationStopped()`，[10]该方法将返回`true`或者`false`。

8.7.3 讨论

1. 例子

简单表单验证。`stopImmediatePropagation()`在某种条件不符合的情况下可以撤销实际的提交绑定：

```
jQuery('form')
    .submit(function(e){
        e.preventDefault(); // Don't submit for real
        if( jQuery('#field').val() == '' )
            e.stopImmediatePropagation();
    })
    .submit(function(e){
```

```
        //只在上面的函数不调用e.stopImmediatePropagation时执行
    });
```

终止所有事件。stopImmediatePropagation()还可以用于暂时禁用元素或者阻塞容器：

```
(function($){

function checkEnabled(e){
    if( jQuery(this).is('.disabled') ){
        e.stopImmediatePropagation(); // Stop all handlers
        e.preventDefault();
    }
};

jQuery.fn.buttonize = function(){
    return this.css('cursor','pointer')
               .bind('click mousedown mouseup',checkEnabled};
};
})(jQuery);
```

2．这种方法的缺点

虽然这一新功能在某些情况下可以救急，但是你必须知道，以这种行为为基础来构建逻辑并不总是安全的。当依赖这种功能时，假定处理程序以预期的方式执行，没有其他处理程序会妨碍这种执行。

虽然jQuery绑定的事件运行顺序与它们添加的顺序一样，但是API对此并没有强大的支持，这意味着在某些浏览器或者特殊的情况下执行可能失败。来自不同插件的绑定也可能发生冲突，因为某个插件可能调用stopImmediatePropagation()，其他插件中绑定的方法就无法执行。这可能导致需要花费很长时间调试的意外问题。

结论是，如果stopImmediatePropagation()确实适用于你的问题，不要担心使用它，但是使用它必须谨慎，并且要符合所有涉及的事件处理程序。

在如下情况下，你应该再三考虑是否使用这种方法：

- 监听器是一个已经被其他插件使用的“流行”DOM元素。
- 事件是click或者ready之类的常见事件。冲突的可能性更大。

另一方面，在如下情况下使用stopImmediatePropagation()相当安全：

- 监听器是动态创建的DOM元素，很少被插件使用。
- 事件是`change-color`或者`addUser`等自定义事件。
- 你希望有意地停止任何绑定的处理程序（和第2个例子中类似）。

8.8 在使用event.target时获取正确的元素

8.8.1 问题

你的代码依赖事件对象的event.target属性，[11]该属性很可能与事件委托组合使用，在这种情况下，只有一个事件处理程序绑定到容器，管理可变数量的后代元素。

在某些情况下，你似乎得不到预期的行为。event.target有时候指向预期元素内的一个元素。

8.8.2 解决方案

event.target属性引用获取事件的元素，也就是具体的元素。

这意味着，如果在一个链接中有张图片而你单击了该链接，event.target将是那张图片而不是链接。

那么，你怎么解决这个问题呢？如果你不打算使用事件委托，那么使用this（函数作用域）或者event.currentTarget（从jQuery 1.3起）应该可以解决。这两个属性总是指向绑定了事件处理程序的元素。如果你确实使用事件委托，那么需要找出预期的双亲元素。

从jQuery 1.3起，可以使用closest()。[12]正如文档中所指出的那样，它返回最靠近的元素，从当前位置一直上溯到双亲，后者匹配某个选择器。

如果你使用的是旧版本的jQuery，可以用如下代码模拟closest()：

```
jQuery.fn.closest = function( selector ){
    return this.map(function(){
        var $parents = jQuery(this).parents();
        return jQuery(this).add($parents).filter( selector )[0];
    });
}
```

对此可以做一些性能上的改进，但是这个简单的版本能够用于一般的用途。

下面是使用closest()最常见情况的一个例子：

```
jQuery('table').click(function(e){
    var $tr = jQuery(e.target).closest('tr');
    // 对表格行作某些操作
});
```

8.8.3 讨论

event.target是jQuery事件系统规范化的事件对象属性之一（在IE上是evet.srcElement）。

那么，为什么在目标元素上触发事件时，即使事件处理程序是绑定到祖先元素上的也会被调用前者呢？答案是：因为有事件冒泡（Event bubbling）。[13]

大部分标准的DOM事件都有冒泡的过程。[14]这意味着，在目标上触发事件之后，事件会上溯到双亲节点，触发同样的事件（以及所有的处理程序）。

这一过程将会持续，直到事件到达document，或者在事件处理程序中调用event.stopPropagation()为止。

因为有了事件冒泡，你不需要总是将事件处理程序绑定到具体的元素，而可以绑定到公共的容器，从那里处理所有元素，这就是事件委托的原理。

8.9 避免多个hover()动画并行显示

8.9.1 问题

我们都至少一次碰到这种问题。你建立了如下代码：

```
jQuery('#something').hover(
     function(){
          //添加一些巧妙的动画
     },
     function(){
          //将动画恢复到初始状态
     }
);
```

例如，你可能在光标每次移过元素时增大它，然后在光标离开元素时将其缩小为原来的大小。

开始一切都很正常，可是当你快速地反复将光标移进移出元素时……这是怎么了？！

jQuery('#something')元素突然多次来回改变大小，直到最后停止。

8.9.2 解决方案

解决方案实际上很简单，太简单了，但是这种问题经常发生，所以我认为这个解决方案很实用。

为了避免这种令人厌恶的效果，你所需要的就是在创建新的动画之前终止该元素上所有现存的动画。

为此，应该使用jQuery的stop()方法。它（正如名称所表示的）停止当前动画，并且也可以选择删除后续的动画。

8.9.3 讨论

1. 示例

我将展示一个变动CSS属性opacity的动画示例，但是它对任何其他属性的做法也一样：

```
jQuery('#something').hover(
     function(){
          jQuery(this).stop().animate({opacity:1}, 1000);
     },
     function(){
          jQuery(this).stop().animate({opacity:0.8}, 1000);
     }
);
```

上述代码中的方法对自定义jQuery动画如slideUp()、slideDown()、fadeIn()等也有效。

这是使用淡入/淡出方法的例子：

```
jQuery('#something').hover(
     function(){
          jQuery(this).stop().fadeTo( 1, 1000 );
     },
     function(){
          jQuery(this).stop().fadeTo( 0.8, 1000 );
     }
);
```

2．还没结束

在这样的情况下还可能发生另一个相关问题：

```
jQuery('#something').hover(
     function(){
          jQuery(this).stop()
               .fadeTo( 1, 1000 )
               .animate( {height:500}, 1000 );
     },
     function(){
          jQuery(this).stop()
               .fadeTo( 0.8, 1000 )
               .animate( {height:200}, 1000 );
     }
);
```

如果你尝试这段代码，快速移动鼠标，元素将再次疯狂地变化，但是只有高度（而不是透明度）发生变化。

原因很简单：jQuery动画默认情况下要进行排队。这意味着如果添加多个动画，它们会按照顺序执行。

在默认情况下，stop()只停止（和删除）当前动画。在上一个例子中，每次都只删除不透明度动画，而将高度动画留在队列中运行。

为了解决这个问题，必须多次调用stop()或者传递true作为第一个参数。这会使stop()清除所有排队的动画。鼠标悬停代码应该是这样的：

```
jQuery('#something').hover(
    function(){
        jQuery(this).stop(true)
            .fadeTo( 1, 1000 )
            .animate( {height:500}, 1000 );
    },
    function(){
        jQuery(this).stop(true)
            .fadeTo( 0.8, 1000 )
            .animate( {height:200}, 1000 );
    }
);
```

8.10 使事件处理程序适用于新添加的元素

8.10.1 问题

你已经绑定了一个或者多个事件处理程序，但它们突然停止工作。

这发生在Ajax请求或者简单的jQuery操作（append()、wrap()等）动态添加新元素之后。

> **注意**
>
> 这个问题很常见，我们都至少碰到过一次。
>
> 8.10.3节介绍幕后的原理。如果你觉得需要在研究解决方案之前对此有深入的理解，可以先看看8.10.3节中的“为什么事件处理程序不见了”。

8.10.2 解决方案

这个反复发生的问题有两种可能的解决方案，各有优劣：

重新绑定

这种方法要求在每次添加新元素时反复调用bind()。

重新绑定很容易实现，不需要任何插件或者新方法。可以简单地将所有绑定放在一个函数中，在每次更新之后再次调用。

事件委托

这种方法依赖事件冒泡，[15]它很快而且是轻量级的，但是需要一定的理解力，有时候可能会有些麻烦。

从jQuery 1.3起，对事件委托有了内置支持。使用的方法很简单，就是用新的live()方法代替bind()。

8.10.3 讨论

1. 为什么事件处理程序不见了

和CSS相反，JavaScript不是一种声明性的语言。不用描述各种行为，它们都会“像魔术一样自动”应用。

JavaScript和许多其他语言一样是命令式语言。开发人员指定需要执行的一系列操作，这些操作在代码执行到该行的时候应用。

当添加如下这段代码时：

```
function handler(){
    alert('got clicked');
}
jQuery('.clickable').bind('click', handler);
```

下面就是所要做的：

1．查找所有CSS类为“clickable”的元素，将其保存在集合里。

2．将“处理程序”函数绑定到集合中每个元素的单击事件。

如果JavaScript/jQuery进行声明性的解释，上述代码的意思如下：

每当单击CSS类为clickable的元素，就运行handler函数。

但是，因为JavaScript/jQuery的解释是命令式的，只有在运行时匹配选择器的元素才会绑定。如果用clickable类添加新元素或者从元素上删除该类，不会添加或者删除这些元素的行为。

2．事件委托简介

事件委托就是在开始时绑定一次，然后被动地监听事件的触发。它依赖浏览器中的许多事件都会冒泡这一事实。

举一个例子，在单击一个 之后，它的双亲节点也会接收单击事件，然后传递给双亲的双亲，依此类推直到它到达document元素。

3．每种方法的优点和缺点

重新绑定。重新绑定很简单：你只是重新添加事件处理程序。它会导致新的问题，例如，为已经绑定事件处理程序的元素添加事件处理程序。有些人添加CSS类来解决这一问题（用某个类标记已经绑定的元素）。

所有这些处理在每次元素更新的时候都需要CPU周期，而且需要创建越来越多的事件处理程序。

同时解决这两个问题的方法之一是使用命名函数作为事件处理程序。如果你始终使用同一个函数，就已经解决了重复的问题，开销也较小。

但是，重新绑定仍然可能随着时间的推移导致越来越高的内存消耗。

事件委托。事件委托只要求在开始的时候绑定，完全没有必要处理重新绑定。这对于开发人员来说相当轻松，而且使代码更加简短，也更加清晰。前面提到的RAM问题不适用于事件委托。页面的内容可能变化，但是活动的事件处理程序总是相同。

但是，事件委托也有需要小心的地方。为了使其正常工作，处理代码（`live()`、插件或者你自己的代码）必须取得发生事件的元素（`event.target`）并遍历它的祖先元素，查看哪一个元素有可以触发的事件处理程序，此外还有一些其他的处理。这意味着，虽然事件委托需要的绑定较少，但是每次触发事件时需要更多的处理。

而且，事件委托无法用于不进行冒泡的事件，例如，`focus`和`blur`。对于这些事件来说，有一个跨浏览器的替代方法，就是在某些浏览器中使用`focusin`和`focusout`。

4．结论

事件委托似乎是更好的方法，但是需要额外的处理。对此，我建议只在真正需要的时候使用实时绑定。有两种常见的情况需要事件委托：

动态元素

有一系列动态变化的DOM元素。

大的列表

当用实时绑定只需要一次绑定，而不是常规方法中的100次时，事件委托运行速度更快。这种方法在开始时更快，占用的内存也较少。

如果没有理由使用`live()`，那么就使用`bind()`。如果以后需要改为实时方式，切换也就是几秒钟的事情。

---

[1]*http://en.wikipedia.org/wiki/Event-driven_programming*。

[2]*http://docs.jquery.com/Events/jQuery.Event#event.data*。

[3]*http://docs.jquery.com/Events/jQuery.Event*。

[4]*http://docs.jquery.com/Events/live*。

[5]*http://plugins.jquery.com/project/LiveQuery*。

[6]*http://plugins.jquery.com/project/ElementReady*。

[7]*http://docs.jquery.com/Events/jQuery.Event#event.stopPropagation.28.29*。

[8]*http://docs.jquery.com/Events/jQuery.Event#event.stopImmediatePropagation.28.29*。

[9]*http://www.w3.org/TR/DOM-Level-3-Events/*。

[10]*http://docs.jquery.com/Events/jQuery.Event#event.isImmediatePropagationStopped.28.29*。

[11]*http://docs.jquery.com/Events/jQuery.Event#event.target*。

[12]*http://docs.jquery.com/Traversing/closest*。

[13]*http://www.quirksmode.org/js/events_order.html*。

[14]*http://en.wikipedia.org/wiki/DOM_events#Common.2FW3C_events*。

[15]*http://www.quirksmode.org/js/events_order.html*。

第9章　高级事件

Ariel Flesler

9.0 导言

本章中的秘诀将处理边界条件问题、高级优化和使代码更酷的某些技巧。这些秘诀大部分是为希望进一步优化jQuery代码的高级开发人员准备的。

正如第8章中一样，我将把代码称作插件，但是并不意味着它必须是一个真正的插件。如果你不想将代码组织为jQuery插件，就要记住我的命名约定。

9.1　在动态加载时运行jQuery

9.1.1　问题

你打算在DOM中添加<script>元素或者用Ajax等其他方式，在页面中动态包含jQuery。

一旦加载jQuery，你认为所有的代码都会开始工作，但是因为某些原因，脚本没有启动。

9.1.2　解决方案

你需要包含一个在jQuery加载之后运行的附加脚本。这个脚本简单地调用jQuery.ready()。这样做以后，一切都会按照预期开始工作。

9.1.3　讨论

1. jQuery.ready()是什么

jQuery.ready()函数在检测到文档就绪时由jQuery的核心调用。一旦调用，自动触发所有document.ready处理程序。

> **注意**
>
> 你不需要担心这个函数是否已经调用（例如，由原始检测调用），从而再次触发所有的document.ready处理程序。
>
> jQuery.ready()内部包含对重复执行的检查。重复的调用将被忽略。

2. 为什么会发生这种情况

document.ready检测主要基于事件完成。根据不同的浏览器绑定某个事件，在文档就绪时这个事件应该触发。

此外，对所有浏览器都绑定window.onload，作为其他选项失败时候的预防措施。

发生的一切就是jQuery只在window.onload事件之后才最终加载到页面中；因此，所有事件处理程序都已经绑定，但是没有一个触发。

通过调用jQuery.ready()，你手动“宣布”document.ready事件，触发所有处理程序并使一切回归正常。

9.2 加速全局事件触发

9.2.1 问题

全局事件触发涉及在所有可用元素上调用某个事件绑定的所有事件处理程序。

调用jQuery.trigger()，不传递任何作为上下文的DOM元素就可以执行这一任务。

这几乎等于在所有绑定了一个或者多个对应事件的元素上调用trigger()，如：

```
jQuery('#a1,#a2,div.b5').trigger('someEvent');
```

全局触发明显更简单，因为你不需要知道需要触发的所有元素。

对于某些情况来说这很实用，但是有时候仍然是一个缓慢的过程。尽管从jQuery 1.3开始，全局触发已经得到优化，它仍然必须遍历注册到jQuery事件系统的所有元素。这可能导致在这样的事件触发时有短暂（或者不那么短）的停顿。

9.2.2 解决方案

可能的解决方案之一是让一个或者多个全局对象担当事件处理程序的角色。这些元素可以是DOM元素，也可以不是。所有全局事件将在这些元素中的一个上绑定和触发。

你不用这么做：

```
jQuery('#text1').bind('change-page', function(e, title){
    jQuery(this).text( 'Page is ' + title );
});
jQuery('#text2').bind('change-page', function(e, title){
    jQuery(this).text( 'At ' + title + ' Page' );
});
jQuery.trigger('change-page', 'Inbox');
```

而代之以：

```
jQuery.page = jQuery({}); //只是一个空对象
jQuery.page.bind('change', function(e, title){
    jQuery('#text1').text( 'Page is ' + title );
});
jQuery.page.bind('change', function(e, title){
    jQuery('#text2').text( 'At ' + title + ' Page' );
});
jQuery.page.trigger('change', 'Inbox');
```

语法上似乎很相似，但是对trigger的每次调用不需要列举jQuery的数据注册表（也就是jQuery.cache）。

即使你决定使用一个DOM元素，原理也相同。DOM元素有时候更合适。例如，如果要创建一个与表格相关的插件，那么将每个<table>元素作为事件处理程序就很有意义。

DOM元素的问题是在许多浏览器中它们是内存泄漏的主要根源。内存泄漏发生在用户离开页面而JavaScript引擎无法释放某些RAM内存时。

当使用DOM元素时，应该注意在这些对象中保存数据的方式。这也就是jQuery提供date()方法的原因。

在大部分情况下，我个人仍然使用常规的JavaScript对象。可以为它们添加属性和函数，而且内存泄漏的可能性（和严重性）比较小。

9.2.3 讨论

1. 优点和缺点

本秘诀的标题已经指出，这种方法更快。你将始终在一个对象上而不是在jQuery.cache的多个条目上触发事件。

这种方法的缺点是每个人都需要知道事件处理程序对象（例子中的jQuery.page），才能绑定或者触发已知的事件。

如果你的目标是封装代码，这可能是个负面因素。[1]

封装的概念在面向对象编程中是高度强迫性的，因此你对这个因素要格外小心。

封装与jQuery编程的相关度总体来说没有那么高，因为jQuery不是面向对象的，大部分用户不太关注代码封装。但是这仍然值得注意。

2. 使监听器具备实际功能

前面提到的监听器对象并不一定是除了bind()、unbind()和trigger()之外什么都没有的简单“哑”对象（据我们所知）。

这些对象实际上可以拥有方法和属性，使其更加实用。

唯一的问题是，如果有如下代码：

```
jQuery.page = jQuery({ number:1 });
```

要访问number属性，就必须这么做：

```
jQuery.page.number; // undefined
 jQuery.page[0].number; // 1
```

这是jQuery对HTML节点和其他变量的处理方式。

但是不要对我失去信心！这个问题很容易解决。编写一个小插件：

```
(function( $ ){

    // 这些方法将从jQuery.fn复制到原型
    var copiedMethods = 'bind unbind one trigger triggerHandler'.split(' ');

    // 空的构造器
    function Listener(){
    };

    $.each(copiedMethods, function(i,name){
        Listener.prototype[name] = $.fn[name];
    });

    // "jQuery.fn.each必须更换
    Listener.prototype.each = function(fn) {
        fn.call(this);
        return this;
    };

    $.listener = function( data ){
        return $.extend(new Listener(), data);
    };

})( jQuery );
```

现在，可以创建具有所需的所有事件相关jQuery方法的对象，但是传递给bind()、unbind()等的函数作用域将是对象本身（例子中的jQuery.page）。

注意，监听器对象并不拥有所有jQuery方法，而只有复制的那些方法。虽然可以添加更多方法，但是它们大部分都无法正常工作。那样做需要更复杂的实现；我们将坚持使用例子中的这些方法，它们能够满足我们对事件的需求。

既然有了这个小型的插件，就能：

```
jQuery.page = jQuery.listener({
    title: 'Start',
    changeTo: function( title ){
        this.title = title;
        this.trigger('change');
    }
});
jQuery.page.changeTo('Inbox');
```

因为现在你能够从处理程序中使用this访问对象，所以没有必要向处理程序传递标题等参数值。相反，可以简单地使用this.title访问该值：

```
jQuery.page.bind('change', function(e){
    jQuery('#text1').text( 'Page is ' + this.title );
});
```

9.3 创建自己的事件

9.3.1 问题

你想在一个元素与事件绑定时为其提供某些行为。

9.3.2 解决方案

使用jQuery.event.special来完成这一工作。这个功能需要一个对象，该对象至少有一个每次为每个元素绑定事件时都要调用的函数，另外还有一个函数用于清理开始时所做的操作。

jQuery.event.special的语法如下：

```
jQuery.event.special.myEvent = {
    // 绑定自定义事件
    setup:function( data, namespaces ){
        this; // 绑定的元素
        // 返回false获得原生绑定,否则将跳过它
    },
    // 清理
    teardown:function( namespaces ){
        this; // 绑定的元素
        // 返回值的意义同上
    }
};
```

在添加事件行为之后，可以这样做：

```
jQuery('#some_element').bind('myEvent', {foo:'bar'}, function(){...});
```

执行上述代码后，将调用setup()函数。

9.3.3 讨论

1．处理事件的所有绑定

前面已经说明，setup()函数指在添加第一个处理程序时调用。

如果在这个事件上封装的逻辑不需要在每次执行新绑定时进行某些操作，这样就已经足够了。

jQuery提供了这种选项，但是从jQuery 1.3.3开始该方法已经有了变化。

如果使用的是旧的版本，那么需要使用jQuery.event.specialAll代替jQuery.event.special。jQuery.event.specialAll将接受同类对象，回调也将接受相同的参数。唯一的区别是返回false不会带来任何改变。

在jQuery 1.3.3中删除了jQuery.event.specialAll。为了拦截对事件的所有绑定，需要在jQuery.event.special命名空间中包含一个add()（以及可选的remove()）函数。这些函数的参数是将要绑定的处理程序，可以选择返回一个替代的新处理程序函数。

2．一个真实的示例

编写一个简单的示例，确保对前面的说明有清晰的理解。将使用jQuery 1.3.3+的方法。

假定我们想要在一个元素被选中（单击）且不在禁用状态的时候触发一个事件。我们将在元素具有CSS类disabled时断定它被禁用。

下面是具体的做法：

```
// 保存这些变量使代码更简短,不要在全局作用域中这么做
var event = jQuery.event;
var $selected = event.special.selected = {
    setup:function( data ){
        event.add(this, 'click', $selected.handler);
        return false;
    },
    teardown:function(){
        event.remove(this, 'click', $selected.handler);
        return false;
    },
    handler:function(){
        var $elem = jQuery(this);
        if( !$elem.hasClass('disabled') )
            $elem.triggerHandler('selected');
    }
};
```

正如你所看到的，提供了自己的selected事件处理程序。在该处理程序中，使用triggerHandler()代替trigger()，因为不需要事件冒泡，也不需要阻止默认操作，所以省略了一些不必要的处理。

3．这种功能的现有用例

jQuery.event.special是添加新的行为而又不用污染jQuery命名空间的一个极佳方法。

jQuery.event.special并不是在任何时候都合适的，但是当需要基于另一个事件（在例子中是click）的自定义事件时，它通常很方便。如果有一个简单地绑定或者模拟事件的插件，这种方法也很有用，可以将该插件当成常规事件“掩盖”起来。

jQuery核心使用jQuery.event.special处理与document.ready事件绑定的事件。实际上，它们被当作常规的事件处理程序存储，但是在第一次绑定该事件时，实际上激活的是检测的代码。

jQuery.event.special还用于透明地处理mouseenter/mouseleave事件（用于hover()）。实现这一功能的所有DOM遍历操作都很好地隐藏在setup()处理程序中。

有一些插件也利用了jQuery.event.special。下面列出这类的插件中的几个：

mousewheel

提供对鼠标滚轮变化的支持。[2]

drag、drop

将拖放支持伪装成简单的事件。[3]

focusin, focusout

这个代码段（不是真正的插件）原来是Jörn Zaefferer编写的，后来通过插件添加，实现focus和blur事件的事件委托。

如果你计划为jQuery添加新的事件，查看这些插件是个好的开始。

9.4 让事件处理程序提供需要的数据

9.4.1 问题

你需要协调其他插件（或者只是简单的jQuery代码）并在执行请求操作之前修改某些变量。

9.4.2 解决方案

使用事件通知其他脚本操作即将执行该操作。

可以获取运行后的事件处理程序中收集而来的数据。

如果没有提供任何数据，可以使用自己选择的某些默认选项。你将看到如何根据使用的jQuery版本完成这一工作。

9.4.3 讨论

1. 在jQuery 1.3+中怎么做

从jQuery 1.3起，由于增加了jQuery.Event，上述任务可以用更漂亮的方式完成。旧的实现方式仍然可用于triggerHandler()，但是对jQuery.trigger()不起作用。

对于现在将要看到的代码，需要创建一个jQuery.Event：

```
var e = jQuery.Event('updateName');
```

现在，为了调用处理程序和获取数值，将把事件对象传递给trigger()，然后从事件对象提取数据：

```
jQuery('#element').trigger(e);
 alert( e.result ); // Charles
```

正如一开始时所说的那样，在绑定了许多处理程序时这种方法不能很好地工作，总的来说，这种方法有点不可靠和脆弱。

那么，如何进行事件处理程序和触发事件的函数之间的通信呢？

答案是，通过传递的事件对象。

传递给trigger()的jQuery.Event对象与每个处理程序接收的第一个参数相同。

这就意味着可以这么做：

```
jQuery('#name').bind('updateName', function(e){
    e.name = this.value;
});

var e = jQuery.Event('updateName');
jQuery('#name').trigger(e);
alert( e.name );
```

上述例子与简单地访问e.result没有太大的不同，但是对于在同一个事件对象上操作的多个事件处理程序又是如何呢？

```
jQuery('#first').bind('update', function(e){
    e.firstName = this.value;
});
jQuery('#last').bind('update', function(e){
    e.lastName = this.value;
});

var e = jQuery.Event('update');
jQuery('#first, #last').trigger(e);
alert( e.firstName );
alert( e.lastName );
```

现在有了让任意数量的事件处理程序为函数提供运行所需数据的一种手段。不言而喻，可以多次调用trigger()，传递相同的事件对象。

前面已经说过，用默认值（如果恰当的话）预设事件对象是明智的。代码不应该依赖于其他代码对某个事件的订阅。

如果没有默认值可用，那么总是可以中断调用或者抛出一个错误。

2．在jQuery 1.3中如何实现

旧版本的jQuery只允许用户获得一个值，该值可以在调用jQuery.trigger()或者triggerHandler()时返回。

具体代码如下：

```
jQuery('#element').bind('updateName', function(){
    return 'Charles';
```

```
});

var name = jQuery('#element').triggerHandler('updateName');
alert( name ); // Charles
```

在返回数据的事件处理程序不超过一个时，这种方法都没问题。在遇到多个事件处理器返回数据的情况时，按照“谁最后到”决定返回哪个处理器的数据。

3．允许事件处理程序阻止操作

这是我们刚刚看到的情况的一种特例。按照设计，事件对象有一个preventDefault()方法，该方法用在原生事件上，终止常用的操作（如链接上的单击），但是在自定义事件上没有实际的用途。

可以利用这个方法，允许其他脚本阻止将要执行的操作。

现在将展示一个操作的示例。虽然使用秘诀9.2介绍的小插件，但是毫无疑问，并不一定要使用它：

```
var remote = jQuery.listener({
    request:function( url, callback ){
        jQuery.ajax({ url:url, success:callback });
    }
});

// 发出一个请求
remote.request('contact.html', function( html ){
    alert( html );
});
```

现在，假定我们想允许外部脚本在必要时终止某些请求。需要这样修改remote.request：

```
var remote = jQuery.listener({
    request:function( url, callback ){
        var e = jQuery.Event('beforeRequest');
        e.url = url;
        this.trigger(e);

        if( !e.isDefaultPrevented() )
            jQuery.ajax({ url:url, success:callback });
    }
});
```

e.isDefaultPrevented()的返回值表示在这个对象上是否调用过e.preventDefault()。

任何外部脚本现在都可以这么做：

```
remote.bind('beforeRequest', function(e){
    if( e.url == 'contact.html' )
        e.preventDefault();
});
```

返回false（在函数中）的效果和e.preventDefault()几乎相同，将停止事件传播，这正是我们预期的。

毫无疑问，在这种情况下，可以使用前面学到的知识，允许处理程序修改URL（或者发送添加的数据）。

9.5 创建事件驱动插件

9.5.1 问题

你希望插件可以从外部控制。人们可以在任何时候“告诉”插件作某些处理。插件可以有许多实例（调用），但是操作应该只在提供的上下文中执行。

9.5.2 解决方案

方法之一是使用事件。

当调用插件时，它在每个匹配的元素上绑定函数，一经触发，这些函数将执行所需的操作。

还可以采用一种次要的方法，每当调用插件时，不考虑方法链，而是返回包含绑定的对象，允许外部操纵（使用来自秘诀9.2的插件）。

这样，可以在相同元素上多次调用插件而不会搞乱事件。

9.5.3 讨论

1. 一个示例

现在，将创建一个简单的幻灯片插件。我将在自己的一个现有插件jQuery.SerialScroll[4]基础上建立这个插件。我在编写这个插件时第一次想到这种方法，老实说，它工作得不错。

将自己的插件命名为slideshow。它接收一个<img>元素和一个URL数组，然后循环显示这些图片。能够移动到上一张/下一张图片，跳转到某张图片，也可以自动循环显示。

从基础的功能开始：

```
(function( $ ){
    $.fn.slideshow = function(options){

        return this.each(function(){
            var $img = $(this),
                current = 0;

            // 添加幻灯片演示行为...
```

```
        });
    };
})( jQuery );
```

现在，将添加几个局部函数，使我们能转移到集合中的不同图片（URL）：

```
function show( index ){
    var total = options.images.length;

    while( index < 0 )
        index += total;

    while( index >= total )
        index-= total;

    current = index;
    $img.attr('src', options.images[index]);
}
function prev(){
    show( current-1 );
}

function next(){
    show( current + 1 );
}
```

可以用事件提供这些功能：

```
$img.bind('prev', prev).bind('next', next).bind('goto',function(e, index){
    show( index );
});
```

自动循环怎么做？再添加几个函数：

```
var auto = false, id;
function start(){
    stop();
    auto = true;
    id = setTimeout(next, options.interval || 2000);
}

function stop(){
    auto = false;
    clearTimeout(id);
}
```

下面是事件：

```
$img.bind('start', start).bind('stop', stop);
```

现在需要在show()中添加几行，在必要时保持自动循环：

```
function show( index ){
    //和以前一样…

    if( auto )
        start();
}
```

这就可以了！现在有了具备后退、前进和自动循环功能的完整幻灯片演示程序。

为了清晰地说明这个示例，我使这个插件完全依赖于外部操纵。

下面是一个具体实现的模型：

```
<ul>
    <li><img id="prev" src="prev.png" /></li>
    <li><img id="slideshow" /></li>
    <li><img id="next" src="next.png" /></li>
  </ul>

...

(function( $ ){
    var $image = $('#slideshow');

    $image.slideshow({
        images: ['1.jpg', '2.jpg', '3.jpg', '4.jpg'],
        interval: 3000
    });

    $('#prev').click(function(){
        $image.trigger('prev');
    });

    $('#next').click(function(){
        $image.trigger('next');
    });

    $image.trigger('goto', 0); // 初始化为0
    $image.trigger('start'); // 我们打算自动循环显示

})( jQuery );
```

注意

这是使用trigger()是因为它很简短，事实却是，如果使用triggerHandler()可能会更快，因为trigger()将产生事件冒泡（从jQuery 1.3起），而你可能并不需要这种功能。

2．如果元素已经有其中一个事件会发生什么？

可能发生一种情况（尽管这很奇怪）：#slideshow元素可能已经绑定了名为prev、next、goto、start或者stop的事件。

如果出现这种情况，解决方案见秘诀8.4。

3．如何允许其他代码清理添加的事件处理程序？

因为没有提供绑定的函数，所以其他外部代码无法解除它们的绑定。

在大部分情况下，可以简单地从事件中解除所有事件的绑定，例如：

```
jQuery('#slideshow').unbind('prev next goto start stop'); // 列举所有事件
//或者
jQuery('#slideshow').unbind(); //盲目地删除所有绑定
```

如果你需要谨慎地进行绑定解除，或者只是想要解除所有相关事件的绑定，查阅秘诀8.3。

4．与其他方法有何区别

现在还有其他的技术能够实现外部操纵。我将对其中一些进行比较：

允许插件接受命令。jQuery UI（以及其他一些插件）使用这种模式。

这种模式就是当插件接受一个字符串作为第一个参数时执行操作，例如：

```
jQuery('#image').slideshow('goto', 1);
```

这种写法比使用事件的方法短一些，但是要求你用公开的方法保存所有必需的数据（在例子中是当前索引），以便后续的数据获取。利用这种模式的人也倾向于使用date()存储这些变量。

如果使用事件，可以简单地使用局部变量，因为事件处理程序可以访问插件的局部作用域。

返回带有方法的对象。Jörn Zaefferer的validate插件（以及其他的一些插件）使用这种模式。

根据编码方式，对象的方法可能访问局部变量。为此必须使用闭包[5]，而闭包是不能滥用的。而且，需要在某个地方（以全局方式）存储这个对象，还要求进行伪面向对象编码（你可能喜欢这种方式，也可能不喜欢）。

可以混合使用这两种方法以及前面介绍过的方法。可以创建一个对象（使用jQuery.listener）并绑定事件然后返回对象，而不是将事件（prev、next等）。绑定到DOM元素。正如我们在秘诀9.2中所看到的，这个监听器对象不仅限于事件。它可以拥有方法甚至在其属性中存储事件数据。

9.6 在调用jQuery方法时得到通知

9.6.1 问题

你希望在使用jQuery修改DOM元素时执行某个操作。这可能涉及修改CSS属性、从文档中删除等。

有些浏览器已经支持突发事件（Mutation events），[6]能够满足上述需求，但是你还无法以跨浏览器的方式使用它们，而且它们也没有集成到jQuery中。

你可能需要的另一个功能是在jQuery方法执行之前修改其参数。同样的原理，你可能需要修改在函数执行之后某方法返回的数据。

9.6.2 解决方案

在某种程度上，这与面向方面的编程相关，[7]但是这里不打算嵌套函数；相反，将一次性地重载所需要的方法，每当调用该方法时就触发事件。

我们将需要在函数运行之前触发一个事件，以便修改函数的参数。还需要在函数运行之后触发一个事件，以便获取返回的数据，甚至在必要时进行修改。

我们来看看如何编写实现这一功能的插件。下面将对每个步骤单独作出说明。

9.6.3 讨论

1. 重载需要的方法

首先，创建一个函数，用自己的函数替代jQuery方法。将其命名为`jQuery.broadcast()`，如果你喜欢其他名称，可以修改：

```
(function($){

    $.broadcast = function(name){
        // 保存原始方法
        var old = $.fn[name];

        $.fn[name] = function(){
            // 广播
```

```
        };
    };
})(jQuery);
```

name必须是我们想要覆盖的方法名称，例如：

```
jQuery.broadcast('addClass');
```

2．在执行之前触发一个事件

既然我们将自己的方法当作jQuery方法，就让我们来看看如何触发一个事件，使我们能够修改输入的参数：

```
//创建一个事件对象
 var e = $.Event('before-'+name);
 // 将参数保存到对象中
 e.args = $.makeArray(arguments);
 //触发事件
 this.trigger(e);
```

假定要广播addClass()，现在可以这么做：

```
jQuery('body').bind('before-addClass',function(e){
  e.args[0]; // CSS类
});
```

3．执行原始方法

现在触发一个事件，但是仍然必须调用旧的addClass()。因为将把返回的数据保存在事件对象中，所以在触发其他事件的时候，仍然可以提供这些数据。

```
e.ret = old.apply(this, e.args);
```

正如你所看到的，不传递原始的arguments数组，而用自己提供的数据代替，以防它们以某种方式修改。

4．在执行之后触发事件

现在将返回的数据保存在事件对象中，可以触发最后一个事件，允许已返回数据的外部修改。

重用同一个事件对象，但是修改事件的名称。

```
e.type = 'after-'+name;
this.trigger(e);
```

5．返回结果

现在剩下的工作就是返回结果数据，继续正常的执行流程。将返回保存在e.ret上的数据，该数据可能已经被某个事件处理程序修改：

```
return e.ret;
```

6．全部整合到一起

下面是开发的完整代码：

```
(function($){

    $.broadcast = function(name){
        var old = $.fn[name];

        $.fn[name] = function(){
            var e = $.Event('before-'+name);

            e.args = $.makeArray(arguments);
            this.trigger(e);

            e.ret = old.apply(this, e.args);

            e.type = 'after-'+name;
            this.trigger(e);

            return e.ret;
        };
    };
})(jQuery);
```

7．还有哪些可以改进之处

我尽量保持示例简明扼要。对这个例子还有许多可以改进的地方，下面是几种思路：

- 使用triggerHandler()代替trigger()：如果不需要事件冒泡，可以简单地使用triggerHandler()，这个方法会使整个过程更快；注意，triggerHandler()只触发集合中第一个元素上的事件。
- 在每个元素上单独运行：在前面的例子中，trigger()在整个集合上运行一次。对于大多数情况这是没有问题的，但是当用在有多个元素的集合上时可能得到出乎意料的结果。

可以调用map()包装在函数中放入的功能，这能使代码在每个元素上都运行一次。

这样做的缺点是速度可能稍慢，由于调用map()，还将生成一个（预期之外）的栈条目。

- 允许外部代码阻止正常地执行：如果使用的是jQuery 1.3或者更高版本，可以利用jQuery.Event的方法。

可以用e.isDefaultPrevented()“询问”事件对象是否有其他代码调用了它的preventDefault()方法。

如果e.isDefaultPrevented()返回true，就不要调用原始函数。

- 避免多次重载同一jQuery方法：这很简单；只要创建一个内部对象，记录重载的方法，然后只要忽略重复的调用就可以了。
- 与jQuery.event.special集成：这样就不必为所要重载的每个方法调用jQuery.broadcast()了。

作为替代，为每个方法在jQuery.event.special中添加一个条目，当其他代码绑定事件的时候从内部调用jQuery.broadcast()。这应该与重复调用的检查组合使用。

9.7 将对象方法作为事件监听器使用

9.7.1 问题

你的对象具有方法和属性，你想要将这些方法（函数）当作事件处理程序。问题是一旦这样做，这些方法将失去对象的引用，你也无法在事件处理程序中引用对象。

9.7.2 解决方案

上述问题的解决曾经很复杂。你必须生成闭包，封装对象，然后将它们传递给bind()。

从jQuery 1.3.3起，bind()方法中已经添加了一个新的参数，允许指定一个对象为作用域或者事件处理程序的this对象，而不需要使用函数闭包。

这使得所需的代码既简短又快速。现在，可以将对象方法作为函数传递，对象本身则作为作用域。

9.7.3 讨论

1. 节点到哪里去了

你迟早都会提这个问题。前面已经提到过，当向bind()传递一个作用域对象时，事件处理程序的this将被覆盖。这意味着无法像往常那样读取该节点……但是这个节点并没有丢失。

当向bind()方法传递一个作用域对象时，事件将向作用域对象提交事件处理程序的this。仍然可以使用event.currentTarget属性确定提交给事件的元素，该属性包含对该DOM元素的引用。

这通常没有必要，因为使用this更简短，但是在这类情况下，这是唯一的出路。

2. 示例

我将创建一个小的示例，说明使用作用域参数的方法，并说明它适用的场合。

对象。例如，需要两个对象，每个对象都有一个将要作为事件处理程序绑定的方法：

这些对象如下：

```
function Person(name){
   this.name = name;
   this.married = false;
}

jQuery.extend( Person.prototype, {
     whatIsYourName: function(){
         alert(this.name);
     },
     updateMarriedState: function(e){
         var checkbox = e.currentTarget;
         this.married = checkbox.checked;
     }
});

var peter = new Person('Peter');
var susan = new Person('Susan');
```

绑定方法。假定有某种表单，表单上有两个复选框（#c1和#c2）。每个复选框操纵上述对象之一的married状态。

```
jQuery('#c1').bind('change', peter.updateMarriedState, peter);
jQuery('#c2').bind('change', susan.updateMarriedState, susan);
```

由于有作用域属性，因此不需要为每次绑定创建新的函数；可以使用对象方法来代替。

这些方法甚至不需要从一开始就附加到对象中。可以这样做：

```
function updatePersonMarriedState(e){
     var checkbox = e.currentTarget;
     this.married = checkbox.checked;
}
jQuery('#c1').bind('change', updatePersonMarriedState, peter);
jQuery('#c2').bind('change', updatePersonMarriedState, susan);
```

正如你所看到的，你实际上并不一定要将这些函数放入对象原型中，保持它们的独立也有实际意义。为什么属于Person的一个方法应该知道复选框和节点的情况？将具体的DOM操纵与数据分离可能更好。

在某些情况下，对象方法完全不必了解节点或者事件对象。在这种情况下，可以直接绑定一个方法，完全不必将DOM与数据混在一起。

如果有两个按钮（#b1和#b2），单击它们时应该显示一个人的姓名，这可以简单地实现：

```
jQuery('#b1').bind('click', peter.whatIsYourName, peter);
jQuery('#b2').bind('click', susan.whatIsYourName, susan);
```

值得一提的是，这两个方法实际上相同：

```
peter.whatIsYourName == susan.whatIsYourName; // true
```

该函数只创建一次，并保存到Person.protype中。

---

[1] *http://en.wikipedia.org/wiki/Encapsulation_(computer_science)*。

[2] *http://plugins.jquery.com/project/mousewheel*。

[3] *http://plugins.jquery.com/project/drag*，*http://plugins.jquery.com/project/drop*。

[4] *http://flesler.blogspot.com/2008/02/jqueryserialscroll.html*。

[5] *http://en.wikipedia.org/wiki/Closure_(computer_science)*。

[6] *http://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-eventgroupings-mutationevents*。

[7] *http://en.wikipedia.org/wiki/Aspect_oriented_programming*。

第10章　从头开始增强HTML表单

Brian Cherne

10.0 导言

不管你是想学习更多JavaScript和jQuery知识，还是只想为眼下的问题编写最直接的解决方案，有时候从头开始编写代码是最好的方法。本章旨在为你提供简单通用的解决方案，帮助你从头开始编写自己的代码。

必须注意的是，虽然从头开始有很大的好处，你所遇见的某些常见的问题（如剩余字符计数、自动调整文本区域大小以及表单验证等）已经被其他人解决。请参见第11章或者访问jQuery论坛和博客，了解社区中评论和测试过的插件能给你带来的帮助。有时候，查看其他人的代码能够帮助你做得更好。

在必要时，我在每个秘诀的“问题”标题下提供某些样板HTML。这不是一个哲学的说法——“裸”HTML没有任何问题。但是，我试图重申一个概念，JavaScript应该用于增强现有的HTML，改进用户交互，应该与HTML完全分开考虑。

> **注意**
>
> 在接下来的秘诀中，我只展示与问题相关的XHTML代码段。请确保代码是通过验证的完整XHTML文档。

而且，我只在秘诀中包含`$(document).ready(function(){...})`，作为讨论的一部分。所有其他的解决方案都假定你将在代码结构的正确位置放置JavaScript——在`.ready()`处理程序或者XHTML代码之后的文件尾部。更多信息请参见第1章。

10.1 在页面加载时将焦点放在一个文本输入字段上

10.1.1 问题

在你的主页上有一个登录表单，你希望页面加载时焦点位于用户名文本输入框。

10.1.2 解决方案

使用jQuery $(selector).focus()方法：

```
//当HTML DOM就绪
$(document).ready(function(){
    //将焦点放在<input id="username" type="text" ...>上
    $('#username').focus();
});
```

10.1.3 讨论

使用$(document).ready()应该足够快。但是，在通过慢速连接读取大型HTML文件之类的情况下，光标的焦点可能比预期更晚地设置——用户可能已经输入了用户名并且正在输入她的密码，这时$(document).ready()执行，将她的光标放回到用户名输入框。这多么烦人！在这种情况下，可以在<input>标记之后使用内联JavaScript，立即设置焦点：

```
<input name="username" id="username" type="text" />

<script type="text/javascript">
    $('#username').focus();
</script>
```

如果你更喜欢将代码集中放在$(document).ready()块中，也可以在设置焦点之前检查文本输入框是否已经输入了文本：

```
// 当HTML DOM就绪
$(document).ready(function(){
    var $inputTxt = $('#username');
    if( $inputTxt.val() == '' ) {
        // 默认情况下将焦点放在用户名输入框上
        $inputTxt.focus();
    }
});
```

当禁用JavaScript时会发生什么情况？用户必须手动单击进入文本输入框开始输入。

10.2 禁用和启用表单元素

10.2.1 问题

你的订单表单有用于运送和账单联络信息的字段。你决定为用户提供表示用户运送信息和账单信息相同的复选框，使界面更加友好。当选中该复选框时，禁用账单字段：

```
<fieldset id="shippingInfo">
    <legend>Shipping Address</legend>

    <label for="shipName">Name</label>
    <input name="shipName" id="shipName" type="text" />

    <label for="shipAddress">Address</label>
    <input name="shipAddress" id="shipAddress" type="text" />
</fieldset>

<fieldset id="billingInfo">
    <legend>Billing Address</legend>

    <label for="sameAsShipping">Same as Shipping</label>
    <input name="sameAsShipping" id="sameAsShipping" type="checkbox"
value="sameAsShipping" />

    <label for="billName">Name</label>
    <input name="billName" id="billName" type="text" />

    <label for="billAddress">Address</label>
    <input name="billAddress" id="billAddress" type="text" />
</fieldset>
```

10.2.2 解决方案1

如果你想要做的只是禁用账单字段，在change事件触发时使用jQuery .attr()和.removeAttr()方法就可以做到：

```
//寻找"sameAsShipping"复选框,监听change事件
$('#sameAsShipping').change(function(){

    if( this.checked ){
        //寻找billingInfo内的所有文本输入字段,禁用它们
        $('#billingInfo input:text').attr('disabled','disabled');
    } else {
        //寻找billingInfo内的所有文本输入字段,启用它们
        $('#billingInfo input:text').removeAttr('disabled');
    }
}).trigger('change'); // close change() then trigger it once
```

10.2.3 解决方案2

虽然选择复选框禁用表单字段已经足以达到用户的要求，但是还可以更进一步，用运送信息中的数据预先填写账单文本字段。

这个解决方案的第一部分在结构上和前一个解决方案相同。但是，除了禁用账单字段之外，还用运送信息字段的数据预先填写它们。下面的代码假定运送和账单所用的<fieldset>元素包含相同数量的文本输入字段，顺序也相同：

```
//寻找"sameAsShipping"复选框,监听change事件
$('#sameAsShipping').change(function(){
    if( this.checked ){
        //找到billingInfo中的所有文本输入字段,禁用它们,然后循环读取每个输入字段
        $('#billingInfo input:text').attr('disabled',
'disabled').each(function(i){

                //寻找billingInfo内的所有文本输入字段,禁用它们
                var valueFromShippingInput =
$('#shippingInfo input:text:eq('+i+')').val();
                //用运送文本字段的值设置账单文本字段值
                $(this).val( valueFromShippingInput );

        }); // close each()
    } else {
        //寻找billingInfo内的所有文本输入字段,启用它们
        $('#billingInfo input:text').removeAttr('disabled');
    }
}).trigger('change'); // close change() then trigger it结束change(),然后触发它
```

本解决方案的第二部分在用户输入运送字段信息时，自动更新账单字段，但是只在禁用账单字段时进行：

```
//寻找shippingInfo文本输入字段,监听keyup和change事件
$('#shippingInfo input:text').bind('keyup change',function(){

        //如果选中"sameAsShipping"复选框
        if ( $('#sameAsShipping:checked').length ){

        //找出文本输入字段中的this对象
        var i = $('#shippingInfo input:text').index( this );
        var valueFromShippingInput = $(this).val();
        $('#billingInfo input:text:eq('+i+')').val( valueFromShippingInput );
    }
}); // 结束bind()
```

10.2.4 讨论

在上述的解决方案中，使用了input:text选择器，避免禁用复选框。

使用.trigger('change')会立刻执行.change()事件。该事件开始时会检查复选框状态，以防它默认选中。而且，事件方法保护Firefox和其他在页面刷新时保留单选按钮和复选框状态的浏览器。

当禁用JavaScript时会发生什么情况？你应该在CSS中默认隐藏复选框。然后使用JavaScript为覆盖前一个CSS规则的双亲元素添加一个类名。在下面的示例代码中，添加了一个附加的 来包围复选框和标签，便于隐藏它们：

```
<style type="text/css" title="text/css">
 #sameAsShippingWrapper { display:none; }
```

```
 .jsEnabled #sameAsShippingWrapper { display:block }
</style>

...

//当HTML DOM就绪时
$(document).ready(function(){
    $('form').addClass('jsEnabled');
});

...

<form>
 ...
 <div id="sameAsShippingWrapper">
    <label for="sameAsShipping">Same as Shipping</label>
    <input name="sameAsShipping" id="sameAsShipping" type="checkbox" ... />
 </div>
 ....
</form>
```

作为用CSS隐藏复选框然后用JavaScript显示它的替代方案，还可以用JavaScript将复选框添加到DOM中。我喜欢保持HTML、CSS和JavaScript的独立，但是有时候这是更好的解决方案：

```
var html_label = '<label for="sameAsShipping">Same as Shipping</label>';
var html_input = '<input name="sameAsShipping" id="sameAsShipping" type="checkbox"
value="sameAsShipping" />';

$( html_label + html_input ).prependTo('#billingInfo").change( ... ).trigger( ... );
```

10.3 自动选择单选按钮

10.3.1 问题

你有一系列单选钮。最后一个按钮的标签为“Other”并关联一个文本输入字段。很自然，你希望如果用户在Other字段中输入文本时选中该按钮：

```
<p>How did you hear about us?</p>
<ul id="chooseSource">
    <li>
        <input name="source" id="source1" type="radio" value="www" />
        <label for="source1">Website or Blog</label>
    </li>
    <li>
        <input name="source" id="source2" type="radio" value="mag" />
        <label for="source2">Magazine</label>
    </li>
    <li>
        <input name="source" id="source3" type="radio" value="per" />
        <label for="source3">Friend</label>
    </li>
    <li>
        <input name="source" id="source4" type="radio" value="oth" />
        <label for="source4">Other</label>
        <input name="source4txt" id="source4txt" type="text" />
    </li>
</ul>
```

10.3.2 解决方案1

在HTML代码中你会注意到，单选按钮、标签和关联的文本输入元素都包装在一个<li>标记里。你不一定需要这种结构，但是它能大大简化准确找到单选按钮的过程——保证只有一个单选按钮兄弟元素：

```
//寻找chooseSource列表总的所有文本输入字段,监听blur事件
$('#chooseSource input:text').blur(function(){

    // 如果在文本输入字段中输入了文本
    if ( $(this).val() != '' ) {
        //寻找单选按钮兄弟元素并设置为选中状态
        $(this).siblings('input:radio').attr('checked',true);
    }
});
```

10.3.3 解决方案2

进一步拓展思路，当单选按钮选中时，可以调用文本字段的.focus()方法。需要注意的是，下面的代码完全替代了前一个解决方案，不采用.blur()方法之后链接.each()方法的方案，而是使用.each()方法，因为这样可以访问所有需要的对象：

```
$('#chooseSource input:text').each(function(){

    // 下面的代码都使用两次,所以把它们存储在函数中以达到更高的效率
    // 文本输入字段
    var $inputTxt = $(this);
    // 关联的单选按钮
    var $radioBtn = $inputTxt.siblings('input:radio');

    // 监听文本输入字段的blur事件
    $inputTxt.blur(function(){
        //如果文本输入字段中有文本
        if ( $inputTxt.val() != '' ) {
            // 选中单选按钮
            $radioBtn.attr('checked',true);
        }
    });

    // 监听单选按钮的change事件
    $radioBtn.change(function(){
        // 如果按钮选中,将焦点放在文本输入字段上
        if ( this.checked ) { $inputTxt.focus(); }
    });

}); //结束each()
```

10.3.4 讨论

jQuery的.siblings()方法仅返回兄弟元素，而不是你试图寻找其兄弟元素的那个HTML元素。所以，$(this).siblings('input:radio')可以改写为$(this).siblings('input')，因为只有一个输入兄弟元素。我喜欢包含:radio选择器，因为这样更加明确，创建的代码也是不言自明的。

使用$('#source5txt').focus(...)直接获得Other文本输入字段，并用它的id属性直接获取单选按钮非常简单。虽然这也是完全可用的方法，但是前面展示的代码更灵活。如果有人决定修改Other单选钮会怎么样呢？如果每个单选钮都关联了文本输入字段又当如何？抽象的解决方案不用多花力气就能够处理这些情况。

为什么使用.blur()代替文本输入字段上的.focus()？虽然.focus()能够更直接地选择关联的单选按钮，但如果用户简单地用Tab键选择不同元素，.focus()可能意外地选中单选按钮。使用.blur()然后检查按钮值可以避免这个问题。

禁用JavaScript会发生什么情况？用户必须手动单击进入文本输入字段开始输入，也必须手动选择单选按钮。你剩下的工作是决定在用户输入文本并选择不同单选按钮的时候验证和处理提交数据的方法。

10.4 用专用的链接选择（反选）所有复选框

10.4.1 问题

你需要用专用的Select All和Deselect All链接选择和反选所有复选框：

```
<fieldset>

    <legend>Reasons to be happy</legend>

     <a class="selectAll" href="#">Select All</a>
     <a class="deselectAll" href="#">Deselect All</a>

    <input name="reasons" id="iwokeup" type="checkbox" value="iwokeup" />
    <label for="iwokeup">I woke up</label>

    <input name="reasons" id="health" type="checkbox" value="health" />
    <label for="health">My health</label>

    <input name="reasons" id="family" type="checkbox" value="family" />
    <label for="family">My family</label>

    <input name="reasons" id="sunshine" type="checkbox" value="sunshine" />
    <label for="sunshine">The sun is shining</label>

</fieldset>
```

10.4.2 解决方案

直接使用class属性获取Select All和Deselect All链接。然后附加相应的.click()处理程序：

```
//找到一组字段中的"Select All"链接并监听单击事件
$('fieldset .selectAll').click(function(event){
    event.preventDefault();
    // 找到所有复选框并选中
    $(this).siblings('input:checkbox').attr('checked','checked');
});

// 找到一组字段中的"Deselect All"链接并监听单击事件
$('fieldset .deselectAll').click(function(event){
    event.preventDefault();
    //找到所有复选框并取消选择它们
    $(this).siblings('input:checkbox').removeAttr('checked');
});
```

10.4.3 讨论

如果你对激活专用链接以及使其无效感兴趣，应该看看本章的秘诀10.5。在那个解决方案中，单个复选框更新切换状态，你需要这样的逻辑相应地激活专用链接以及使其失效。

当禁用JavaScript时会发生什么情况？你应该默认用CSS隐藏链接。然后使用JavaScript为双亲元素添加一个类名，覆盖前一个CSS规则：

```
<style type="text/css" title="text/css">
 .selectAll, .deselectAll { display:none; }
 .jsEnabled .selectAll, .jsEnabled .deselectAll { display:inline; }
</style>

...

//当HTML DOM就绪时
$(document).ready(function(){
    $('form').addClass('jsEnabled');
});
```

10.5 用一个切换开关选中（反选）所有复选框

10.5.1 问题

你需要用一个切换开关（这里是另一个复选框）选中和反选所有复选框。此外，如果单独选中某些（或者所有）复选框，切换动作应该自动变换它们的状态：

```
<fieldset>

    <legend>Reasons to be happy</legend>

     <input name="reasons" id="toggleAllReasons" type="checkbox" class="toggle" />
     <label for="toggleAllReasons" class="toggle">Select All</label>

    <input name="reasons" id="iwokeup" type="checkbox" value="iwokeup" />
    <label for="iwokeup">I woke up</label>

    <input name="reasons" id="health" type="checkbox" value="health" />
    <label for="health">My health</label>

    <input name="reasons" id="family" type="checkbox" value="family" />
    <label for="family">My family</label>

    <input name="reasons" id="sunshine" type="checkbox" value="sunshine" />
    <label for="sunshine">The sun is shining</label>

</fieldset>
```

10.5.2 解决方案

用class属性和:checkbox选择器直接找到切换开关。然后循环读取每个找到的切换开关，用.siblings()确定相关联的复选框，并附加change事件监听器：

```
//找到字段集中的"Select All"切换开关,循环读取找到的所有开关
$('fieldset .toggle:checkbox').each(function(){

    //以下代码使用多次,将其存储起来获得更高的效率
    // 切换开关复选框
    var $toggle = $(this);
    // 其他复选框
    var $checkboxes = $toggle.siblings('input:checkbox');

    // 监听切换开关的change事件
    $toggle.change(function(){
        if ( this.checked ) {
            // 如果选中,选择所有复选框
            $checkboxes.attr('checked','checked');
        } else {
            //如果没有选中,反选所有复选框
            $checkboxes.removeAttr('checked');
        }
    });

    //监听每个单独复选框(不包含切换开关)的change事件
```

```
    $checkboxes.change(function(){
        if ( this.checked ) {
            //如果复选框选中而其他复选框都选中,则选中切换开关
            if ( $checkboxes.length == $checkboxes.filter(':checked').length ) {
                $toggle.attr('checked','checked');
            }
        } else {
            //如果复选框没有选中,反选切换开关
            $toggle.removeAttr('checked');
        }
    }).eq(0).trigger('change'); //然后仅触发第一个复选框的change事件
}); //结束each()
```

10.5.3 讨论

使用.eq(0).trigger('change')直接执行第一个复选框的.change()事件。这将设置切换开关状态，并且保护Firefox和其他在页面刷新时保留单选按钮和复选框的浏览器。

.eq(0)用来仅仅触发第一个复选框的change事件。没有.eq(0)，.trigger('change')将在每个复选框上执行，因为它们共享同一个切换开关，所以只需要运行一次。

当禁用JavaScript时会发生什么情况？应该默认用CSS隐藏切换开关复选框和标签。然后使用JavaScript为双亲元素添加一个类名，这将覆盖前一条CSS规则：

```
<style type="text/css" title="text/css">
 .toggle { visibility:hidden; }
 .jsEnabled .toggle { visibility:visible; }
</style>

...

//当HTML DOM就绪时
$(document).ready(function(){
    $('form').addClass('jsEnabled');
});
```

10.6 添加和删除Select元素中的选项

10.6.1 问题

你有一个用于选择颜色的下拉框，希望在其中添加新的颜色或者删除其中的选项。

```
<label for="colors">Colors</label>
<select id="colors" multiple="multiple">
    <option>Black</options>
    <option>Blue</options>
    <option>Brown</options>
</select>

<button id="remove">Remove Selected Color(s)</button>

<label for="newColorName">New Color Name</label>
<input id="newColorName" type="text" />

<label for="newColorValue">New Color Value</label>
<input id="newColorValue" type="text" />

<button id="add">Add New Color</button>
```

10.6.2 解决方案

使用.appendTo()方法在下拉框中添加新选项：

```
//找到"Add New Color"按钮
$('#add').click(function(event){
    event.preventDefault();

    var optionName = $('#newColorName').val();
    var optionValue = $('#newColorValue').val();

    $('<option/>').attr('value',optionValue).text(optionName).appendTo('#colors');
});
```

使用.remove()方法删除选项：

```
//找到"Remove Selected Color(s)"按钮
$('#remove').click(function(event){
    event.preventDefault();

    var $select = $('#colors');

    $('option:selected',$select).remove();
});
```

10.6.3 讨论

使用.attr()和.text()方法填充<option>元素：

```
$('<option/>').attr("value",optionValue).text(optionName).appendTo('#colors');
```

然而，可以改写同一行代码，在不使用该方法的情况下，在一步中构建<option>元素：

```
$('<option value="'+optionValue+'">'+optionName+'</option>').appendTo('#colors');
```

像这样连接所有<option>数据在速度上可能快不到1毫秒，人们几乎无法注意到。我更喜欢用.attr()和.text()方法来填充<option>元素，因为我认为这样更容易理解，也更容易调试和维护。由于性能问题可以忽略不计，因此使用哪一种方法就完全看开发人员的偏好了。

禁用JavaScript会发生什么情况？需要提供一个服务器端的替代方案来处理按钮单击，用户必须等待结果最终重新加载。

10.7 根据字符计数自动跳到下一个控件

10.7.1 问题

你有一个表单，让用户在线注册一个产品，你要求用户输入打印在安装盘上的序列号。序列号是一个16位的数字，分布在4个输入字段中。理想情况下，为了加快用户数据的输入，在每个输入字段填满时，你希望焦点自动地设置为下一个输入字段，直到用户输完序列号：

```
<fieldset class="autotab">
    <legend>Product Serial Number</legend>
    <input type="text" maxlength="4" />
    <input type="text" maxlength="4" />
    <input type="text" maxlength="4" />
    <input type="text" maxlength="4" />
</fieldset>
```

10.7.2 解决方案

在<fieldset class="autotab">中寻找所有<input>元素。使用jQuery的.bind()方法监听keydown和keyup方法。在一些应该忽略的按键上退出绑定函数，因为它们对于自动前后切换输入字段没有意义。当<input>元素填满时，根据maxlength属性，用.focus()将焦点设置为下一个<input>元素。相反，当使用Backspace键时，如果<input>元素为空，用.focus()将焦点设置为上一个<input>元素：

```
$('fieldset.autotab input').bind('keydown keyup',function(event){

    //触发事件的键码
    var keyCode = event.which;

    //我们希望忽略如下按键:
    // 9 Tab, 16 Shift, 17 Ctrl, 18 Alt, 19 Pause Break, 20 Caps Lock
    // 27 Esc, 33 Page Up, 34 Page Down, 35 End, 36 Home
    // 37 Left Arrow, 38 Up Arrow, 39 Right Arrow, 40 Down Arrow
    // 45 Insert, 46 Forward Delete, 144 Num Lock, 145 Scroll Lock
    var ignoreKeyCodes =
',9,16,17,18,19,20,27,33,34,35,36,37,38,39,40,45,46,144,145,';
    if ( ignoreKeyCodes.indexOf(',' + keyCode + ',') > -1 ) { return; }

    //我们希望在keydown事件上忽略backspace,
    //仅让它完成自己的工作,而不改变焦点
    if ( keyCode == 8 && event.type == 'keydown' ) { return; }

    var $this = $(this);
    var currentLength = $this.val().length;
    var maximumLength = $this.attr('maxlength');

    //如果是backspace键而没有更多字符,返回
    if ( keyCode == 8 && currentLength == 0 ) {
        $this.prev().focus();
    }
```

```
    //如果已经填满了输入字段,转到下一个
    if ( currentLength == maximumLength ) {
        $this.next().focus();
    }
});
```

10.7.3 讨论

为什么要绑定keydown和keyup事件？

可以只使用keydown事件。但是，当用户完成第一个输入字段时，下一个按键会使焦点设置为第二个输入字段，没有任何视觉提示。通过使用keyup事件，在填写第一个输入字段之后，第二个输入字段获得焦点，光标放在输入字段的开始位置，大部分浏览器会用边框或者其他高亮状态表示焦点。而且，在当前输入字段为空时，Backspace键需要keyup事件才能将焦点设置为前一个输入字段。

也可以只使用keyup事件。但是，如果光标在第二个输入字段中，打算使用Backspace键清除该字段，一旦删除了所有字符，焦点就会移到第一个输入字段。遗憾的是，第一个字段已经填满，所以下一个击键会因为maxlength属性而丢失，然后keyup事件将把焦点又设置为第二个输入字段。丢失击键是很糟糕的事情，所以在keydown上执行相同的检查，这样在字符丢失之前会将光标移到下一个输入字段。

因为这一逻辑不是CPU密集的，所以可以同时绑定keydown和keyup事件没什么问题。在其他情况下，可能需要采用更具选择性的方案。

你将会注意到，ignoreKeycodes变量是一个字符串。如果我们打算动态地构建该变量，创建一个数组，然后使用JavaScript的.join(', ')或.toString()方法会更快捷。但是因为字符串值始终相同，从一开始就将它编码为一个字符串更简单。因为担心出错，我让ignoreKeyCodes以逗号开始和结束。这样，当搜索前后加上逗号的keycode时，就能保证只找到你想找的数字——如果搜索的是9，不会找到19或者39。

注意，没有用来阻止在最后一个<input>元素上执行$this.next().focus()的代码。这里利用了jQuery链。如果$this.next()什么也没找到，则方法链停止——它无法用.focus()将焦点设置在找不到的元素上。在不同的场景中，预先缓存任何已知的.prev()和.next()元素可能是有意义的。

当禁用JavaScript时会发生什么情况？什么也不会发生，用户必须手动单击，从一个文本输入字段转到下一个字段。

10.8 显示剩余字符串计数

10.8.1 问题

你的公司网站上有一个联系人表单。这个表单有一个<textarea>元素，用户可以在该元素中自由表达自己的想法。但是，你知道时间就是金钱，不希望你的员工在这上面阅读短篇小说，所以要限制他们不得不阅读的消息长度。在这一过程中，你也想告诉最终用户还可以输入多少个字符。

```
<textarea></textarea>
<div class="remaining">Characters remaining: <span class="count">300</span></div>
```

10.8.2 解决方案

以所有.remaining消息为目标，对每条消息，寻找相关的<textarea>元素和.count子元素中列出的最大字符数。为<textarea>绑定一个update函数，在用户输入文本的时候捕获：

```
//对于找到的每个"Characters remaining: ###"元素
$('.remaining').each(function(){

    // 查找并存储读出的计数和相关的文本区域/输入字段
    var $count = $('.count',this);
    var $input = $(this).prev();

    //  .text()返回的是一个字符串,乘于1将得到一个数字(用于计算)
    var maximumCount = $count.text()*1;

    //在keyup、paster和input事件上调用的更新函数
    var update = function(){

        var before = $count.text()*1;
        var now = maximumCount - $input.val().length;

        //检查确保用户没有超过限制
        if ( now < 0 ){
            var str = $input.val();
            $input.val( str.substr(0,maximumCount) );
            now = 0;
        }

        //只在必要时修改DOM
        if ( before != now ){
            $count.text( now );
        }
    };

    //监听change事件(参见如下的讨论)
    $input.bind('input keyup paste', function(){setTimeout(update,0)} );

    //在开始时调用update,以防输入域预先填写
    update();
```

```
}); // 结束.each()
```

10.8.3 讨论

上述代码足够通用，考虑了指定页面任何数量的“Character remaining”消息和<textarea>元素。如果你打算构建一个内容管理或者数据输入系统，这可能很实用。

为了在用户试图用鼠标将数据复制和粘贴到<textarea>时提供保护，需要绑定input和paste事件。不能使用mouseup事件，因为在选择浏览器上下文菜单的一项时不会触发它。input事件是HTML5（工作草案）的一部分，已经被Firefox、Opera和Safari所实现。它在用户输入时触发，不管输入设备为何（鼠标还是键盘）。Safari在本书编写期间还有一个缺陷，无法在<textarea>元素上触发输入事件。Safari和Internet Explorer都能理解<textarea>元素上的paste事件，理解用于捕捉击键的keyup事件。

附加keyup、input和paste是多余的，但是在本例中是有利的。update函数很简单，因而没有任何性能问题，它只在必要的时候操纵DOM，所以第一个update之后的调用都是多余的，什么也不会做。

对多余事件的替代方案之一是在<textarea>元素拥有焦点时使用setInterval。在指定时间间隔之后调用同一个update函数，如果与keyup事件配对，你就能够在按键的时候立刻更新，而且任意的更新时间间隔（如300毫秒）就是信息粘贴到<textarea>元素的时间。如果update函数较为复杂或者代价较高，这可能是更好的替代方案。

当将事件绑定到表单元素时，有时候使用超时对函数调用稍作延迟很重要。在前一个例子中，Internet Explorer在剪贴板中的文本实际在添加到<textarea>元素中之前触发paste事件。因此，剩余字符数的计算在用户单击或者按下另一个键之前都是不正确的。通过使用setTimeout(update, 0)，把更新函数放在调用栈的最后，在浏览器添加文本之后才触发：

```
$input.bind('input keyup paste', function(){setTimeout(update,0)} );
```

当禁用JavaScript时会发生什么情况？你应该在CSS中默认隐藏“Characters remaining”消息。然后使用JavaScript为双亲元素添加一个类名，覆盖前一条CSS规则。还有，在服务器端再次检查消息的长度也很重要：

```
<style type="text/css" title="text/css">
 .remaining { display:none; }
 .jsEnabled .remaining { display:block; }
</style>

...

//当HTML DOM就绪时
$(document).ready(function(){
    $('form').addClass('jsEnabled');
});
```

10.9 限制文本输入字段内容为特定的字符

10.9.1 问题

你的购物车页面有一个数量字段，你希望确保用户只能在该字段中输入数字：

```
<input type="text" class="onlyNumbers" />
```

10.9.2 解决方案

用onlyNumbers类寻找所有元素，并监听keydown和blue事件。keydown事件处理器将阻止用户在字段中输入非数字字符。blur事件处理器是一个防范措施，清除任何通过上下文菜单或者浏览器“编辑”菜单中的粘贴功能输入的数据：

```
$('.onlyNumbers').bind('keydown',function(event){

    //按键的keycode
    var keyCode = event.which;

    // 48～57标准键盘的数字键
    var isStandard = (keyCode > 47 && keyCode < 58);

    // 96～105扩展键盘的数字键(也称小键盘)
    var isExtended = (keyCode > 95 && keyCode < 106);

    // 8 Backspace, 46向前删除
    // 37左箭头, 38上箭头, 39右箭头, 40下箭头
    var validKeyCodes = ',8,37,38,39,40,46,';
    var isOther = ( validKeyCodes.indexOf(',' + keyCode + ',') > -1 );

    if ( isStandard || isExtended || isOther ){
        return true;
    } else {
        return false;
    }

}).bind('blur',function(){

    //匹配非数字字符的正则表达式
    var pattern = new RegExp('[^0-9]+', 'g');

    var $input = $(this);
    var value = $input.val();

    //用正则表达式清除字段值
    value = value.replace(pattern, '');
    $input.val( value )
});
```

10.9.3 讨论

keydown事件是即时发生的，阻止用户在字段中输入非数字字符。也可以用与blur事件共享相同处理器的keyup事件来代替。但是，用户将会看到非数字字符出现，然后很快消失。我更喜欢从一开始就阻止输入这些字符，避免闪烁。

blur事件可以阻止复制和粘贴的非数字字符进入文本字段。在前一个场景中，假定用户试图测试JavaScript的限制（我也会这么做）或者试图从一个电子表格中复制和粘贴数据。在我看来，两者都不需要立刻进行更正。但是，如果你的情况要求更及时地更正，请参阅秘诀10.8的“讨论”部分中有关捕捉粘贴事件中更改的信息。

如果你的情况不同，希望用户从电子表格中复制和粘贴数据，一定要注意：我所使用的正则表达式没有考虑小数点。所以，数字“1，000”会清理为“1000”而数字“10.00”也会清理为“1000”。

你将会注意到，validKeyCodes变量是一个以逗号开始和结束的变量。在秘诀10.7已经提到，这样做是因为我害怕出错——当搜索两边加上逗号的keyCode时，保证你只会找到所要找的数字。

当禁用JavaScript时会发生什么情况？用户可以输入他们喜欢的任何字符。一定要在服务器端验证代码。不要依靠JavaScript提供干净的数据。

10.10 用Ajax提交表单

10.10.1 问题

你想用Ajax提交一个表单：

```
<form action="process.php">

    <!-- value changed via JavaScript -->
    <input type="hidden" name="usingAJAX" value="false" />

    <label for="favoriteFood">What is your favorite food?</label>
    <input type="text" name="favoriteFood" id="favoriteFood" />

    <input type="submit" value="Tell Us" />

</form>
```

10.10.2 解决方案

找到<form>元素，拦截其submit事件：

```
$('form').submit(function(event){

    // 我们想用Ajax提交表单(避免闪烁)
    event.preventDefault();

    //在这里放置验证代码(如果有的话)
    // ...

    //通知服务器端进程使用了Ajax
    $('input[name="usingAJAX"]',this).val( 'true' );

    //保存对表单的引用
    var $this = $(this);

    //从表单元素捕捉URL
    var url = $this.attr('action');

    // 准备要发送的表单数据
    var dataToSend = $this.serialize();

    // 告诉我们服务器端进程信息的回调函数
    var callback = function(dataReceived){

        //隐藏表单(幸好存储了它的引用)
        $this.hide();

        //在例子中,服务器返回一个HTML片段,只要将它附加到DOM中就可以了
        // 期望的结果: <div id="result">Your favorite food is pizza! Thanks for
        // telling us!</div>
        $('body').append(dataReceived)
    };
```

```
    //接收数据类型(预计的是HTML片段)
    var typeOfDataToReceive = 'html';

    //现在发送表单等待返回数据
    $.get( url, dataToSend, callback, typeOfDataToReceive )

}); //结束.submit()
```

10.10.3 讨论

当禁用JavaScript时会发生什么情况？将提交表单，整个页面用服务器端脚本返回的结果刷新。使用JavaScript将`<input type="hidden" name="usingAJAX" />`元素的值从`false`改为`true`。这使服务器端脚本知道应该发回什么样的响应——完整的HTML页面或者预期的Ajax响应数据。

10.11 验证表单

10.11.1 问题

你希望验证一个表单。开始，你应该建立一些基本的CSS。对这一增强来说，真正重要的唯一样式是div.errorMessage选择器的display:none声明和div.showErrorMessage选择器的display:block声明。其余的样式只是对外观的美化：

```
<style type="text/css" title="text/css">
    div.question {
        padding: 1em;
    }
    div.errorMessage {
        display: none;
    }
    div.showErrorMessage {
        display: block;
        color: #f00;
        font-weight: bold;
        font-style: italic;
    }
    label.error {
        color: #f00;
        font-style: italic;
    }
</style>
```

下面的HTML片段是构造表单的一个例子。<div class="question>元素纯粹用于布局，对验证代码并不重要。每个<label>元素的for属性将其与id属性相同的表单元素关联。这是标准的HTML，之所以强调它，是因为JavaScript也使用这个属性（反向）寻找给定表单元素的正确<label>。类似地，你会注意到错误消息的id属性为errorMessage_加上相关表单元素的name属性。这种结构看上去似乎多余，但是复选框和单选按钮按照name属性分组，每一个组只能有一条错误消息：

```
<form action="process.php">

<!-- TEXT -->
<div class="question">
    <label for="t">Username</label>
    <input id="t" name="user" type="text" class="required" />
    <div id="errorMessage_user" class="errorMessage">Please enter your username.</div>
    </div>

<!-- PASSWORD -->
<div class="question">
    <label for="p">Password</label>
    <input id="p" name="pass" type="password" class="required" />
    <div id="errorMessage_pass" class="errorMessage">Please enter your password.</div>
</div>

<!-- SELECT ONE -->
<div class="question">
<label for="so">Favorite Color</label>
```

```
    <select id="so" name="color" class="required">
    <option value="">Select a Color</option>
        <option value="ff0000">Red</option>
        <option value="00ff00">Green</option>
        <option value="0000ff">Blue</option>
    </select>
    <div id="errorMessage_color" class="errorMessage">Please select your favorite
color.</div>
</div>

<!-- SELECT MULTIPLE -->
<div class="question">
    <label for="sm">Favorite Foods</label>
    <select id="sm" size="3" name="foods" multiple="multiple" class="required">
        <option value="pizza">Pizza</option>
        <option value="burger">Burger</option>
        <option value="salad">Salad</option>
    </select>
    <div id="errorMessage_foods" class="errorMessage">Please choose your favorite
foods.</div>
</div>

<!-- RADIO BUTTONS -->
<div class="question">
    <span>Writing Hand:</span>
    <input id="r1" type="radio" name="hand" class="required"/>
    <label for="r1">Left</label>
    <input id="r2" type="radio" name="hand" class="required" />
    <label for="r2">Right</label>
    <div id="errorMessage_hand" class="errorMessage">Please select what hand you
write with.</div>
</div>

<!-- TEXTAREA -->
<div class="question">
    <label for="tt">Comments</label>
    <textarea id="tt" name="comments" class="required"></textarea>
    <div id="errorMessage_comments" class="errorMessage">Please tell us what you
think.</div>
</div>

<!-- CHECKBOX -->
<div class="question">
    <input id="c" type="checkbox" name="legal" class="required" />
    <label for="c">I agree with the terms and conditions</label>
    <div id="errorMessage_legal" class="errorMessage">Please check the box!</div>
</div>

<input type="submit" value="Continue" />

</form>
```

10.11.2 解决方案

解决方案的第一部分相当简单。找到<form>元素，拦截submit事件。当提交表单时，循环读取必要的表单元素，检查必要的元素是否有效。如果表单没有错误，则（也仅在这种情况下）触发submit事件：

```
$('form').submit(function(event){
```

```
    var isErrorFree = true;

    //循环读取必要的表单元素,检查它们是否有效
    $('input.required, select.required, textarea.required',this).each(function(){
        if ( validateElement.isValid(this) == false ){
            isErrorFree = false;
        };
    });

    // Ajax替代方案:
    // event.preventDefault();
    // if (isErrorFree){ $.get( url, data, callback, type ) }
    // if (isErrorFree){ $.post( url, data, callback, type ) }
    // if (isErrorFree){ $.ajax( options ) }

    return isErrorFree;

}); //结束 .submit()
```

解决方案的第二部分是真正发生验证的地方。isValid()方法开始时存储打算验证的常用数据。然后，在switch()语句中验证元素。最后，在<label>和div.errorMessage元素中添加或者删除类名。

```
var validateElement = {

    isValid:function(element){

        var isValid = true;
        var $element = $(element);
        var id = $element.attr('id');
        var name = $element.attr('name');
        var value = $element.val();

        // <input>使用在标签中写入的type属性
        // <textarea>固有的类型为"textarea"
        // <select>固有的类型为"select-one"或"select-multiple"
        var type = $element[0].type.toLowerCase();

        switch(type){
            case 'text':
            case 'textarea':
            case 'password':
                if ( value.length == 0 ||
value.replace(/\s/g,'').length == 0 ){ isValid = false; }
                break;
            case 'select-one':
            case 'select-multiple':
                if( !value ){ isValid = false; }
                break;
            case 'checkbox':
            case 'radio':
                if( $('input[name="' + name +
'"]:checked').length == 0 ){ isValid = false; };
                break;
        } // 结束switch()
        //将使用$(selector)[method]代替$(selector).method
        //明智地选择正确的方法
        var method = isValid ? 'removeClass' : 'addClass';

        // 显示错误消息[addClass]
```

```
        // 隐藏错误消息[removeClass]
        $('#errorMessage_' + name)[method]('showErrorMessage');
        $('label[for="' + id + '"]')[method]('error');

        return isValid;

    } // 结束validateElement.isValid()
}; // 结束validateElement object
```

10.11.3 讨论

这个解决方案中的验证相当简单，检查项目如下：

- <input type="text">、<input type="password">和<textarea>有空格之外的某些数据。
- <select>元素有默认选项之外的选择。请注意，<select>元素有两类：“select-one”（单选）和 “selectmultiple”（多选）（参见本节的第二个代码段中的HTML代码和前一个代码段中的JavaScript验证）。“单选”<select>中的第一个<option>元素必须有value=""，验证才能正常工作。“多选”<select>没有这种要求，因为它的<option>元素可以反选。
- 对应name组内的<input type="radio">和<input type="checkbox">至少有一个元素选中。

使用switch(){}语句是因为它比多条if(){}else if(){}语句更高效，而且允许集中处理共享验证的元素，而用break；语句分隔不同的组。

validateElement对象处于全局作用域，其意图是在其他表单上重用。通过包含验证方法，它还保持了全局作用域的整齐——未来，可以为validateElement对象添加助手方法，而不要担心全局命名冲突。例如，可以这样实现stripWhitespace()方法：

```
var validateElement = {

    stripWhitespace : function(str){
        return str.replace(/\s/g,'');
    },
    isValid : function(element){

        //... snipped code ...//

            case 'text':
            case 'textarea':
            case 'password':
                //如果文本长度在去掉空格之后为0,它就是无效的
                if ( this.stripWhitespace(value).length == 0 ){ isValid = false; }
                break;

        //...省略的代码...//

    } // 结束validateElement.isValid()
}; // 结束validateElement object对象
```

当显示和隐藏错误消息时，使用方括号记法调用jQuery的.addClass()和.removeClass()方法：

```
//用$(selector)[method]代替$(selector).method
//明智地选择正确的方法
var method = isValid ? 'removeClass' : 'addClass';

//显示错误消息[addClass]
// 隐藏错误消息[removeClass]
$('#errorMessage_' + name)[method]('showErrorMessage');
$('label[for="' + id + '"]')[method]('error');
```

上述代码中的方括号记法在功能上与点记法相同：

```
if (isValid) {
    $('#errorMessage_' + name).removeClass('showErrorMessage');
    $('label[for="' + id + '"]').removeClass('error');
} else {
    $('#errorMessage_' + name).addClass('showErrorMessage');
    $('label[for="' + id + '"]').addClass('error');
}
```

当进行提交验证时，点记法更加清晰易读。但是，扩展方括号记法的解决方案，可以用change事件（在初始验证之后）重新验证元素。这使得当用户的新答案实际上有效时可以立刻得到反馈，而不要求他们单击提交按钮。下列代码无法像预期那样工作（真正的解决方案参见下一段），但是说明了在何处.unbind()和.bind() change事件：

```
// 用$(selector)[method]代替$(selector).method
//明智地选择正确的方法
var method = isValid ? 'removeClass' : 'addClass';

//显示错误消息[addClass]
//显示错误消息[removeClass]
$('#errorMessage_' + name)[method]('showErrorMessage');
$('label[for="' + id + '"]')[method]('error');
//在初始验证之后,允许元素在修改后重新验证
$element
    .unbind('change.isValid')
    .bind('change.isValid',function(){ validateElement.isValid(this); });
```

注意

因为在每次验证的时候解除绑定和绑定change事件，所以添加了.isValid事件命名空间，更加直接地针对具体的事件函数。这样，如果表单元素绑定了另一个change事件，它们将会保留。

前一段代码的问题不是语法而是逻辑。你会注意到，HTML中的单选按钮有class="required"属性。这意味着，当验证整个表单的时候，每个单选按钮都验证，而且（更重要的是）每个单选按钮的<label>添加或者删除一个类来表示错误的情况。但是，如果允许用元素专用的change事件进行重新验证，只有特定单选按钮的<label>将更新——其他都保持错误状态。为了处理这种情况，change事件必须查看同一name组中的所有单选按钮和复选框，同时影响所有的<label>类：

```
// 用$(selector)[method]代替$(selector).method
// 明智地选择正确的方法
var method = isValid ? 'removeClass' : 'addClass';

// 显示错误消息[addClass]
// 显示错误消息[removeClass]
$('#errorMessage_' + name)[method]('showErrorMessage');

if ( type == 'checkbox' || type == 'radio' ) {

    //如果是单选按钮或者复选框,寻找同名的所有输入字段
    $('input[name="' + name + '"]').each(function(){
        //更新每个输入元素的<label>标记(this==<input>)
        $('label[for="' + this.id + '"]')[method]('error');
    });

} else {

    //其他所有元素只要更新一个<label>
    $('label[for="' + id + '"]')[method]('error');

}

// 在初始验证之后,允许元素在修改后重新验证
     .unbind('change.isValid')
     .bind('change.isValid',function(){ validateElement.isValid(this); });
```

如果上述代码用点记法重写，就需要重写两倍的代码。此外，将这一新逻辑应用到合适的地方，name组中只有一个单选按钮（或者复选框）需要有class="required"属性，就能使该组的其他所有元素得到正确的调整。

当禁用JavaScript的时候会发生什么情况？将提交表单而不进行客户端验证。一定要在服务器端验证代码。不要依靠JavaScript提供干净的数据。如果服务器端代码返回带有错误的表单，可以用相同方式使用相同元素上的同一个类。没有必要使用内联样式标记或者编写自定义代码对服务器端错误做出不同的处理。

第11章　用插件增强HTML表单

Jörn Zaefferer

11.0 导言

表单是Web应用程序用户很常用的交互手段。改进这种交互就能改善应用程序的工作效率。

jQuery和各种插件为更好的交互提供了现成和可自定义的解决方案，其核心就是渐进增强。

每个问题都可以用jQuery解决方案从头开始解决，但是使用插件有许多好处：

- 避免重复开发；
- 提供在不同浏览器中得到很好测试的功能；
- 节约花在细节上的许多精力；
- 提供经过调整，可以工作在极限情况下的功能。

每个秘诀都将讨论插件的长处和不足，强调重新开发可能有意义的场合。

11.0.1 基本方法

使用jQuery插件的基本方法始终如一。你首先包含jQuery，然后在页面上包含插件文件。有些插件还需要一个样式表。大部分插件需要某些标记和选择标记元素的一行代码，然后对其加以处理。因为常用的命名惯例，插件“slideshow”（幻灯片放映）应该这样使用：

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="jquery.slideshow.css"/>
    <script src="assets/jquery-latest.js"></script>
    <script src="assets/jquery.slideshow.js"></script>
    <script type="text/javascript">
    jQuery(document).ready(function($){
        $("#slideshow").slideshow();
    });
    </script>
</head>
<body>
    <div id="slideshow">...</div>
</body>
</html>
```

幻灯片放映所需的标记与slider插件或者表单验证有很大的不同，这正是每个插件的文档和示例所要说明的，也将在下面的秘诀中介绍。

11.1 验证表单

11.1.1 问题

大部分注册表单需要输入电子邮件地址、密码（两次）、用户名和其他一些信息（如生日）。这种表单适用于电子邮件服务、网店或者论坛。

很容易想象，John Doe希望在某个网点上购买一台新的显示器，注册的时候还要输入一个验证码（一组可以辨认的随机字符，用来区分人和机器人）。他花费一些时间填写完整的表单，然后提交该表单。大约5秒之后，表单再次显示，顶部有一些错误消息：他忘了填写街道字段。他改正错误之后再次提交。又过了5秒钟：现在他忘了填写密码和验证码字段！

这样迟钝的反馈非常令人沮丧，可能毁掉本来很好的用户体验，特别是在安全方面的考虑限制了功能的时候——在这个例子中导致了空白的密码和验证码字段。

11.1.2 解决方案

改善这种情况的方法之一是添加客户端验证。基本思路是尽快给用户反馈，避免他们厌烦。不应该提交无效的表单，这样就能避免重复填写密码或者验证码的情况。

在字段中填写*john.doe@gmail，com*这样的无效电子邮件之后，高亮显示的错误也是有意义的。在用户有机会正确填写字段之前高亮显示错误字段没有任何帮助：在需要至少两个字符的字段上，用户输入第一个字符之后在字段上显示“输入太短”完全没有作用。

validation插件（*http://jquery-cookbook.com/go/plugin-validation*）很好地满足了这些需求。

要开始使用这个插件，首先下载，解压文件，将`jquery.validate.js`复制到项目中。下面的例子展示了一个评论表单，表单上有姓名、电子邮件、URL和实际评论字段。调用插件方法`validate()`建立表单的验证。验证规则用类和属性以内联方式指定：

```
<!DOCTYPE html>
<html>
<head>
   <script src="assets/jquery-latest.js"></script>
   <script src="assets/jquery.validate.js"></script>
   <style type="text/css">
     * { font-family: Verdana; font-size: 96%; }
     label { width: 10em; float: left; }
     label.error { float: none; color: red; padding-left: .5em; vertical-align: top; }
     div { clear: both; }
     input, textarea { width: 15em; }
     .submit { margin-left: 10em; }
   </style>
   <script type="text/javascript">
     jQuery(document).ready(function($){
       $("#commentForm").validate();
     });
   </script>
</head>
<body>
   <form id="commentForm" method="get" action="">
```

```
    <fieldset>
      <legend>A simple comment form with submit validation and default messages</legend>
      <div>
        <label for="cname">Name</label>
        <input id="cname" name="name" class="required" minlength="2" />
      </div>
      <div>
        <label for="cemail">E-Mail</label>
        <input id="cemail" name="email" class="required email" />
      </div>
      <div>
        <label for="curl">URL (optional)</label>
        <input id="curl" name="url" class="url" value="" />
      </div>
      <div>
        <label for="ccomment">Your comment</label>
        <textarea id="ccomment" name="comment" class="required"></textarea>
      </div>
      <div>
        <input class="submit" type="submit" value="Submit"/>
      </div>
    </fieldset>
  </form>
</body>
</html>
```

有required类的任何字段都必须接受检查，看看有没有内容。这个例子中的其他方法包括：

email

检查字段中是否包含有效的电子邮件地址。

url

检查字段是否包含有效的URL。

minlength

检查字段是否至少有*x*个字符；这里*x*通过属性minlength="2"指定。

11.1.3 讨论

验证插件推广客户端验证的一种特殊方法：在浏览器中进行尽可能多的工作，仅在特殊情况下寻求服务器的帮助，这些情况（如检查用户名是否仍然可用）需要用到远程方法（*http://jquery-cookbook.com/go/plugin-validation-remote-method*）。

另一种不同的方法是不在客户端和服务器端重复验证规则和方法，而是通过Ajax将整个表单发送到服务器（通常是在表单提交的时候），然后在服务器端使用已经编写好的相同逻辑。这样做的缺点是用户反馈较慢，因为每次击键都发送一个请求是不现实的。编写服务器验证的时候也不太会考虑Ajax验证，所以也不容易重用它。在这种情况下，你必须预先进行规划。

验证插件可在以后添加到表单上，和远程验证不同，没有必要以某种方式改写应用程序。这对于博客上的简单评论表单和一些内联网应用上更加复杂的表单以及两者之间的任何表单都很实用。

这个插件最重要的组成部分是规则和方法。方法包含验证逻辑，如使用正则表达式确定一个值是不是有效电子邮件地址的E-mail方法。规则将字段与方法连接起来，一条规则就是一个输入字段和一个方法的配对。电子邮件地址字段有一条将其标记为必需字段的规则，以及一条将其标记为电子邮件地址的规则。

1．方法

验证插件有19个内置方法。最重要的方法是required——它指定必须填写的字段。如果省略它，在空字段上的大部分其他方法将被忽略。唯一的例外是equalTo方法，该方法检查字段内容是否与其他字段相同，即使空字段也适用，这条规则本身最常用于“确认密码”字段。

email、url、date、dateISO、dateDE、number、numberDE、digits和creditcard方法都检查某些数据类型，对不同地区有简单的变种。例如，number要求美国数字格式（如1，000.00），而numberDE要求德国数字格式（如1.000，00）。

min和max以及range方法检查数字值，而minlength、maxlength和rangelength检查字符数量。

在选择列表或者复选框中，min、max和range验证选中的选项或者复选框的数量。

在文件输入字段中，accept方法可以很方便地检查文件扩展名，默认情况下查找.gif、.png、.jpg或者.jpeg。

remote方法是将其他地方的验证逻辑委托给服务器端的唯一方法。它以指向某个服务器端资源的URL作为参数，该URL可以是一个进行数据库查询的脚本，例如，检查用户名是否已经存在或者指定电子邮件地址是否已经注册的脚本。使用远程方法验证用户名和电子邮件地址的注册表单例子可以在*http://jquery-cookbook.com/go/plugin-validation-remote-demo*上找到。

自定义方法是按照应用程序相关需求扩展插件的好手段。你可能有一个表单，用户输入的URL必须以某个公司域名开始。自定义方法可以封装必要的验证：

```
jQuery.validator.addMethod("domain", function(value, element) {
    return this.optional(element) || /^http:\/\/mycorporatedomain.com/.test(value);
}, "Please specify the correct domain for your documents");
```

jQuery.validator.addMethod的第一个参数是自定义方法的名称，必须是有效的JavaScript标识符。第二个参数是实现真正验证的一个函数。如果它返回真值，就认为输入是有效的。它使用this.optional(element)确定输入是否有值以及是否应该跳过——所有默认方法都使用相同的调用。第三个参数指定新方法的默认信息。

接受参数的方法的写法非常类似：

```
jQuery.validator.addMethod("math", function(value, element, params) {
    return this.optional(element) || value == params[0] + params[1];
}, jQuery.format("Please enter the correct value for {0} + {1}"));
```

在这个例子中，默认的信息在该插件提供的模板助手jQuery.format的帮助下指定。圆括号中的索引占位符在验证运行时由实际参数替代。

自定义方法也可以重用现有的方法，这对为单个方法指定不同的默认信息很有用。在这个例子中，required方法用不同的默认信息化名为customerRequired：

```
$.validator.addMethod("customerRequired", $.validator.methods.required,
"Customer name required");
```

插件中的additionalMethods.js捆绑了一组现成的自定义方法。

2．规则

指定规则有4种不同的方法：两种采用代码形式，另两种作为元数据嵌入。前一个例子使用了类和属性等元数据，这是插件默认支持的方式。在元数据插件（*http://jquery-cookbook.com/go/plugin-metadata*）可用时，规则可以各种方式迁入，例如，在class属性中：

```
<input type="text" name="email" class="{required:true, email:true}" />
```

这段代码中的类在圆括号中包含了JavaScript文本，语法与在代码中通过rules选项指定规则非常相似：

```
$("#myform").validate({
    rules: {
        name: {
            required: true,
            minlength: 2
        },
        email: {
            required: true,
            email: true
        },
        url: "url",
        comment: "required"
    }
});
```

name、email、url和comment等对象键值始终引用的是元素名称而不是ID。

注意，url和comment使用了简写，其中只需要一条规则。当用minlength等参数指定规则时不能使用这种方法。

有些规则需要在以后添加，这是第4种方法——rules插件方法：

```
//首先初始化验证
$("#myform").validate();
// 以后的某个时刻,添加更多的规则
$("#username").rules("add", { minlength: 2});
```

规则也可以这样删除：

```
$("#username").rules("remove", "required");
```

当在登录表单上实现“忘记密码”链接时，这个方法很方便：

```
$("#loginform").validate({
    username: "required",
    password: "required"
});
$("a#forgotPassword").click(function(e) {
    $("#password").rules("remove", "required");
    $("#loginform").submit();
    $("#password").rules("add", "required");
    return false;
});
```

单击事件代码从密码中删除“required”规则，试图提交表单（触发验证），并重新添加规则。这样，用户名字段仍然得到验证，如果验证失败，密码字段再次变成必填字段（在正常表单提交的情况下）。

依赖性。字段的验证行为往往取决于单击链接之外的一些因素。使用required方法的参数能够处理这些情况。参数可以是一个选择器或者一个回调函数。选择器在依赖性可用简单的表达式编写的时候很有用。电子邮件字段只在时事通信复选框选中的时候才是必填的：

```
email: {
    required: "#newsletter:checked"
}
```

回调可用于任意复杂性的表达式，例如，字段依赖于多个其他字段的状态：

```
email: {
    required: function(element) {
        return $("#newsletter:checked").length && $("#telephone:blank");
    }
}
```

自定义表达式。前一个例子使用:blank表达式，只选择根本没有值或者空白的元素。该插件还提供了:filled表达式，它和:blank意义相反。jQuery本身提供了:checked，验证插件添加了与此相反的:unchecked。在指定按钮或者复选框的依赖性时，这两者都很有用。

虽然也可以使用:not表达式获得:filled或:checked的相反意义，但是:blank和:unchecked使选择器更易读，从而更容易理解。

3．错误消息

和规则类似，指定各种消息也有几种方法，既可以通过代码也可以采用内联方式。内联消息从title属性中读取：

```
<input name="email" class="required email" title="A valid email address is
required" />
```

这将为每条规则生成一条错误消息。备选的内联方法是使用元数据插件（参见前面的“规则”小节）：

```
<input name="email" class="{required:true, email:true, messages:{required:"Required",
email: "Not a valid email address"}}"/>
```

可以用这种方法指定每条规则的消息，也可以使用messages选项：

```
$("#myform").validate({
    messages: {
        email: {
            required: "Required",
            email: "Not a valid email address"
        }
    }
});
```

这里的键值email指的也是输入字段的名称而非ID，和规则的指定方法一样。

对于更动态性的场景，可以使用rules插件方法：

```
$("#myform").validate();
// 稍后
$("#email").rules("add", {
    messages: {
        email: "A valid email address, please!"
    }
});
```

如果使用title属性的替代品，同时使用常规的标题，可以禁止插件检查消息的属性：

```
$("#myform").validate({
    ignoreTitle: true
});
```

本地化。默认的消息采用英语（除了dateDE和numberDE以外）。此外，该插件提供（在本书编写的时候）17种本地化信息。用法很简单：将所需的message_xx.js文件复制到项目中，在验证插件之后包含它们。例如，下面是瑞典语本地化消息所用的代码：

```
<script src="assets/jquery-latest.js"></script>
<script src="assets/jquery.validate.js"></script>
<script src="assets/messages_se.js.js"></script>
```

有了上述代码，“Please enter a valid email address”（请输入有效的电子邮件地址）将会变成“Ange en korrekt e-postadress”。

错误元素。默认情况下，错误消息将插入DOM中发生错误元素的下一个节点。把错误消息当作标签元素插入，把for属性设置为验证元素的id。带有for属性使用标签利用了浏览器的特性——单击标签会将焦点设置在输入字段上。所以默认情况下，用户可以单击错误消息，将焦点移到无效字段上。

如果需要不同的元素类型，可以使用errorElement选项：

```
$("#myform").validate({
    errorElement: "em"
});
```

插件仍然使用for属性，但是浏览器提供的自动链接失效。

布局。如果你想要自定义错误消息插入的位置，errorPlacement选项就能派上用场。假定有一个表单，使用表格布局，第一列包含常规标签，第二列是输入字段，第三列是错误消息：

```
<form id="signupform" method="get" action="">
    <table>
        <tr>
            <td class="label">
                <label id="lfirstname" for="firstname">First Name</label>
            </td>
            <td class="field">
                <input id="firstname" name="firstname" type="text" value=""
maxlength="100" />
            </td>
            <td class="status"></td>
        </tr>
        <!-- more fields -->
    </table>
</form>

$("#signupform").validate({
    errorPlacement: function(error, element) {
        error.appendTo( element.parent("td").next("td") );
    }
});
```

另一种常见的需求是将一般的消息显示在表单上方。errorContainer选项有助于实现这种功能：

```
$("#myform").validate({
    errorContainer: "#messageBox1"
});
```

在这个例子中，ID为messageBox1的元素在表单无效的时候显示，在表单有效时隐藏。

errorContainer选项也可以与errorLabelContainer选项合并使用。当指定这两个选项时，错误标签不会放在输入元素的旁边，而是添加到表单上方或者下方的一个元素中。和errorContainer及wrapper组合使用，信息可以添加到表单上方的一个错误列表中：

```
<div class="container">
        <h4>There are a few problems, please see below for details.</h4>
        <ul></ul>
</div>
<form id="myform" action="">
<!-- form content -->
</form>
var container = $('div.container');
// validate the form when it is submitted
$("#myform").validate({
    errorContainer: container,
    errorLabelContainer: $("ul", container),
    wrapper: 'li'
});
```

4．处理提交

一旦表单有效，就必须提交它。默认情况下和任何其他的表单提交一样。要通过Ajax提交表单，可以使用submitHandler选项和表单插件（更多细节参见秘诀11.6）：

```
$(".selector").validate({
    submitHandler: function(form) {
        $(form).ajaxSubmit();
    }
});
```

invalidHandler回调用于在无效提交时运行代码。下面的例子显示忘记填写的字段的摘要：

```
$("#myform").validate({
    invalidHandler: function(e, validator) {
        var errors = validator.numberOfInvalids();
        if (errors) {
            var message = errors == 1
                ? 'You missed 1 field. It has been highlighted below'
                : 'You missed ' + errors + ' fields. They have been highlighted below';
            $("div.error span").html(message);
            $("div.error").show();
        } else {
            $("div.error").hide();
        }
    }
});
```

Maketo演示程序展示了这种行为（*http://jquery-cookbook.com/go/plugin-validation-marketo-demo*）。

5．局限性

那么，不使用插件而从头开始编写一个验证解决方案在什么时候有意义呢？插件有一些局限性：有些表单中的成组输入字段（如复选框）有不同的name属性，难以作为整体来验证。有相同名称的一系列输入字段也无法验证，因为每个单独的输入字段都需要有独特的名称。如果你坚持对于输入字段采用唯一名称的命名约定，对一组复选框或者单选钮只使用一个名称，插件就能工作得很好。

如果应用程序只有一个登录表单，使用插件可能就有些小题大做了，从文件尺寸上也不合理；但是，如果将插件用于网站的其他地方，它也就可以用在登录表单上。

11.2 创建固定格式的输入字段

11.2.1 问题

某些输入类型很容易出错，如信用卡号。在第一时间内没有发现的简单录入错误可能在以后产生可怕的错误。这也适用于日期或者电话号码，它们之间有一些共同的特征：

- 固定长度
- 大部分是数字
- 在某些位置上有定界符

11.2.2 解决方案

Masked input插件（*http://jquery-cookbook.com/go/plugin-masked-input*）是一个能够改进反馈的jQuery插件。它适用于一个或者多个输入字段，限制输入的字符，同时自动插入定界符。

在下面的例子中，把电话号码输入字段限制为固定格式：

```
<!DOCTYPE html>
<html>
<head>
    <script src="assets/jquery-latest.js"></script>
    <script src="assets/jquery.maskedinput.js"></script>
    <script>
    jQuery(document).ready(function($) {
        $("#phone").mask("(999) 999-9999");
    });
    </script>
</head>
<body>
    <form>
        <label for="phone">Phone</label>
        <input type="text" name="phone" id="phone" />
    </form>
</body>
</html>
```

除了jQuery以外，还包含插件文件。在文档就绪事件的回调函数中，选择ID为phone的输入字段，调用mask方法。唯一的参数指定了限定的格式，表示该字段格式为美国电话号码。

11.2.3 讨论

指定固定格式时可以使用4个具有特别意义的字符：

a

字母字符a～z、A～Z。

9

0～9的任意数字。

*

任何字母和数字字符，也就是a～z、A～Z和0～9。

?

在此之后可以是任意字符。

任何其他字符（如phone掩码中的圆括号或者连字符）都被认为是字面量，插件自动将它们插入输入字段中，用户无法删除。

默认情况下，插件为每个可变字符插入下划线（_）。对于电话号码示例，输入字段获得焦点时将显示如下值：

```
(___) ___-____
```

当用户开始输入时，如果输入的是有效字符（数字），第一个下划线将被替代。其他字面量也会被跳过。

下划线占位符可以通过传递一个附加的参数自定义：

```
$("#phone").mask("(999) 999-9999", {placeholder: " "});
```

上述代码显示空格代替下划线。

也可以定义新的格式字符：

```
$.mask.definitions['～'] = '[+-]';
$("#eyescript").mask("～9.99～9.99 999");
```

上面定义了新的格式字符～，允许值为+和-，指定的方式和正则表达式的字符类相同。以后，～可以用在固定格式中。

问号可以建立有固定部分和可选部分的格式。有可选分机号的电话号码可以这样定义：

```
$("#phone").mask("(999) 999-9999? x99999");
```

当组合固定格式和验证插件（秘诀11.1）时，重要的是为字段定义合适的规则。否则，验证插件可能将固定格式插件的占位符当作有效的输入，在用户刚插入第一个字符时，如果无效的字段被标识为有效，用户肯定不会高兴的。

局限性

这个插件主要的局限性是固定长度的要求。它不能用于可变长度的输入字段，如货币值。例如，“$ 999，999.99”要求输入100 000.00～999 999.99之间的值，而不能接受此范围之外的数值。

11.3 自动补全文本字段

11.3.1 问题

有两种HTML输入字段类型允许用户从一系列现有值中选择一个：单选按钮和选择列表。单选按钮对于多达8项的列表很适合，而选择列表可以有多达30～150个列表项，具体取决于数据类型。当用户也能输入新值的时候，这两种方法都无法满足——在这种情况下，它们通常伴随一个“其他”字段。当列表很大（可能有500或者500 000项）时，两种方法都无能为力。

11.3.2 解决方案

jQuery UI autocomplete窗口组件（*http://jquery-cookbook.com/go/widget-autocomplete*）能够解决选择列表所无法解决的多种问题。在最简单的情况下，显示数据存储在一个JavaScript数组中：

```
<label for="month">Select a month:</label>
<input id="month" name="month" />

var months = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
'August', 'September', 'October', 'November', 'December'];
$("#month").autocomplete({
    source: months
});
```

这里对月份输入应用了自动补全插件，数据在一个普通的JavaScript数组中。

当数据不存在于客户端时，插件可以从服务器端资源获得数据：

```
$("#month").autocomplete({
    source: "addresses.php"
});
```

然后，插件向服务器端资源发送一个GET请求，在q参数中附加用户输入的值，如`addresses.php?q=ma`。插件预期的响应是以换行符分隔的值列表：

```
Mainstreet
Mallstreet
Marketstreet
```

11.3.3 讨论

使用插件时的第一个决策是确定数据在本地还是远端。

使用本地数据，完整数据集已经存在于浏览器的内存中，它可能已经作为页面的一部分或者通过单独的Ajax请求加载。在任何一种情况下，它只加载一次。这种模式在数据较小而且是静态的时候很实用——少于500行数据且在选择的时候不修改值。本地数据的主要好处是找到匹配值的速度极快。

远程数据从服务器以小块的形式加载（合理的块尺寸为最多100行）。既可以使用小的数据集也可以使用非常大的数据集（例如，超过50万行数据）。由于数据从服务器下载，因此寻找匹配值比本地数据要慢。通过加载足够大的数据块可以缓解速度问题，这样可以在客户端进行过滤，而不用更多的服务器请求。

11.4 选择一个取值范围

11.4.1 问题

想象一个汽车搜索界面：用户输入可接受的价格范围，在修改该值的同时，也更新处于对应范围的备选汽车列表。用于此类输入的HTML表单元素（普通文本输入、单选按钮、选择列表）还不够好。一方面，这些元素都要求输入一个精确值。另一方面，它们无法表达价格范围。而且也不能改变整个价格范围，用户不得不一个接一个地更新起始和结束值。

11.4.2 解决方案

jQuery UI slider窗口组件（*http://jquery-cookbook.com/go/widget-slider*）能够将两个文本输入框转换为一个范围滑块。范围的起始和结束值可以用鼠标或者光标键拖动。

默认的滑块应用到一个简单的 ，没有任何选项：

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

$("#slider").slider();
```

要使用范围滑块，除了UI主题之外，还必须包含jQuery、jQuery UI核心以及滑块slider.js文件：

```
<link rel="stylesheet" href="ui.core.css" />
<link rel="stylesheet" href="ui.slider.css" />
<link rel="stylesheet" href="ui.theme.css" />
<script type="text/javascript" src="jquery-1.3.2.js"></script>
<script type="text/javascript" src="ui.core.js"></script>
<script type="text/javascript" src="ui.slider.js"></script>
```

以上代码在页面中添加了一个漂亮的滑块，但是目前它实际上没有任何用处。

在汽车搜索的例子中，我们打算将选中的值放入输入字段，显示给用户：

```
<p>
    <label for="amount">Price range:</label>
    <input type="text" id="amount" style="border:0; color:#f6931f;
font-weight:bold;" />
</p>

<div id="slider-range"></div>
```

基于标记，可以创建范围滑块：

```
var slider = $("#slider-range").slider({
    range: true,
    min: 0,
    max: 500,
    values: [75, 300],
    slide: function(event, ui) {
        $("#amount").val('$' + ui.values[0] + ' - $' + ui.values[1]);
    }
```

```
});
$("#amount").val('$' + slider.slider("values", 0) + ' - $' + slider.slider("values",
1));
```

将range选项设置为true告诉插件创建两个手柄（而不是一个）。min和max选项指定可选范围；values选项是开始位置。

slide回调在手柄移动时触发。在例子中，它更新amount输入字段并显示所选的价格范围。

11.4.3 讨论

将滑块绑定到一个文本输入框是选择之一；另一种选择是将其绑定到一个选择列表，并使用选择列表的选项作为数值。

我们以客房预订为例，用户在系统中输入最小床位数。最大床位数为6；因此，滑块是个不错的选择。使用渐进增强，可以用一个滑块，将其变化反馈到<select>元素，对其加以改进：

```
<select name="minbeds" id="minbeds">
    <option>1</option>
    <option>2</option>
    <option>3</option>
    <option>4</option>
    <option>5</option>
    <option>6</option>
</select>

var select = $("#minbeds");
var slider = $('<div id="slider"></div>').insertAfter(select).slider({
    min: 1,
    max: 6,
    range: "min",
    value: select[0].selectedIndex + 1,
        slide: function(event, ui) {
    select[0].selectedIndex = ui.value-1;
}
});
$("#minbeds").click(function() {
    slider.slider("value", this.selectedIndex + 1);
});
```

现有标记没有任何语义含义，因此不使用它，而是动态生成 并将其插入DOM，紧接在<select>之后。

因为只有一个值，所以用value选项代替values选项。直接用选择列表的DOM属性selectedIndex初始化该选项。属性从0开始，所以加上1。

当按键或者通过鼠标拖动手柄更新滑块时，通过设置selectedIndex属性更新选择列表，设置的值从传递给每个ui事件的ui对象中读取。在初始化时加上的偏移值1现在要减去。

尽管只有一个手柄，但还是设置了range选项。该选项的参数可以是字符串，也可以是布尔值：设置为min显示从滑块起始位置到手柄位置的范围；设置为max显示从滑块终点到手柄之间的范围。这有助于形象地表示旅馆客房的最小床位数。

最后，为选择列表绑定了一个单击事件，在用户直接修改选择列表的时候更新滑块。还可以隐藏选择列表，但是需要添加另一种形式的标签，显示选中的数字值。

插件还支持两种例子中未介绍的选项：

- 设置`animate: true`，在单击滑块某个位置的时候，手柄以动画的形式移动到目标位置。
- 设置`orientation: vertical`显示垂直滑块，代替默认的水平滑块。

还有其他一些事件，提供更细粒度的控制：

- `start`在滑动开始时调用。
- `stop`在滑动停止时调用。
- `change`在滑动停止且滑块值变化的时候调用；这个事件在滑块变动触发高代价操作（如向服务器发送请求、更新图形等）时特别有用。当然，它会使滑块的行为更不明显，因为滑动的时候没有实时的反馈。

11.5 输入范围约束值

11.5.1 问题

滑块很适合处理粗略的输入，并使其更加形象，但是不适合收集精确数据，如布局组件中的像素值，这类值需要以很小的增量（逐个像素）微调。对于标准输入控件，必须使用键盘：单击字段，删除当前值并输入一个新值，对每个增量都必须重复这一操作。

11.5.2 解决方案

jQuery UI spinner窗口组件（*http://jquery-cookbook.com/go/widget-spinner*）为输入控件添加了上下按钮解决这个问题，这样鼠标交互也能处理光标上下移动之类的键盘事件。

你所需要的就是一个常规文本输入控件：

```
<input id="value" name="value" />
```

然后对其应用Spinner插件：

```
$("#value").spinner();
```

这将创建和定位上下按钮并添加必要的键盘处理事件。

使用Spinner插件添加增/减量按钮，单击按钮或者设置焦点为输入字段并使用光标键均可。

这个插件还将输入限制为数字值——在输入框中输入abc，在控件失去焦点时它将被默认值所替代。除非另外指定，否则默认值为0。

11.5.3 讨论

Spinner插件提供几个选项进一步限制输入：

- `min`设置低限，例如，–10或者100。
- `max`设置高限，例如，10或者200。
- `stepping`限制增量值，例如5；默认为1。

当使用Spinner输入货币值时，可以用`currency`选项，在输入框中显示相应的符号。

下面的例子组合以上选项，创建一个捐赠表单：

```
<label for="currency">Currency</label>
<select id="currency" name="currency">
    <option value="$">US $</option>
    <option value="€">EUR€</option>
    <option value="¥">YEN¥</option>
</select>
```

```
<br/>
<label for="amount">Select the amount to donate:</label>
<input id="amount" name="amount" value="5" />
```

我们有一个用于货币的选择列表和输入金额的文本输入框：

```
var currency = $("#currency").change(function() {
    $("#amount").spinner("option", "currency", $(this).val()).blur();
});
$("#amount").spinner({
    currency: currency.val(),
    min: 5,
    max: 1000,
    step: 5
});
```

我们在货币选择列表上绑定修改事件，当选择改变时更新微调框的currency选项。

微调框本身用当前值初始化，同时设置min、max和step，将值设置在5～1000，增量为5。例如，5，10，15，20，以此类推。

Google地图整合

输入字段的值也可能是小数；在这种情况下，可以使用decimal选项指定小数点后允许的位数。在下面的例子中，显示一个Google地图，使用微调框指定经纬度值。

首先，包含Google Maps API脚本：

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?
sensor=false"></script>
```

然后，可以为微调框添加标记和实际的地图，以及一些简单的样式：

```
<style>
    #map { width:500px; height:500px; }
</style>

<label for="lat">Latitude</label>
<input id="lat" name="lat" value="44.797916" />
<br/>
<label for="lng">Longitude</label>
<input id="lng" name="lng" value="-93.278046" />

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

据此，可以初始化地图，并与微调框链接：

```
function latlong() {
    return new google.maps.LatLng($("#lat").val(),$("#lng").val());
}
function position() {
    map.set_center(latlong());
}
$("#lat, #lng").spinner({
    precision: 6,
    change: position
});
```

```
var map = new google.maps.Map($("#map")[0], {
    zoom: 8,
    center: latlong(),
    mapTypeId: google.maps.MapTypeId.ROADMAP
});
```

position函数将地图中心设置为从微调框获取的经度和纬度值。初始化时把decimal选项设置为6，并为change选项传递position函数。这样，地图在一个微调框出现变化时更新。然后用Google Maps API初始化地图。

这个例子中微调框的缺点是递增和递减只影响小数点之前的数字，所以滚动效果比较粗糙。increment选项将1以下的值舍入为1，所以它没有作用。

11.6 在后台上传文件

11.6.1 问题

文件上传是许多Web应用程序的一部分，但是浏览器对此的支持很差。最大的问题是缺乏对上传状态的反馈，同时用户的任何操作都会干扰上传。简单的进度栏能够改进反馈，但是需要在服务器端进行相当多的工作，而干扰性操作的问题仍然没有得到解决。

11.6.2 解决方案

为了改进解决方案，文件上传应该在后台进行。这就允许应用程序继续接受其他用户输入。

jQuery表单插件（*http://jquery-cookbook.com/go/plugin-form*）大大简化了从原生浏览器上传到Ajax后台上传之间的转换。采用如下的表单：

```
<form id="uploadform">
    <input type="file" id="fileupload" name="fileupload" />
    <input type="submit" value="Upload!" />
</form>
```

需要添加的就是对ajaxForm的调用：

```
$("#uploadform").ajaxForm();
```

但是，在后台上传而没有任何完成上传的反馈是不充分的，因此使用success选项显示成功上传的相关提示：

```
$("#uploadform").ajaxForm({
    success: function() {
        alert("Upload completed!");
    }
});
```

11.6.3 讨论

ajaxForm方法将自身绑定到表单的提交事件，这就允许它包含用于在Ajax请求中提交表单的按钮。后者在使用ajaxSubmit时不可用。ajaxSubmit方法在表单提交在其他地方处理（例如，验证插件）时很有用。为了整合验证和Ajax提交，ajaxSubmit应该在submitHandler选项中使用：

```
$("#commentform").validate({
    submitHandler: function(form) {
        $(form).ajaxSubmit({
            success: function() {
                $(form).clearForm();
                alert("Thanks for your comment!");
            }
        });
```

```
    }
});
```

除了alert以外，表单插件还提供了clearForm方法，该方法从表单中删除所有值，简化了用户上传其他文件的工作。

11.7 限制输入文本的长度

11.7.1 问题

限制文本区域中的字符数量是常见的工作，如Twitter评论所限制的140个字符和YouTube限制的500个字符。在用户提交表单之后告诉他输入字符太多令人沮丧，所以显示剩余字符的指示器很有意义。

11.7.2 解决方案

maxlength插件（*http://jquery-cookbook.com/go/plugin-maxlength*）在文本区域前后添加一个“Characters left: x”，解决了这个问题。该插件应用到文本输入框或者文本区域之后，寻找具有charsLeft类的元素，更新字符计数：

```
<form action="/comment">
   <p>Characters left: <span class="charsLeft">10</span></p>
   <textarea name="commentbody" maxlength="10"></textarea>
</form>

$('textarea').maxlength();
```

为了减少干扰，可以用jQuery创建必要的元素，使表单的标记更简单：

```
<form action="/comment">
   <textarea name="commentbody" maxlength="10"></textarea>
</form>

var textarea = $('textarea');
$('<p>Characters left: <span class="charsLeft">10</span></p>').insertBefore(textarea);
textarea.maxlength();
```

11.7.3 讨论

以Twitter为例，文本区域允许你超过140个字符的限制，但是你无法提交。这对粘贴不符合140个字符限制的长文本在以后编辑大有帮助。为了用maxlength插件获得类似效果，可以将hardLimit选项设置为false。但是，这不会影响实际提交，提交可以在其他地方（例如，秘诀11.1中的验证插件）处理。

该插件还支持用单词计数代替字符计数，做法是将words选项设置为true。

还可以设置feedback选项，而不让插件搜索默认的.charsLeft选择器。

下面的例子使用上述三个选项：

```
<form action="/comment">
    <textarea name="commentbody" maxlength="10"></textarea>
    <p><span>x</span> characters left</p>
</form>

$('textarea').maxlength({
```

```
    feedback: "p>span",
    hardLimit: false,
    words: true
});
```

11.8 在输入字段上方显示标签

11.8.1 问题

页面布局在输入元素前面没有足够的空间显示标签，输入的功能因此而产生混淆，只使用标题又不够醒目。

搜索和登录表单通常屈服于空间的限制。输入字段前面没有足够的可视空间来显示标签，但是没有标签，输入字段的功能就显得模糊。`title`属性不足以解决这个问题，因为它很难定位，需要用户将光标放在输入字段上并停在那里。

11.8.2 解决方案

最常见的例子——搜索字段，可以通过在字段内显示“search”来解决，浅灰色的字样是为了强调它只是一个标签，而不是实际的搜索文本。当焦点位于该字段的时候，就删除文本。当焦点离开字段时，除非输入其他文本，否则恢复默认的显示。

空间有限的登录表单是另一个较常见的例子，由用户名和密码字段组成。密码字段需要显示水印为普通文本，而输入的（或者浏览器预先填写的）密码必须进行模糊处理。

在两种情况下，都不应该将水印当作正常值提交。

水印（watermark）插件（*http://jquery-cookbook.com/go/widget-watermark*）在实际输入上方显示一个标签元素，当输入字段获得焦点时隐藏该标签，而在空字段失去焦点的时候再次显示，从而解决了这一问题。

这个解决方案在字段上方使用标签而不是修改字段内的文本，因此它也可以用于密码字段，而不需要在提交的时候清除水印值。

默认的用法是调用水印插件方法，并传递所要显示的值：

```
$("#search").watermark("Search");
```

11.8.3 讨论

除了向插件传递值之外，还可以使用元数据插件（*http://jquery-cookbook.com/go/plugin-metadata*）在标记中将显示文本指定为元数据，这种方法在使用多个水印或者在服务器端生成水印时更加实用：

```
<form id="loginform">
    <input type="text" id="email" name="email"
class="{watermark:'E-Mail Address'}" />
    <input type="password" id="password" name="password"
class="{watermark:'Your password'}" />
</form>

$("#loginform input").watermark();
```

元数据有一个缺点，它不能在渐进增强的基础上使用。为了改善这种情况，应该像常规表单一样使用标签元素，由插件将标签放在正确的位置上：

```
<form id="loginform">
    <div>
        <label for="email">E-Mail Address</label>
        <input type="text" id="email" name="email" />
    </div>
    <div>
        <label for="password">Your password</label>
        <input type="password" id="password" name="password" />
    </div>
</form>
```

在这个例子中，水印插件应用到标签而不是输入字段上：

```
$("#loginform label").watermark();
```

然后，该插件使用各个标签的`for`属性寻找相关的输入字段，将标签放在输入字段上。

11.9 根据内容增大输入字段

11.9.1 问题

文本区域是界面的一个部分，往往过大或者过小，这取决于用户的输入。如果文本区域太大，其他重要的元素就会落到视野之外；如果太小，用户就不得不进行滚动。

11.9.2 解决方案

使用弹性（elastic）插件（*http://jquery-cookbook.com/go/plugin-elastic*）从默认的较小高度开始，当用户输入一定数量的文本时自动增长其高度。

插件的用法很简单，从一个文本区域开始：

```
<textarea id="commentbody"></textarea>
```

对其应用插件：

```
$("#commentbody").elastic();
```

11.9.3 讨论

弹性插件在文本区域上绑定一个定时器和一个blur事件以观察变化。当内容变化时，它将内容复制到一个隐藏的文本区域，并应用原有的样式，计算新的高度，如果新高度超过原始文本区域的高度，启动一个动画改变高度。这使得文本区域可以随着内容添加或者删除而增大和缩小。

另一种方法是由用户调整文本区域的大小。Safari默认为任何文本区域提供这种那个功能。jQuery UI resizable插件（*http://jquery-cookbook.com/go/plugin-resizable*）在其他浏览器上也能添加同样的功能。从同一个文本区域开始，应用resizable插件，自定义handle选项，仅在右下角显示一个手柄：

```
$("#resizable").resizable({
    handles: "se"
});
```

利用上述语句并包含jQuery UI基本主题，就可以在文本区域下方显示手柄。添加如下CSS可以将手柄移到文本区域的右下角：

```
.ui-resizable-handle {
    bottom: 17px;
}
```

11.10 选择日期

11.10.1 问题

日期输入对于搜索事件、航班或者旅馆，以及在注册表单中输入生日来说是必需的。常见的解决方案是使用三个选择列表，分别用于日期、月份和年。这对于生日输入是可行的，但是在搜索某段时间内的航班时显得非常笨拙。

11.10.2 解决方案

jQuery UI的日期选择器（Datepicker，*http://jquery-cookbook.com/go/plugin-datepicker*）能够解决上述问题，它提供了日历和许多对各种应用程序进行优化的自定义选项。

默认的日期选择器只要简单地应用到输入字段即可生效：

```
<label for="startAt">Start at:</label>
<input type="text" name="startAt" id="startAt" />

$("#startAt").datepicker();
```

上述代码将绑定必要的事件，在输入字段获得焦点时显示日期选择器，默认日期是当前日期。Next和Previous按钮可以用于选择下一个月或者上一个月，月历则用来选择具体的日期。

为了更好地使用日期选择器，需要使其适应于应用程序。对于航班搜索的例子，可以假定用户搜索后三个月内的航班，因此，它一次性显示从当前日期下一周开始的三个月：

```
<label for="from">From</label>
<input type="text" id="from" name="from"/>
<label for="to">to</label>
<input type="text" id="to" name="to"/>
```

从两个输入字段开始，这两个字段都与相应的标签关联，然后向二者应用日期选择器：

```
var dates = $('#from, #to').datepicker({
    defaultDate: "+1w",
    changeMonth: true,
    numberOfMonths: 3,
    onSelect: function(selectedDate) {
        var option = this.id == "from" ? "minDate" : "maxDate";
        dates.not(this).datepicker("option", option, new Date(selectedDate));
    }
});
```

日期选择器的默认日期是当前日期加上一周，用defaultDate选项指定。通过changeMonth: true还显示了修改月份的一个选择列表。选项numberOfMonths: 3指定同时显示3个日历。

onSelect选项是用户选择日期时触发的一个事件。当起始日期选定时，把结束日期的minDate选项设置为起始日期，当结束日期选定时，设置起始日期的maxDate选项。

这样，用户可以任意选择两个日期，与选择另一个日期时，输入已经限制在明确的范围内。

11.10.3 讨论

默认情况下，当输入字段接收焦点时显示日期选择器。使用showOn选项，可以配置日历在单击输入字段旁边的日历图标时才显示：

```
$("#datepicker").datepicker({
    showOn: 'button',
    buttonImage: 'images/calendar.gif',
    buttonImageOnly: true
});
```

buttonImage选项指定用作按钮的一个图片的路径，buttonImageOnly指定仅使用该图片，而不是具有嵌入图片的按钮元素。

showOn选项还支持both值，表示在输入字段获得焦点和单击按钮时都显示日期选择器。

11.10.4 本地化

jQuery UI datepicker支持41个地区的本地化，以ui.datepicker-*xx*.js文件的形式提供，其中的*xx*是地区代码。每个文件都为$.datepicker.regional添加一个属性。ui.date picker-ar.js添加如下属性：

```
$.datepicker.regional['ar'] = {
    closeText: 'قل اغ!',
    prevText: 'قب اسل ا&#x3c;',
    nextText: 'لي اتل ا&#x3e;',
    currentText: 'م ويل ا',
    dayNames: ['الخم','الأربعاء', 'الثلاثاء', 'الاثنين', 'الأحد', 'السبت'
الجمعة', 'يس'],
    dayNamesShort: ['خم', 'أربعاء', 'ثلاثاء', 'اثنين', 'أحد', 'سبت'
'جمعة', 'يس'],
    dayNamesMin: ['خم', 'أربعاء', 'ثلاثاء', 'اثنين', 'أحد', 'سبت'
جمعة', 'يس'],
    dateFormat: 'dd/mm/yy',
    firstDay: 0,
    isRTL: true
};
```

要引用如下属性，可以将日期选择器初始化为阿拉伯地区：

```
$("#datepicker").datepicker($.datepicker.regional.ar);
```

使用$.extend也可以混合其他选项：

```
$("#datepicker").datepicker($.extend({}, $.datepicker.regional.ar, {
    changeMonth: true,
```

```
    changeYear: true
});
```

通过{}创建一个空对象字面量，然后使用$.extend将地区选项及changeMonth和changeYear值复制到空对象中，然后用该对象初始化日期选择器。

第12章　jQuery插件

Mike Hostetler

12.0 导言

jQuery JavaScript程序库的首要目标是成为开源世界中其他JavaScript程序库快速而简洁的替代品。追求这一目标的关键原则是确保jQuery核心满足大部分开发人员的需求，同时保持快速和简洁。jQuery核心可能无法完全满足开发人员的需求。开发人员也可以编写对jQuery核心功能的扩展，这些扩展可能对许多jQuery用户有帮助，但是不应该包含在jQuery核心中。

jQuery的设计使其可以用多种方法扩展。本章中的秘诀将把读者带入jQuery插件的世界。

12.1　从哪里寻找jQuery插件

12.1.1　问题

你试图用jQuery构建应用，其中需要一些jQuery核心中不存在的功能。这一问题以前的开发人员也可能碰到，所以你认为可能存在一个插件。你从哪里开始查找插件，又应该如何评估找到的插件呢？

12.1.2　解决方案

通过如下jQuery插件库搜索：

jQuery插件库

http://plugins.jquery.com

Google代码

http://code.google.com

GitHub

http://github.com

Google特殊查询

http://google.com

SourceForge

http://sourceforge.net

12.1.3　讨论

可以在多个网站上找到jQuery插件。因为jQuery插件的特性，某些开放源码托管网站可能吸引比其他网站更多的jQuery插件。此外，jQuery项目在*http://plugins.jquery.com*上有jQuery插件的一个集中存储库。

最好查看所有可用资源，收集多种可能使用的插件进行审核。用于解决相同问题的插件往往采用完全不同的方法，或者构建于不同版本的jQuery核心程序库。

当搜索jQuery插件时，寻找最近更新和最新版本插件的最佳步骤如下：

1. 通过jQuery插件存储库搜索

jQuery项目托管一个插件存储库（*http://plugins.jquery.com*），在本书编写时其中已经有超过1200个插件。大部分插件作者都将它们的插件提交到这个存储库。

jQuery插件存储库中托管的插件分为几个类别，这些类别有助于缩小搜索范围。插件可以采用多种分类，还必须按照API兼容性列出，确保你所找到的插件能够与特定版本的jQuery核心程序库协调一致。最后，你还可以按照发布日期浏览插件，这样你就能在喜爱的插件发布新版本时与时俱进。

2．通过GoogleCode搜索

GoogleCode（*http://code.google.com/*）托管提供了非常丰富的jQuery插件存储库。如果你在主插件存储库上找不到一个插件，往往可以在GoogleCode上找到。

3．通过GitHub搜索

GitHub（*http://github.com*）是代码托管界一颗冉冉升起的新星，许多jQuery插件作者正在转向它。越来越多的插件进入该网站，它也成为搜索特定插件时值得一试的网站。GitHub最吸引人的特征之一是利用Git源代码管理系统的功能，以友好的方式对存储库加以分类的能力。如果需要修改现有插件，利用GitHub的特性，能够以极具吸引力的方式跟踪上游的更新。

在GitHub上查找插件的最佳方式是利用GitHub杰出的搜索功能。GitHub支持多种高级搜索运算符。这些选项的更多细节可以在*http://github.com/search*上看到。在明确地搜索某一个jQuery插件时，用JavaScript搜索将返回最佳结果。

4．进行Google搜索

前面推荐的都是著名的插件来源，但是通过Google进行全网搜索也很有用。因为搜索的对象越来越多，所以需要对可能的结果进行筛选。使用几种建议的搜索方法能够更快地找到插件：

```
{searchterm} "jquery*.js" - Best practice plugin naming is jquery-{myplugin}.js or
jquery.{myplugin}.js

 {searchterm} "*jquery.js" - Alternate best practice plugin naming
```

5．通过SourceForge搜索

SouceForge（*http://sourceforge.net*）上托管的jQuery实际上很少。但是，该网站上的许多项目提供jQuery支持工具，例如，IDE代码完成扩展。如果你没有其他选择，或者打算搜索某些独特的内容，SourceForge是可以快速搜索的好地方。

12.2　何时应该编写一个jQuery插件

12.2.1　问题

搜索满足需求的现有jQuery插件之后，找到的插件可能无法满足你的需求，或者构造的方法让你无法正常使用。编写一个可以与有相同需求的人们共享的新jQuery插件是否值得？

12.2.2　解决方案

对这一问题没有固定的解决方案。可用jQuery插件已经很多，但是有些情况下，对于特定的需求找不到可用的插件。

在我看来，编写和发行自己的jQUery插件可以根据下面三点决定：

- 其他人是否可能有相同的问题？
- 你愿意提供什么样的支持？
- 你希望在什么程度上参与社区？

12.2.3　讨论

1．如果有潜在的受众，就构建一个插件

如果你面对的问题尚未解决，而且其他开发者也可能碰到相同的问题，那么在你之前其他人如何解决这个问题是关键点。假定你在此时已经完成了寻求解决方案的一些准备工作。在搜索过程中，在论坛帖子或者未回答的邮件列表问题上，能够找到对某个插件需求情况的线索。确定一个插件是否值得构建并不容易，决策最终取决于计划构建该插件的人。但是，对潜在受众的总体感觉值得探索。

自行构建和发布插件的另一个潜在原因是现有的插件符合你的需求，但是不能完全实现你所要的功能。如果是这种情况，编写一个补丁并将补丁发回给原作者，请求其包含到插件中，是值得考虑的。通过提交现有项目的补丁加入开放源码过程能够更高效地利用开发人员最珍贵的资源——时间。

2．了解和表达你愿意提供的支持水平

如果编写自己的插件是最佳选择，事先考虑和规划有助于确保你自己的开放源码过程顺利进行。不管何时，只要决定发布自己的代码，首要的关注点都是许可证。jQuery核心项目有MIT和GPL两个许可证，但是许多其他的开放源码许可证也值得考虑。在维基百科上（*http://en.wikipedia.org/wiki/Open_source_license*）能找到对错综复杂的开放源码许可证的透彻讨论。

其次，考虑和传达插件作者愿意为下载和使用代码的其他人提供的支持也很重要。选择简单地发布代码而不提供任何支持是有效的选择，这比由于担心潜在的支持

问题而自己保留代码要好得多。关键是沟通；在插件的注释中编写有关支持计划的简单说明大有帮助。

如果你愿意为自己发布的插件提供更深入的支持，几个杰出的源代码托管网站提供了一些特性，可以帮助你支持自己的插件。托管插件的最佳场所列表参见秘诀12.1。

3．规划其他人的参与

最后，要认真思考并评估是否愿意接受其他人的参与。参与是开放源码生态系统的关键组成部分，从发布插件开始就表达你的意图是明智的。允许参与的吸引力在于你可以从其他人的工作中获益。接受其他人参与的插件更容易吸引其他用户，一部分原因是这样显示了代码的活力，还有一部分原因是有活力的代码更值得信赖。

传达参与路径很关键。不管你的意图如何，当用户发现你发布的代码，这些代码就可能吸引某种参与。制定计划，以开放和公开的方式推动参与是必不可少的。最后的忠告：简单地发布你的电子邮件，允许人们发送评论和提问邮件，以此来推动参与是个坏主意，原因有二。首先，电子邮件不是能够向潜在用户表现活力的公开论坛，其次，这样做使插件的作者成为将参与活动整合到插件中的一个瓶颈。

12.3 编写第一个jQuery插件

12.3.1 问题

你已经决定要编写一个jQuery插件。如何用jQuery编写插件？应该遵循什么样的最佳实践？

12.3.2 解决方案

jQuery的设计使得编写插件非常简单直接。你可以编写方法或者函数扩展现有的jQuery对象。在包含jQuery代码库之后声明如下JavaScript，代码就可以使用新的自定义方法和函数。

1. 编写自定义jQuery方法

jQuery方法可以链接，因此可以利用jQuery选择器。jQuery方法通过用方法名称扩展jQuery.fn对象定义。因为jQuery对象必须能处理多种结果，你必须将自定义功能包装在对each()函数的调用中，以便将代码应用到所有结果：

```
jQuery.fn.goShop = function() {
   return this.each(function() {
      jQuery('body').append('<div>Purchase: ' + this.innerHTML + '</div>');
   });
};
```

访问这个新的插件和正常的jQuery调用一样简单，只需要利用新的方法名称：

```
jQuery('p').goShop();
```

2. 编写自定义jQuery函数

把函数附加到jQuery主对象中，设计为从jQuery选择之外调用：

```
jQuery.checkout = function() {
   jQuery('body').append('<h1>Checkout Successful</h1>');
};
```

这个新函数可以正常地操纵和调用：

```
jQuery.checkout();
```

12.3.3 讨论

在主jQuery对象上附加新方法和函数是jQuery的一个强大功能。程序库中内置的许多核心方法也使用相同的技术。利用jQuery中现有的这一基础，jQuery用户和插件

用户都可以快速而简洁地添加新功能，扩展现有功能，并以最适合的形式构造jQuery代码。这种灵活性是关键的特征，使jQuery及其插件得到更广泛的受众。通过新方法还是函数来扩展jQuery的选择取决于开发人员的需求。一般来说，最好是以添加新方法为中心来扩展jQuery，因为这使得新方法可以与其他方法链接，并且允许方法中的代码利用jQuery的选择器引擎。

12.4 向插件传递选项

12.4.1 问题

你的第一个插件在jQuery中添加了一个方法。但是，如果正确地输出其中的一些选项，对用户更有帮助。将选项传递给子定义方法的最佳方法是什么？

12.4.2 解决方案

选项最好通过选项对象传递到自定义插件方法中。使用一个选项对象传递参数能够生成更清晰，更容易使用的代码，并且为将来提供更好的灵活性。

当允许插件使用选项时，提供合理的默认值是明智的。提供合理的默认值之后，插件为用户提供覆盖默认值的方法也很重要。通过声明默认选项对象，以用户提供的选项和jQuery extend()方法覆盖默认选项，就很容易实现上述两个目标，然后可以在代码中利用这些选项：

```
jQuery.fn.pulse = function(options) {
   //合并传入的选项和默认选项
   var opts = jQuery.extend({}, jQuery.fn.pulse.defaults, options);

return this.each(function() {
   // 跳动!
   for(var i = 0;i<opts.pulses;i++) {
      jQuery(this).fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
   }

      //重置为正常值
      jQuery(this).fadeTo(opts.speed,1);
   });
};

// 跳动插件默认选项
jQuery.fn.pulse.defaults = {
   speed: "slow",
   pulses: 2,
   fadeLow: 0.2,
   fadeHigh: 1
};
```

通过指定选项的默认值，使用插件的开发人员在调用函数时就可以提供尽可能少的选项。在定义了插件入口方法之后放置选项默认值很重要；否则，你将会碰到一个错误：

```
// 仅覆盖一个选项
jQuery('p').pulse({pulses: 6});

// 覆盖所有选项
jQuery('p').pulse({speed: "fast", pulses: 10, fadeLow: 0.3, fadeHigh: 0.8});
```

最后，将选项指定为插件函数中作为子插件函数附加的对象，默认选项在项目中只能覆盖一次。开发人员以后可以指定自己的默认选项组，最大限度地减少获得想要的行为时必须的代码量：

```
// 上面包含插件代码

// 重置默认跳动选项
jQuery.fn.pulse.defaults = {
   speed: "fast",
   pulses: 4,
   fadeLow: 0.2,
   fadeHigh: 1
};
//在这个调用中将使用新的默认值
jQuery('p').pulse();
```

12.4.3 讨论

在插件中支持选项是为插件带来极大灵活性的强大方法。支持丰富选项集的插件更可能符合广泛受众的需求，执行更多种类的任务，而且通常比不支持选项的插件更流行。

在插件中包含一组默认选项是给使用插件的开发人员带来灵活性，以及插件实现方式选择的另一种方法。默认选项还带来了一个方便的特性：插件可以始终依赖定义的某些选项，从而减少了检查某个选项是否传递的代码量。这使插件用户在每次调用插件时，能够覆盖单个选项、多个选项甚至所有选项。最后，通过在jQuery对象中附加默认选项，这些选项可以在全局范围内覆盖选项，为用户提供另一种新颖且具有创造性的工具。

12.5 在插件中使用$快捷方式

12.5.1 问题

其他JavaScript插件库使用$快捷方式。jQuery本身只将$作为快捷方式，主对象命名为jQuery。如何确保插件保持与其他插件和程序库的兼容性？

12.5.2 解决方案

jQuery本身使用$函数作为jQuery对象的自定义别名。当jQuery设置为兼容模式时，它将$别名的控制权传回给定义$的原始程序库。插件可以使用相同的技术。

将插件包装在一个匿名函数中并立即执行该函数，$快捷方式就保留在插件内部。插件之外的代码可以正常使用$。在插件中，$将正常引用jQuery对象：

```
;(function($) {
   $.fn.pulse = function(options) {
      //将传入的选项与默认选项合并
      var opts = $.extend({}, $.fn.pulse.defaults, options);

      return this.each(function() {
         // 跳动!
         for(var i = 0;i<opts.pulses;i++) {
            $(this).fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
         }
         // 重置为正常值
         $(this).fadeTo(opts.speed,1);
      });
   };

   //跳动插件默认选项
   $.fn.pulse.defaults = {
      speed: "slow",
      pulses: 2,
      fadeLow: 0.2,
      fadeHigh: 1
   };
})(jQuery);
```

12.5.3 讨论

将你发布的代码包装在一个匿名函数中非常简单直接，这一步骤添加了多个特性，确保插件代码能够更好地运行于用户所在的更加广阔的世界里。

在函数定义的开始添加一个分号有助于避免其他开发人员在他们的程序库中忘记包含最后的分号。JavaScript语言默认用换行符打断语句，但是许多用户用精简工具将项目中的整组JavaScript压缩为单个文件。这一过程删除了行结束符，可能导致紧接着的代码出错。添加初始的分号是防止这种情况的简单技巧。

左圆括号之后就是匿名函数的定义。在匿名函数中，定义一个函数，该函数传递一个用来代替全名jQuery对象的变量。在这种情况下，将$当作一个变量使用。定义附加函数是必需的，这是因为JavaScript对作用域的处理方式。在Java和C++等更传统的语言中，作用域限制在块语句中。在JavaScript中，作用域包装在函数中。因此，这里使用函数实际上是为了建立一个作用域边界，以便在这个边界内定义插件。

接下来是插件的新版本，唯一的改动是jQuery对象的使用方式。因为匿名地包装这一插件，并限制了$变量的作用域，所以现在可以自由地使用$而不会与任何其他代码冲突。

最后一行将作用域函数和匿名函数分别包装在右方括号和右圆括号中，并在匿名函数定义之后立即调用它。这是告诉函数传入jQuery对象的地方，该对象在函数中重命名为$。最后，用分号结束新语句，避开JavaScript精简和压缩工具导致的错误。

$快捷方式在编写JavaScript代码时很有用。它减小了代码尺寸，促进了好的代码设计，从而变得极端流行并为人熟知。因此，许多程序库使用了$快捷方式，将其纳入自己的上下文。由于每种程序库都支持自己的$快捷方式版本，很容易发生冲突。通过将插件代码包装在一个匿名函数中，可以确保插件保持$快捷方式周围代码的作用域级别，减少和其他JavaScript程序库冲突的可能性。

前面已经说明过，在匿名函数中包装插件有一个副作用，就是创建闭包。在JavaScript中利用闭包有助于为需要定义的方法或者变量提供正确的命名空间，进一步减少了变量或者函数名称与其他代码冲突的几率。

12.6 在插件中包含私有函数

12.6.1 问题

你的插件代码不断增长，需要进行组织。如何实现插件之外无法访问的私有方法？

12.6.2 解决方案

利用秘诀12.4开始介绍的插件设计模式，在用于包装插件的匿名函数中可以正常定义私有函数。因为私有函数处于匿名函数内部，所以外部代码无法看到它。外部代码只能看到附加到jQuery对象中的函数或者方法。

```
;(function($) {

   $.fn.pulse = function(options) {

      //合并传入选项和默认选项
      var opts = $.extend({}, $.fn.pulse.defaults, options);

      return this.each(function() {
         doPulse($(this),opts);
      });
   };

    function doPulse($obj,opts) {
      for(var i = 0;i<opts.pulses;i++) {
         $obj.fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
      }

      //重置为正常值
      $obj.fadeTo(opts.speed,1);
}
   // 跳动插件默认选项
   $.fn.pulse.defaults = {
      speed: "slow",
      pulses: 2,
      fadeLow: 0.2,
      fadeHigh: 1
   };

})(jQuery);
```

12.6.3 讨论

因为现在已经将插件包装在匿名函数中，在插件中定义私有函数很简单，只需要像平常一样添加一个新的函数。

用公共和私有方法分类和组织插件为用户和插件作者带来了许多好处。随着插件不断成熟和从社区中得到反馈，可以利用公共和私有方法在不同的插件版本中提供一

致的API。API的一致性是插件成功的重要因素。

随着插件的成长，将代码分解为私有和公共信息的能力在代码组织中也有显著的好处。精心组织的代码更容易阅读、维护和测试。经过严格测试的清晰代码出错的可能性也更小。

12.7 支持元数据插件

12.7.1 问题

许多插件利用元数据插件将自定义选项传递给方法。如何整合元数据插件？

12.7.2 解决方案

利用元数据插件很简单：检查插件是否可用，然后用元数据参数扩展插件选项。使用这一技术，可以在调用插件时提供默认选项，并通过标记中的元数据覆盖每个操作对象的默认选项：

```
<!--包含元数据插件 -->
<script type="text/javascript" src="metadata/jquery.metadata.js"></script>

<!-- 包含元数据的标记示例-->
<p class="{pulses: 8, speed: 'slow'}">Starship Enterprise</p>
<p>Battlestar Galactica</p>
<p class="{speed: 100}">Serenity</p>

;(function($) {
   $.fn.pulse = function(options) {
     //将传入的选项与默认选项合并
     var opts = $.extend({}, $.fn.pulse.defaults, options);

     return this.each(function() {

       //为这个特定节点合并元数据元素
       var o = $.metadata ? $.extend({}, opts, $.metadata.get(this)) : opts;

        doPulse($(this),o);
     });
   };

   function doPulse($obj,opts) {
     for(var i = 0;i<opts.pulses;i++) {
        $obj.fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
     }

     //重置为正常值
     $obj.fadeTo(opts.speed,1);
   }

   // 跳动插件默认选项
   $.fn.pulse.defaults = {
     speed: "slow",
     pulses: 2,
     fadeLow: 0.2,
     fadeHigh: 1
   };
})(jQuery);
```

12.7.3 讨论

包含元数据插件是jQuery插件可以在另一个插件上构建的极好例证。jQuery插件生态系统很大，你很有可能利用其他的插件。

为了包含和使用元数据插件，首先必须在脚本中包含它。元数据插件和jQuery一样在Google代码上托管。元数据插件的工作原理是允许在HTML中嵌入附加数据，同时仍然产生有效的HTML。利用这一点，让用户在需要操作的项目的类元素中嵌入与元素相关的选项。

选项用标准的JSON嵌入HTML。可以嵌入所有选项，也可以一个都不嵌入，这取决于你的用户。使用元数据插件的其他方法和选项在其文档页面（*http://docs.jquery.com/Plugins/Metadata*）上可以找到描述。

在插件中，首先检查用户是否包含元数据插件，这能确保我们保持这一可选的附加特性，并在必要时提供后向兼容性。因为元数据插件操作单个元素，所以分解选项的处理操作。第一步是调用插件时提供的选项。这些选项由默认选项扩展，创建了插件第一次实例化时的出发点。第二步是用为每个元素定义的元数据扩展这些局部默认选项。如果元数据插件存在，用元数据选项扩展局部默认选项所需要的就是这些了。

元数据插件为插件用户提供了传递参数的另一种选择。为潜在用户提供选项是展示你对插件的投入，成为jQuery生态系统好公民的极好手段。元数据插件还通过将自定义选项嵌入HTML元素中，很好地减少用户的编码量。

12.8 为插件添加静态函数

12.8.1 问题

除了通过jQuery函数使插件可用，你还希望输出一个静态函数。如何为jQuery插件添加一个静态函数？

12.8.2 解决方案

为插件添加一个静态方法需要按照添加方法的相同方式扩展jQuery对象。不同之处只是这些函数在调用时不使用jQuery选择器：

```
;(function($) {
   $.fn.pulse = function(options) {
      //将传入的选项与默认选项合并
      var opts = $.extend({}, $.fn.pulse.defaults, options);

      return this.each(function() {

         // 为特定节点合并元数据元素
         var o = $.metadata ? $.extend({}, opts, $.metadata.get(this)) : opts;

         doPulse($(this),o);
      });
   };

   function doPulse($obj,opts) {
      for(var i = 0;i<opts.pulses;i++) {
         $obj.fadeTo(opts.speed,opts.fadeLow).fadeTo(opts.speed,opts.fadeHigh);
      }

      // 重置为正常值
      $obj.fadeTo(opts.speed,1);
   }

   // 定义添加的基础对象
   $.pulse = {};
     // 静态函数
     $.pulse.impulse = function($obj) {
       var opts = {
       speed: 2500,
       pulses: 10,
       fadeLow: 0.2,
     fadeHigh: 0.8
     };
    doPulse($obj,opts);

    }
    // 静态函数
    $.pulse.warpspeed = function($obj) {
      var opts = {
        speed: 25,
        pulses: 100,
```

```
      fadeLow: 0.2,
      fadeHigh: 0.8
    };
    doPulse($obj,opts);
  }

  //跳动插件默认选项
  $.fn.pulse.defaults = {
    speed: "slow",
    pulses: 2,
    fadeLow: 0.2,
    fadeHigh: 1
  };
})(jQuery);
```

调用插件中的静态方法非常简单，只要显式地传递一个需要操作的有效对象：

```
// 在第一个返回元素上调用impulse方法
jQuery.pulse.impulse(jQuery('p:first'));

// 在第一个返回元素上调用warpspeed方法
jQuery.pulse.warpspech(jQuery('p:first'));
```

12.8.3 讨论

在插件作用域内添加一个静态函数只需要为插件外的代码添加调用它的方法。这可以通过将函数附加到jQuery对象中实现。

在前一个例子中，已经添加了一个命名空间对象以更好地组织代码。如果插件需要的是一个静态函数，完全可以输出静态函数，而不需要添加命名空间对象。添加命名空间对象之后，只要和平常一样定义函数，并将它们附加到创建的命名空间对象中即可。这样就将函数输出到全局命名空间，同时允许函数的内容访问私有函数和变量。

利用这个静态函数很简单，只要用它附加到的jQuery对象调用它就可以了。调用这个函数不需要使用jQuery选择器，所以为了在一个DOM元素上进行操作，该元素必须显式地传递给函数。

附加到jQuery对象中的静态函数是jQuery程序库灵活性的另一个例子。整个插件可以通过添加以新的有趣方法扩展jQuery核心的静态函数构造。静态函数可以是提供给插件的入口点，也可以是一个有用的简单快捷方法，这样的打包方式使它更容易与其他开发人员共享。不管是何种需求，在构建自己的jQuery插件时，静态函数都可能是有用和强大的工具。

12.9 用Qunit对插件进行单元测试

12.9.1 问题

你打算通过为jQuery插件创建单元测试，提升其质量和可靠性。如何编写和交付jQuery插件的测试？

12.9.2 解决方案

为jQuery插件编写单元测试的最简单方法是利用QUnit，这是jQuery项目使用的同一个单元测试框架。可以用QUnit编写JavaScript测试，然后把它们和插件一同交付给用户，在他们自己的浏览器中运行：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd ">
<html xmlns="http://www.w3.org/1999/xhtml " xml:lang="en" lang="en">
<head>
   <script type="text/javascript" src="../jquery-1.3.2.min.js"></script>
   <script type="text/javascript" src="metadata/jquery.metadata.js"></script>
   <script type="text/javascript" src="jquery.pulse.js"></script>

   <link rel="stylesheet"
href="http://jqueryjs.googlecode.com/svn/trunk/qunit/testsuite.css " type="text/css"
media="screen" />
   <script type="text/javascript"
src="http://jqueryjs.googlecode.com/svn/trunk/qunit/testrunner.js "></script>
</head>
<body>

   <script type="text/javascript">
     module("Testing the jQuery Pulse Plugin");
     test("Test Pulse with basic options", function() {
        $("div.starship").pulse();
     equals($("#enterprise").css("opacity"),1,"The element should be visible");
     equals($("#galactica").css("opacity"),1,"The element should be visible");
   });

   test("Test Impulse", function() {
     $.pulse.impulse($("#galactica"));

     equals($("#galactica").css("opacity"),1,"The element should be visible");
   });

   test("Test Warp Speed", function() {
     $.pulse.warpspeed($("#enterprise"));

     equals($("#enterprise").css("opacity"),1,"The element should be visible");
   });
   </script>

   <div id="main">
     <div class="starship" id="enterprise">USS Enterprise - NC-1701-A</div>
     <div class="starship" id="galactica">Battlestar Galactica</div>
   </div>
```

```
</body>
</html>
```

12.9.3 讨论

学习如何有效地测试代码超出了本章的范围。前一个例子中编写的测试是为了简单地说明单元测试所能完成的任务。第18章将更详细地介绍单元测试，特别是QUnit框架。有关如何使用QUnit，可以测试的程序类型以及如何有效地测试代码的内容请参见第18章。

将单元测试和你的插件一同交付是向开发人员展示你对所发布的代码的成功和稳定性所做工作的另一种好方式。这能建立用户对你的信任，说明你的插件是jQuery插件生态系统的好成员。测试还使插件用户更容易找到在其他运行时环境（如不同的浏览器）中可能滋生的缺陷。这样，作为插件作者的你能够更好地处理在有效的试验台上找到的bug，而作为插件开发者的你也能直接处理找到的bug。

第13章　从头开始创建界面组件

Nathan Smith

13.0 导言

jQuery UI窗口组件集提供了许多现成的功能，但是你有时可能选择创建符合特定需求的自定义元素。你可能希望更好地控制HTML标记，或者只是想要得到轻量级的JavaScript代码。不管哪一种原因，本章将展示如何简洁地编写项目中所用的自定义组件。这些秘诀在编写的时候注重易用性，相比配置更重视简洁性。

秘诀13.1展示如何创建自定义工具提示，用于需要通过提供附加内容或者指南吸引用户注意力的场合。秘诀13.2将说明如何构建文件树样式的菜单，使用户能够深入探索网站层次结构。在秘诀13.3中你将学习如何创建垂直折叠的组件。秘诀13.4将说明如何使用页间链接和对应的目标创建文档标签。秘诀13.5说明如何通过相应的操作创建基本模态窗口。秘诀13.6解释如何构建简单的下拉式菜单。秘诀13.7研究按钮控制的图片旋转器的创建，重用了秘诀13.4中的页间链接技术。秘诀13.8从秘诀13.3中吸取经验，创建水平面板代替垂直折叠组件。

注意

下面的范例贯穿本章，在各个例子中不再特意强调。

每个秘诀一开始检查文档中是否确实存在必要的元素。如果不存在则退出函数。如果该条件不符合就没有必要进行什么后续操作，这样就阻止了不必要地执行代码：

```
// 元素存在吗?
if (!$('#foobar').length) {

 // 如果不存在,退出
 return;
}
```

在本章中都会使用一段通用的代码来撤销只用于触发JavaScript事件的后续链接。`blur()`方法用于去掉虚线边框，否则这个边框会一直存在（直到用户单击其他位置），`return false`告诉浏览器不要跳转到链接的`href`属性所指网页：

```
// Nofollow.
this.blur();
return false;
```

为了真正启动动态功能，每个秘诀都以对jQuery的`document.ready()`函数的调用而结束，确保在试图应用事件监听器等功能之前DOM（不一定包括所有图像资源）已经加载完毕：

```
//启动各项操作
$(document).ready(function() {
 init_foobar();
});
```

有些秘诀在HTML文档的`<head>`元素中有如下代码。大部分人认为`document.write()`是JavaScript中过时的方法，因为这个命令强制浏览器暂停以渲染

页面。但是，当使用CSS预先隐藏的内容以后将通过JavaScript显示时，这正是我们需要的效果：

```
<script type="text/javascript">
/* <![CDATA[ */
document.write('<link rel="stylesheet" type="text/css" href="preload.css" />');
/* ]]> */
</script>
```

本质上，在页面开始渲染之前，应用到<head>的一个CSS文件预先隐藏所有的内容，这些内容将在以后用户与页面交互的时候显示。我们用JavaScript编写CSS引用的原因是为了在禁用JavaScript的时候，所有内容可见且完全可以访问。有关这种技术的更多内容，参阅Peter-Paul Koch的“Three JavaScript articles and one best practice”（*http://www.quirksmode.org/blog/archives/2005/06/three_javascrip_1.html#link4*）。

13.1　创建自定义工具提示

13.1.1　问题

有时，图形元素或者界面特征可能需要进一步的说明，但是因为空间的限制（或者美学的考虑），设计人员可能不希望因为添加解释性文本而占据了宝贵的屏幕空间。在这种情况下有必要为用户提供指导，当用户刚入手时需要这种指导，而在熟悉了界面之后他们的需求就会减少。在这类情况下，工具提示是理想的解决方案。但是，HTML用于创建工具提示的资源很少，而title="..."属性往往不能满足我们的要求。

工具提示对于用户界面说明可能是一个好的解决方案，尤其是与某种可以设置的用户首选项（如“不要再次显示”）绑定的时候。但是，常常滥用动态的工具提示，最明显的是在博客上，页面上每个具有title="..."属性的元素在光标移过的时候都会显示工具提示。这样的情况应该避免，因为如果通过工具提示，所有元素都被当作特殊情况对待，那么提示的重要性也就下降了，实际上，页面上没有任何东西得到强调。这等同于大声喊出一句话中的每个单词。和所有Web项目一样，内容的上下文应该决定所采用的方法，反之则不然。

13.1.2　解决方案

为了解决上述问题，可以使用jQuery获得页面上感兴趣的区域中的光标位置，然后动态地在离原点一定距离的位置上放置一个 元素，该元素包含操作说明、附加信息（在电子商务中）或者任何开发人员需要显示的信息。这可以在</body>结束标记之前创建一个动态生成的 ，允许它拥有比页面其他元素更高的z-index来实现，如图13-1所示。此外，为了使工具提示获得优先，可以明确地指定极高的z-index——9999。

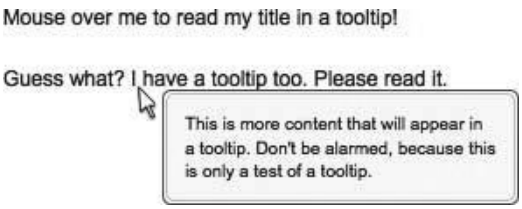


图13-1　用jQuery生成的工具提示

1．工具提示——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Creating Custom Tooltips</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="tooltip.css" />
<script type="text/javascript" src="../_common/jquery.js"></script>
<script type="text/javascript" src="tooltip.js"></script>
</head>
```

```
<body>
<div id="container">
   <p>
     <span class="tooltip" title="This is my title. There are many like it, but
         this one is mine. You will see it as you hover your mouse over me.">
        Mouse over me to read my title in a tooltip!
     </span>
   </p>
   <p>
     <span class="tooltip" title="This is more content that will appear in a
         tooltip. Don't be alarmed, because this is only a test of a tooltip.">
        Guess what? I have a tooltip too. Please read it.
     </span>
   </p>
</div>
</body>
</html>
```

2. 工具提示——jQuery代码

```
// 初始化
function init_tooltip() {

   // 元素存在吗?
   if (!$('.tooltip').length) {

      //如果不存在则退出
      return;
   }

   //插入工具提示(隐藏)
   $('body').append('<div id="tooltip_outer"><div id="tooltip_inner"></div></div>');

   // 空变量
   var $tt_title, $tt_alt;

   var $tt = $('#tooltip_outer');
   var $tt_i = $('#tooltip_inner');

   //监听悬停
   $('.tooltip').hover(function() {

      // 存储title,清空它
      if ($(this).attr('title')) {
         $tt_title = $(this).attr('title');
         $(this).attr('title', '');
      }

      // 存储alt,清空它
      if ($(this).attr('alt')) {
         $tt_alt = $(this).attr('alt');
         $(this).attr('alt', '');
      }

      // 插入文本
      $tt_i.html($tt_title);

      // Show tool tip.显示工具提示
      $tt.show();
   },
   function() {
```

```
      // 隐藏工具提示
      $tt.hide();

      // 清空文本
      $tt_i.html('');

      // 修复title
      if ($tt_title) {
         $(this).attr('title', $tt_title);
      }

      // 修复alt
      if ($tt_alt) {
         $(this).attr('alt', $tt_alt);
      }

   // 监听移动
   }).mousemove(function(ev) {

      // 事件坐标
      var $ev_x = ev.pageX;
      var $ev_y = ev.pageY;

      //工具提示坐标
      var $tt_x = $tt.outerWidth();
      var $tt_y = $tt.outerHeight();

      // 元素坐标
      var $bd_x = $('body').outerWidth();
      var $bd_y = $('body').outerHeight();

   //移动工具提示
      $tt.css({
         'top': $ev_y + $tt_y > $bd_y ? $ev_y-$tt_y : $ev_y,
         'left': $ev_x + $tt_x + 20 > $bd_x ? $ev_x-$tt_x-10 : $ev_x + 15
      });
   });
}

// 启动所有操作
$(document).ready(function() {
   init_tooltip();
});
```

13.1.3 讨论

值得一提的是，`$('.tooltip')`并不是读取元素的最高效方式。出于演示目的，页面上的所有标记都进行了解析，这与`document.getElementsByTagName('*')`等价。根据文档的大小和浏览器种类，这样做可能相当慢。所以，当实际使用这些代码时，一定要指定所搜索的标记。例如，可以使用`$('a.tooltip, span.tooltip')`代替`$('.tooltip')`。虽然更现代的浏览器将把这种类选择器映射到`getElementsByClassName`或`querySelectorAll`（如果可用的话），但是较老的浏览器必须首先循环读取标记名称，然后确定相关的类是否存在。

假定存在一个或者多个匹配`class="tooltip"`的元素，将动态标记附加到页面的最后，正文结束标记之前。因为在CSS文件中对`#tooltip_outer` ID应用`isplay: none`，所以这些标记还不能看到。

接下来，创建空变量$tt_title和$tt_alt。这些变量用于临时存储匹配的class="tool tip"元素的title和alt(如果存在)属性。聪明的读者可能感到奇怪：“我们感兴趣的不是title属性吗？为什么要关心alt?”这个问题问得好。除了title之外，还存储alt属性，这只是为了防止class="tooltip"用于图片的情况。Internet Explorer会显示自己的工具提示，包含alt的内容，而我们不希望出现这种情况。

剩下的代码处理class="tooltip"元素。当光标悬停于这类元素之上时，存储title和alt属性，然后将两个属性设置为空字符串。这样，浏览器的默认工具提示就不会干扰自定义工具提示。把title属性的内容复制到#tooltip_inner，然后显示#tooltip_outer。

同样，当鼠标离开目标元素时，我们希望撤销鼠标第一次进入时发生的操作。隐藏#tooltip，把#tooltip_inner内容设置为空串，而title和alt属性恢复为原始值。

最后，.mousemove()方法在光标进入class="tooltip"元素范围内时监控它的移动。工具提示相对于鼠标位置偏移量，出现在光标的右侧；也就是说，除非工具提示有超出浏览器宽度的危险。在这种情况下，可能出现水平滚动条，而我们不想这样。为了避免这样的混乱，用一些程序逻辑将工具栏换到光标的左侧。垂直方向也一样，如果工具提示离页面底部太远，它将换到鼠标指针的上方。

13.2　使用文件树扩展器导航

13.2.1　问题

在有多层信息架构的内容密集网站上，有时候需要表现多级的嵌套数据。如果所有信息全部显示，将很难控制并且占据过多的页面垂直空间。这时就需要文件树范式。这种功能在Windows资源管理器（不要和Internet Explorer混淆）的桌面UI上最容易看到，用户可以展开和折叠目录层次。

13.2.2　解决方案

通过在嵌套的无序列表上使用jQuery的后代元素选择器，可以根据需要隐藏/显示树结构的多余部分。这可以通过为顶级的无序列表添加class="tree"并组合使用CSS和JavaScript显示子级别来实现，这样会形成类似图13-2中所示的一棵树。此外，使用事件委托支持许多层次，避免为多个元素附加事件监听器的开销。作为替代，事件在最高级别的<ul class="tree">元素上通过jQuery的.live()方法捕捉。

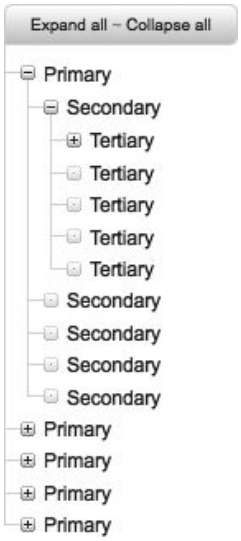


图13-2　通过文件树表现多层数据

1．文件树——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us" lang="en-us">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Navigating a File Tree Expander</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="tree.css" />
<script type="text/javascript">
/* <![CDATA[ */
document.write('<link rel="stylesheet" type="text/css" href="preload.css" />');
```

```
/* ]]> */
</script>
<script type="text/javascript" src="../_common/jquery.js"></script>
<script type="text/javascript" src="tree.js"></script>
</head>
<body>
<div id="container">
   <p class="tree_controls">
     <a href="#" class="expand_all">Expand all</a> ~
     <a href="#" class="collapse_all">Collapse all</a>
   </p>
   <ul class="tree">
     <li>
        <a href="#" class="tree_trigger"> </a> Primary
        <ul class="tree_expanded">
          <li>
             <a href="#" class="tree_trigger"> </a> Secondary
             <ul class="tree_expanded">
               <li>
                  <a href="#" class="tree_trigger"> </a> Tertiary
                  <ul>
                    <li>
                       <span class="tree_slug"> </span> Quaternary
                    </li>
                    <li>
                       <span class="tree_slug"> </span> Quaternary
                    </li>
                    <li>
                       <span class="tree_slug"> </span> Quaternary
                    </li>
                    <li>
                       <span class="tree_slug"> </span> Quaternary
                    </li>
                    <li>
                       <span class="tree_slug"> </span> Quaternary
                    </li>
                  </ul>
                  ...
               </li>
               ...
             </ul>
             ...
          </li>
          ...
        </ul>
        ...
     </li>
     ...
   </ul>
</div>
...
</body>
</html>
```

2．文件树——jQuery代码

```
//初始化
function init_tree() {

   //元素存在吗?
   if (!$('ul.tree').length) {

      // 如果不存在则退出
```

```
    return;
  }

  //展开和折叠
  $('p.tree_controls a.expand_all, p.tree_controls a.collapse_all').click(function() {

    // 查看类
    if ($(this).hasClass('expand_all')) {
        $(this).parent('p').next('ul').find('a.tree_trigger')
               .addClass('trigger_expanded')
               .end().find('ul').addClass('tree_expanded');
        return false;
    } else {
        $(this).parent('p').next('ul').find('a.tree_trigger')
               .removeClass('trigger_expanded')
               .end().find('ul').removeClass('tree_expanded');
    }

    // Nofollow.
    this.blur();
    return false;
});

// 监听文件树的单击
$('ul.tree a.tree_trigger').live('click', function() {
  // 下一个<ul>是隐藏的吗?
  if ($(this).next('ul').is(':hidden')) {
    $(this).addClass('trigger_expanded').next('ul')
           .addClass('tree_expanded');
  } else {
    $(this).removeClass('trigger_expanded').next('ul')
           .removeClass('tree_expanded');
  }

  // Nofollow.
  this.blur();
  return false;
  });

  // 为最后一个<li>添加类
  $('ul.tree li:last-child').addClass('last');

  // 修改触发器状态
  $('ul.tree_expanded').prev('a').addClass('trigger_expanded');
}
// 启动
$(document).ready(function() {
  init_tree();
});
```

13.2.3 讨论

文件树代码从在类名为expand_all和collapse_all的链接附加事件处理程序开始。如果单击某一个链接，则向上遍历DOM到双亲节点<p>，通过下一个<ul>，然后向下进入其子节点。每个具有class="tree_trigger"属性的子链接接受trigger_expanded类，后续的<ul>接受tree_expanded类。这些类名与改变其外观的CSS规则对应。对于触发链接，它们拥有一个扩展的图标。而对于列表，它们现在用display: block代替display: none。

“实时”事件监听器监听树中的单击。本质上，监听的是<ul class="tree">中的单击，然后确定单击是否发生在具有class="trigger"属性的链接上。如果是，执行相关的代码。使用.live()的好处和直接为每个链接添加单击处理程序相反：代码与所有匹配条件的现有和将来的元素相关。这是双重的好处：你不用浪费时间为许多元素附加事件监听器，而且如果动态内容通过Ajax插入，它们也会受到“实时”事件监听器的影响。

接下来，通过JavaScript为每个双亲的:lastchild添加一个样式钩子class="last"。这样就能通过一条浅灰色的线段，定位模拟树的链接性的背景图像。最后，如果在页面加载时，任何子元素<ul>已经用class="tree_expanded"硬编码为可见，遍历DOM，为最近的触发链接添加class="tree_trigger_expanded"类。

13.3 展开折叠控件

13.3.1 问题

人们使用折叠控件的情形可能类似于文件树。两者的样式很类似：附加信息开始时不可见，在用户进一步交互的时候显示。但是，它们之间也有不同，折叠控件不是用来包含整个数据分类的，而更多地用作将注意力吸引到网站或者产品多个侧面的一种新颖界面。折叠控件的一个例子可以在*http://www.apple.com/iphone*上看到。各个信息面板可以在用户空闲的时候展开，不需要完全占据分配给边栏的垂直空间。折叠就像图书馆里的高密度或者活动书架一样节约空间，使得每个通道能够存放多个架子，而不像固定货架那样，每个通道上只能摆放一个。

值得注意的是，jQuery UI折叠组件是高度可自定义的，并且可以指定与其余UI部件相匹配的主题/皮肤。你可以在*http://jqueryui.com/demos/accordion*上看到折叠的示例。使用官方窗口组件的好处是能得到jQuery UI的正式支持，并且能够持续发展，变得更加健壮。潜在的缺点是如果你所需要的只是一个简单的折叠组件，就需要较多的额外代码。反过来说，选择构建自定义折叠组件的原因之一就是为了较小的代码占用量。这造成了无法实现像素级别的精确动画以及必须设置每个折叠面板的像素高度的缺点。建议你综合考虑两种方案，选择最适合项目的做法。

13.3.2 解决方案

使用jQuery杰出的DOM遍历功能，也就是邻近兄弟元素选择器，就可以编写一个处理多个折叠元素的通用脚本。此外，这个脚本还能在必要时处理添加到折叠控件的更多元素。图13-3展示了尚未展开的折叠控件，而图13-4展示了展开后折叠控件的内容。

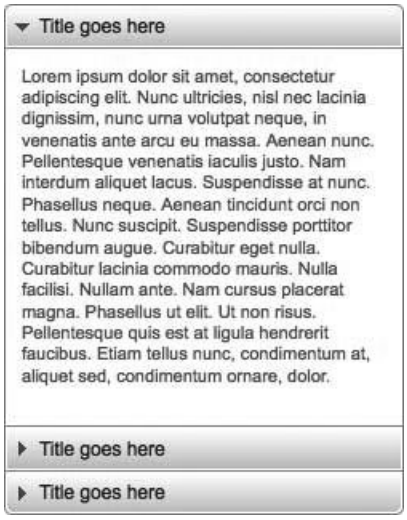


图13-3 等待用户展开的折叠控件

图13-4　展开的折叠控件

1．折叠——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us" lang="en-us">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Expanding an Accordion</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="accordion.css" />
<script type="text/javascript">
/* <![CDATA[ */
document.write('<link rel="stylesheet" type="text/css" href="preload.css" />');
/* ]]> */
</script>
<script type="text/javascript" src="../_common/jquery.js"></script>
<script type="text/javascript" src="accordion.js"></script>
</head>
<body>
<div id="container">
   <dl class="accordion">
     <dt>
        <a href="#"><span></span> Title goes here</a>
     </dt>
     <dd>
        <p>
          Lorem ipsum...
        </p>
     </dd>
     <dt>
        <a href="#"><span></span> Title goes here</a>
     </dt>
     <dd>
        <p>
          Lorem ipsum...
        </p>
     </dd>
     <dt>
        <a href="#"><span></span> Title goes here</a>
     </dt>
```

```
    <dd>
      <p>
        Lorem ipsum...
      </p>
    </dd>
  </dl>
  ...
</div>
</body>
</html>
```

2．折叠——jQuery代码

```
// 初始化
function init_accordion() {

  //元素存在吗?
  if (!$('dl.accordion').length) {

    // 如果不存在则退出
    return;
  }

  //收集所有折叠项
  $('dl.accordion').each(function() {

    //显示第一个折叠项
    $(this).find('dt:first a').addClass('accordion_expanded')
           .end().find('dd:first').show();
    // 通过CSS添加圆角
    $(this).find('dt:last').addClass('last');
  });

  // 单击事件监听器
  $('dl.accordion dt a').click(function() {

    // 获得双亲节点<dl>
    var $dl = $(this).parents('dl:first');

    // 获得下一个<dd>
    var $dd = $(this).parent('dt').next('dd');

    //为最后一个<dt>添加类
    function findLast() {
    if ($dl.find('dd:last').is(':hidden')) {
      $dl.find('dt:last').addClass('last');
    }
  }
  // 元素是否可见?
  if ($dd.is(':hidden')) {

  // 展开<dd>,隐藏其他元素
  $dd.slideDown('fast').siblings('dd:visible').slideUp('fast', findLast);

    // 修改箭头状态,从<dt>删除"last"类
    $(this).addClass('accordion_expanded').parent('dt')
           .removeClass('last').siblings('dt').find('a')
           .removeClass('accordion_expanded');
  }

  // Nofollow.
  this.blur();
```

```
    return false;
  });
}
//启动
$(document).ready(function() {
    init_accordion();
});
```

13.3.3 讨论

上述函数从寻找类为accordion的所有定义列表元素开始，并对它们应用jQuery的.each()方法。在每个元素中，给第一个<dl>链接指定accordion_expanded类，显示第一个<dd>*（由于CSS的display: none属性，其他<dd>元素仍然隐藏）。此外，把最后一个<dt>设置class="last"，可以在支持的浏览器中为其设置独特的圆角样式。这与文件树的示例不同，在文件树中为缺乏：lastchild的浏览器打了补丁。对折叠控件来说，将删除class="last"并根据用户交互重新应用。

代码的第二部分处理折叠控件的核心。折叠控件的<dt>中存在的所有链接都指定了单击事件监听器。当单击任何链接时，向上遍历DOM到双亲<dt>元素，然后经过下一个<dd>。如果这个<dd>元素是隐藏的，通过jQuery的.slideDown()方法用动画显示它，同时在其他兄弟<dd>元素上调用.slideUp()。完成这一动作之后，执行回调函数findLast，该函数根据与最后一个可见的<dt>相伴的<dd>是否隐藏，确定是否赋予class="last"属性。

如果最后一个<dd>可见，不进行任何操作，因为这个<dd>元素本身已经通过针对:lastchild的CSS设置了圆角。同样，聪明的读者会感到奇怪：“为什么我们不为不能识别:lastchild的Internet Explorer 6和7打补丁？”原因是，Internet Explorer 6和7不支持:lastchild，它们也不支持通过CSS设置圆角，所以在这里补丁得不到什么好处。

最后，在单击的<dt>链接上添加accordion_expanded类，并在所有其他<dt>链接上删除该类。这导致除了最近单击的<dt>链接之外，每个<dt>上的箭头都指向右方，表示它们折叠起来。

13.4　选择文档中的不同选项卡

13.4.1　问题

你的一个页面上可能有许多数据，这些数据由于网站架构的原因必须放在一起，而不能分别显示在不同的页面上。在这种情况下，选项卡式的界面通常比带有标题和段落的冗长文档更好。这时，选项卡的工作方式就和桌面应用程序上的一样，它不会离开你所在的页面，而是将与各个选项卡关联的信息带到幕前，如图13-5所示。Yahoo!主页就是这种功能的一个例子。

图13-5　使用选项卡帮助用户浏览信息

13.4.2　解决方案

通过捕捉页间锚链接的href="..."属性，可以使用jQuery寻找目标ID，隐藏其兄弟节点，并将目标元素显示到前景中。这是一个简单的jQuery应用，但是能带来很好的效果。

1．选项卡——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us" lang="en-us">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Tabbing Through a Document</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="tabs.css" />
<script type="text/javascript">
/* <![CDATA[ */
document.write('<link rel="stylesheet" type="text/css" href="preload.css" />');
/* ]]> */
</script>
<script type="text/javascript" src="../_common/jquery.js"></script>
<script type="text/javascript" src="tabs.js"></script>
</head>
<body>
<div id="container">
   <ul class="tabs">
     <li>
       <a href="#tab_content_primary_01" class="current">Tab Link 01</a>
     </li>
```

```
    <li>
      <a href="#tab_content_primary_02">Tab Link 02</a>
    </li>
    <li>
      <a href="#tab_content_primary_03">Tab Link 03</a>
    </li>
    <li>
      <a href="#tab_content_primary_04">Tab Link 04</a>
    </li>
    <li>
      <a href="#tab_content_primary_05">Tab Link 05</a>
    </li>
  </ul>
  <div class="tab_content_wrap">
    <div id="tab_content_primary_01" class="tab_content">
      <p>
        <strong>Content Area 01</strong>
      </p>
      <p>
        Lorem ipsum...
      </p>
    </div>
    <div id="tab_content_primary_02" class="tab_content">
      <p>
        <strong>Content Area 02</strong>
      </p>
      <p>
        Duis ultricies ante...
      </p>
    </div>
    <div id="tab_content_primary_03" class="tab_content">
      <p>
        <strong>Content Area 03</strong>
      </p>
      <p>
        Morbi fringilla...
      </p>
    </div>
    <div id="tab_content_primary_04" class="tab_content">
      <p>
        <strong>Content Area 04</strong>
      </p>
      <p>
        Sed tempor...
      </p>
    </div>
    <div id="tab_content_primary_05" class="tab_content">
      <p>
        <strong>Content Area 05</strong>
      </p>
      <p>
        Nulla facilisi...
      </p>
    </div>
  </div>
  ...
</div>
</body>
</html>
```

2. 选项卡——jQuery代码

```
// 初始化
function init_tabs() {

   // 元素存在吗?
   if (!$('ul.tabs').length) {

      // 如果不存在,退出
      return;
   }

   //显示初始内容区域
$('div.tab_content_wrap').each(function() {
      $(this).find('div.tab_content:first').show();
   });

   // 监听选项卡上的单击
   $('ul.tabs a').click(function() {

      // If not current tab.如果不是当前选项卡
      if (!$(this).hasClass('current')) {

         // 修改当前指示器
      $(this).addClass('current').parent('li').siblings('li')
              .find('a.current').removeClass('current');

         //显示目标,隐藏其他选项卡
         $($(this).attr('href')).show().siblings('div.tab_content').hide();
      }
      // Nofollow.
      this.blur();
      return false;
   });
}

//启动
$(document).ready(function() {
   init_tabs();
});
```

13.4.3 讨论

当函数开始运行时，显示第一个选项卡的内容区域，由于preload.css文件中的display: none样式规则，其余部分保持隐藏。

除此之外，必须做的就是监听<ul class="tabs">中的任何链接被单击。如果单击的链接还没有class="current"属性，我们就知道它的内容没有显示，所以为单击的链接设置class="current",并从具有该类的其他兄弟元素中删除该类。接下来，捕捉同一个页面某个ID的href="..."或者单击的链接，通过jQuery的.show()方法显示该元素，同时隐藏可见的所有兄弟选项卡内容区域。注意，如果你想改进功能，例如，在选项卡状态变化时触发自定义事件或者通过Ajax加载远程内容，一定要研究一下官方的jQuery UI Tab窗口组件。

13.5　显示简单的模态窗口

13.5.1　问题

由于流行的弹出窗口拦截功能已经包含在大部分浏览器中，因此我们再也不能可靠地使用window.open()创建对话框了。更加流行和可用的替代解决方案是在当前页面上创建一个模态的覆盖层，这个覆盖层将优先显示，直到用户与之交互或者撤除它。

值得注意的是，jQuery UI对话框部件是高度可自定义的，并且能够指定一个与其余UI部件匹配的主题/皮肤。你可以在*http://jqueryui.com/demos/dialog*上看到它的一个例子。使用官方窗口组件的好处是受到jQUery UI社区的正式支持，并将持续发展，越来越健壮。潜在的缺点是如果你需要的是一个简单的模态窗口，官方组件需要更多的额外代码。反过来说，选择构建自定义的模态组件的原因之一就是为了较小的代码占用量。建议你综合考虑两种方案，选择最适合项目的做法。

> **注意**
>
> 如果你想要更健壮的解决方案，特别是针对显示多种内容，尤其适合图片库，那么ThickBox就再合适不过了。这是一个由Cody Lindley（本书合著者之一）编写的流行jQuery插件。你可以在*http://jquery.com/demo/thickbox/*上看到它的运行示例。

13.5.2　解决方案

使用jQuery，我们可以轻松地求出浏览器视窗的宽度和高度，并在整个网站设计上创建一个灰色层。然后，可以用CSS定位将自己的模态窗口（实际上是一个 层）居中放在网站内容之前，将用户的注意力吸引到它上面，如图13-6所示。这个窗口可以显示各种类型的内容，包括图像、Ajax加载的HTML片段和页面中的标记。

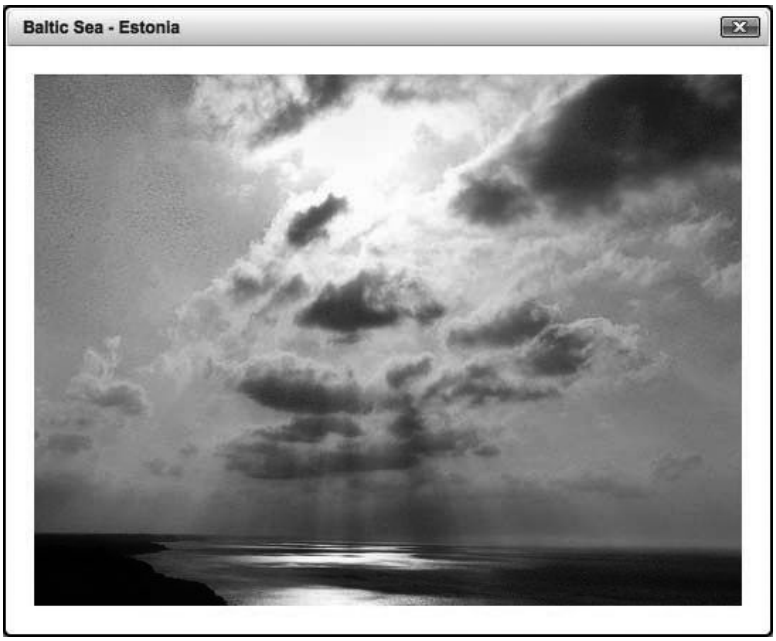


图13-6　用jQuery创建的模态窗口

1．模态——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us" lang="en-us">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Displaying a Simple Modal Window</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="modal.css" />
<script type="text/javascript" src="../_common/jquery.js"></script>
<script type="text/javascript" src="modal.js"></script>
</head>
<body>
<div id="container">
   <a href="#modal_anchor" class="modal">In-page</a>
   | <a href="modal_markup.html#load_me" class="modal">Remote markup</a>
   | <a href="modal_text.txt" class="modal">Remote text</a>
   | <a href="../_common/photo_1.jpg" class="modal" title="Baltic Sea - Estonia">Image
file</a>.
   <br />
   <br />
   <select><option>-- SHOULD BE OVERLAPPED IN IE6 --</option></select>
   <br />
   <br />
   <div id="modal_anchor">
      <p>
         This content will be copied into the modal window, if its #anchor is targeted.
      </p>
   </div>
   Lots of line breaks, to simulate scrolling content...
   <br />
   It's the end of the world, as we know it, and I feel fine.
</div>
</body>
</html>
```

2．模态——jQuery代码

```
// 初始化
function init_modal() {

   // 元素存在吗?
   if (!$('a.modal').length) {

      //如果不存在,退出
      return;
   }

   // 检测IE6(布尔值)
   var $IE6 = typeof document.addEventListener !== 'function' && !window.XMLHttpRequest;

   // 进行某些运算
   function sizeModal() {

      //模态窗口尺寸
      var $modal = $('#modal_window');
      var $modal_width = $modal.outerWidth();
      var $modal_height = $modal.outerHeight();
```

```
  var $modal_top = '-' + Math.floor($modal_height / 2) + 'px';
  var $modal_left = '-' + Math.floor($modal_width / 2) + 'px';

  // 设置模态
  $('#modal_window').css('margin-top', $modal_top)
                    .css('margin-left', $modal_left);
}

/* 用于IE6 */
function positionModal() {
  //强制设定模态窗口位置
  $('#modal_wrapper').css('top', $(document).scrollTop() + 'px');
}

// 显示模态窗口
function showModal() {
  if ($IE6) {
    positionModal();
  }
  // 显示包装器
  $('#modal_wrapper').show();

  // 设置尺寸
  sizeModal();

  // 显示模态窗口
  $('#modal_window').css('visibility', 'visible').show();

  // 在图像加载时调整大小
  $('#modal_content img').each(function() {
    $(this).load(function() {
      $(this).removeClass('modal_placeholder').show();
      sizeModal();
    });
  });
}

//在<body>的最后插入模态窗口
$('body').append('
                  <div id="modal_wrapper">
                    <!--[if IE 6]>
                      <iframe id="modal_iframe"></iframe>
                    <![endif]-->
                    <div id="modal_overlay"></div>
                    <div id="modal_window">
                      <div id="modal_bar">
                        <strong>Modal window</strong>
                        <a href="#" id="modal_close">Close</a>
                      </div>
                      <div id="modal_content"></div>
                    </div>
');

// 查找模态链接
$('a.modal').click(function() {

//检查href="..."
var $the_link = $(this).attr('href');

// 确定链接目标
if ($the_link.match(/^#./)) {

  // 如果是#anchor内容
```

```
      $('#modal_content').html($($(this).attr('href')).html());
      showModal();

   } else if ($the_link.match(/.jpg$/) ||
              $the_link.match(/.png$/) ||
              $the_link.match(/.gif$/)) {
      //如果是图像内容
      $('#modal_content').html('
         <p id="modal_image_wrapper">
           <img src="' + $the_link + '" class="modal_placeholder" />
         </p>
      ');
      showModal();

   } else {

      // 如果是外部Ajax内容
      $('#modal_content').load($(this).attr('href')
                          .replace('#', ' #'), '', showModal);
   }

   // 确定模态窗口标题
   if ($(this).attr('title')) {

      // 插入标题
      $('#modal_bar strong').html($(this).attr('title'));

   } else if ($(this).html() !== '') {

      // 插入链接文本
      $('#modal_bar strong').html($(this).html());
   }

   // Nofollow.
   this.blur();
   return false;
});

//隐藏模态元素
$('#modal_overlay, #modal_close').click(function() {

   //隐藏模态窗口
   $('#modal_wrapper').hide();

   // 隐藏,因为图像可能在以后加载
   $('#modal_window').css('visibility', 'hidden');

   // 解除图像监听器绑定
   $('#modal_content img').each(function() {
      $(this).unbind();
   });

   // 删除模态内容
   $('#modal_content').html('');

   // 重置模态窗口标题
   $('#modal_bar strong').html('Modal window');

   // Nofollow.
   this.blur();
   return false;
});
```

```
// 如果是IE6,监听浏览器滚动
if ($IE6) {
   $(window).scroll(function() {
     if ($('#modal_wrapper').is(':visible')) {
        positionModal();
     }
   });
  }
}

// 启动
$(document).ready(function() {
   init_modal();
});
```

13.5.3 讨论

模态窗口解决方案从定义一个布尔变量开始，该变量表示浏览器是否为Internet Explorer 6。为了确定变量值，进行了两次快速求值。一旦知道了我们所面对的是IE6，就可以修补功能。你还将注意到，在动态创建的标记中包含一个条件注释：

```
<!--[if IE 6]><iframe id="modal_iframe"></iframe><![endif]-->
```

乍一看，这条注释令人困惑，因为如果浏览器是Internet Explorer 6，插入的空iframe不引用任何页面内容。这样做的原因是为了欺骗IE6，使其允许模态窗口覆盖页面中可能存在的任何<select>表单元素。如果不采用这一变通方法，页面上的所有<select>表单元素将“刺穿”模态覆盖层，导致用户迷失方向。

接下来，创建一个函数，它包含在视窗中居中显示模态窗口所需的所有计算。虽然可以在CSS中完成这些计算和简单地硬编码宽度和高度，但是这个函数的灵活性更好。

实现这个JavaScript的开发人员只需要在CSS中设置模态窗口的宽度，函数负责其他的功能，甚至考虑到不同高度的内容。下一个函数只用于Internet Explorer 6，修补了IE6对CSS position: fixed属性支持的缺失。对于其他的所有浏览器，模态窗口都能够在用户滚动查看长文档时保持水平和垂直居中。但是在IE6中，需要明确地告诉模态窗口在用户滚动时调整其位置。我们将在文件的后面调用这个函数完成调整。

显示模态窗口实际上很简单。将所有必要的代码集中在showModal()中，它包括当浏览器为IE6时调用positionModal()。showModal()显示模态包装器 并调用sizeModal()，根据内容的高度居中模态窗口并调整大小。一旦正确设置大小，模态窗口本身就会显示出来。对任何动态插入的图像附加了一个onload函数，这是考虑到浏览器在图像完全缓存之前不知道它的尺寸。注意，showModal()在文件的后面才会真正调用。

当文档加载时，将模态窗口的标记附加在</body>结束标记之后。

把单击监听器附加到所有具有class="modal"属性的链接上。当单击模态链接时，为了确定加载的内容类型，进行一系列求值。首先，如果链接以#符号开始，后面是一个或者多个字符，我们就知道这是链接的页内内容。第二种情况涉及图像。如果href以.jpg、.png或.gif结束，则创建一个<img>标记，将href属性复制到src中。第三种情况是，如果前面的条件都不符合，我们很可能正在处理一个外部页面。在这种情况

下，调用jQuery的.load()方法，从页面（如果存在#符号，则从具体的ID）读取HTML并插入模态窗口。

下一块代码为模态覆盖层（灰色背景）添加单击事件监听器和关闭按钮。如果单击其中一个，模态窗口将会隐藏，所有模态窗口图像从它们的事件监听器中删除，将把模态窗口内容设置为空串，模态窗口的标题栏中的文本也将重置。

最后是专用于IE6的一个窗口滚动事件监听器。如果模态窗口包装器可见（包括其他与模态窗口相关联的其他元素），则在用户滚动页面时连续调用positionModal()。这能确保模态窗口模拟position:fixed，停留在正确的位置。

13.6　构建下拉菜单

13.6.1　问题

不可避免地会有这样的客户或者老板，他们希望网站导航结构当中每个要素都是“一点即中”。虽然这个愿望无可厚非，但是将指向网站各个部分的链接都放在一个页面上可能导致严重的混乱。这时候需要下拉菜单。

13.6.2　解决方案

在桌面程序和操作系统中，这些菜单通常通过单击一个项激活，之后你可以看到多个子项和分类。但是按照Web规范，下拉式菜单是在用户把鼠标悬停于顶级链接之上时出现的，如图13-7和图13-8所示。组合使用CSS :hover规则和定位技术，大部分繁重的工作根本不需要太多的JavaScript就可以完成。jQuery只需要提供对IE6的小改进。

图13-7　准备就绪的下拉菜单

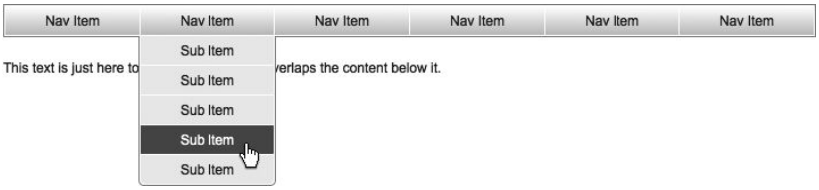


图13-8　使用中的下拉菜单

开发人员应该注意：要考虑无法灵活使用鼠标的用户的可访问性。就像古语：“如果你只拥有锤子，那么什么东西看上去都像钉子。”在将下拉菜单作为简单现成的解决方案之前，必须确认对项目的信息架构已经深思熟虑。要确信下拉模式是最佳选择。例如，Microsoft Word长期以来都因荒谬的下拉菜单级数和可切换选项（其中大部分都是普通用户从未接触过的）而闻名，在Office 2007中这一界面进行了重新设计，改成了绰号为“Ribbon”的标签式UI。突然间，曾经模糊的选项由于更好的执行界面而变得可以正常使用了。

1．下拉菜单——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us" lang="en-us">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Building Drop-Down Menus</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="dropdown.css" />
<!--[if IE 6]>
<script type="text/javascript" src="../_common/jquery.js"></script>
```

```
<script type="text/javascript" src="dropdown.js"></script>
<![endif]-->
</head>
<body>
<div id="container">
   <ul class="dropdown">
      <li class="dropdown_trigger">
         <a href="#">Nav Item</a>
         <ul>
            <li>
               <a href="#">Subitem</a>
            </li>
            <li>
               <a href="#">Subitem</a>
            </li>
            <li>
               <a href="#">Subitem</a>
            </li>
            <li>
               <a href="#">Subitem</a>
            </li>
            <li>
               <a href="#">Subitem</a>
            </li>
            <li>
               <a href="#">Subitem</a>
            </li>
         </ul>
      </li>
      ...
   </ul>
   <p>
      This text is just here to show that the menu overlaps the content below it.
   </p>
   ...
</div>
</body>
</html>
```

2. 下拉菜单——jQuery代码

```
// 初始化
function init_dropdown() {

   // 元素存在吗?
   if (!$('ul.dropdown').length) {

      // 如果不存在,退出
      return;
   }

   // 为悬停添加监听器
   $('ul.dropdown li.dropdown_trigger').hover(function() {

      //显示后续的<ul>
      $(this).find('ul').fadeIn(1);
   },
   function() {

      //隐藏后续的<ul>
      $(this).find('ul').hide();
   });
}

//启动
$(document).ready(function() {
```

```
    init_dropdown();
});
```

13.6.3 讨论

在这个例子中，jQuery只在浏览器是IE6时才有用。你可能觉得奇怪：“为什么fadeIn调用时只使用1毫秒的动画？”这是为了修复IE6中的一个缺陷：显示垂直CSS边框时的问题。从视觉上，这与.show()相同，但是不会有任何边框问题。除此之外，其他代码都很简单。当具有class="dropdown_trigger"属性的列表项上有鼠标悬停时，显示后续的<ul>。当鼠标离开该区域时，隐藏<ul>。这就是全部的逻辑！注意，我们有条件地包含jQuery程序库，只用于IE6。如果你正在阅读本书，很可能不只想将jQuery用在这样一个特殊的演示程序上。在那种情况下，就把包含jQuery的语句移到条件注释之外。

13.7　交叉消隐的循环图像

13.7.1　问题

在包含大标题或者“英雄”图片的网页上（常见于电子商务网站），许多不同的产品和部门竞争顶部位置。折中的方法往往是简单地将一系列图像循环淡入淡出。这样做当然不错，但是在使用中常常令人沮丧，因为太多网站都忽视了一项需求：暂停循环以便慢慢收集试图表达的信息。

恒定动画的影响应该加以考虑。大部分用户已经学会忽略包含许多动作的烦人广告。设计人员/开发人员必须考虑不想看到动画同时又想阅读页面其余内容的用户。更糟糕的是在用户试图查看循环幻灯片中的某一张时，却只能看着它继续变化。因此，本秘诀中加入了播放/暂停功能，以免用户陷入没完没了的动画循环。

13.7.2　解决方案

使用jQuery的`.fadeIn()`和`.fadeOut()`方法，可以创建一个细致的交叉消隐动画，循环读取一个数组，根据设置好的定时器修改每幅图像的不透明度。通过实现从13.4节学来的代码，可以创建指向每张图片的链接，不仅使目标图像出现在前景上，而且将标志变量`pause`设置为`true`或者`false`，从而启动或者停止动画。这样就能实现真正可用的图片循环播放器，而不仅仅是纯粹的视觉效果。

1．图片循环播放器——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Cross-fading Rotating Images</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="rotator.css" />
<script type="text/javascript">
/* <![CDATA[ */
document.write('<link rel="stylesheet" type="text/css" href="preload.css" />');
/* ]]> */
</script>
<script type="text/javascript" src="../_common/jquery.js"></script>
<script type="text/javascript" src="rotator.js"></script>
</head>
<body>
<div id="container">
   <div id="rotator_wrapper">
     <ul id="rotator">
        <li id="photo_1">
          <img src="../_common/photo_1.jpg" alt="Photo" />
        </li>
        <li id="photo_2">
          <img src="../_common/photo_2.jpg" alt="Photo" />
        </li>
        <li id="photo_3">
          <img src="../_common/photo_3.jpg" alt="Photo" />
```

```
        </li>
        <li id="photo_4">
          <img src="../_common/photo_4.jpg" alt="Photo" />
        </li>
        <li id="photo_5">
          <img src="../_common/photo_5.jpg" alt="Photo" />
        </li>
      </ul>
      <ul id="rotator_controls">
        <li>
          <a href="#photo_1" class="current">1</a>
        </li>
        <li>
          <a href="#photo_2">2</a>
        </li>
        <li>
          <a href="#photo_3">3</a>
        </li>
        <li>
          <a href="#photo_4">4</a>
        </li>
        <li>
          <a href="#photo_5">5</a>
        </li>
      </ul>
      <a href="#" id="rotator_play_pause">PAUSE</a>
   </div>
</div>
</body>
</html>
```

2．图片循环播放器——jQuery代码

```
// 初始化
function init_rotator() {

   // 元素存在吗?
   if (!$('#rotator').length) {

      //如果不存在,退出
      return;
   }

   // 旋转速度
   var speed = 2000;

   // 暂停设置
   var pause = false;

   // rotate函数
   function rotate(element) {

      // 如果用户交互则停止
      if (pause) {
         return;
      }

      // 下一个/第一个<li>
      var $next_li = $(element).next('li').length ?
                     $(element).next('li') :
                     $('#rotator li:first');
```

```
    //下一个/第一个控制链接
    var $next_a = $('#rotator_controls a.current').parent('li').next('li').length ?
                  $('#rotator_controls a.current').parent('li').next('li').find('a') :
                  $('#rotator_controls a:first');

    //动画
    $('#rotator_controls a.current').removeClass('current');
    $next_a.addClass('current');

    // 继续
    function doIt() {
       rotate($next_li);
    }

    // 淡出<li>
    $(element).fadeOut(speed);

    // 显示下一个<li>
    $($next_li).fadeIn(speed, function() {

       // 稍作延迟
       setTimeout(doIt, speed);
    });
  }

  // 为控件添加单击监听器
  $('#rotator_controls a').click(function() {

    // 修改按钮文本
    $('#rotator_play_pause').html('PLAY');

    //显示目标图片,隐藏其他<li>
    $($(this).attr('href')).show().siblings('li').hide();

    // 添加class="current"并从其余元素中删除该类
    $(this).addClass('current').parent('li').siblings('li')
            .find('a').removeClass('current');;

    //暂停动画
    pause = true;

    // 不要追踪链接
    this.blur();
    return false;
  });

    // 暂停/播放动画
    $('#rotator_play_pause').click(function() {

       // 按钮表示什么?
       if ($(this).html() === 'PAUSE') {

          // 停止循环
          pause = true;

          // 修改文本
          $(this).html('PLAY');

       } else {

          // 删除class="pause"
          pause = false;
```

```
            // 开始循环
            rotate('#rotator li:visible:first');

            // 修改文本
            $(this).html('PAUSE');
        }

        // 不要追踪链接
        this.blur();
        return false;
    });

    //隐藏除了第一个以外的所有<li>
    $('#rotator li:first').siblings('li').hide();

    //等待页面加载
    $(window).load(function() {

        // 开始循环
        rotate($('#rotator li:visible:first'));
    });
}
// 启动
$(document).ready(function() {
    init_rotator();
});
```

13.7.3 讨论

本秘诀从定义两个关键变量speed（表示毫秒的数字值）和pause（告诉循环播放器是否播放的布尔值）开始。speed开始时设置为2秒，pause设置为false，允许循环播放器在页面加载时自动播放。

在rotate()函数中，设置了一个$next_li变量，它将对应当前动画显示的下一个<li>或者数组中的第一个<li>（到达数组最后，需要重新开始时）。<ul id="rotator_controls">中的链接适用相同的逻辑，为当前活跃的图像按钮添加一个视觉指示器。在两秒钟的延迟之后，整个图片序列再次启动。

如果演示程序在这里结束，得到的就是令人厌烦的无控制图片循环。幸运的是，可以重用来自选项卡式文档秘诀中的页内锚链接技术。简单地为<ul id="rotator_controls">中的链接指定单击监听器，然后在显示目标图片的同时隐藏其他图片。还添加了一个播放/暂停按钮，用来启动/停止循环播放器的动画。

最后是使一切动起来的代码。<ul id="rotator">中除了第一个<li>之外，其他元素都隐藏起来，当窗口加载结束时，开始显示动画。注意，$(window).load()与$(document).ready()有所不同，因为前者等待所有资源（包括图片）完全加载，这对于图片循环播放器来说特别重要。而后者只等待HTML结构完整，这对于在页面其余图片尚在加载中应用功能很重要。两者同样重要，但是各有其适用的场合。

13.8 滑动面板

13.8.1 问题

有时候，你可能需要水平显示多种选择，并且为此做一些修饰，但是由于选择太多而超过了布局的宽度。或者，你可能只是寻求一种方式，说明这里有某些有趣的用户交互。不管是哪一种用途，滑动面板方法（有时称作水平折叠）都是表现这类信息的一种可能方法。图13-9展示了一个关闭的面板，而图3-10显示滑出后的可视面板。

图13-9 关闭的水平面板

图13-10 打开的水平面板

13.8.2 解决方案

在本秘诀中，我们将复习折叠演示中应用的一些概念，但是这里的动画在水平方向，而不像折叠中那样从垂直方向展开和折叠内容。还使用了CSS定位技巧来解决同时移动的面板中小的计算误差。不用担心每个面板之间的同步，具有完美像素的动画执行当中的精度也达到了几分之一秒，只要取得<ul class="panels">中的最后一个<li>，并将其绝对定位到<ul>的右上角就可以了。那样，当所有面板的总宽度在移动时加起来大于<ul>的总宽度时，最后一个<li>就不会转到下一行。

1. 面板——HTML代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<meta http-equiv="imagetoolbar" content="false" />
<title>jQuery Cookbook - Ch.13 - Sliding Panels</title>
<link rel="stylesheet" type="text/css" href="../_common/basic.css" />
<link rel="stylesheet" type="text/css" href="panels.css" />
<script type="text/javascript" src="../_common/jquery.js"></script>
<script type="text/javascript" src="panels.js"></script>
</head>
<body>
<div id="container">
   <ul class="panels">
      <li>
         <a href="#">1</a>
      </li>
      <li>
```

```
      <a href="#">2</a>
    </li>
    <li>
      <a href="#">3</a>
    </li>
    <li>
      <a href="#">4</a>
    </li>
    <li>
      <a href="#">5</a>
    </li>
  </ul>
  <ul class="panels">
    <li>
      <a href="#">A</a>
    </li>
    <li>
      <a href="#">B</a>
    </li>
    <li>
      <a href="#">C</a>
    </li>
    <li>
      <a href="#">D</a>
    </li>
    <li>
      <a href="#">E</a>
    </li>
  </ul>
</div>
</body>
</html>
```

2．面板——jQuery代码

```
//初始化
function init_panels() {

  // 元素存在吗?
  if (!$('ul.panels').length) {

    //如果不存在,退出。
    return;
  }

  // 动画速度
  var speed = 200;

  //为最后一个<li>添加类
  $('ul.panels li:last-child').addClass('last');

  // 以mouseover事件开始
  $('ul.panels li').hover(function() {

    // 修改目标<li>
    $(this).stop().animate({
      width: '360px',
      fontSize: '150px'

    // 速度
    }, speed)
```

```
      //修改兄弟<li>
      .siblings('li').stop().animate({
         width: '135px',
         fontSize: '50px'

      // 速度
      }, speed);
   },

   //以mouseout事件结束
   function() {

      // 恢复目标<li>
      $(this).stop().animate({
         width: '180px',
         fontSize: '100px'

      //速度
      }, speed)

      // 恢复兄弟<li>
      .siblings('li').stop().animate({
         width: '180px',
         fontSize: '100px'

      //速度
      }, speed);
   });
}

// 启动
$(document).ready(function() {
   init_panels();
});
```

13.8.3 讨论

本秘诀从定义speed变量开始。在例子中，把速度设置为200毫秒。然后，为每个位于最后的<li>添加class="last"。接着，附加悬停事件监听器（实际上，这个事件监听器映射到mouseover和mouseout两个事件，这里不纠缠技术细节了）。当鼠标悬停于某个<li>上时，元素的宽度用动画设置为40%，字体尺寸设置为150px，同时把其他<li>设置为15%的宽度和50px的字体。同样，当鼠标退出<li>时，把所有<li>元素的宽度设置为20%，把字体尺寸设置为100px。

第14章　使用jQuery UI构建用户界面

Richard D. Worth

14.0 导言

两年前，Stefan Petre编写了一个名为Interface的程序包，捆绑了一组相当流行的jQuery插件。这些插件提供了很棒的交互，如拖放、选择、排序和调整大小，以及工具提示、自动补全和折叠等杰出的窗口组件。jQuery 1.2版本修改了一些API，这也就要求对Interface进行兼容性的更改，但是Interface再也没有得到更新。

jQuery UI（*http://jqueryui.com*）是由Paul Bakaus发起的，它继续了Interface未竟的事业。jQuery UI是一组具有一致API和完整文档的插件，在所有主流浏览器中都经过测试。使用它，就能创建丰富的Web界面和富互联网应用（RIA）。而且，这些插件配合得很好，具有易用、可访问、可扩展和“可主题化”的特性。

jQuery UI是jQuery的姐妹项目。jQuery UI 1.0版本发行于2007年9月。版本1.5发行于2008年6月。在1.6开发的中途，开发团队改变了方向，最终发行的版本1.7有了一些重大的变化，其中最特别的是jQuery UI CSS Framework的推出（*http://jqueryui.com/doc/Theming/API*）。之后的jQuery UI 1.6是为了遗留程序的兼容性而发行的。最新的稳定版本是1.7.2，包含如下的交互、窗口组件和特效。

14.0.1 交互

- 可拖动（拖动）
- 可放置（或放置）
- 可改变大小
- 可选择
- 可排序

14.0.2 窗口组件

- 折叠控件
- 日期选择器
- 对话框
- 进度条
- 滑块
- 选项卡

14.0.3 特效

- 百叶窗、弹跳、剪切、上/下/左/右拉动、爆炸、折叠、高亮、震动、伸缩、摇晃、上/下/左/右滑动、变换
- 颜色动画
- 类动画（有时间间隔的`addClass/removeClass/toggleClass`）

14.0.4 基本用法

本章放弃了对这些交互、窗口组件和特效的一些较常见用法的介绍，因为它们在jQuery UI网站上的演示程序中（*http://jqueryui.com/demos/*）已经作了很好的介绍。这些演示程序的完整源代码和描述以及完整的文档都包含在jQuery UI的下载文件中。

14.0.5 本章组织结构

前两个秘诀帮助你下载jQuery UI或者在内容分发网络（Content Delivery Network，CDN）中应用它，并将其包含在你的网页中以供使用。

接下来的7个秘诀介绍jQuery UI API。这个API建立于jQuery插件模式的基础上，但是已经发展到包含了jQuery UI窗口组件所需的API，成为一种独特风格的jQuery插件。也就是说，它们是状态和方法调用。所以，除了在init函数中指定选项，还可以在init之后修改选项。也可以调用jQuery UI插件上的方法，改变状态，以编程方式触发自定义事件。

本章余下的内容关注一个项目，将多个jQuery UI窗口组件组合起来，创建一个包含用于音乐播放器的灵活和可主题化控件的用户界面。

14.1 包含整个jQuery UI套件

14.1.1 问题

你打算包含整个jQuery UI套件。这可能是因为你不知道将会使用套件的哪些部分，不会使用哪些部分。也可能是因为你使用的套件功能很多，包含整个套件比起你所要使用的各个单独部件更容易或者更有效。

14.1.2 解决方案

链接到一个jQuery UI主题，然后是jQuery脚本的一个兼容版本，接着是jQuery UI脚本：

```
<link rel="stylesheet" type="text/css" href="themename/jquery-ui.css" />
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" src="jquery-ui.js"></script>
```

14.1.3 讨论

本章介绍的是最新的稳定版本jQuery UI 1.7.2。它至少需要jQuery 1.3。当下载jQuery UI时，ZIP包里就包含了兼容的jQuery版本。

和托管自己的jQuery和jQuery UI版本不一样，可以使用Google的AJAX Libraries API。只要将脚本URL改成下面这样就可以了：

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.7.2/jquery-ui.min.js "></script>
```

Google还托管了20多个jQuery UI ThemeRoller画廊中的主题。

```
<link rel="stylesheet" type="text/css" href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.7.2/themes
```

每个主题包含13张图片，在主题CSS中通过相对URL引用。

你可以用base、black-tie、blitzer、cupertino、dark-hive、dot-luv、eggplant、excite-bike、flick、hot-sneaks、humanity、le-frog、mint-choc、overcast、pepper-grinder、redmond、smoothness、south-street、start、sunny、swankypurse、trontastic、ui-darkness、ui-lightness或者vader代替{themename}。每个主题的预览可以参见jQuery UI ThemeRoller画廊（*http://jqueryui.com/themeroller/#themeGallery*）。

jQuery UI的主题在下一章中将详作介绍。对于我们来说，我们一定会包含某一个主题，因为主题是必需的。

14.2 包含单独的一两个jQuery UI插件

14.2.1 问题

你只打算使用一两个jQuery UI窗口组件，不想导入整个程序库和所有主题CSS。你想要的只是使用插件中必需的部分。

14.2.2 解决方案

你只想要Sortable和Tabs插件。包含单个jQuery UI组件而非整个套件有两个选择：

- 使用jQuery UI下载构建器（Download Builder，*http://jqueryui.com/download*）创建定制的jQuery UI，仅仅包含你感兴趣的插件。在这个例子中，选择Sortable和Tabs插件。下载构建器自动选择任何依赖组件，在这个例子中是UI Core。下载的ZIP包含一个.js文件，该文件带有UI Core、Sortable和Tabs插件：

```
js/jquery-ui-1.7.2.custom.min.js
```

在页面上jQuery脚本之后包含这个文件，两个文件在相同的文件夹中：

```
<script type="text/javascript" src="js/jquery-1.3.2.min.js"></script>
<script type="text/javascript" src="js/jquery-ui-1.7.2.custom.min.js"></script>
```

- 下载jQuery UI开发包，在一个自定义Download Builder ZIP中引用*development-bundle*文件夹，或者使用SVN（*http://jqueryui.com/docs/Subversion*）。每个单独的插件文件都在ui子目录中，单独引用每一个文件：

```
<script type="text/javascript" src="jquery-1.3.2.js"></script>
<script type="text/javascript" src="ui/ui.core.js"></script>
<script type="text/javascript" src="ui/ui.sortable.js"></script>
<script type="text/javascript" src="ui/ui.tabs.js"></script>
```

如果采用第二种选择（开发包），每个单独插件所用的CSS也存在于单独的文件中。需要包含核心CSS、每个插件专用的CSS和主题CSS：

```
<link rel="stylesheet" type="text/css" href="themes/base/ui.core.css" />
<link rel="stylesheet" type="text/css" href="themes/base/ui.tabs.css" />
<link rel="stylesheet" type="text/css" href="themes/base/ui.theme.css" />
```

在本例中，选择的插件之一Sortable没有任何插件专用的CSS。

14.2.3 讨论

不管使用JavaScript还是CSS，在使用单个大型包含文件和多个较小包含文件之间都需要做出权衡。这并不像“在开发中使用单个插件，在生产中使用大文件”那么明晰。例如，在性能测试不是主要关注点时，在开发中指定整个套件可能比较简单。但是在生产环境中，可能只需要包含每个页面所用的插件文件，以便最大限度地减少加载量。

另一方面，出于调试目的，在开发中让脚本和CSS指向各个插件文件可能有好处，而在生产中可以使用Google AJAX Libraries API和访问者的缓存弥补文件大小造成的问题，即使文件包含了未使用的函数也没有太大的关系。理想的设置取决于架构、使用插件的多少和开发及生产环境的具体需求。

14.3 用默认选项初始化jQuery UI插件

14.3.1 问题

你打算尽快开始使用jQuery UI插件，接受内置的默认选项。

14.3.2 解决方案

所有jQuery UI插件的调用和传统jQuery插件类似，所以在获得一组匹配元素之后，只要将插件名称作为jQuery对象上的一个函数就行了：

```
<script type="text/javascript">
$(function() {
    $('#topnav').tabs();
});
</script>
```

14.3.3 讨论

因为JavaScript是区分大小写的，所以在jQuery UI插件的名称上要多加小心。和jQuery API一样，所有jQuery UI插件都以小写字母开始，大部分都只有一个单词。如果需要超过一个单词，从第二个单词开始都以大写字母开始。目前还没有任何jQuery UI插件的名称超过一个词，示例如下：

```
$('p.long').succinct();

$('.short').longerPluginName();
```

初始化元素有一个类ui-pluginname。例如，下面是调用$('div').draggable();前后的HTML：

```
<div>A simple DIV</div>

<div class="ui-draggable">A simple DIV</div>
```

这种情况也有一些例外。调用.dialog()的元素得到ui-dialog-content类，并且包装在具有ui-dialog类的一个生成元素里。另一个例外是，如果在文本输入字段上调用.datepicker()，输入字段将不具有ui-datepicker类，但在输入字段获得焦点时显示的 具有ui-datepicker类。

下面是初始化jQuery UI插件时需要注意的几点：

- 如果在超过一个元素的集合上调用jQuery UI插件的init方法，将在每个元素上单独调用init。所以，下列的语句

  ```
  $('img').draggable();
  ```

  和如下语句等价：

  ```
  $('img').each(function() {
      $(this).draggable();
  });
  ```

- 每个DOMElement仅可以被每个jQuery UI插件初始化一次。任何未来的init调用，不管是否指定选项，都会被忽略。本章后面有关于在init之后改变选项以及销毁插件的秘诀，该插件撤销init操作。如果你真的要这么做，可以在那之后再次调用init。
- 所有选项都是可选的。始终可以安全地调用插件名称方法初始化jQuery UI插件。这不仅安全，而且非常实用。每个插件都设计了最常用的默认选项。如果这些选项不合你的心意，参见下两个秘诀。

14.4 用自定义选项初始化jQuery UI插件

14.4.1 问题

你打算使用jQuery UI插件，但是希望采用插件作者选择的默认值之外的选项。

14.4.2 解决方案

在一个选项散列中覆盖默认选项，作为插件init方法的第一个参数：

```
$('#myDiv').dialog({
    height: 100,  //覆盖默认值:'auto'
    width: 350   // 覆盖默认值:300
});
```

14.4.3 讨论

在init上指定的任何选项值将覆盖默认值。所有未指定的选项值则维持默认值。

选项散列不管包含的是所有默认值还是一些默认值加上某些自定义值，都是插件初始化状态的基础。这一状态对应DOMElement和jQuery UI插件的组合。例如，可以用两个不同的jQuery UI插件初始化一个元素，每个插件都有一个颜色选项：

```
$('#myDiv').foo({ color: 'black' });
$('#myDiv').bar({ color: 'green' });
```

现在，对于#myDiv，foo颜色是什么？是黑色。bar颜色是什么？绿色。两种颜色都独立于CSS颜色。在后面的一些秘诀中，将研究如何查询元素的插件值，以及如何设置新的值。

还要注意一点，myDiv已经初始化了foo和bar的值，它就不再受到插件默认值的影响了。默认值只是在init中用作插件初始化状态的一个模板。

14.5 创建你自己的jQuery UI插件默认值

14.5.1 问题

每当你创建jQuery UI对话框时，就会发现有几个选项总是指定相同的值：

```
$('#msg').dialog({
    height: 300,
    width: 400,
    draggable: false,
    modal: true,
    buttons: {
        'OK': function(event, ui) {
            $(this).dialog('close');
        }
    }
    ...
});
```

你希望代码像过去那样简洁。如何才能得到`$('#msg').dialog();`那样的简单之美？

14.5.2 解决方案

在`init`之前扩展`$.ui.pluginname.defaults`覆盖插件默认值：

```
$.extend($.ui.dialog.defaults, {
    height: 300,
    width: 400,
    draggable: false,
    modal: true,
    buttons: {
        'OK': function(event, ui) {
            $(this).dialog('close');
        }
    }
});
...
$('#msg').dialog();
...
$('#note').dialog();
```

14.5.3 讨论

如果你只想改进可读性，可以简单地将选项放入一个变量中，将其传递给插件的`init`方法：

```
var options = {
    height: 300,
    width: 400,
    draggable: false,
    modal: true,
    buttons: {
        'OK': function(event, ui) {
            $(this).dialog('close');
        }
    }
};

$('#msg').dialog(options);
```

但是本秘诀不仅是关于可读性和代码美学的，还与修改不是由你编写的插件的默认行为有关。而且，这里介绍的方法可以让你回到简单的无选项`init`：

```
$('#msg').dialog();
```

正如Dave Methvin的名言“除非读懂你的想法，否则就无法得到更简洁的。”

当然，你可以像前一个秘诀里那样向插件的init方法传递自定义选项，覆盖这些自定义默认选项。

不要忘记，插件选项是在init的时候从默认值中复制和扩展的。所以，在 已经初始化为对话框后扩展$.ui.dialog.defaults对该对话框没有影响，即使init没有使用自定义选项也一样。这种扩展只对默认值覆盖之后初始化的对话框有效。

14.6 获取和设置jQuery UI插件选项

14.6.1 问题

你需要在初始化之后检查或者修改jQuery UI插件选项的值。

14.6.2 解决方案1：获取选项值

调用插件的option方法，传递选项名称：

```
var active = $('#myDiv').accordion('option', 'active');
```

当仅用选项名称调用时，option方法获取并返回值，所以它不能链接。

14.6.3 解决方案2：设置值

调用插件的option方法，传递选项名称和新值：

```
$('#myDiv').accordion('option', 'active', 3);
```

当用选项名称和值调用时，option方法设置新值并返回jQuery对象，所以它可以链接。

14.6.4 讨论

option方法获取/设置值的方法遵循了jQuery getter和setter方法（如.css()和.attr()）的模式。如果提供一个值，该方法就是setter方法；如果省略值，就是getter方法。

和其他jQuery setter方法一样，可以传递一个散列给option方法，一次设置多个选项：

```
$('#myDiv').accordion('option', {
    active: 2,
    collapsible: true
});
```

14.7 调用jQuery UI插件方法

14.7.1 问题

你需要以编程方式使用jQuery UI插件进行某些操作。

14.7.2 解决方案

调用jQuery UI插件名称方法，传递想要调用的插件方法名称作为第一个参数。例如，使用如下代码关闭一个对话框：

```
$('#msg').dialog('close');
```

如果方法需要参数，在方法名之后传递它们。例如，选择第三个选项卡，使用如下代码：

```
$('#nav').tabs('select', 2); // tabs插件的select方法接受从0开始的索引
```

14.7.3 讨论

每个jQuery UI插件都至少提供4个常用的基本方法：

- option
- enable
- disable
- destroy

前一个秘诀中已经介绍了option方法。destroy方法将在以后的秘诀中介绍。enable和disable方法的功能不言自明，它们的工作原理是设置插件的disabled选项，该选项默认为false：

```
$('img').draggable('disable');

$('#mySlider').slider('enable');
```

调用这些方法还将切换元素上的ui-pluginname-disabled类，这个类可以用于样式或者选择。

要查看插件目前是否禁用，可以使用option方法获得disable选项的值：

```
var isDisabled = $('#tempature').slider('option', 'disabled');
```

14.8 处理jQuery UI插件事件

14.8.1 问题

你需要对jQuery UI插件上发生的事件做出反应，或者得到相应的通知。这些事件可能是对话框打开、折叠面板关闭或者选中某个选项卡。

在这个秘诀中，将处理一个被拖放到可放置元素上的可拖动元素，这会触发可放置元素上的drop事件。

14.8.2 解决方案1：向事件名称选项传递一个回调函数

在init或者以后使用的option方法中，可以声明一个回调函数，在事件发生时调用：

```
//在初始化时声明一个事件回调选项
$('#shopping-cart').droppable({
     drop: function(event, ui) {
          addProduct(ui.draggable);
     }
});

// 在初始化之后用option方法声明事件回调
$('#shopping-cart').droppable();
...
$('#shopping-cart').droppable('option', 'drop', function(event, ui) {
     addProduct(ui.draggable);
});
```

注意，这个解决方案只考虑每个事件触发器调用一个函数的情况。可以使用代理方法或者下面所介绍的绑定解决方案调用多个处理函数。

14.8.3 解决方案2：用事件类型绑定自定义事件

使用jQuery .bind()方法，绑定到事件类型：

```
//在初始化时声明事件回调
$('#shopping-cart').bind('drop', function(event, ui) {
     addProduct(ui.draggable);
});
```

这种绑定可以在插件元素或者某些容器上利用自定义事件冒泡和委托来完成。

14.8.4 讨论

每个jQuery UI事件接受两个参数event和ui。event参数类似于传递给所有浏览器事件（如click和keypress）的事件参数。不同之处在于，这是一个自定义事件对象。和浏览器事件一样，事件的类型可以在evnet.type中找到。

许多jQuery UI插件事件通常会被对应的浏览器事件触发。例如，可拖动序列事件dragstart、drag、dragstop很有可能被浏览器的mousedown、mousemove和mouseup事件触发。如果自定义事件被浏览器事件触发，浏览器事件将保存在event.originalEvent属性中。如果需要确定操作是通过键盘、鼠标还是以编程方式进行的，这个属性就很有用。如果需要找出鼠标单击或者移动时是否按下了功能键，该属性也能派上用场。

ui参数是一个散列，包含了这时特别适用于事件的值，以及调用option或者其他插件方法无法得到的值。例如，当在可放置元素上放下一个可拖动元素时，把可拖动元素传递给ui.draggable中的drop

事件。这个ui散列的内容对于每个插件事件来说都是独特的。

注意，事件名称和事件类型往往不同。例如，可拖动元素（Draggable）和滑块（Slider）都有start事件，这是事件名称，而它们的事件类型分别是dragstart和sliderstart。因为每个插件都有自己的选项命名空间，所以它们都可以拥有相同的选项名start：

```
$('img').draggable({
    start: function(event, ui) {
        //event.type == 'dragstart'
    }
});
$('#mySlider').slider({
    start: function(event, ui) {
        //event.type == 'slidestart'
    }
});
```

但是，因为事件在同一个命名空间中绑定和触发，所以需要一个前缀，以保持事件类型唯一：

```
$('img').bind('dragstart', function(event, ui) {
        //event.type == 'dragstart'
    }
});
$('#mySlider').bind('slidestart', function(event, ui) {
        //event.type == 'slidestart'
    }
});
```

这个前缀通常就是插件的名称，所以事件类型为dialogfocus、tabsadd和progressbarchange等。在某些情况下，如果某个自定义动词前缀更合适，则用它代替。所以，用dragstart代替draggablestart，用slidestart代替sliderstart。

如果事件类型前缀恰好与事件名称完全相同，为了避免dragdrag或slideslide这样的重叠，前缀会去掉。在这些情况下，事件类型将与事件名称相同，如drag和slide。

14.9 销毁jQuery UI插件

14.9.1 问题

你已经用某个特定插件完成了操作，希望自己的元素回到原来的形态。这时进行的操作不仅是disable；而是撤销init方法的操作。

14.9.2 解决方案

调用destroy方法：

```
$('#queue').sortable('destroy');
```

14.9.3 讨论

调用destroy方法将完全撤销插件元素初始化所作的操作。它将删除init所添加的任何类或者任何后来的方法调用或者事件。如果init导致元素被包装，该元素将被解包。整个过程就像一个大的撤销操作。

销毁jQuery UI插件不会从DOM中删除元素，它只删除保存在元素上的插件状态，将元素尽可能恢复到init之前的状态。在jQuery UI插件卸载之后，可以再次初始化它。

如果你打算销毁并删除某个插件元素，可以简单地调用.remove()，jQuery UI将在删除元素时自动调用destory方法。即使元素被多个jQuery UI插件初始化，情况也一样。

14.10　创建jQuery音乐播放器

14.10.1　问题

你需要一个支持常见界面控件集的音乐播放器，不管音乐是由Flash播放器还是HTML5音频或者是其他浏览器音频功能播放的。你需要可访问、灵活和可主题化的控件。基本的功能有：

- 播放
- 暂停
- 显示和控制当前回放点的轨道条
- 显示歌曲缓冲数量的进度计
- 音量

除了这些基本功能之外，还需要另一个功能。音乐播放器必须是可缩放的。不管是浏览器、用户还是应用程序调整窗口的大小（最大可以是全屏），相同的界面应该在任何大小的窗口中都能正常工作。

14.10.2　解决方案

用jQuery UI来构建一个音乐播放器。将用jQuery UI CSS Framework的图标创建播放和暂停按钮，并用jQuery UI Slider插件创建轨道条。进度计使用jQuery UI Progressbar插件。最后，音量控制开关也是一个jQuery UI Slider。在一个常见的容器中包装这些元素，提供某个漂亮的窗口组件主题，不仅每个控件都可以设置主题，而且整个音乐播放器可以设置主题。

> **注意**
>
> 我们不打算将这个音乐播放器构建为一个可重用的插件。我们只准备将jQuery UI窗口组件连接在一起，使用户感觉像是一个组件在工作。但是这个音乐播放其本身不是jQuery插件或者jQuery UI插件。在这个秘诀中，它只是一组HTML、JavaScript和CSS。那样，我们可以专注于底层jQuery UI插件的使用，而不用担心从现有插件中构建新插件的额外复杂性。

1. HTML5音频

为了保持简洁，这里打算使用HTML5媒体元素API（*http://dev.w3.org/html5/spec/Overview.html#htmlmediaelement*）的最小子集。这个API存在于许多最新的浏览器（如Firefox 3.5）中。把它当作兼容性图层来使用，这样就可以轻松地替换Flash Player等其他回放机制。在这个秘诀中，需要从音频API中得到如下接口：

- 开始或者恢复播放（`play`）
- 暂停播放（`pause`）
- 获得歌曲长度（`duration`）
- 获取当前播放点（`timeupdate`）
- 转到歌曲中的某一点（`currentTime`）
- 获取歌曲播放音量（`volumeChange`）
- 设置特定音量（`volume`）

假定文档中存在一个HTML5音频元素（*http://dev.w3.org/html5/spec/Overview.html#audio*），下面是兼容性图层的代码：

```
var $audio = $('audio'), audioEl = $audio[0];
var audio = {
        currentTime: 0,
        duration: secondsTotal,
        volume: 0.5,
        set: function(key, value) {
                this[key] = value;
                try { audioEl[key] = value; } catch(e) {}
                if (key == 'currentTime') {
                        $audio.trigger('timeupdate');
                }
```

```
                if (key == 'volume') {
                        $audio.trigger('volumechange');
                }
        },
        play: function() {
                audioEl.play && audioEl.play();
        },
        pause: function() {
                audioEl.pause && audioEl.pause();
        }
};
$audio.bind('timeupdate', function() {
        audio.currentTime = audioEl.currentTime;
});
audio.set('currentTime', 0);
audio.set('volume', 0.5);
```

2．音乐播放器

为音乐播放器使用CSS类mplayer。这也将是主 的类，并作为所有CSS规则和jQuery选择器的前缀。下面是“裸”播放器的CSS和HTML：

```
.mplayer { position: relative; width: 40%; height: 2.5em; margin: 50px 0 100px 0; }

<div class="mplayer ui-widget"></div>
```

将宽度设置为40%，以便能够从一开始就获得一个灵活的播放器。调整浏览器大小并观察播放器大小的变化。当播放器非空时更容易看到效果。

除了mplayer类之外，主 还有一个ui-widget类，用于确定其中的元素设置了合适的样式。有关利用jQuery UI CSS Framework类设置主题的更多内容参见下一章。

一个空的 ，没有任何JavaScript，这就形成了一个不可见也没有任何动静的音乐播放器。下面添加一个播放按钮，启动音乐。

3．播放和暂停按钮

jQuery UI中还没有按钮插件。可以用一个a元素和一些语义命名的jQuery UI CSS Framework图标类来建立一个：

下面是CSS：

```
.mplayer .buttons-container { position: absolute; top: 10px; left: 10px; }
.mplayer .buttons-container .playpause { height: 1.2em; width: 1.2em; display: block;
        position: relative; top: -2px; left: -2px; }
.mplayer .buttons-container .playpause .ui-icon { margin: -1px 0 0-1px; }
.mplayer .playpause .ui-icon-play, .paused .playpause .ui-icon-pause { display: none; }
.paused .playpause .ui-icon-play { display: block; }
```

下面是HTML：

```
<div class="mplayer ui-widget">
        <div class="buttons-container">
                <a class="playpause ui-state-default ui-corner-all" href="#">
                        <span class="ui-icon ui-icon-play"></span>
                        <span class="ui-icon ui-icon-pause"></span>
                </a>
        </div>
</div>
```

通过几条CSS规则，就能拥有一个暂停/播放两用按钮。根据前面的CSS，一次只能看到一个图标，显示的是播放还是暂停取决于div.mplayer是否有paused类。但是不同设计人员可以用相同的HTML，让两个图标都可见，但是根据歌曲是否播放使用不同的颜色和不透明度。

下面是JavaScript：

```
$('.mplayer .playpause').click(function() {
        var player = $(this).parents('.mplayer');
        if (player.is('.paused')) {
                $('.mplayer').removeClass('paused');
                audio.play();
        } else {
                $('.mplayer').addClass('paused');
                audio.pause();
        }
        return false;
})
.hover(function() { $(this).addClass('ui-state-hover'); },
        function() { $(this).removeClass('ui-state-hover'); })
.focus(function() { $(this).addClass('ui-state-focus'); })
.blur(function() { $(this).removeClass('ui-state-focus'); });
$('.mplayer').addClass('paused');
```

按钮需要用JavaScript进行如下操作：

- 在单击按钮时，根据div.mplayer上有没有paused类，调用audio.play()或audio.pause()函数。
- 切换.mplayer上的paused类。
- 对鼠标和键盘的foucus、hover和blur事件做出反应。这是按钮插件方便的地方（有一个按钮插件正在开发中），但是对于这种简单的图标按钮，不需要太多的代码。

不要忘记return false;语句，因为按钮是一个href属性为#的<a>元素。

加载了jQuery、jQuery UI和UI Lightness主题之后，图14-1展示了只带播放/暂停按钮的音乐播放器外观。

图14-1　播放和暂停按钮

如果单击播放按钮，它应该变成暂停按钮。如果再次单击，按钮应该变回播放按钮。还要注意，鼠标悬停时的效果和用键盘的Tab键将焦点移进/移出按钮时的视觉提示。如果你在一个支持audio元素的浏览器中且元素的src属性指向一个支持的音乐文件，在单击播放按钮时应该听到音乐。

4．当前和总时间标签

下一步是添加两个标签，一个显示在歌曲中的当前位置，另一个显示歌曲的总时间。这些标签相当简单。

下面是CSS：

```
.mplayer .currenttime { position: absolute; top: 0.6em; left: 2.2em;
        width: 3em; text-align: center; background: none; border: none; }
.mplayer .duration { position: absolute; top: 0.6em; right: 2.2em;
        width: 3em; text-align: center; background: none; border: none; }
```

下面是HTML：

```
<div class="mplayer ui-widget">
        <div class="buttons-container">
                <a class="playpause ui-state-default ui-corner-all" href="#">
                        <span class="ui-icon ui-icon-play"></span>
                        <span class="ui-icon ui-icon-pause"></span>
                </a>
        </div>
        <span class="currenttime ui-state-default"></span>
        <span class="duration ui-state-default"></span>
</div>
```

下面是JavaScript：

```
function minAndSec(sec) {
        sec = parseInt(sec);
        return Math.floor(sec / 60) + ":" + (sec % 60 < 10 ? '0' : '') +
Math.floor(sec % 60);
}
$('.mplayer .currenttime').text(minAndSec(audio.currentTime));
$('.mplayer .duration').text(minAndSec(secondsTotal));

$audio
        .bind('timeupdate', function(event) {
                $('.mplayer .currenttime').text(minAndSec(audio.currentTime));
        });
```

我们将当前时间放在左侧，总时间放在右侧，中间为轨道条留出空间（见图14-2）。因为我们希望当前时间始终反映出在歌曲中的位置，所以绑定音频的timeupdate通知事件。这个事件本身不会告诉我们currentTime。因此，转向audio.currentTime属性。需要一个小的函数将其格式化为分：秒，因为音频图层中的时间以秒表示：

图14-2　当前和总时间标签

5．表示歌曲位置的滑块轨道

我们已经有了一些进展，下一步是轨道栏。它由一个简单的 组成，但是我们打算通过调用.slider()为它添加轨道和手柄。我们将使用Slider插件的range:'min'选项，以便使0：00和当前时间之间的区域加上阴影。对了，还要将max选项设置为歌曲的长度（以秒表示）。如果歌曲长度为3.5分钟，将max设置为210。不需要任何计算，因为audio.duration已经告诉我们歌曲的总秒数。Slider的其他默认值对我们来说可以正常使用：max:0、step:1。

CSS如下：

```
.mplayer .track { top: 11px; margin: 0 5.2em; margin-top: -2px;
        border-style: none; }
.mplayer .track .ui-slider-handle { border-left-width: 0; height: 1.1em;
        top: -0.24em; width: 2px; margin-left: -3px; }
```

HTML如下：

```
<div class="mplayer ui-widget">
        <div class="buttons-container">
                <a class="playpause ui-state-default ui-corner-all" href="#">
                        <span class="ui-icon ui-icon-play"></span>
                        <span class="ui-icon ui-icon-pause"></span>
                </a>
        </div>
        <span class="currenttime ui-state-default"></span>
        <div class="track"></div>
        <span class="duration ui-state-default"></span>
</div>
```

JavaScript如下：

```
$('.mplayer .track')
        .slider({
                range: 'min',
                max: audio.duration,
                slide: function(event, ui) {
                        $('.ui-slider-handle', this).css('margin-left',
                                (ui.value < 3) ? (1 - ui.value) + 'px' : '');
                        if (ui.value >= 0 && ui.value <= audio.duration) {
                                audio.set('currentTime', ui.value);
                        }
                },
```

```
                change: function(event, ui) {
                        $('.ui-slider-handle', this).css('margin-left',
                                (ui.value < 3) ? (1 - ui.value) + 'px' : '');
                }
        })
        .find('.ui-slider-handle').css('margin-left', '0').end()
        .find('.ui-slider-range').addClass('ui-corner-left').end();
$audio
        .bind('timeupdate', function(event) {
                $('.mplayer .track').each(function() {
                        if ($(this).slider('value') != audio.currentTime) {
                                $(this).slider('value', audio.currentTime);
                        }
                });
                $('.mplayer .currenttime').text(minAndSec(audio.currentTime));
        });
```

滑块手柄居中对齐，意味着在min值，手柄的左半部超出滑块的左侧，而在max值，手柄的右半部超出滑块的右侧。我们已经采用了比常规更细的手柄，并且去掉了左边框，使其更好地靠近边缘，但是在接近min的地方仍然需要稍作调整。这就是如下代码行的用途：

```
slide: function(event, ui) {
       $('.ui-slider-handle', this).css('margin-left',
                (ui.value < 3) ? (1 - ui.value) + 'px' : '');
       if (ui.value >= 0 && ui.value <= audio.duration) {
               audio.set('currentTime', ui.value);
       }
},
change: function(event, ui) {
        $('.ui-slider-handle', this).css('margin-left',
                 (ui.value < 3) ? (1 - ui.value) + 'px' : '');
}
```

而且，在slide回调函数中，在让音频转到某一点之前检查该值是否有效。当用户拖动滑块时就是这种情况，我们必须移动歌曲中的播放点。这一做法考虑到了“刷洗”的情况。如果仅在change回调中处理这种情况，在用户单击或者拖动滑块手柄到新的播放点时，音频在用户放开鼠标之前不会变化。图14-3展示了创建的滑块。

图14-3　表示歌曲位置的滑块轨道

6．轨道中显示缓冲的进度栏

接下来我们准备做些有趣的事情。如果我告诉你，我们可以在同一个元素上调用两个不同的jQuery UI插件，那会怎么样呢？在这个例子中它能工作得很好。我们已经有一个轨道栏，它是作为 创建的，在它上面调用.slider()。除了为.track元素添加一个ui-slider类，jQuery UI的Slider插件创建两个元素——滑块手柄（.ui-slider-handle）和滑块范围（.ui-slider-range，因为指定了range:min），并附加到轨道上。幸运的是，这就是需要对 所做的全部修改了。 仍然是一个 。所以，赋予它两个功能，并调用.progressbar()。这将在显示当前时间的区域后面显示缓冲的情况。让我们来看看效果。

CSS如下：

```
.mplayer .ui-progressbar .ui-progressbar-value { border-style: none; }
```

JavaScript如下：

```
var secondsCached = 0, cacheInterval;
$('.mplayer .track')
        .progressbar({
                value: secondsCached / secondsTotal * 100
        })
        .find('.ui-progressbar-value').css('opacity', 0.2).end();
```

```
cacheInterval = setInterval(function() {
        secondsCached += 2;
        if (secondsCached > secondsTotal) clearInterval(cacheInterval);
        $('.mplayer .track.ui-progressbar')
                .progressbar('value', secondsCached / secondsTotal * 100);
}, 30);
```

这里没有HTML，因为重用了前一部分中的.track元素。你也许还没有注意到，缓冲的代码完全是虚构的，但是，它能够正常工作；它并不能真正表现歌曲的缓冲，只是模拟而已。但是它工作得很棒！如果你确实有一个音乐资源可以加载并缓冲，而且你的音频API支持缓冲通知，你就可以绑定到有关的事件，并且设置前面看到的进度栏的数值（0～100之间）。和Slider不同，你不能为进度条指定自定义的max值。但是那确实有意义，对吗？进度介于0%～100%。

好了，我们已经有了一些概念验证代码。当页面加载时，缓冲进度将很快向前延伸，就像文件飞速加载一样，但是又不完全像本地文件的加载。这样的视觉效果很有趣。图14-4展示了创建的进度栏。缓冲进度指示器是另一个需要虚构的东西吗？因为它不是真正的缓冲进度，你可以忽略它。会发生什么情况？这取决于你的音频API和后端。所以，如果你没有或者不想要缓冲进度，可以省略，或者为了美观而留着它。

图14-4　轨道中的进度栏用于显示缓冲

7．音量滑块

需要添加一个音量控件。滑块对此很合适，从volume: 0拖动到volume:1，将step设置为0.01：

```
$('.mplayer .volume').slider({
        max: 1,
        step: 0.01,
        value: audio.volume,
        slide: fnSlide,
        change: fnChange
});
```

就这样，简单得让人不敢相信。可是，为什么不能这么简单呢？上述代码无疑能够正常工作，但是它会占据一些空间，方向也是个问题。如果将它设计为水平方向（Slider插件的默认值），就会和表示播放进度的滑轨争夺水平空间。更不用说我们还想削减播放器的空间了。那么，应该为滑块添加orientation:'vertical'选项吗？这也能够正常工作，但是这意味着播放器现在为了容纳音量控制需要有100个像素高。其余控件只需要30个像素的高度。一定还有更好的办法。

确实有更好的解决方案。在不用的时候隐藏音量滑块的滑动条。我们保持滑块手柄可见，并为它添加一个小喇叭图标。然后将控件高度设置为0，隐藏其余部分。当用户把光标悬停于手柄之上时，将高度设置为100个像素。在光标移出时，将移除滑动条，高度又回到0。而且，由于它的容器是在相对包装器中绝对定位的，因此在它完全可见时不会影响到播放器的整体高度。

还有一个问题，当滑动条显示时，假定音量为0.1（10%），这意味着手柄靠近底部。手柄应该跳下来，还是向上？当用户滑动手柄时又该怎么样呢？如果用户从10%拖动到90%然后放开鼠标会怎么样呢？当滑动条再次隐藏时，它又会跳回去，真麻烦！

所以，这就是我们需要努力的地方。我们将在拖动中保持手柄固定。用户向上拉增加音量，向下则降低音量。滑动条包含range:"min"选项，手柄下面的阴影部分将相应地上下移动。

CSS如下：

```
.mplayer .volume-container { position: absolute; top: 12px; right: 12px; }
.mplayer .volume { height: 0; margin-top: 5px; }
```

HTML如下：

```
<div class="mplayer ui-widget">
        <div class="buttons-container">
                <a class="playpause ui-state-default ui-corner-all" href="#">
                        <span class="ui-icon ui-icon-play"></span>
                        <span class="ui-icon ui-icon-pause"></span>
                </a>
        </div>
        <span class="currenttime ui-state-default"></span>
        <div class="track"></div>
        <span class="duration ui-state-default"></span>
        <div class="volume-container">
                <div class="volume">
                        <a href="#" class="ui-state-default ui-corner-all
ui-slider-handle">
                                <span class="ui-icon ui-icon-volume-on"></span>
                        </a>
                </div>
        </div>
</div>
```

JavaScript如下：

```
$('.mplayer .volume')
        .slider({
                max: 1,
                orientation: 'vertical',
                range: 'min',
                step: 0.01,
                value: audio.volume,
                start: function(event, ui) {
                        $(this).addClass('ui-slider-sliding');
                        $(this).parents('.ui-slider').css({
                                'margin-top': (((1 - audio.volume) * -100) + 5) + 'px',
                                'height': '100px'
                        }).find('.ui-slider-range').show();
                },
                slide: function(event, ui) {
                        if (ui.value >= 0 && ui.value <= 1) {
                                audio.set('volume', ui.value);
                        }
                        $(this).css({
                                'margin-top': (((1 - audio.volume) * -100) + 5) + 'px',
                                'height': '100px'
                        }).find('.ui-slider-range').show();
                },
                stop: function(event, ui) {
                        $(this).removeClass('ui-slider-sliding');
                        var overHandle = $(event.originalEvent.target)
                                .closest('.ui-slider-handle').length > 0;
                        if (!overHandle) {
                                $(this).css({
                                        'margin-top': '',
                                        'height': ''
                                }).find('.ui-slider-range').hide();
                        }
                },
                change: function(event, ui) {
                        if (ui.value >= 0 && ui.value <= 1) {
                                if (ui.value != audio.volume) {
                                        audio.set('volume', ui.value);
                                }
                        }
                }
        })
        .mouseenter(function(event) {
                if ($('.ui-slider-handle.ui-state-active').length) {
                         return;
                }
                $(this).css({
                        'margin-top': (((1 - audio.volume) * -100) + 5) + 'px',
                        'height': '100px'
                }).find('.ui-slider-range').show();
        })
        .mouseleave(function() {
```

```
                $(this).not('.ui-slider-sliding').css({
                        'margin-top': '',
                        'height': ''
                }).find('.ui-slider-range').hide();
        })
        .find('.ui-slider-range').addClass('ui-corner-bottom').hide().end();
```

在拖动滑动条时，调整以负值表示的margin-top选项值反比于当前值，保持手柄静止。这一切发生在如下的代码中：

```
$(this).parents('.ui-slider').css({
        'margin-top': (((1 - audio.volume) * -100) + 5) + 'px',
        'height': '100px'
})
```

图14-5展示了播放器中的音量滑块。

图14-5　音量滑块

这种交互需要识别你不是在与鼠标相反的方向上拖动移动的滑动条。但是与此同时，鼠标、阴影部分的大小和音量的变动应该在逻辑上达成一致：向下表示降低音量，向上表示增大音量。而且，如果你愿意，可以让光标悬停以显示滑动条，将光标移到所要设置的音量位置，然后单击。

8．窗口组件背景和顶部样式

现在要为几个元素添加jQuery UI CSS Framework类，以和内部控件相匹配的方式设置播放器的样式：

CSS如下：

```
.mplayer .bg { position: absolute; width: 100%; height: 100%; top: 0;
        bottom: 0; left: 0; right: 0; border: none; }
.mplayer .rod { position: absolute; top: -2px; left: -0.4%; right: -0.4%;
        width: 100.8%; height: 3px; overflow: hidden; border: none; }
.mplayer .hl { position: absolute; top: 2px; left: 1%; right: 1%; width: 98%;
        height: 1px; overflow: hidden; border: none; }
.mplayer .hl2 { position: absolute; top: 2px; left: 2%; right: 2%; width: 96%;
        height: 3px; overflow: hidden; border: none; }
```

JavaScript如下：

```
$('.mplayer').each(function() {
        $('.bg:first', this).css('opacity', 0.7);
        $('.bg:last', this).css('opacity', 0.3);
})
$('.mplayer .rod').css('opacity', 0.4);
$('.mplayer .hl').css('opacity', 0.25);
$('.mplayer .hl2').css('opacity', 0.15);
```

HTML如下：

```
<div class="mplayer ui-widget">
        <div class="bg ui-widget-header ui-corner-bottom"></div>
        <div class="bg ui-widget-content ui-corner-bottom"></div>
        <div class="rod ui-widget-header"></div>
        <div class="hl ui-widget-content"></div>
        <div class="hl2 ui-widget-content"></div>
        <div class="buttons-container">
                <a class="playpause ui-state-default ui-corner-all" href="#">
                        <span class="ui-icon ui-icon-play"></span>
```

```
                        <span class="ui-icon ui-icon-pause"></span>
                </a>
        </div>
        <span class="currenttime ui-state-default"></span>
        <div class="track"></div>
        <span class="duration ui-state-default"></span>
        <div class="volume-container">
                <div class="volume">
                        <a href="#" class="ui-state-default ui-corner-all
ui-slider-handle">
                                <span class="ui-icon ui-icon-volume-on"></span>
                        </a>
                </div>
        </div>
</div>
```

在这里，使用不透明度和分层从任何jQuery UI主题中得到更多的阴影。图14-6展示完成的作品：

图14-6　窗口组件背景和顶部样式

最后，图14-7展示了jQuery UI音乐播放器在几个预建的jQuery UI主题中的样例。

图14-7　在几个不同的ThemeRoller主题中的jQuery UI音乐播放器

第15章　jQuery UI主题

*Maggie Wachs, Scott Jehl, Todd Parker*和*Patty Toland*
（*Filament Group, Inc.*）

15.0 导言

jQuery UI的优势之一是它很容易集成到各种网站和应用程序中。集成取得成功的主要因素之一是jQuery UI为其窗口组件应用和较大规模的网站或者系统设计相一致的观感的能力。

jQuery UI的设计专门简化了自定义主题。可以为窗口组件创建高度自定义的视觉样式——不仅自动包含颜色和纹理，还用如下插件提供了完备的交互状态：

- jQuery UI CSS Framework是一个全面的CSS类集，用于在不同窗口组件中应用一致的样式和行为。
- ThemeRoller是jQuery UI的主题创建工具。

上述这些工具组合起来提供了一种方法，可以轻松而一致地修改jQuery UI官方窗口组件和自定义组件的观感，使它们无缝地融入你的网站或者应用程序。

本章关注的是如何最大限度地利用这些工具，不管你是用它们自定义jQuery UI官方窗口组件还是将它们加入你的自定义开发工作流。我们将从jQuery UI CSS的概要和ThemeRoller的使用方法开始，然后将按照如下顺序介绍4个有关主题的秘诀：

1. 用ThemeRoller设置jQuery UI窗口组件的样式；

2. 覆盖jQuery UI布局和主题样式；

3. 将ThemeRoller主题应用到非jQuey UI组件；

4. 在一个页面上引用多个主题。

每个秘诀都从一个基本的设计难题示例开始，并逐步应用各种技术自定义主题。为此，你将经常看到某个秘诀引用本章另一个秘诀的情况。

在开始设置窗口组件样式之前，理解所有jQuery UI类的构造方式以及和ThemeRoller的协同方式非常重要。

理解jQuery UI CSS的各个组件

当创建jQuery UI CSS时，首要目标是简化设置过程，使开发人员能够快速无缝地部署窗口组件，并且在代码中无须设计复杂的标记或者CSS。

早期，在我们集成第三方JavaScript窗口组件到自有项目的经历中，自定义程序库窗口组件的外观比正确设置和运行脚本难得多。在自定义设计的脚本中，核心和窗口组件插件脚本在后台处理复杂的任务，可配置的选项简化了自定义的工作，与此不同，窗口组件的样式通常由单个类或者标记来完成。我们必须在标记中确定类，并在修改之前分析样式规则的结构，这通常需要花费好几个小时筛查代码和CSS，使用Firebug找出在哪里指定类或者查找内联样式，更新背景图像，然后替换样式规则或者在标记中编辑类，使外观近似于项目的设计（这还只有在代码和CSS都进行了合理的组织并且保持一致时才能做到）。

对于我们来说，这感觉上是个退步；对基本没有设置样式的窗口组件添加自定义的观感应该比从已经设置样式的窗口组件中提取CSS，然后尝试领会哪条规则可以安全地替换要容易得多。我们决定开发更好的方法，一致性地应用样式，使它们能够在一组窗口组件和较大的网站或者应用设计系统中协调地工作。

为了解决jQuery UI中的这个问题，我们将jQuery UI CSS分成几个逻辑组件，类似于脚本的结构，并将窗口组件正常工作必需的核心结构样式（位置、浮动和内边距）从可自定义的主题样式（颜

色和背景图片）中独立出来。这样，开发人员可以修改、使窗口组件匹配项目的类现在可以分为两个基本的类别：

- 窗口组件相关类包括格式化特定窗口组件结构和布局所需的所有样式，包括位置、间距和尺寸，以及其他帮助其正常工作的布局相关样式。例如，选项卡所用的窗口组件类包括浮动选项卡，使其在水平的行中显示并选择性地隐藏相关选项卡内容面板的样式。

窗口组件相关类包含在下载一个或者多个jQuery UI窗口组件的同时下载的CSS中（从秘诀15.1中可以学到下载和引用jQuery UI CSS的方法）。类得名于它们所控制的特定窗口组件，类名通常以ui-[widgetname]前缀开始，例如，ui-tabs。

- 框架类为所有窗口组件应用自定义主题观感——包括基本字体、背景颜色和纹理、字体和图表颜色、形状（圆角半径）和交互状态反馈。框架类按照基本用途取名，例如，有些类提供状态反馈（ui-state-default、ui-state-hover）或者应用圆角（ui-corner-all、ui-corner-top），它们的设计意图是在整个网站或者应用程序中重用。实际上，它们可以应用到任何窗口组件，包括jQuery UI或者其他JavaScript程序库创建的组件或者自定义窗口组件。

在实践中，指定这些类的组合来设置jQuery UI窗口组件的样式——一个或者多个描述性的窗口组件相关类和一系列通用的框架类，这些类共同创建最后的外观。例如，查看折叠组件标题的标记：

```
<h3 class="ui-accordion-header ui-state-active ui-corner-top">code</h3>
```

上述设计中应用了三个类，为组件带来非常特殊的样式：

- ui-accordion-header是这个组件特有的窗口组件相关类；它设置结构化样式规则（位置、尺寸、内外边距），但是不适用于任何颜色或者图片。
- ui-state-active是一个框架类，添加主题颜色和背景图片显示组件的活动状态。
- ui-corner-top也是一个框架类，指定标题的顶部有圆角。

尽管这种方法意味着为一些元素指定多个类，但是这种强有力的系统使得为不限数量的窗口组件（甚至是自定义组件）应用非常轻量级的主题非常容易。谨慎地将结构样式从主题中分离出来还意味着你可以在任何时候投入新的主题，而不用担心破坏现有的窗口组件。

我们还想简化新观感的创建，或者精确地匹配现有设计，而不需要深入学习CSS或者Adobe Photoshop之类照片编辑工具的专业知识。ThemeRoller使开发人员可以编辑由框架类设置的样式规则，而没有必要接触CSS或者进行人工图像制作。

ThemeRoller是一个Web应用程序，为设计和下载jQuery UI自定义主题提供了有趣和直观的界面。ThemeRoller提供了修改如下主题样式的工具：

- **所有窗口组件的基本字体**。基本字体设置用于主题中所有窗口组件的标准字形、大小和粗细（正常或者加粗）。默认情况下，字体的大小以“em”为单位。建议使用ems代替像素，这样在用户操纵浏览器文本大小时，文本将随着窗口组件容器伸缩，但是如果你喜欢，也可以指定像素大小。和标准CSS一样，提供字体范围以防用户计算机上没有安装首选字体，并以通用字体样式如“serif”或者“sans-serif”作为字体样式字符串的结尾，是一种好的习惯。
- **圆角半径**。圆角半径可以一致地应用到主题中的所有窗口组件，使它们有圆角的外观。每个半径值后面必须有单位：固定半径采用像素值，采用em值能够响应文本大小的变化，值为0则采用方形的边角。较小的像素值使窗口组件的边角更接近方形，而较大的值使其更圆。

> **注意**
>
> 在本书编写的时候，CSS3中的边角设置和我们在Framework中相同，但还没有得到一些现代浏览器（包括Internet Explorer）的支持。请参见秘诀15.1中的补充材料，学习如何为这些浏览器增加圆角支持。

- **标题、工具栏和内容区域**。这些工具设置带有半透明纹理的背景颜色以及边框、文本和图标的颜色。例如，标题样式可用于对话框或者日期选择器的标题栏，以及滑块或者进度条的选中区域，

而内容样式用于选中的折叠控件或者选型卡的内容区域。

- **可单击元素的默认、活动和悬停状态**。用户交互有三种不同的状态：default是标准的可单击状态，hover用于光标放在对应项之上时的视觉反馈，active用在对应项被选中时。每个可单击状态都由带有半透明纹理的背景颜色以及边框、文本和图标的颜色表示。记住，每种状态都应该有足够的差异，才能为用户带来充分的反馈。
- **高亮和错误状态**。这是用于表达系统状态的特殊样式。高亮状态用在吸引用户注意力的信息上，还可以表示日历组件中当天的日期，在Ajax屏幕发生更新时也很有用。错误状态用于表示出现了需要用户注意的问题，例如，显示表单验证问题或者警告用户系统故障。高亮状态和错误状态都由带有半透明纹理的背景颜色以及边框、文本和图标的颜色定义。这些状态应该与主题中标准内容文本和背景颜色产生对比，两者之间也应该有足够的差异，以便清晰地表示吸引注意力和表达警告信息的意图。
- **作为覆盖层的模态屏幕**。模态屏幕是处于模态对话框和下面的页面内容之间的一个层次，通常用于在模态对话框显示的时候暂时禁用其他内容。这一工具设置模态屏幕的背景颜色和不透明度样式。如果你对特定的窗口组件完全不需要模态覆盖，可以通过窗口组件的modal选项切换。
- **阴影样式**。和高亮及错误状态一样，阴影样式可以有选择地应用到覆盖层。阴影有背景颜色、纹理和不透明度（和标题及可单击元素一样），还有指定阴影和组件左上角偏移量的阴影厚度，以及圆角半径。为了使阴影均匀地围绕组件，顶部和左侧偏移量应该为负数且等于阴影厚度。和标准圆角半径一样，可以设置以像素或者em值为单位的阴影圆角半径，或者输入0产生方形的边角。

ThemeRoller界面让你直接编辑前面所述的所有框架类样式，预览在jQuery UI窗口组件上的设计变化。一旦创建主题，ThemeRoller自动生成和打包所有必需的CSS和背景图片——你只要下载生成的主题样式表并在项目中引用即可。（在jQuery网站的ThemeRoller部分或者*http://themeroller.com*可以找到ThemeRoller）

现在我们对jQuery UI CSS和ThemeRoller已经有了了解，下面将有4个使用它们自定义主题的秘诀。首先，我们将从用ThemeRoller创建主题和设置组件样式的简单步骤开始（秘诀15.1）；然后我们将转向更复杂的步骤——覆盖框架类以得到更多自定义的主题（秘诀15.2）；接着会在这个项目中使用框架类（秘诀15.3）；最后研究如何在单个页面上为复杂界面使用多个主题（秘诀15.4）。

> **注意**
>
> 有些设计人员和开发人员对为网站或者应用程序中的jQuery UI和ThemeRoller就绪组件编辑和预览主题感兴趣，我们为他们开发了一个可下载的ThemeRoller书签工具。要了解有关这个书签的内容并下载，参见*http://ui.jquery.com/themeroller*。

15.1　用ThemeRoller设置jQuery UI窗口组件样式

15.1.1　问题

你的网站或者应用程序所用的jQuery UI窗口组件必须匹配已经确定的设计。

15.1.2　解决方案

使用ThemeRoller，这是一个简单的Web应用程序，用于编辑jQuery UI CSS框架类，自定义jQuery UI窗口组件的观感。

> **注意**
>
> 本秘诀有如下假设：
>
> - 你有CSS工作原理的基本知识，具体来说就是样式层叠、优先顺序和使用选择器类、id或者元素限定范围的方法。（关于建议的资源，请参考15.5节）。
> - 你已经熟悉了jQuery UI CSS类。（如果你不熟悉，可以复习15.0.1节）

我们来看一个例子。

我们正在设计一个新的旅游预订网站，具体地说，我们构建预订航班的一部分界面。网页的设计有一组选择预定类型的选项卡组成（航班、汽车租赁或者套餐），航班预订选项卡包含一个输入乘客数量的表单、选择出发和到达城市的选择列表，设置出发和返回日期的日历以及一个提交按钮（见图15-1）。

图15-1　旅游程序的最终目标设计

在这个秘诀中，将使用jQuery UI窗口组件来实现选项卡和日历选择器，并用ThemeRoller中创建的一个自定义主题设置它们的样式（除了标准的ThemeRoller输出之外，还可以修改主题样式表，使其更接近于你的设计——你将在秘诀15.2～15.4中看到具体的做法）。

第1步：打开ThemeRoller

打开jQuery UI网站（*http://jqueryui.com*），从顶部的导航条中选择Themes，或者直接访问*http://themeroller.com*。

ThemeRoller的界面分为两个主要部分，如图15-2所示：

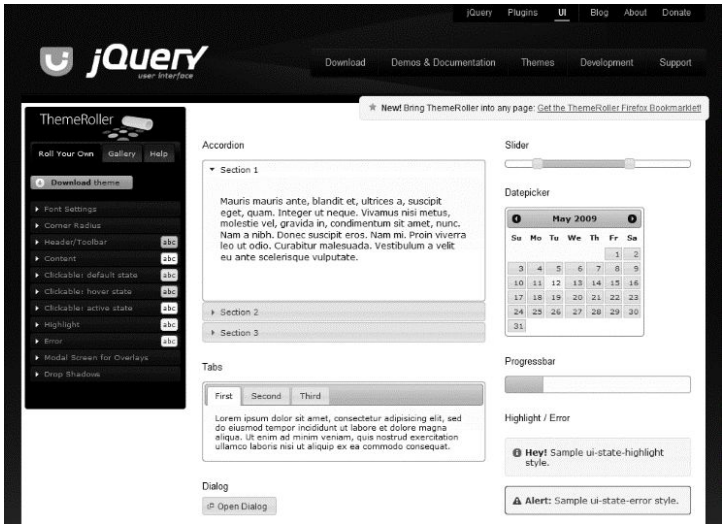


图15-2　ThemeRoller的默认视图，左边是工具栏窗格，右边是窗口组件预览窗格

- 左列是ThemeRoller工具栏窗格，提供设置和修改主题中所有样式设置的工具。
- 右边是示例窗口组件预览窗格，用于预览样式选择——每个窗口组件都交互式地显示各种样式（例如，用鼠标查看悬停和活动样式），当用工具栏编辑样式时实时更新。

ThemeRoller工具栏提供两种独特的方法自定义主题，可以从工具栏顶部的选项卡访问：

- Roll Your Own选项卡（见图15-3）是用来创建主题的自定义样式的工作区。可自定义的设置分为几个部分，每个部分都有进行快速样式选择的输入框和工具，包括用于所有窗口组件的基本字体和圆角半径设置，以及背景颜色和纹理、文本颜色和图标颜色的设置。

图15-3　ThemeRoller的Roll Your Own选项卡（A）提供修改字体、圆角半径和一组交互状态颜色的控件；Gallery选项卡（B）提供对各种预构建主题的单击访问

各个部分在默认情况下是关闭的，在标签的右侧以小图标的形式显示当前样式。在编辑的时候按照需要打开/关闭各个部分，右边的窗口组件预览示例实时反映你的修改。

> **注意**
>
> JavaScript对于使用ThemeRoller不是必要的。如果禁用JavaScript，会显示一个Preview按钮，单击它可查看更改。

- **Gallery选项卡**（见图15-3）提供一系列预先配置的主题，可以下载或者作为进一步自定义主题的起点。

第2步：创建和预览主题

对于旅游预订应用，我们将选择与最终设计接近的画廊主题Sunny（如图15-4所示）。

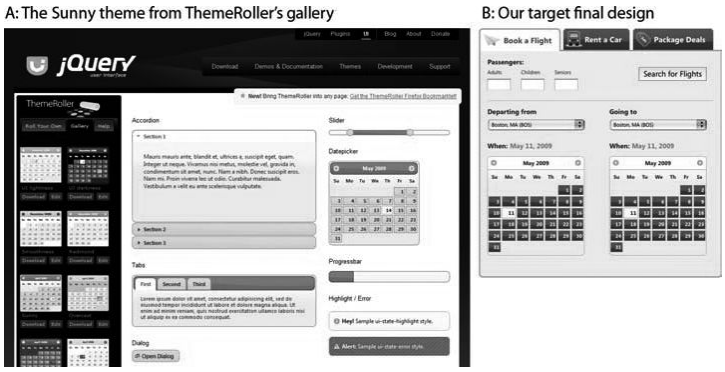


图15-4　ThemeRoller的画廊主题为自定义设计提供了广泛的起点；Sunny（A）与我们的目标主题（B）共享许多样式

Sunny指定了与最终设计类似的总体背景、字体和字体颜色，但是一些样式还需要编辑才能更接近于设计——例如，Sunny的选项卡是黄色的，背景为灰色，而我们的选项卡是暗灰色的，背景为白色。

单击画廊中Sunny图片之下的Edit按钮（转移到Roll Your Own视图）或者单击画廊中的Sunny图片激活它，然后单击工具栏顶部的Roll Your Own选项卡，可以轻松修改这些设置。

一旦在Roll Your Own选项卡中打开Sunny主题的设置，工具栏就预先用该主题的所有设置填充，并且可以开始编辑。调整如下设置，使Sunny主题匹配设计：

图15-5　字体设置和圆角设置部分

- **设置所有窗口组件的基本字体**：Sunny主题中的默认字体和目标设计似乎非常相似，但是可以简单地打开Font Settings（字体设置）部分（如图15-5所示），确认设置正确，或者改变字体系列、粗细和大小的值。字体系列可以接受多个以逗号分隔的字体名称（和标准CSS记法相同）。下面是一些设计注意事项和技巧：
  - 默认情况下，字体大小以“em”为单位。建议使用em代替像素大小，这样窗口组件文本将会在用户操纵浏览器文本大小的时候随着窗口组件容器缩放。
  - 提供一组字体，预防在用户计算机上没有安装首选字体的情况。字体串以“Serif”或者“Sans-serif”等通用字体结束是一个好的习惯。
- **应用圆角半径**。我们的设计在日期选择器和选项卡上包含了圆角。可以在ThemeRoller中打开Corner Radius（圆角半径）部分设置jQuery UI窗口组件的圆角半径，输入一个数值，然后是单位：用于固定半径的像素或者使半径与文本大小对应的em。较小的像素值使窗口组件的边角更方，而较大的值使边角更圆。将该值设置为0产生正方的边角。

> **注意**
>
> 在本书编写期间，有些现代浏览器（最明显的是Internet Explorer）不支持CSS3的borderradius属性，因此无法显示框架类所应用的圆角。边角显示为方形。如果你的设计包含圆角且必须在所有浏览器中一致性地显示，你可能应该使用ddRoundies等JavaScript程序库来显示圆角。
>
> 我们已经在Filament Group lab上编写了一个基本的教程，说明在项目中加入ddRoundies的方法：“Achieving Rounded Corners in Internet Explorer for jQuery UI with DD_roundies”（*http://www.filamentgroup.com/lab/achieving_rounded_corners_in_internet_explorer_for_jquery_ui_with_dd_roundi*）。

- **将默认选项卡和按钮变为灰色**。未选中的选项卡和折叠部分的标题或者日期选择器按钮一样，都是可单击元素，每个元素都指定一个类来表示其当前可单击状态：默认状态、悬停状态或者活动状态。这里将把默认状态背景颜色从灰色改为黄色，并更新文本和边框颜色以匹配设计（见图15-6）：

图15-6　ThemeRoller的可单击元素默认状态设置部分

1．打开“Clickable: default state”（可单击元素：默认状态）部分。

2．将光标焦点放到背景颜色字段（包含了以#为前缀的十六进制值），并选择新的暗灰色或者输入一个十六进制值；在本例中，输入值#333333。

3．文本颜色为暗灰色，现在与背景融为一体，所以还要更新默认状态的文本颜色，以便和背景产生对比。把文本颜色值改为#FFFFFF。

4．和文本一样，出现在标题中的图标也是灰色的，需要更新才不会消失在灰色背景中。为其指定颜色值#EEEEEE，这种颜色是背景的补色，但是对比度不会高于文本。

5．最后，将边框颜色从黄色改为浅灰色；输入数值#D2D2D2。

6．按下Tab或者Enter键，或者单击页面的其他地方，在右边预览窗口组件的变化。

- **更新悬停状态，匹配新的选项卡颜色**：可单击元素的悬停状态样式在光标放在选项卡、折叠部分或者日期选择器等可单击组件上时显示。既然默认的状态为灰色，就调整悬停状态的背景和文本颜色，使用互补的深灰色背景与白色文本和图标协调一致：

  1．打开“Clickable: hover state”部分。

  2．在背景颜色字段中输入值#111111。

  3．将文本和图标颜色更新为#FFFFFF。

  4．还要将边框颜色设置为比默认边框更深的灰色——#888888，更好地匹配设计。

- **将选项卡和日期选择器的标题背景改为白色**：标题样式出现在多个jQuery UI窗口组件中：在选项卡的后面，在日期选择器顶部的月/年反馈信息和导航按钮，作为滑块范围，作为进度条完成指示器。在设计中，标题是浅白色，带有灰色文本和深黄色的图标：

  1．打开Header/Toolbar（标题/工具栏）部分。

  2．在背景颜色字段中输入十六进制值#FFFFFF。

  3．单击背景输入字段旁边的纹理图标，选择第一个选项“flat”。（光标悬停于任何纹理图片上时会看到名称。）这样会删除背景图片，使得样式只设置一个平面背景颜色。

  4．将背景不透明度设置为100%，确保标题完全不透明。

  5．文本颜色为白色，在新的背景上不明显，所以将其改为暗灰色（#333333）以匹配默认的可单击状态。

  6．最后，将边框和图标颜色改为#EDAA12，文本颜色改为白色——#FFFFFF。

- **将内容容器边框颜色改为黄色**。内容边框出现在折叠部分的周围，确定选项卡、对话框、滑块、日期选择器和进度条的容器。在设计中该边框和标题边框使用相同的浅黄色：

  1．打开Content（内容）部分。

  2．将焦点放在边框颜色字段，输入值#EDAA12。

- **更新“活动”状态边框颜色，与容器融为一体**。更新容器边框颜色之后，你将会看到选中的折叠部分和选中的选项卡仍然有暗灰色的边框。这个颜色用可单击元素活动状态类设置：

  1．打开“Clickable: active state”部分。

  2．将焦点放在边框颜色字段，输入值#EDAA12。

> **注意**
>
> 你可以在任何时候将页面记入书签“保存”一个主题；ThemeRoller在每次刷新预览窗格时用相关的样式更新URL。将自定义主题记入书　签——甚至将多个主题记入书签进行比较——并从书签中重新加载主题进行修改或者为了下载而对其进行微调。
>
> 对于任何从ThemeRoller下载的主题，在样式表中都包含了完整的主题URL。打开样式表（如jqueryui-1.7.1.custom.css），搜索以如下内容开始的注释：

```
"To view and modify this theme,visit http://jqueryui.com/themeroller/..."
```

这时，我们已经使ThemeRoller主题尽可能地匹配旅游预订应用的设计（见图15-7）。现在这个主题已经可以下载。

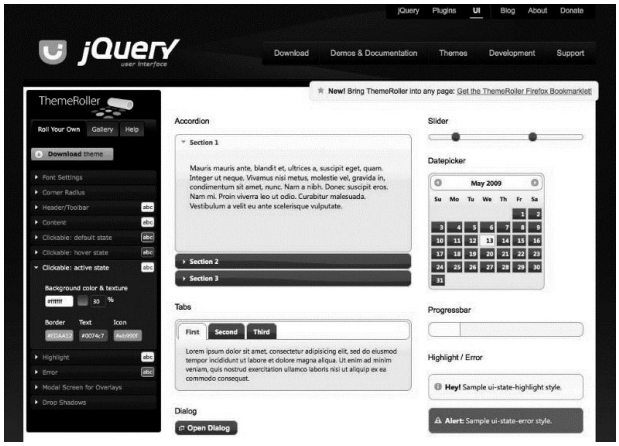


图15-7　最后的自定义ThemeRoller主题与设计很匹配

第3步：下载jQuery UI窗口组件和主题

单击ThemeRoller工具栏的Roll Your Own选项卡顶部的“Download theme”（下载主题）按钮，将打开jQuery UI下载构造器页面（见图15-8）。

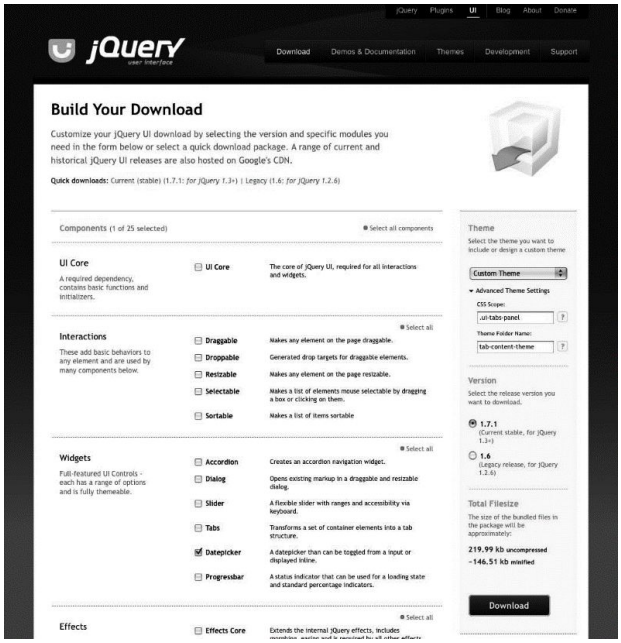


图15-8　jQuery下载器将UI核心、
你选择的交互和窗口组件以及你的主题合并为一个ZIP文件

在Theme下的右侧栏目中，你将会看到下拉式菜单中预先选中的Custom Theme（自定义主题）。

> **注意**
>
> 如果你从画廊中选择了一个默认主题且不作修改，就会看到该主题的名称（如Smoothness）。

接下来，我们将选择和主题一起下载的jQuery UI组件。默认情况下选择所有组件；简单地反选你不想下载的组件，或者单击Components（组件）部分顶部的“Deselect all components”（反选所有

组件），只下载主题样式表。对于旅游预订应用，需要jQuery UI核心脚本和用于选项卡和日期选择器的插件脚本。

最后，选择你想使用的jQuery UI版本；默认选中最新的稳定版本。单击Download按钮，在本地保存ZIP文件。下载的ZIP文件将命名为jquery-ui-1.7.1.custom.zip。

（下载构建器的Theme部分中的Advanced Theme Settings（高级主题设置）部分允许你下载部分主题——我们将在秘诀15.4中详细解说。）

警告

jQuery UI CSS随jQuery UI新版本的发行而更新——例如，新的发行版本不仅包含更新过的脚本，而且包含对CSS的修改和更新。
在本书编写的时候，jQuery UI的版本是1.7，本章中的秘诀和技术仅适用于该版本的主题功能。

第4步：将文件合并到项目目录中

接下来，打开前一步下载的ZIP文件，查看其内容。jQuery UI文件编排为如下目录结构；文件夹和文件的顺序因为操作系统而有所不同（图15-9展示了在Mac OS X上打开的文件夹）。

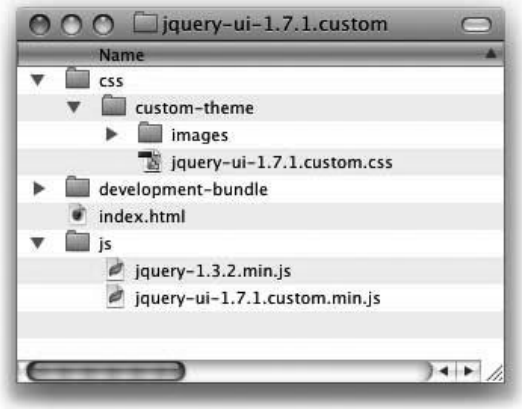


图15-9　jQuery下载文件夹结构

index.html

选中的UI组件的标记样例标记。

注意

如果你选择不下载任何组件，这个文件将不会包含在下载中。

/css

包含一个有如下文件的主题文件夹（例如，*custom-theme*）：

- 一个图片目录，包含框架图标和背景纹理。
- 你的主题样式表，如*jquery-ui-1.7.1.custom.css*，包含你刚刚编辑的样式，以及组件（如果下载）正常工作所必需的窗口组件相关样式。

/js

已经过编译的jQuery UI JavaScript文件。

/development-bundle

用于创建*css*和*js*文件夹中编译后版本的独立组件脚本和CSS，开放源码许可证文本和用jQuery UI进行高级开发所必需的相关资源。

当开发自己的项目时，一定要审核*index.html*中的标记，以它和*http://jqueryui.com*上的演示和文档作为指南，将组件标记和脚本集成到项目中。

对于旅游应用，我们将复制*css*目录中的主题文件夹并粘贴到项目的样式目录中；为了简单起见，将样式文件夹命名为CSS。

> **警告**
>
> 维护在主题文件夹中建立的目录结构很重要，这样图标图片就能被主题类正常引用。如果你修改了主题目录结构，在你决定升级到新版本的jQuery UI脚本和CSS时，可能不得不重复进行这些修改。

第5步：在项目中引用主题样式表

最后，在页面的<head>元素中包含对主题样式表的引用。

记住，样式表引用应该始终出现在对jQuery UI脚本的引用之前，以便让CSS先加载；因为有些窗口组件依赖CSS才能正常工作，所以这一步是必需的。

我们将在旅游预订应用中所有脚本加载之前引用主题样式表（粗体表示）：

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>travel application | Book a Flight, Rent a Car, or Find Package
Deals</title>
    <!-- jQuery UI styles -->
    <link rel="stylesheet" type="text/css" href="css/custom-theme/jquery-ui-
1.7.1.custom.css" />

    <!-- jQuery core & UI scripts -->
    <script type="text/javascript" src="js/jquery-1.3.2.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui-1.7.1.custom.min.js"></script>

    <script type="text/javascript">
    $(function(){
        $('#travel').tabs();
        $("#departure-date-picker").datepicker({altField: '#departure-date',
altFormat: 'MM d, yy'});
        $("#arrival-date-picker").datepicker({altField: '#arrival-date',
altFormat: 'MM d, yy'});
    });
    </script>
</head>
```

当jQuery UI窗口组件标记和样式已经处于项目中的合适位置时，在浏览器中预览页面，以确定正确应用样式。预览旅游应用说明主题正确应用——正如你在图15-10中所看到的，默认选项卡为灰色，标题是白色而文本和图标的颜色与选择匹配。

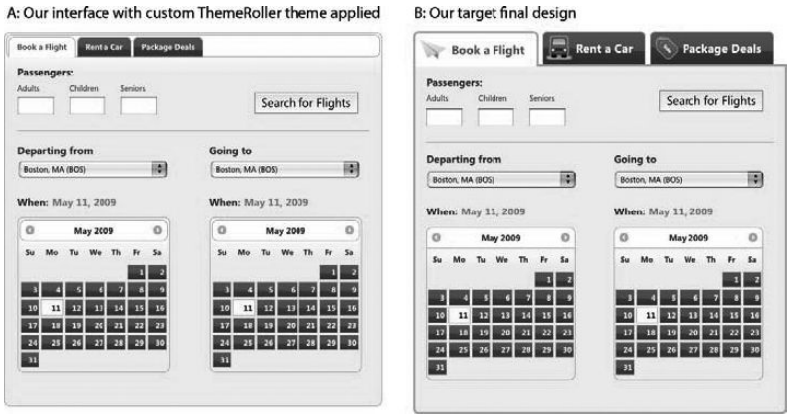


图15-10　应用自定义ThemeRoller主题的设计界面（A）和最终目标设计（B）

但是，界面明显需要更多的工作才能匹配目标设计（见图15-10）：选项卡太小，无法看到它们的自定义图标，日期选择器标题应该出现在顶部，可是它却包围在日期选择器窗口组件内。在秘诀15.2中，我们将研究如何对主题样式进行小的调整，使这些元素更好地匹配设计。

注意

如果页面中的jQuery UI窗口组件没有选择主题样式表，再次确认页面中引用的主题样式表目录路径，更正录入错误。如果不能解决这个问题，暂时禁用任何非jQuery样式或者脚本，测试它们对主题样式表的加载或者显示是否有影响，并修复相关的缺陷。

15.1.3　讨论

因为ThemeRoller主题是为了提供全面的体验和应用到多个窗口组件而建立的，所以考虑各种框架类之间的相互作用很有帮助。如果你选择从头开始创建自己的主题，或者从本质上修改现有的主题，下面是应该考虑的一些要点：

要为标题和工具栏以及内容区域创建统一的背景，使选项卡的“开启”状态无缝地与可见内容面板联系起来，并使内容区域背景和边框与可单击元素活动状态背景和边框相匹配。

对于可单击元素，状态之间的区别应该足够清晰，为用户提供足够的反馈。下面是确保状态相互协调以提供特别的视觉差异的两种方法：

- 为可单击元素的默认和活动可单击状态使用镜像图片纹理，实现三维观感。例如，用于默认按钮状态的“高亮”纹理和用于活动按钮状态的“嵌入”纹理很相配。当单击按钮时，它的外观就像实际按下一样。
- 如果对可单击元素和悬停状态使用相同的纹理，要确保背景颜色和图像不透明度有足够的差异（通常至少有10%的差别），以提供清晰的视觉变化。

为多个样式使用相同的图片，优化主题的加载速度。例如：

- 当为多个状态使用相同的图标颜色时，样式表进行的HTTP请求较少，改进了页面的性能。
- 也可以为多种状态使用相同的背景图片（颜色加上纹理不透明度）。如果这样做，重要的是确保其他样式元素（边框、文本和图标颜色）有明显的差别。

注意

为了改变自定义主题而又不用从头开始，打开原始主题样式表，搜索以“To view and modify this theme, visit http://jqueryui.com/themeroller/...，”开始的注释，并将主题URL复制和粘贴到浏览器的地址栏，打开ThemeRoller，预先加载主题的设置。

15.2 覆盖jQuery UI布局和主题样式

15.2.1 问题

在ThemeRoller中创建并在你的项目中创建和引用的自定义（或者标准画廊）主题匹配目标设计，但还没有完全匹配。你需要修改样式，但是同时希望确保对这些样式的编辑不会给新版本jQuery UI脚本和CSS的更新带来困难。

15.2.2 解决方案

创建自定义覆盖样式，限制在需要附加非ThemeRoller样式的组件中，并对它们进行组织，使得它们不会与标准的jQuery UI CSS文件发生冲突或者覆盖。

警告

后面的秘诀有如下假设：

- 你有CSS工作原理的基本知识，具体地说是样式层叠、优先级，以及用选择器类、ID或者元素限制样式表的方法。（建议的资源请参见15.5节。）
- 你已经熟悉了用ThemeRoller创建和编辑主题的方法（如果还不熟悉，复习秘诀15.1，该秘诀详细描述了创建和应用主题的方法。）

当下载jQuery UI脚本和主题样式表时，每个jQuery UI窗口组件都已经设置了现成的样式；不需要任何修改就可以将窗口组件和样式加入网站。但是默认的样式可能不完全匹配项目中的设计约定。例如，你可能想要减少标题的内边距或者使用自定义背景图片。

我们重拾前一个秘诀未完成的工作，继续完成旅游预订应用。正确地创建、下载和应用了主题样式表；但是，选项卡和日期选择器的默认jQuery UI样式与项目的设计不太匹配，如图15-11所示。

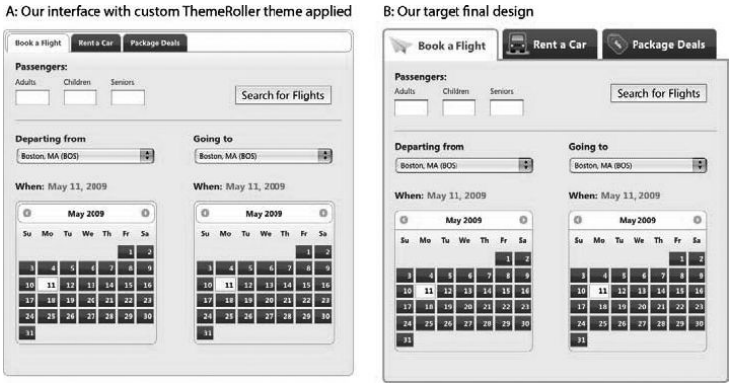


图15-11 应用自定义ThemeRoller主题的设计界面（A）和设计人员提供的目标设计（B）

第1步：检查jQuery UI插件所用的窗口组件标记和样式

首先，检查jQuery UI组件标记中的类如何指定，更好地理解应用这些类的方法（以及可能的覆盖方式）。

从选项卡标记开始。当在页面上初始化jQuery UI选项卡时，插件脚本为窗口组件标记指定多个类，这在下面说明（请注意，这些标记是由插件脚本变换或者插入的，是完成的产品，而不是JavaScript运行之前页面中就有的标记）。

特别要注意以ui-tabs前缀开始的类，这是选项卡的窗口组件相关类，在代码中以粗体显示：

```
<div class="ui-tabs ui-widget ui-widget-content ui-corner-all" id="travel">
    <ul class="ui-tabs-nav ui-helper-reset ui-helper-clearfix ui-widget-header
ui-corner-all">
        <li class="ui-state-default ui-corner-top ui-tabs-selected ui-state-active">
<a href="#travel-flight" id="tab-flight">Book a Flight</a></li>
        <li class="ui-state-default ui-corner-top"><a href="#travel-car"
id="tab-car">Rent a Car</a></li>
        <li class="ui-state-default ui-corner-top"><a href="#travel-package"
id="tab-package">Package Deals</a></li>
    </ul>
    <div id="travel-flight" class="ui-helper-clearfix ui-tabs-panel
ui-widget-content ui-corner-bottom"></div><!-- /flight -->
    <div id="travel-car" class="ui-tabs-panel ui-widget-content ui-corner-bottom
ui-tabs-hide"></div><!-- /car -->
    <div id="travel-package" class="ui-tabs-panel ui-widget-content ui-corner-bottom
ui-tabs-hide"></div><!-- /package -->
</div><!-- /travel -->
```

这些类设置控制窗口组件布局的样式，并根据窗口组件的设计使其正常工作。在本例中，这些样式将一个无序的链接列表和 元素转换为具有相关内容面板的选项卡。（本章前面的15.0.1节详细地讨论了窗口组件相关类。）

这些类还确定了窗口组件的各个组成部分（如标题或者内容面板），因此很适合编写覆盖规则调整布局特征或者添加绘制图标等自定义效果。选项卡的窗口组件类标记了如下组件：

ui-tabs

包装选项卡导航和内容的外部容器。

ui-tabs-nav

导航选项的容器。选项卡列表项目和链接用后代选择器（即，ui-tabs-nav li）设置样式。

ui-tabs-selected

选中选项卡“开启”状态，由脚本动态设置。一次只有一个选项卡指定该类。

ui-tabs-panel

映射到选项卡的内容区域。

ui-tabs-hide

内容面板的默认状态。在用户选择显示之前一直隐藏。

要查看与这些类关联的样式规则，可以打开主题样式表并寻找（Ctrl/Command+F快捷键）或者滚动到以ui-tabs开始的代码块。注意，这些规则只适用于布局特性，如位置、内边距或者边框宽度，而不设置任何主题样式（如背景或者边框颜色）：

```
.ui-tabs { padding: .2em; zoom: 1; }
.ui-tabs .ui-tabs-nav { list-style: none; position: relative;
padding: .2em .2em 0; }
.ui-tabs .ui-tabs-nav li { position: relative; float: left;
border-bottom-width: 0 !important; margin: 0 .2em -1px 0; padding: 0; }
.ui-tabs .ui-tabs-nav li a { float: left; text-decoration: none;
padding: .5em 1em; }
.ui-tabs .ui-tabs-nav li.ui-tabs-selected { padding-bottom: 1px;
border-bottom-width: 0; }
.ui-tabs .ui-tabs-nav li.ui-tabs-selected a, .ui-tabs .ui-tabs-nav
li.ui-state-disabled a, .ui-tabs .ui-tabs-nav li.ui-state-processing
a { cursor: text; }
.ui-tabs .ui-tabs-nav li a, .ui-tabs.ui-tabs-collapsible .ui-tabs-nav
li.ui-tabs-selected a { cursor: pointer; } /* first selector in group seems
obsolete, but required to overcome bug in Opera applying cursor: text overall
```

```
if defined elsewhere... */
.ui-tabs .ui-tabs-panel { padding: 1em 1.4em; display: block; border-width: 0;
background: none; }
.ui-tabs .ui-tabs-hide { display: none !important; }
```

> **注意**
>
> 如果你也下载选项卡插件，主题样式表将包含ui-tabs样式规则。

第2步：创建覆盖样式表

我们已经找到了安全地微调窗口组件外观的最佳方法，就是编写新的样式规则覆盖jQuery UI主题样式并将这些“覆盖规则”附加到单独的样式表中。覆盖规则按照jQuery UI CSS类名编写，必须按照主题样式表中的顺序排列；因为样式表按照顺序读取，所以最后的样式规则总是具有优先权。

jQuery UI程序库不断发展，用更加简练的代码包含更多功能。通过在单独的文件中维护覆盖样式，你可以根据需要或多或少地自定义窗口组件样式，并且仍然保持在必要时升级jQuery UI文件以及在保持覆盖规则不变的情况下重写现有主题样式表的能力。覆盖规则可以在专用样式表中列出以覆盖主题样式，如果你想要限制链接到页面的文件数量（从而限制对服务器的请求数量），也可以将覆盖规则附加到整个项目的主样式表之后。

在本秘诀的工作中，将把覆盖样式附加到项目的主样式表travel.css中为应用程序开发的自定义样式块之后：

```
/* ----- 旅游应用程序的自定义样式*/
body { font-size: 62.5%; }
fieldset { padding: 0 0 1.5em; margin: 0 0 1.5em; border: 0; }
p, label { padding: 0 0 .5em; margin: 0; line-height: 1.3; }
p label { display: block; }
...
/* ----- jQuery UI窗口组件的覆盖规则*/
/*这里是选项卡的背景样式 */
...
```

在页面的主题规则之后引用travel.css：

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>travel application | Book a Flight, Rent a Car, or Find Package
Deals</title>

    <!-- jQuery UI styles -->
    <link rel="stylesheet" type="text/css" href="css/custom-theme/jquery-ui-
1.7.1.custom.css" />

    <!-- overrides & custom styles for the travel application -->
    <link rel="stylesheet" type="text/css" href="css/travel.css" />
....
```

第3步：编辑覆盖样式表中的样式规则

我们已经检查了窗口组件类的命名和应用方式，以及在项目中引用覆盖样式的方法，现在用自定义的选项卡导航栏和日期选择器标题样式更新旅游预订应用程序。首先处理选项卡。

限定覆盖范围。我们为选项卡所创建的设计专用于旅游预订应用程序，而且不一定希望在整个应用程序中的每个选项卡窗口组件中应用相同的自定义（如图标或者字体大小）。为了确保这些样式仅适用于旅游应用程序，将把覆盖规则限制为旅游应用程序的唯一ID。

每条新规则都从应用到要修改的组件上的窗口组件相关类开始；例如，当为选项卡的导航栏修改样式时，将编写用于.ui-tabs-nav类的规则：

```
.ui-tabs-nav { /* our override style rule */ }
```

然后用预定的ID travel将规则限制在旅游应用程序的范围中：

```
#travel .ui-tabs-nav { /* our override style rule */ }
```

编写覆盖规则。应用主题样式表之后，选项卡面板如图15-12所示：单独的选项卡很小，周围有一个边框，该边框和最外层容器之间有几个像素的内边距。

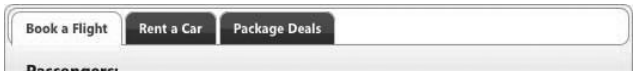


图15-12　在覆盖之前应用了ThemeRoller主题的选项卡

但是，该设计（见图15-13）需要带有图标而没有背景的大选项卡——它们好像出现在选项卡内容之上。

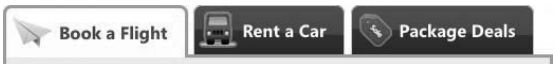


图15-13　目标选项卡设计

为了覆盖默认的选项卡样式，下面进行一些样式修改：

1．首先，删除最外层的边框。整个选项卡组件四周包围着一个像素宽的边框，并有几个像素的内边距。为了让选项卡出现在内容面板之上，将两者删除：

```
#travel.ui-tabs { padding: 0; border-width: 0; }
```

注意

在我们的范围ID #travel和.ui-tabs类之间故意不留空格，因为两者适用于标记中的同一个元素：

```
<div id="travel" class="ui-tabs ui-widget ui-widget-content ui-corner-all">
```

2．接下来，把选项卡导航栏的底部扁平化（设置底部圆角半径为0）并删除顶部和两侧的边框。还将删除所有额外的内边距，使选项卡和组件的左侧等高，还将把边框的宽度加厚到3个像素，以匹配该设计：

```
#travel .ui-tabs-nav {
    border-width: 3px;
    border-top-width: 0;
    border-left-width: 0;
    border-right-width: 0;
    -moz-border-radius-bottomleft: 0;
    -webkit-border-bottom-left-radius: 0;
    -moz-border-radius-bottomright: 0;
    -webkit-border-bottom-right-radius: 0;
    padding: 0;
}
```

3．选项卡有些过于靠近，所以添加右侧的外边距：

```
#travel .ui-tabs-nav li {
margin-right: .5em;
}
```

4．更新选中的选项卡.ui-tabs-selected，使其与选项卡内容区域有所联系。把边框宽度增加到3个像素以匹配设计，然后校正选项卡和内容之间的缝隙。选项卡和内容面板之间的距离与选项卡导航栏边框的厚度相关，所以可以应用3个像素的底部外边距来弥补这一缝隙：

```
#travel .ui-tabs-nav li.ui-tabs-selected {
    border-width: 3px;
    margin-bottom: -3px;
}
```

5．接下来，将应用自定义图标。因为每个图标对选项卡来说都是唯一的，所以可以用每个选项卡的唯一ID将每个图标作为背景图片。（从技术上说，这不是覆盖样式，但是在设置选中选项卡图标样式时需要引用这些规则。）

```
#tab-flight {
    background: url(../images/icon-tab-flight.png) no-repeat .3em center;
        padding-left: 50px;
}
#tab-car {
        background: url(../images/icon-tab-car.png) no-repeat .1em center;
        padding-left: 45px;
}
#tab-package {
        background: url(../images/icon-tab-package.png) no-repeat .1em center;
        padding-left: 45px;
}
```

6．选中的选项卡使用略有不同的图标，背景为白色而不是灰色。为每个选项卡添加一条规则，删除选中状态的窗口组件相关类ui-tabs-selected：

```
#travel .ui-tabs-nav li.ui-tabs-selected #tab-flight {
          background-image: url(../images/icon-tab-flight-on.png);
}
#travel .ui-tabs-nav li.ui-tabs-selected #tab-car {
          background-image: url(../images/icon-tab-car-on.png);
}
#travel .ui-tabs-nav li.ui-tabs-selected #tab-package {
          background-image: url(../images/icon-tab-package-on.png);
}
```

7．选项卡还应该有更多的内边距和较大的字体尺寸：

```
#travel .ui-tabs-nav a {
      font-size: 1.5em;
      padding-top: .7em;
      padding-bottom: .7em;
}
```

8．为完成选项卡，调整内容面板的边框，使其匹配在选中的选项卡上设置的3像素边框：

```
#travel .ui-tabs-panel {
      border-width: 3px;
      border-top-width: 0;
      padding-top: 1.5em;
}
```

选项卡已经匹配设计，接下来更新日期选择器的标题。如图15-14所示，利用一些调整，可以使日期选择器的标题组件（日历上包含导航箭头和月/年反馈信息的条块）出现在上方而不是日期选择器内部。

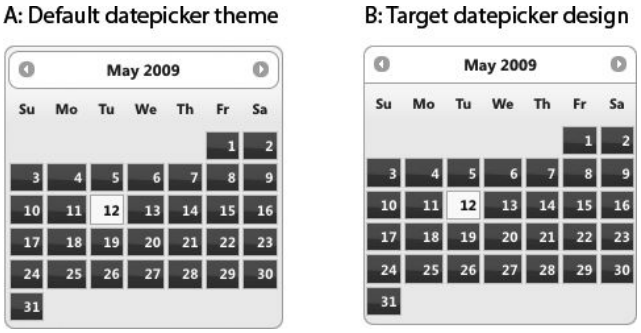


图15-14　应用了ThemeRoller主题的日期选择器（A）和目标设计（B）

和选项卡一样，当初始化页面上的日期选择器插件时，脚本将窗口组件标记写到包含jQuery UI窗口组件相关类和框架类的页面，设置其结构化和主题外观。在日期选择器标记的节选版本中，你可以看到窗口组件相关类遵循命名约定，以ui-datepicker开始，标识各个组件：

```
<div id="ui-datepicker-div" class="ui-datepicker ui-widget ui-widget-content
ui-helper-clearfix ui-corner-all ui-helper-hidden-accessible">
    <div class="ui-datepicker-header ui-widget-header ui-helper-clearfix
ui-corner-all">
    <a class="ui-datepicker-prev ui-corner-all">title="Prev"><span class="ui-icon
ui-icon-circle-triangle-w">Prev</span></a>
    <a class="ui-datepicker-next ui-corner-all" title="Next"><span class="ui-icon
ui-icon-circle-triangle-e">Next</span></a>
    <div class="ui-datepicker-title">
       <span class="ui-datepicker-month">January</span><span class="ui-datepickeryear">
2009</span>
     </div>
    </div>
    <table class="ui-datepicker-calendar">
      <thead>
      <tr>
        <th class="ui-datepicker-week-end"><span title="Sunday">Su</span></th>
        ...
      </tr>
      </thead>
      <tbody><tr>
        <td class="ui-datepicker-week-end ui-datepicker-other-month"> 1 </td>
        ...
    </tr>
    </tbody>
   </table>
   <div class="ui-datepicker-buttonpane ui-widget-content">
    <button type="button" class="ui-datepicker-current ui-state-default
ui-priority-secondary ui-corner-all">Today</button>
    <button type="button" class="ui-datepicker-close ui-state-default
ui-priority-primary ui-corner-all">Done</button>
   </div>
</div>
```

为日期选择器窗口组件类指定如下默认样式规则：

```
.ui-datepicker { width: 17em; padding: .2em .2em 0; }
.ui-datepicker .ui-datepicker-header { position:relative; padding:.2em 0; }
.ui-datepicker .ui-datepicker-prev, .ui-datepicker .ui-datepicker-next {
position:absolute; top: 2px; width: 1.8em; height: 1.8em; }
.ui-datepicker .ui-datepicker-prev-hover, .ui-datepicker .ui-datepicker-next-hover {
top: 1px; }
.ui-datepicker .ui-datepicker-prev { left:2px; }
.ui-datepicker .ui-datepicker-next { right:2px; }
.ui-datepicker .ui-datepicker-prev-hover { left:1px; }
.ui-datepicker .ui-datepicker-next-hover { right:1px; }
.ui-datepicker .ui-datepicker-prev span, .ui-datepicker .ui-datepicker-next span {
display: block; position: absolute; left: 50%; margin-left: -8px; top: 50%;
margin-top: -8px; }
```

```
.ui-datepicker .ui-datepicker-title { margin: 0 2.3em; line-height: 1.8em;
text-align: center; }
.ui-datepicker .ui-datepicker-title select { float:left; font-size:1em;
margin:1px 0; }
.ui-datepicker select.ui-datepicker-month-year {width: 100%;}
.ui-datepicker select.ui-datepicker-month,
.ui-datepicker select.ui-datepicker-year { width: 49%;}
.ui-datepicker .ui-datepicker-title select.ui-datepicker-year { float: right; }
.ui-datepicker table {width: 100%; font-size: .9em; border-collapse: collapse;
margin:0 0 .4em; }
.ui-datepicker th { padding: .7em .3em; text-align: center; font-weight: bold;
border: 0; }
.ui-datepicker td { border: 0; padding: 1px; }
.ui-datepicker td span, .ui-datepicker td a { display: block; padding: .2em;
text-align: right; text-decoration: none; }
.ui-datepicker .ui-datepicker-buttonpane { background-image: none; margin: .7em
0 0 0; padding:0 .2em; border-left: 0; border-right: 0; border-bottom: 0; }
.ui-datepicker .ui-datepicker-buttonpane button { float: right; margin: .5em .2em
.4em; cursor: pointer; padding: .2em .6em .3em .6em; width:auto; overflow:visible; }
.ui-datepicker .ui-datepicker-buttonpane button.ui-datepicker-current { float:left; }
...
```

这只是日期选择器样式规则的一个子集；要查看所有样式，可以在主题样式表中寻找以ui-datepicker开始的样式代码块。

回到旅游应用程序，编写几条覆盖规则，使日期选择器的标题类似该设计：

1．首先删除将标题与日期选择器外部容器分开的内边距：

```
#travel .ui-datepicker { padding: 0; }
```

2．和选项卡导航栏一样，应该将底部弄平并删除顶部和侧面的边框：

```
#travel .ui-datepicker-header {
    border-top-width: 0;
    border-left-width: 0;
    border-right-width: 0;
    -moz-border-radius-bottomleft: 0;
    -webkit-border-bottom-left-radius: 0;
    -moz-border-radius-bottomright: 0;
    -webkit-border-bottom-right-radius: 0;
}
```

3．最后，删除悬停状态下“上一个”和“下一个”导航箭头中的边框和背景图片：

```
#travel .ui-datepicker-prev-hover,
#travel .ui-datepicker-next-hover {
    border-width: 0;
    background-image: none;
}
```

应用覆盖样式之后，旅游应用现在和最终设计已经完全匹配了（见图15-15）。

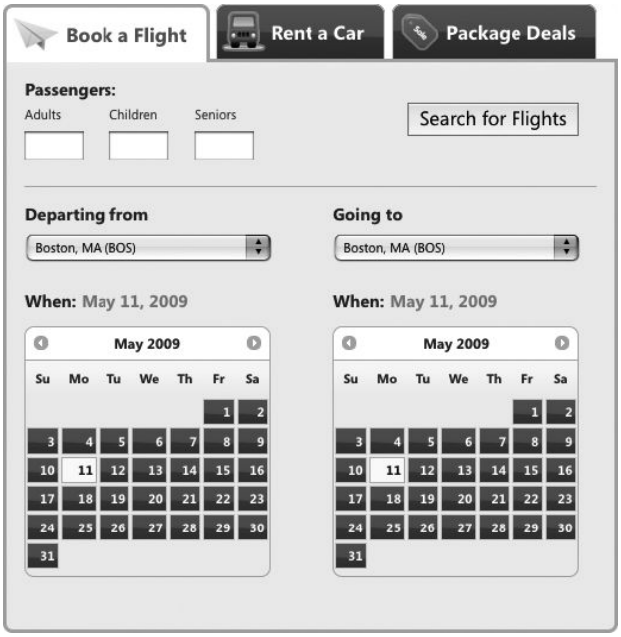


图15-15　最终设计，应用了标准的ThemeRoller和覆盖的样式

15.2.3　讨论

考虑一下，你是想为项目中的所有窗口组件应用覆盖规则，还是只想为一部分窗口组件覆盖主题样式。如果以不同方式显示组件的可能性很小，可以将覆盖样式的范围限定为容器元素的类或者ID，这样就不会修改窗口组件的默认格式。

下面是对编辑的一些提示：

- 如果你希望删除窗口组件标题上的底部边框，可以用`border-bottomwidth:0;`代替`border-bottom:0;`。前者将保留边框样式和颜色，可以恢复它们。
- 对于具有相同类的堆叠元素，可以只禁用其中一个中的背景图片，让背景颜色的差异显露出来。
- 如果需要修改窗口组件某一部分的颜色，设计主题以容纳变化，而不是在样式表中硬编码颜色。
- 如果需要删除一个边框，但是想要保留它用于结构化布局，可以将其设置为透明的。对IE来说，安全的方式是将边框样式设置为`dashed`。
- 只要有可能，就在内边距和外边距等结构化尺寸上使用em单位，更重要的是要将这个单位用在字体尺寸上。编写样式将标准窗口组件尺寸设置为1em，不要将尺寸减小到0.8em以下，以便保持文本的清晰度。

15.3 为非jQuery UI组件应用主题

15.3.1 问题

其他页面组件（如内容框、按钮和工具栏）在jQuery UI窗口组件旁边，有着类似的交互和行为，但是它们的设计不匹配。

15.3.2 解决方案

可以为非jQuery UI元素指定框架类，应用与ThemeRoller样式元素相同的主题。（额外的好处是，这些元素将在应用已更新的ThemeRoller主题时自动更新。）

> **注意**
>
> 后面的秘诀有如下的假设：
>
> - 你有CSS工作原理的基本知识，具体地说是样式层叠、优先级，以及用选择器类、ID或者元素限制样式表的方法。（建议的资源请参见15.5节。）
> - 你已经熟悉了用ThemeRoller创建和编辑主题的方法（如果还不熟悉，复习秘诀15.1，该秘诀详细描述了创建和应用主题的方法）。

在前两个秘诀中，使用ThemeRoller创建和下载一个主题，然后编写几条CSS规则覆盖默认主题样式，并使其更接近于最终的设计。现在，进一步采取另一个步骤，将框架类应用到项目中的元素，使它们与jQuery UI窗口组件以及创建的主题协调一致。

第1步：检查可用的框架类，确认可以应用到组件的类

框架类是用ThemeRoller创建主题时下载的jQuery UI主题样式表的一部分。它们根据用途命名并应用背景颜色和纹理、边框和字体颜色、圆角和图标等主题样式。框架类内置于jQuery UI窗口组件，但是它们也适用于任何其他元素——例如，开发或者从第三方扩展的自定义窗口组件——可以在整个网站或者应用程序范围内实现一致的观感。

下面将概述组成框架的类、每个类应用的样式和在你自己的代码中引用它们的一般原则。

> **注意**
>
> 除非另作说明，框架类设置的所有样式（包括任何文本、链接和图标样式）都由子元素继承。

布局助手类隐藏内容或者修复常见的结构问题，如将浮动子元素完全包装在一个容器中：

`.ui-helper-hidden`

应用`display:none`。这样隐藏的内容无法被屏幕阅读器访问。

`.ui-helper-hidden-accessible`

将元素定位在页面之外使其不可见，但是仍然可以被屏幕阅读器访问。

`.ui-helper-reset`

删除继承的内边距、外边距、边框、文本装饰和`liststyle`；将`line-height`设置为1.3，`font-size`设置为100%。

`.ui-helper-clearfix`

强制非浮动容器元素完全包装浮动子元素。

窗口组件容器类只应该应用到作为容器命名原因的元素上，因为它们的子链接将从它们那里继承样式：

.ui-widget

在整个窗口组件上应用主题的字体系列和字体尺寸，并明确地为子表单元素设置相同的字体系列和1em的字体大小以强制继承。

.ui-widget-header

应用加粗字体。

.ui-widget-content

应用边框颜色、背景颜色和图片以及文本颜色。

交互状态设置可单击元素（如按钮、折叠标题和选项卡）的样式，在用户与之交互时提供相应的状态反馈；每个类都应用边框颜色、背景颜色和图片以及文本颜色。-hover、-focus和-active类的用途是替换与之等价的CSS伪类（:hover、:active、:focus），必须用客户端脚本指派给一个元素。状态类这样设计是为了避免样式冲突，以及由于CSS内置伪类所增加的选择器复杂性（如果伪类对项目是必需的，可以将它们添加到秘诀15.2中所描述的覆盖样式表中）。

- .ui-state-default
- .ui-state-hover
- .ui-state-focus
- .ui-state-active

交互提示设置内容的样式，以高亮或者错误消息、禁用的表单元素或者视觉层次结构的形式表达反馈。所有的类都应用边框颜色、背景颜色和图片以及文本颜色：

.ui-state-highlight

指定该类暂时高亮显示一个组件。

.ui-state-error

将该类指派给包含错误消息的任何组件。

.ui-state-error-text

只应用“错误”文本和图标颜色而没有背景。

.ui-state-disabled

用低的不透明度使表单元素看上去像禁用的样子，从而可以和其他用于设置该元素样式的类协同工作。该元素在应用该类时仍然可用；要禁用元素的功能，可以使用表单元素属性disabled。

.ui-priority-primary

当按钮的操作优先于另一个按钮时（即，Save按钮优先于Cancel按钮）指定该类。应用粗体文本。

.ui-priority-secondary

当按钮的操作是另一个按钮（例如，Cancel按钮）的辅助功能时指定该类。应用普通粗细的字体和较低的不透明度。

图标类以方向箭头和信息性符号（例如，X或者垃圾桶表示删除按钮）的形式提供附加反馈。用两个类为元素应用图标：

```
.ui-icon
```

基类，将元素尺寸设置为16像素的正方形，隐藏所有文本并设置ThemeRoller生成的图标精灵图作为背景。

```
.ui-icon-[type]
```

这里的“type”是将要显示的图标图形的一个描述符，这个类型是单个词（ ui-icon-document、ui-icon-print）或者几个单词、数字和简写的组合；例如，.ui-icon-carat-1-n将显示一个指向北方的插入符号，而.ui-icon-arrow-2-e-w将显示一个指向东方-西方的双箭头图标。

注意

因为ui-icon基类影响元素的尺寸并且隐藏所有内部文本，所以为它们自己的元素（如标记）指定图标是一个好的做法，以便样式对任何子内容或者元素都不会有不利的影响。为了可访问性，在图标的标记中包含简短的描述，它从用户的角度来说是隐藏的，但是仍然可用于屏幕阅读器。

而且，具有.ui-icon类的每个元素都根据双亲容器的状态指定一幅背景精灵图像。例如，在一个.ui-state-default容器内的图标元素将以ThemeRoller中设置的ui-state-default icon图标颜色显示图标。

jQuery UI提供一整套完整的框架图标（见图15-16）。在ThemeRoller中可以将光标悬停于窗口组件示例栏中的一个图标，预览它们的默认和悬停交互状态，还可以将光标悬停于一个图标上，查看它的类名。

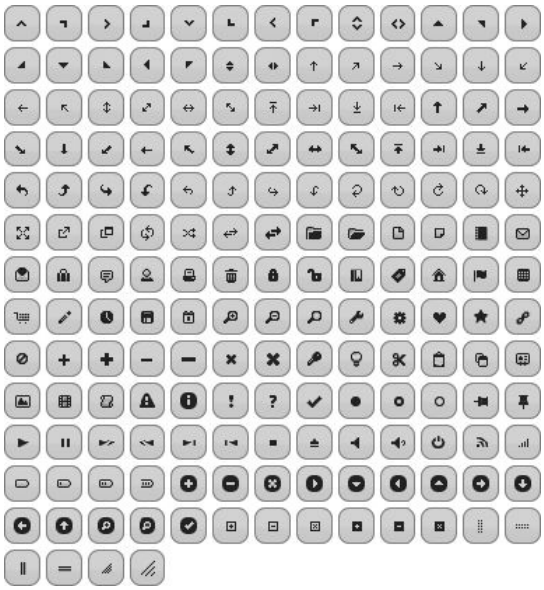


图15-16　jQuery UI在单幅精灵图像中包含整组主题图标；它们的交互状态可以在ThemeRoller中预览

圆角助手类为容器的所有圆角的一个子集应用圆角。圆角类名称的最后一段表示圆角应用的位置，如：

```
.ui-corner-tl
```

左上

```
.ui-corner-tr
```

右上

.ui-corner-bl

左下

.ui-corner-br

右下

.ui-corner-top

左上和右上

.ui-corner-bottom

左下和右下

.ui-corner-right

右上和右下

.ui-corner-left

左上和左下

.ui-corner-all

所有4个角

覆盖和阴影类可以用于为网站或者应用程序增加深度和尺寸：

.ui-widget-overlay

为模态屏幕应用100%宽度和高度的尺寸、背景和不透明度，模态屏幕是处于模态对话框和页面内容之间的一层，常用于使页面内容在模态窗口显示时暂时禁用。

.ui-widget-shadow

应用背景、圆角、不透明度顶部/左侧偏移量以定位窗口组件背后的阴影，以及阴影厚度（类似于边框宽度）。

因为这些框架类为jQuery UI窗口组件应用主题样式且可以用于设置页面上任何组件的样式，所以可以在一个界面中全面使用它们，创建统一的外观。在本秘诀中，将研究如何指定三种框架类：

- 可单击元素状态类，包括.ui-state-default、.ui-state-hover和.ui-state-active
- 圆角类.ui-corner-all
- 禁用表单元素的交互提示类.ui-state-disabled

第2步：应用可单击元素状态框架类

让我们来继续调整旅游预订应用的外观。

应用ThemeRoller中创建的一个主题并通过覆盖规则修改默认样式之后，旅游应用的航班选择器界面几乎已经完成：jQuery UI组件中的可单击元素有一致的外观——默认情况下，选项卡和日期选择器按钮都为暗灰色，具有玻璃的纹理。

但是Search for Flights提交按钮没有遵循这种设计，而是类似于标准的无样式浏览器按钮（见图15-17）。我们希望它更像精致的主题样式。

图15-17　除了Search for Flights按钮没有设置样式之外，界面接近完成

为了使搜索按钮和旅游应用中的其他可单击元素类似，为之指定设置可单击元素状态样式的框架类（.ui-statedefault、.ui-state-hover和.ui-state-active），然后编写一段简短的jQuery脚本，在用户与按钮交互时应用样式。还将用与选项卡和日期选择器组件相同的半径应用圆角。

首先，为按钮设置默认状态类，使其与其他可单击元素相匹配。将简单地（或者从主题样式表中复制）将ui-defaultstate写入按钮的class属性：

```
<button id="search-flights" class="ui-state-default">Search for Flights</button>
```

因为可单击元素（如选项卡）具有圆角，所以为按钮的四个角都添加圆角，在class属性中附加ui-corner-all类：

```
<button id="search-flights" class="ui-state-default ui-corner-all">Search for
Flights</button>
```

在标记中添加这些属性之后，已经为搜索按钮应用了可单击元素的默认主题样式，而且使其“可主题化”，以后，如果我们决定为旅游应用程序窗口组件创建和应用新的主题，搜索按钮将采用新样式表中的默认可单击元素和圆角样式。

最后，应用悬停和mousedown（活动）状态，为用户提供与按钮交互时的视觉反馈（见图15-18）。

图15-18　三个框架类用于指定可单击元素状态

为了更新悬停和mousedown状态下的按钮外观，编写一个小的jQuery脚本。因为已经下载并在页面中包含了最新版本的jQuery核心程序库，而且在DOM就绪时初始化了窗口组件插件，所以将为DOM就绪代码块附加一个函数，切换为搜索按钮指定的状态类。正如下面的脚本块所指出的，hover事件包含

两个函数，第一个函数在mouseover事件发生的时候删除默认状态类并添加悬停状态，第二个函数则在mouseout事件发生时反转指定的类，mousedown事件则用活动类代替默认和悬停状态类：

```
$(function(){
     // 初始化选项卡和日期选择器
          $('#travel').tabs();
          $('#departure-date-picker').datepicker({altField: '#departure-date', altFormat:
'MM d, yy'});
          $('#arrival-date-picker').datepicker({altField: '#arrival-date', altFormat: 'MM
d, yy'});

     //搜索按钮悬停和活动状态
          $('#search-flights')
               .hover(
                    function(){ $(this).removeClass('ui-state-default').addClass ('ui-state- hover');},
                    function(){ $(this).removeClass('ui-state-hover').addClass ('ui-state- default');}
               )
               .mousedown(
                    function(){ $(this).removeClass('ui-state-default, ui-stat-e
hover').addClass('ui-state-active'); }
               );
});
```

注意

为什么在CSS伪类（:hover、:active、:focus）能够起到相同作用的时候还要编写脚本更新按钮状态？我们在设计jQuery UI CSS时权衡了这个问题，出于几个关键原因，我们决定不使用伪类：

- 伪类给样式表带来了一定的复杂性，使其几乎无法保持简洁，包含伪类要求我们考虑这些状态可能发生冲突的所有情况。
- 伪类加剧了CSS的膨胀，可能显著地增加样式表的尺寸。
- 因为有些浏览器（如老旧但仍然流行的Internet Explorer版本）只支持链接元素上的伪类，所以必须为所有可单击状态创建类。

最终，按钮外观类似图15-19。

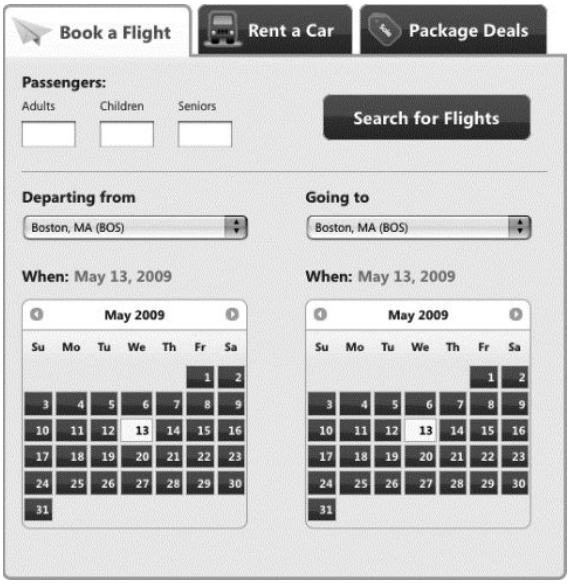


图15-19　最终设计，搜索按钮应用了主题类

现在，按钮已经设置了匹配应用程序的样式，可以有条件地添加一个交互提示类ui-state-disabled，提供当禁用按钮时的视觉反馈（见图15-20）。例如，假定航班预定表单的所有字段都必须提交。在这种情况下，搜索按钮应该在用户为每个字段输入有效值之前保持禁用；当表单完成时，启用该按钮提交表单。

图15-20　添加ui-state-disabled状态，使表单元素看上去是禁用的

要为搜索按钮应用禁用的外观，为默认按钮附加框架类ui-state-disabled。（两个类对于最终外观都是必要的，因为禁用状态样式只是降低默认按钮的不透明度。）

```
<button id="search-flights" class="ui-state-default ui-state-disabled ui-corner-
all">Search for Flights</button>
```

应用禁用状态类只改变按钮的外观而不影响其功能：它仍然能接受用户输入以提交表单。为了确保该按钮真的禁用，一定要为按钮标记添加disabled属性和值：

```
<button id="search-flights" class="ui-state-default ui-state-disabled ui-corner-all"
disabled="disabled">Search for Flights</button>
```

15.3.3　讨论

框架类在整个应用程序中重用，它们本身就为开发人员提供了一组用于设置应用程序中的相关组件（如旅游应用的Search for Flights按钮，甚至你自己的窗口组件）的完备类。因为框架类按照用途命名，所以将它们应用到自定义窗口组件的各个组成部分相当直观：

- 可单击元素状态类可以添加到需要悬停或者活动状态的按钮、链接或者其他元素中。
- 圆角类可以应用到具有块属性的任何元素。
- 布局助手类可以用在整个布局结构中，修复浮动容器或者切换内容可见性。
- 交互提示类可以赋予必须表达视觉优先级或者错误消息的元素。

为非jQuery UI元素添加框架类还使它们成为可主题化的元素；如果你决定用ThemeRoller编辑和下载更新后的主题，新主题也会自动为这些元素应用样式。

15.4 在一个页面上引用多个主题

15.4.1 问题

在你的应用程序中必须应用多个主题，这些主题显示在单个页面上。例如，你的jQuery UI选项卡必须按照首要主题设置样式，而选项卡面板中的组件遵循不同的主题。

15.4.2 解决方案

用ThemeRoller创建第二个主题，并在下载过程中将新的主题与某个类、某个ID或者其他范围选择器关联，选择性地应用到窗口组件或者应用程序中的组件上。

> **注意**
>
> 后面的秘诀有如下的假设：
>
> - 你有CSS工作原理的基本知识，具体地说是样式层叠、优先级，以及用选择器类、ID或者元素限制样式表的方法。（建议的资源请参见本章最后的附录。）
> - 你已经熟悉了用ThemeRoller创建和编辑主题的方法（如果还不熟悉，复习秘诀15.1，该秘诀详细描述了创建和应用主题的方法）。

jQuery UI主题的意图是在jQuery UI窗口组件和整个应用程序的其他界面组件中形成一致的观感，但是有时候设计更为复杂，必须根据组件在应用程序中出现的位置，为某些组件应用不同的观感。

在旅游应用示例中，假设设计人员审核最后的设计，觉得在所有可单击元素上使用暗灰色使得预订类型选项卡和其中的表单字段难以区分。他决定顶部的选项卡应该保持当前的样式，但是选项卡中的所有交互性组件（包括日期选择器和搜索按钮）应该有不同的样式，默认状态为黄色。图15-21展示了当前的设计和新的设计。

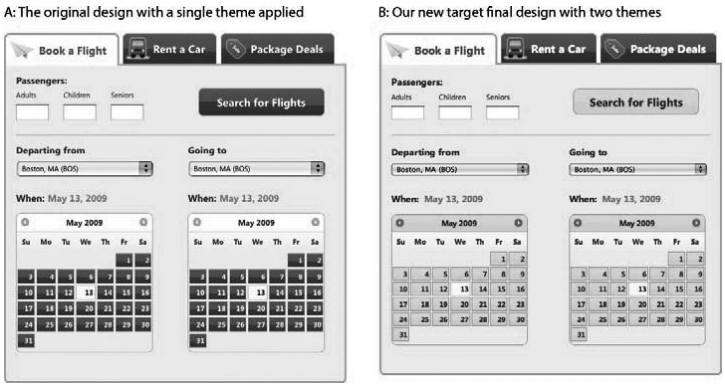


图15-21 原始主题（A）将所有交互式元素的可单击默认状态设置为灰色；新设计保持了顶部选型卡的颜色，但是将选项卡中的所有交互式组件改为黄色

为选项卡内容创建样式例外有几种方法。正如秘诀15.2描述的，可以编写和引用覆盖规则，为日期选择器和按钮修改默认主题样式。为此，必须使用设计编辑工具（如Adobe Photoshop）计算出所有新的颜色十六进制值，然后生成新的黄色背景图片。

也可以在ThemeRoller中创建匹配次要主题（在本例中是黄色的可单击元素）的新主题，明确限制其作用范围为选项卡内容范围，然后在原始主题样式表之后应用它。jQuery UI下载构建器提供简单的界面来限制主题范围：下载页面上的Advanced Theme Settings（高级主体设置）区域可以设置为指定一个范围选择器——类、ID或者其他层次结构的CSS选择器——允许你准确地指出附加主题设置哪些组件的样式。

回到旅游预订应用，这时我们已经完成了秘诀15.1～15.3中描述的步骤：

- 创建和下载一个主题，并在项目中引用（见秘诀15.1）。
- 编写附加的覆盖规则修改几个主题默认样式（见秘诀15.2）。
- 为搜索按钮添加几个框架类，应用主题样式（见秘诀15.3）。

现在，我们将研究如何限制第二个主题的范围，并将其应用到项目中。

第1步：用ThemeRoller创建另一个主题

打开jQuery UI网站（*http://jqueryui.com*）从顶部的导航栏选择Themes，或者直接访问*http://themeroller.com*。

创建原始主题设置设计中使用的所有窗口组件。但是，在本例中只设置选项卡内容面板内的组件；现在，可以不理会顶部的导航选项卡。

正如在秘诀15.1中所做的那样，从Sunny主题开始，因为它在默认情况下几乎匹配新设计中的黄色可单击状态和标题样式。

> **注意**
>
> 你可以使用现有的自定义主题作为起点，没有必要从头开始。为此，打开主题样式表，搜索以“To view and modify this theme, visit http://jqueryui.com/themeroller/...”开始的组件，并将主题URL复制和粘贴到浏览器的地址栏，打开ThemeRoller并预加载主题设置。

Sunny主题非常接近于新的目标设计，只有两个例外：日期选项卡上的标题是灰色的，而该设计是黄色，内容区域和活动状态边框颜色比设计中指定的褐色更深。我们将返回到Roll Your Own选项卡，调整几个设置：

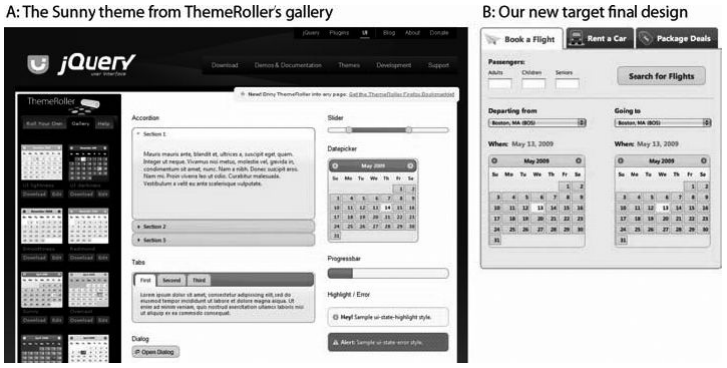


图15-22　新目标设计有黄色的可单击状态和选项卡内容标题，接近于Sunny画廊主题

- **将标题背景从灰色改为黄色：**附加主题的背景颜色和边框必须匹配“可单击元素默认状态”。

1．打开Header/Toolbar部分。

2．在背景颜色字段输入#FECE2F；不需要对纹理或者不透明度设置作任何修改。

3．因为白色文本目前在黄色背景上不容易分辨，所以将它的颜色变深，以匹配应用程序其他地方的灰色文本；输入颜色值#333333。

4．同样，因为日期选择器标题中的图标需要与背景有更大的对比，所以将它设置为中等的褐色，输入#A27406。

5．最后，将边框颜色改为#D19405。

- **将内容和活动状态边框改为浅褐色：**内容边框出现在折叠组件周围，定义选项卡、对话框、滑块、日期选择器和进度条外部容器。

  1．打开Content部分。

  2．更新边框颜色匹配标题边框，#D19405。

  3．按下Tab或者Enter键，或者单击页面其他区域，在右边预览窗口组件的变化。

第2步：限制新主题范围并下载

当你结束Sunny主题的编辑，单击Roll Your Own选项卡工具栏中的“Download theme”按钮，导航到jQuery UI下载构建器。

在编辑下载构建器设置之前，需要确定使用哪一个范围选择器将新的主题应用到旅游应用程序的内容面板。应该确保只影响选项卡内容，而不会改变应用到顶部导航选项卡的原始主题。

范围选择器是类、ID或者具体标识我们想要设置样式的元素双亲容器的和HTML标记。最好选择范围最为有限的范围选择器，这样你就不会因为疏忽而将样式应用到应该使用基本主题样式的元素。在旅游预订应用中，范围选择器应该标识包围选项卡内容而不包围选项卡导航面板的容器。

当我们查看在应用程序中生成的标记时，可以看到每个内容面板都指定了ui-tabs-panel类：

```
<div class="ui-tabs ui-widget ui-widget-content ui-corner-all" id="travel">
    <ul class="ui-tabs-nav ui-helper-reset ui-helper-clearfix ui-widget-header uicorner-
all">
        <li class="ui-state-default ui-corner-top ui-tabs-selected ui-state-active">
<a href="#travel-flight" id="tab-flight">Book a Flight</a></li>
        <li class="ui-state-default ui-corner-top"><a href="#travel-car" id="tabcar">
Rent a Car</a></li>
        <li class="ui-state-default ui-corner-top"><a href="#travel-package"
id="tab-package">Package Deals</a></li>
    </ul>
    <div id="travel-flight" class="ui-helper-clearfix ui-tabs-panel
ui-widget-content ui-corner-bottom"></div><!-- /flight -->
    <div id="travel-car" class="ui-tabs-panel ui-widget-content ui-corner-bottom
ui-tabs-hide"></div><!-- /car -->
    <div id="travel-package" class="ui-tabs-panel ui-widget-content
ui-corner-bottom ui-tabs-hide"></div><!-- /package -->
</div><!-- /travel -->
```

因为内容面板标记单独出现在选项卡导航之后，我们可以安全地将新的样式应用范围设置为ui-tabs-panel类，不会影响顶部选项卡应用的样式。

标识了范围选择器，就可以返回到jQuery UI下载构建器。在Theme下的右侧栏目中，指定新主题在应用程序中的范围。单击Advanced Theme Settings展开这一部分，你将看到两个输入字段（见图15-23）。

图15-23　jQuery UI下载构建器的高级主题设置展开，提供CSS范围和新主题文件夹字段

- CSS Scope（CSS范围）接受范围选择器（类、ID或者HTML标记）。在编译主题样式表时，下载构建器用该值作为每条样式规则的前缀，仅将样式标题应用到指定容器中的元素。

对于旅游预订应用程序，输入限制范围的类.ui-tabs-panel。一定要包含前导的句点（.），或者，如果指定一个ID，要包含hash（#）——这些标记对于样式表正常显示是必要的。

> **注意**
>
> 当这个字段留空时，主题应用到程序中的所有组件，没有限制在任何特定容器范围内。

- Theme Folder Name（主题文件夹名称）接受一个新主题包含在下载的ZIP中的文件夹名称；这个文件夹包含主题样式表和图片文件。该值默认为选中主题的名称，在例子中应该是“custom-theme”，因为在到达下载构建器之前用ThemeRoller设计了一个自定义主题。

当在第一个字段中输入CSS范围时，下载构建器根据范围建议一个文件夹名称。这是有帮助的，但是你可能想用对项目目录结构更有意义的名称覆盖建议名称。

当对于旅游预订应用，编写自己的文件夹名称，用“tab-content-theme”更好地描述文件夹内容。

既然已经设置了CSS范围和文件夹名称，就选择使用新主题的所有jQuery UI窗口组件。

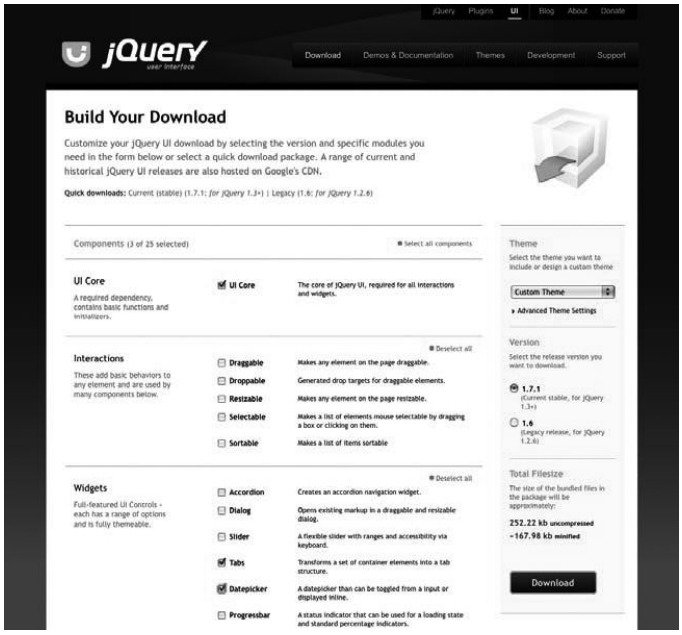


图15-24　在jQuery UI Download页面中填写高级主题设置并选择所有使用限制主题的窗口组件，下载该主题

获得新组件

你需要下载使用受限主题的窗口组件，以便在受限CSS中包含相应的样式。这些窗口组件所用的JavaScript自动包含在下载内容中。但是，如果你只需要限制范围的主题，可以放弃JavaScript，因为它可能是你已经拥有的代码的复制品。

如果你在这一步意识到需要下载项目所用的不包含在原始下载内容中的组件（例如，你需要添加一个进度条），我们强烈建议你不要和受限范围主题一起下载新的窗口组件。JS文件的合并太过复杂。

作为替代，我们建议重新完成整个下载过程，在项目中添加新的组件：在ThemeRoller中重新打开原始主题，然后下载项目中使用的所有jQuery UI组件。这样，你就可以简单地覆盖原始主题样式表，用包含应用程序中所有窗口组件的文件替代jQuery UI JavaScript文件。要做到这一点，只要打开原始主题样式表，搜索以“To view and modify this theme, visit http://jqueryui.com/themeroller/...”开头的注释，将主题URL复制和粘贴到浏览器的地址栏，打开ThemeRoller并预先加载主题设置，然后单击“Download theme”选择附加的组件。

选择你想要使用的jQuery UI版本（默认选择最新的稳定版本），单击Download按钮，并在本地存储ZIP文件（该文件的名称类似*jquery-ui-1.7.1.custom.zip*）。

第3步：将文件合并到项目目录

下载文件夹包含CSS目录，该目录中有你的受限范围主题文件夹；组件JavaScript（js），可能是你已经使用（为了安全，在覆盖任何文件之前都要再次核对）的代码的复制品；开发包（development-bundle）包含用于创建*css*文件夹中汇编版本的单独*css*文件，开放源码许可证文本和高级开发必需的相关资源。文件夹和文件的内容根据操作系统而有所不同（图15-25显示了在Mac OS X中打开的文件夹）。

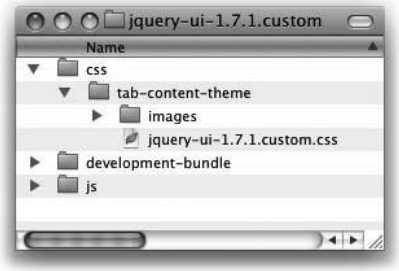


图15-25　当下载受限范围主题时，jQuery下载文件夹结构的截图

现在，复制和粘贴*tab-content-theme*文件夹到旅游预订项目的样式目录中。

警告

维护主题文件夹中已经确立的文件结构很重要，这样主题类能够正常引用图标图片。如果你修改了主题目录结构，很可能在你决定升级到新版本的jQuery UI脚本和CSS时要重复进行这些修改。

新的主题文件夹将和样式目录中的原始主题文件夹放在一起，如图15-26所示。

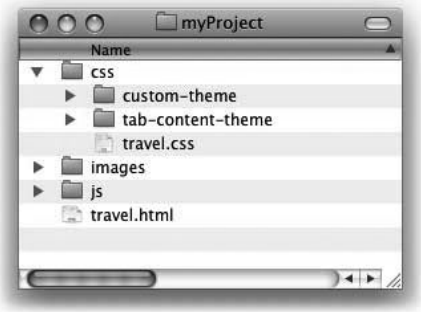


图15-26　把受限范围主题文件夹附加到样式目录中

第4步：在你的项目中引用受限范围的主题样式

我们将在原始主题样式表之后，所有jQuery UI脚本之前引用受限范围的样式表。在页面上引用主题样式表的顺序不重要：

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Travel widget | Book a Flight, Rent a Car, or Find Package Deals</title>

    <!-- jQuery UI styles -->
    <link rel="stylesheet" type="text/css" href="css/custom-theme/jquery-ui-
1.7.1.custom.css" />
     <link rel="stylesheet" type="text/css" href="css/tab-content-theme/jquery-ui-
1.7.1.custom.css" />

    <!-- jQuery core & UI scripts -->
    <script type="text/javascript" src="js/jquery-1.3.2.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui-1.7.1.custom.min.js"></script>

    <script type="text/javascript">
    $(function(){
        $('#travel').tabs();
        $("#departure-date-picker").datepicker({altField: '#departure-date',
altFormat: 'MM d, yy'});
        $("#arrival-date-picker").datepicker({altField: '#arrival-date', altFormat:
'MM d, yy'});
```

```
    });
</script>
</head>
...
```

当主题样式表链接就绪，我们在浏览器中预览页面，确认正常应用样式。因为我们将新主题的范围限制在选项卡内容面板，所以样式仅应用到内容的窗口组件，而不是上面的选项卡，如图15-27所示。

图15-27　应用受限范围主题的最终应用程序

注意

这种技术的另一个例子可以参见这篇文章：http://www.filamentgroup.com/lab/using_multiple_jquery_ui_themes_on_a_single_page/。

15.5　附录：其他CSS资源

为了最大限度地利用jQuery UI CSS Framework和ThemeRoller，具备CSS工作原理的基本知识是很有用的，具体地说，就是样式层叠、优先权和使用选择器类、ID或者元素限定范围的方法。

建议将如下书籍和在线资源作为这些概念的入门：

CSS Basics Tutorial

http://www.cssbasics.com/

CSS Cheat Sheet

http://lesliefranke.com/files/reference/csscheatsheet.html

Designing with Web Standards

http://www.zeldman.com/dwws/

Web Standards Solutions

http://www.friendsofed.com/book.html?isbn=1590593812

Eric Meyer on CSS

http://www.ericmeyeroncss.com/

第16章　jQuery、Ajax、数据格式：HTML、XML、JSON、JSONP

Jonathan Sharp

16.0 导言

Web开发人员用一些数据格式和协议在浏览器和服务器之间传送信息。本章提供处理和使用最常用数据格式、Ajax技术和jQuery的秘诀。

16.1 jQuery和Ajax

16.1.1 问题

你打算在不离开访问者当前所在页面的情况下，从服务器请求一些附加数据。

16.1.2 解决方案

下面是简单的Ajax请求：

```
(function($) {
    $(document).ready(function() {
        $('#update').click(function() {
            $.ajax({
                type: 'GET',
                url: 'hello-ajax.html',
                dataType: 'html',
                success: function(html, textStatus) {
                    $('body').append(html);
                },
                error: function(xhr, textStatus, errorThrown) {
                    alert('An error occurred! ' + ( errorThrown ? errorThrown:
xhr.status );
                }
            });
        });
    });
})(jQuery);
```

16.1.3 讨论

jQuery的Ajax架构核心是jQuery.ajax()方法。这个方法为所有浏览器到服务器的请求和响应提供了基础。所以，我们要更详细地研究这个方法。为了初始化到服务器的一个请求，把包含请求参数的一个设置对象传递给$.ajax方法。可用选项很多，最常用的请求选项是type、url、complete、dataType、error和success：

```
var options = {
    type: 'GET'
};
```

当启动Ajax请求时首先要考虑的选项是服务器HTTP请求类型，大部分情况下是GET或者POST类型：

```
var options = {
    type: 'GET',
    url: 'hello-ajax.html',
```

```
    dataType: 'html'
};
```

接下来我们要注意URL和dataType选项。URL的含义不言自明，值得一提的是后面的交互。当将cache选项设置为false时，jQuery将附加一个get变量_=<random number>（例如，/server-ajax-gateway?_=6273551235126），该变量用于避免浏览器、代理和服务器发送缓存的响应。最后，dataType选项指定服务器响应的期望数据格式。例如，如果你希望服务器返回HTML，则选项值html是合适的：

```
var options = {
    type: 'GET',
    url: 'hello-ajax.html',
    dataType: 'html',
    error: function(xhr, textStatus, errorThrown) {
        alert('An error occurred! ' + errorThrown);
    },
    success: function(data, textStatus) {
        $('body').append( data );
    }
};
```

我们要定义的下两个选项是两个回调方法，一个称为error，另一个称为success。它们的标题恰当地表示了功能，error在请求中有错误时调用，success在成功响应的时候调用（由服务器返回的响应类型是否为200确定）。complete是另一个常用的选项，定义了在响应成功或者错误之后执行的一个回调方法：

```
var options = {
    type: 'GET',
    url: 'hello-ajax.html',
    dataType: 'html',
    complete: function(xhr, textStatus) {
        //处理响应的代码
    }
};
```

一旦设置定义完毕，就可以继续执行请求：

```
var options = {
    type: 'GET',
    url: 'hello-ajax.html',
    dataType: 'html',
    complete: function(xhr, textStatus) {
        //处理响应的代码
}
};
$.ajax( options );
```

还可以用内联方式设置选项：

```
$.ajax({
    type: 'GET',
    url: 'hello-ajax.html',
    dataType: 'html',
    complete: function(xhr, textStatus) {
        //处理响应的代码
    }
});
```

最终解决方案请求文件hello-ajax.html并在成功请求返回时将内容（html）附加到<body>元素中。如果请求失败，则触发error方法，用一条消息警告用户：

```
(function($) {
    $(document).ready(function() {
        $('#update').click(function() {
            $.ajax({
                type: 'GET',
                url: 'hello-ajax.html',
                dataType: 'html',
                success: function(html, textStatus) {
                    $('body').append(html);
                },
                error: function(xhr, textStatus, errorThrown) {
                    alert('An error occurred! ' + errorThrown);
                }
            });
        });
    });
})(jQuery);
```

16.2 在整个网站上使用Ajax

16.2.1 问题

你有一个大的Web应用程序，在整个代码库中都发生Ajax调用，需要为整个应用程序的所有请求定义默认设置。

16.2.2 解决方案

```
(function($) {
    $(document).ready(function() {
        $('#loadingIndicator')
            .bind('ajaxStart', function() {
                $(this).show();
            })
            .bind('ajaxComplete', function() {
                $(this).hide();
            });
        $.ajaxSetup({
            cache: true,
            dataType: 'json',
            error: function(xhr, status, error) {
                alert('An error occurred: ' + error);
            },
            timeout: 60000, // Timeout of 60 seconds
            type: 'POST',
            url: 'ajax-gateway.php'
        }); //结束 $.ajaxSetup()
    }); // 结束.read()
})(jQuery);
```

16.2.3 讨论

在开发较大的应用程序时，往往通过一个公用的Ajax网关传递所有请求。使用$.ajaxSetup()方法，我们可以设置Ajax请求默认设置。这会使整个应用程序中的Ajax请求变得简单：

```
$.ajax({
    data: {
        //我从服务器请求的数据
    },
    success: function(data) {
        //现在更新用户界面
    }
});
```

简单提一句，因为超时选项的值以毫秒表示（秒数×1000），所以超时值6000代表6秒。设置该值时要考虑的一件事是全球互联网的扩展。在你的访问者或者用户

中，有些人所在位置的网络延迟可能超过你所在地区用户的期望值。所以，不要将该值设置得太低（例如，5秒）。指定较高的值（如30或者60秒）可以让具有高延迟连接（例如，使用卫星信道）的用户仍然能够享用你的应用程序。

在前一个例子中，对ajax-gateway.php进行POST请求。发生的错误将由$.ajaxSetup()中定义的错误函数处理。对于特定的请求，仍然可以按照如下方式覆盖设置：

```
$.ajax({
    url: 'another-url.php',
        data: {
            // 服务器请求数据
        },
        success: function(data) {
            //现在更新用户界面
    }
});
```

把前一个请求发送给another-url.php而不是ajax-gateway.php。

jQuery Ajax架构的好处之一是ajaxComplete、ajaxError、ajaxSend、ajaxStart、ajaxStop和ajaxSuccess等全局事件。这些事件可以用*.bind('event',* callback)方法或者快捷方法*.event*(callback)设置。下面的例子展示了为ajaxError事件绑定回调的两种方法：

```
(function($) {
    $(document).ready(function() {
        $('#loadingIndicator')
            .ajaxError(function() {
                //你的代码
            });
        //或者使用bind()
        $('#loadingIndicator')
            .bind('ajaxError', function() {
                //你的代码
            });
    });
})(jQuery);
```

下面是可用事件的简要描述，以及触发的顺序：

ajaxStart

如果没有其他请求正在进行，则在Ajax请求开始时触发。

ajaxSend

在每个单独请求发送之前触发。

ajaxSuccess或ajaxError

在请求成功或者不成功时触发。

ajaxComplete

每当请求完成（不管成功还是出错）时触发。

ajaxStop

如果没有其他Ajax请求正在进行中则触发。

在下一个秘诀中，更详细地介绍这些事件。

16.3 使用带有用户反馈的简单Ajax

16.3.1 问题

你需要在Ajax请求进行中为用户显示一个状态指示器，并在完成时隐藏它。

16.3.2 解决方案

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

        //在单击doAjaxButton按钮时启动Ajax请求
        $('#doAjaxButton').click(function() {
            $.ajax({
                url: 'ajax-gateway.php',
                    data: { val: "Hello world" },
                    dataType: 'json',
                success: function(json) {
                    //数据处理代码
                    $('body').append( 'Response Value: ' + json.val );
                }
            });
        });
    });
})(jQuery);
```

16.3.3 讨论

jQuery Ajax实现的一个巨大好处是输出全局Ajax事件，可以由每个Ajax请求在所有元素上触发。在下面的解决方案中，用快捷方法将其中两个事件ajaxStart和ajaxStop绑定到ID为ajaxStatus的XHTML元素上。当Ajax请求在单击#doAjaxButton时触发，也调度ajaxStart事件，并调用#ajaxStatus元素上的show()。注意，这些事件是自动触发的，是使用$.ajax()(或者$.get()等快捷方法）的副作用。这为在应用程序范围内的请求（如提交Ajax请求）提供了一个简洁的无耦合解决方案：

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

```
            .ajaxStop(function() {
                $(this).hide();
            });

        //在单击doAjaxButton时启动Ajax请求
        $('#doAjaxButton').click(function() {
            $.ajax({
                url: 'ajax-gateway.php',
            data: { val : 'Hello world' },
            dataType: 'json',
                success: function(json) {
                    //数据处理代码
                    $('body').append( 'Response value: ' + json.val );
                }
            });
        });
    });
})(jQuery);
```

我们来看看其他的一些事件和局部与全局Ajax事件之间的差异。局部Ajax事件（用$.ajaxSetup()建立或者在调用$.ajax()时定义)包括beforeSend、success、error和complete。这些事件内联定义，与每个Ajax请求紧密耦合。全局Ajax事件与本地事件交错，但是由绑定它们的元素触发，而且使用jQuery的原生事件处理架构。下面是处理局部Ajax事件（如complete事件）的简单回顾：

```
$.ajax({
    type: 'GET',
    url: 'ajax-gateway.php',
    dataType: 'html',
    complete: function(xhr, textStatus) {
        //处理响应的代码
    }
});
```

现在我们来看看成功的Ajax请求后触发事件的分类、顺序和作用域：

- ajaxStart（全局）
- beforeSend（局部）
- ajaxSend（全局）
- success(局部）
- ajaxSuccess(全局）
- complete（局部）
- ajaxComplete(全局）
- ajaxStop（全局）

对于不成功的Ajax请求，触发事件的顺序如下，success和ajaxSucess分别被error和ajaxError代替：

- ajaxStart(全局)
- beforeSend(局部)
- ajaxSend(全局）

- error(局部)
- ajaxError(局部)
- complete(局部)
- ajaxComplete(全局)
- ajaxStop(全局)

ajaxStart和ajaxStop是全局作用域里的两个特殊事件。它们的不同在于处理多个并发请求时的表现。ajaxStart在请求发出且没有其他请求正在进行中时触发。ajaxStop在一个请求完成且没有其他请求正在进行中时触发。这两个事件在发出多个并发请求的时候只触发一次：

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

        //在单击doAjaxButton时启动Ajax请求
        $('#doAjaxButton').click(function() {
            $.ajax({
                url: 'ajax-gateway.php',
                complete: function() {
                    //数据处理代码
                }
            });
            $.ajax({
                url: 'ajax-data.php',
                complete: function() {
                    //数据处理代码
                }
            });
        });
    });
})(jQuery);
```

global设置可以传递给$.ajax()方法，该设置的值可以是true或者false。通过将global设置为false，可以禁止全局事件触发。

注意

如果你在应用程序中遇到性能问题，可能是大量元素事件传播的代价所致。在这种情况下，将global设置为false可改进性能。

beforeSend回调是一个局部事件，允许在请求发送之前修改XMLHttpRequest对象（作为参数传递）。在下面的示例中，为请求指定一个自定义HTTP标头。从回调返回false可以取消请求：

```
(function($) {
    $(document).ready(function() {
        //当单击doAjaxButton时启动Ajax请求
        $('#doAjaxButton').click(function() {
            $.ajax({
                url: 'ajax-gateway.php',
                  beforeSend: function(xmlHttpRequest) {
                      xmlHttpRequest.setRequestHeader('X-SampleHeader',
'Hello world');
                  },
                complete: function() {
                    //数据处理代码
                }
            });
        });
    });
})(jQuery);
```

现在如果我们修改解决方案来考虑所有这些事件，就会得到如下代码：

```
(function($) {
    $(document).ready(function() {
        $('#ajaxError')
            .ajaxError(function(evt, xhr, ajaxOptions, error) {
               $(this)
                 .html( 'Error: ' + ( xhr ? xhr.status : '' )
                              + ' ' + ( error ? error :'Unknown' ) )
                 .show();
            })
            .ajaxSuccess(function() {
                $(this).hide();
            });
        $('#ajaxStatus')
            .ajaxStart(function() {
                $(this).show();
            })
            .ajaxSend(function() {
                $(this).html('Sending request...');
            })
             .ajaxStop(function() {
                $(this).html('Request completed...');
                var t = this;
                setTimeout(function() {
                    $(t).hide();
                }, 1500);
            });

        //在单击doAjaxButton时启动Ajax请求
        $('#doAjaxButton').click(function() {
            $.ajax({
                url: 'ajax-gateway.php',
                complete: function() {
                    //数据处理代码
                }
            });
        });
    });
})(jQuery);
```

16.4 使用Ajax快捷方法和数据类型

16.4.1 问题

你需要向服务器发出一个GET Ajax请求，并将结果HTML内容放在ID为contents的一个 中。

16.4.2 解决方案

```
(function($) {
$(document).ready(function() {
        $('#contents').load('hello-world.html');
});
})(jQuery);
```

16.4.3 讨论

本秘诀和其他秘诀略有不同，我们将纵览jQuery提供的多个函数和快捷方法，清晰地表现它们之间的差异。

jQuery提供了一些发出Ajax请求的快捷方法。根据前面介绍Ajax的秘诀，存在如下的快捷方法：.load()、$.get()、$.getJSON()、$.getScript()和$.post()，但是我们首先回顾自己的解决方案：

```
$('#contents').load('hello-world.html');
```

.load(url)方法对hello-world.html发出一个GET Ajax请求，并在#contents中放置结果的内容。.load()方法有两个可选的参数data和callback。data参数可以是一个映射对象（或者JavaScript对象）或者一个字符串（自jQuery 1.3起）。下面的例子传入值为world的hello变量。（这和如下的URL一样：helloworld.html?hello=world）

```
$('#contents').load('hello-world.html', { hello: 'world' });
```

第三个可选参数是一个回调函数，在请求完成（success或者error)时调用。在下面的例子中，请求完成时触发一条警告消息：

```
$('#contents').load('hello-world.html', { hello: 'world' }, function() {
    alert('Request completed!');
});
```

接下来要研究的两个方法是$.get()和$.post()。这两个方法都接受相同的参数，$.get()方法发送一个GET HTTP请求，而$.post()方法发送一个POST HTTP

请求。我们将研究一个使用$.get()请求的示例。$.get()方法接受的参数有url、data、callback和type。前三个参数的作用和前面的load()方法一样，所以只介绍最后一个参数type：

```
$.get(
    'hello-world.html',
    { hello: 'world' },
    function(data) {
        alert('Request completed!');
    },
   'html'
);
```

type参数可以接受如下值：xml、html、script、json、jsonp或text。这些type值确定来自Ajax请求的响应文本在传递到回调函数之前如何处理。在前一个例子中，因为指定type参数html，所以回调函数的数据参数将为DOM对象的形式。使用xml作为type参数值将传递xml DOM对象。如果type参数指定为script，服务器返回的结果数据将在触发callback方法之前执行。json和jsonp格式都会将一个JavaScript对象传递给callback方法，jsonp的不同之处在于jQuery将在请求中传递一个方法名称，将回调方法映射到请求定义的匿名函数。这样做允许跨域请求。最后，text格式正如其名：以字符串形式将普通文本传递给callback方法。

现在，我们来看看最后两个快捷方法：$.getJSON()和$.getScript()。$.getJSON()方法接受url、data和callback参数。$.getJSON()实际上是$.get()方法和为JSON或者JSONP设置的相关参数的组合体。下面的例子将向Filckr发起一个JSONP请求，从公共时间线（public timeline）请求照片：

```
$.getJSON(
    'http://www.flickr.com/services/feeds/photos_public.gne?
format=json&jsoncallback=?',
    function(json) {
    }
);
```

因为这个请求是跨域的，jQuery自动将请求当作JSONP处理，并填写对应的callback函数名。这也意味着，jQuery将在文档中插入一个<script>标记初始化请求，代替XMLHttpRequest对象。Flickr的API允许设置jsoncallback get变量以指定callback函数名。你将会注意到URL的jsoncallback=?部分。jQuery将智能地用对应的函数名自动替换?。默认情况下，jQuery将附加一个callback=变量，但是允许像例子中展示的那样对其进行修改。callback替换在GET和POST请求URL上都有效，但是对于数据对象中传入的参数无效。使用JSON的方法参见秘诀16.7和秘诀16.8，Flickr示例的完整JSONP实现参见秘诀16.9。

$.getScript()通过Ajax请求或者动态插入用于跨域支持的<script>执行请求，然后评估返回的数据并最终触发所提供的回调函数。在下面的例子中，将在文档中添加一个脚本，然后调用回调中提供的一个函数：

```
// hello-world.js
function helloWorld(msg) {
    alert('Hello world! I have a message for you: ' + msg);
}

// hello-world.html
(function($) {
    $(function() {
        $.getScript('hello-world.js', function() {
            helloWorld('It is a beautiful day!');
        });
    });
})(jQuery);
```

16.5 使用HTML片段和jQuery

16.5.1 问题

你打算取得一个HTML字符串，并将其转换为一系列DOM节点，然后插入文档。

16.5.2 解决方案

```
(function($) {
    $(document).ready(function() {
        $('<div>Hello World</div>')
            .append('<a href="http://jquery.com">A Link</a>')
            .appendTo('body');
    });
})(jQuery);
```

16.5.3 讨论

操纵HTML字符串是使用jQuery的常见任务。在jQuery的核心中有一个用于将标记字符串转换为DOM表现的简洁接口。可以简单地传递一个HTML字符串来代替选择器。（下面的代码不适用于XML；将XML转换为DOM的方法参见秘诀16.6）

```
$('<div>Hello World</div>');
```

这时，HTML已经转换为DOM表现形式，可以用jQuery进行操纵。可以用任何jQuery方法操作这个片段：

```
$('<div>Hello World</div>')
    .append('<a href="http://jquery.com">A Link</a>')
    .appendTo('body');
```

警告

值得注意的是，在HTML片段附加到文档之前，有些视觉属性（如width和height）可能无法使用。所以在下面的例子中，调用.width()将返回0。

```
$('<div>Hello World</div>').width();
//返回'0'
```

16.6 将XML转换为DOM

16.6.1 问题

你需要将XML字符串转换为DOM对象供jQuery使用。

16.6.2 解决方案

```
<h1 id="title"></h1>

(function($) {
    $(document).ready(function() {
        var xml = '<myxml><title>Hello world!</title></myxml>';
        var title = $.xmlDOM( xml ).find('myxml > title').text();
        $('#title').html( title );
    });
})(jQuery);
```

16.6.3 讨论

jQuery邮件列表上常见的一个问题是如何将XML字符串转换为jQuery可以操作的DOM表现形式。当用响应类型xml发起Ajax请求时，浏览器将自动把返回的XML文本解析为DOM对象。

那么，如果你有一个需要用jQuery处理的XML字符串时应该怎么做？xmlDOM插件提供了XML字符串的原生跨浏览器解析，返回一个jQuery包装的DOM对象。可以在一步内转换和访问XML：

```
(function($) {
    $(document).ready(function() {
        var xml = '<myxml><title>Hello world!</title></myxml>';
        var title = $.xmlDOM( xml ).find('myxml > title').text();
        $('#title').html( title );
    });
})(jQuery);
```

另一种常见的方法是将DOM对象作为jQuery（上下文）的第二个参数，如：

```
(function($) {
    $(document).ready(function() {
        var $xml = $.xmlDOM( '<myxml><title>Hello world!</title></myxml>' );
        var title = $('myxml > title', $xml).text();
        $('#title').html( title );
    });
})(jQuery);
```

这样，就可以对传入的上下文对象运行jQuery选择；否则，jQuery对文档对象运行查询。

作者编写的xmlDOM插件可以从*http://jquery-cookbook.com/go/plugin-xmldom*下载。

16.7 创建JSON

16.7.1 问题

你有一个JavaScript对象，包含需要进行序列化以便于存储和读取的数据。

16.7.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var messageObject = { title: 'Hello World!', body: 'It\'s great to be
alive!' };
        var serializedJSON = JSON.stringify( messageObject );
    });
})(jQuery);
```

16.7.3 讨论

JavaScript对象标记法（JavaScript Object Notation，JSON）是用于在浏览器和服务器之间交换数据的常见数据格式。它具有轻量级的特性，很容易用JavaScript使用和解析。首先看一个简单的对象：

```
var messageObject = { title: 'Hello World!', body: 'It\'s great to be alive!' };
```

在这个例子中，有一个简单的对象，该对象有两个属性`title`和`body`。存储序列化版本的对象相当简单。序列化版本如下：

```
var serializedJSON = '{"title":"Hello World!","body":"It\'s great to be alive!"}';
```

使用JSON的两个常见任务是序列化（将对象编码为字符串形式）和反序列化（将字符串形式解码为对象）。目前，只有一些浏览器具有内置的JSON处理功能（如Firefox 3.1以上版本和Internet Explorer 8）。其他浏览器计划添加支持，因为JSON现在已经是ECMA 3.1规范的一部分。同时，使用JSON数据有两种主要的方法。Douglas Crockford编写了JSON编码和解码的JavaScript实现，可以从*http://jquery-cookbook.com/go/json*下载。利用JSON程序库序列化前面的对象：

```
var serializedJSON = JSON.stringify( messageObject );
```

现在有一个字符串表现形式，可以从Ajax请求把它发送到服务器或者在表单中提交它。

16.8 解析JSON

16.8.1 问题

你得到一串JSON数据，需要将其转换为对象格式。

16.8.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var serializedJSON = '{"title":"Hello World!","body":"It\'s great to be
alive!"}';
        var message = JSON.parse( serializedJSON );
    });
})(jQuery);
```

16.8.3 讨论

正如前一个秘诀中讨论的，我们现在将关注JSON字符串的解析或者解码。

> **警告**
>
> 重要的是，这里概述的一些方法是不安全的，可能导致潜在的安全问题。一定要确保数据源是可信任的。

利用JSON数据的最简方法是对消息执行`eval()`方法。但是，这种方法有一些固有的安全问题，因为`eval()`是围绕整个JavaScript规范而不只是JSON子集。这意味着，恶意人士可以执行JSON字符串中嵌入的代码。所以，不建议使用这种方法。作为替代，使用前一个秘诀提到的Douglas Crockford的JSON程序库。（注意，他的程序库也利用`eval()`，但是对数据进行预处理以确保数据安全。）

```
var serializedJSON = '{"title":"Hello World!","body":"It\'s great to be alive!"}';
var message = JSON.parse( serializedJSON );
```

这样，就可以像其他JavaScript对象一样使用消息对象：

```
alert( "New Message!\nTitle: " + message.title + "\nBody: " + message.body);
```

JSON程序库以及其他JSON资源可以从*http://jquery-cookbook.com/go/json*下载。

16.9 使用jQuery和JSONP

16.9.1 问题

你想要利用Filckr公共照片流中的一系列照片，并显示前三张图片。

16.9.2 解决方案

```
(function($) {
    $(document).ready(function() {
        var url  = 'http://www.flickr.com/services/feeds/photos_public.gne?
jsoncallback=?';
        var params = { format: 'json' };
        $.getJSON(url, params, function(json) {
            if ( json.items ) {
                $.each( json.items, function(i, n) {
                    var item = json.items[i];
                    $('<a href="' + item.link + '"></a>')
                        .append('<img src="' + item.media.m + '" />')
                        .appendTo('#photos');
                    // 显示前三张照片(返回false将退出循环)
                    return i < 2;
                });
            }
        });
    });
})(jQuery);
```

16.9.3 讨论

当构建网站或者应用程序时安全是一个关键问题，在Ajax出现之后更是如此。Web浏览器对于请求强制实施同源策略，这意味着请求被限制为与页面URL相同的域或者当前域的子域。例如，*http://www.example.com*上的一个页面允许发出对*http://www.example.com*和*http://x.www.example.com*的Ajax请求，但是不能对*http://example.com*或*http://y.example.com*发出请求。随着语义Web的发展以及Flickr等网站开始为其他用户和服务提供API以便使用，Web浏览器实施的安全策略成为了一个障碍。从不使用同源策略的领域之一是带有src属性的脚本元素。*http://www.example.com*上的网页可以包含一个来自*http://static.example2.com*的脚本，但是动态包含脚本和管理程序流程成为一个问题。因此，JSONP发展为克服同源限制的一个标准。

> **警告**
>
> 重要的是，这里概述的一些方法是不安全的，可能导致潜在的安全问题。一定要确保数据源是可信任的。而且，当在页面中包含脚本元素时，整个脚本将有权访问整个HTML DOM和它所包含的私有或者敏感数据。恶意脚本有可能将这些数据发送给不受信任的一方，要采取将脚本放入沙箱等预防措施。扩展的安全性超出了本秘诀的范围，但是我们希望确保你意识到这个问题。

JSONP通过带有src属性的<script>标记使用请求数据，开发人员可以实现一个回调函数包装数据，管理程序流程。首先看一条简单的JSON消息。

```
{"title":"Hello World!","body":"It's great to be alive!"}
```

下面是回调中包装的相同消息：

```
myCallback({"title":"Hello World!","body":"It's great to be alive!"})
```

当请求的资源加载到浏览器中时，将调用myCallback函数，以JSON对象作为第一个参数。开发者可以实现如下myCallback函数来处理数据：

```
function myCallback(json) {
    alert( json.title );
}
```

现在回顾一下Flickr解决方案。首先定义Flickr Web服务的URL，然后声明一个params对象作为get变量。jsoncallbcak参数是一个由Flickr服务定义的特殊参数，允许传入一个函数名。因为已经将该参数设置为?，所以jQuery将自动生成一个函数名称并将其与回调方法绑定。

注意

jQuery通过URL中的=?检测JSONP（跨域请求）。不能将这个参数传递给params数组。

```
var url = 'http://www.flickr.com/services/feeds/photos_public.gne?
jsoncallback=?';
var params = { format: 'json' };
```

接下来，调用jQuery的$.getJSON()方法，传入url、params和回调函数，该函数将接受一个JSON对象。在回调方法中，检查和确保项的数组存在，然后使用jQuery的$.each()遍历前三个项，创建一个链接，将图片附加到链接上，再将链接附加到ID为photos的一个元素。最后，回调函数将在第三次循环（i=2）时返回false，中断循环。

```
$.getJSON(url, params, function(json) {
    if ( json.items ) {
        $.each( json.items, function(i, n) {
              var item = json.items[i];
            $('<a href="' + item.link + '"></a>')
                .append('<img src="' + item.media.m + '" />')
                .appendTo('#photos');
            return i < 2;
        });
    }
});
```

通过组合JSON数据格式和JSONP的跨域能力，Web开发人员可以创建新的应用程序，以创新的方式聚合和转换数据，发展语义Web。

第17章　在大项目中使用jQuery

Rob Burns

17.0 导言

jQuery常用于为网站添加小的用户界面改进。但是，对于更大和更复杂的Web应用程序，jQuery也相当有用。本章的各个秘诀展示了jQuery如何用于解决更有实质性和交互性的Web内容的需求。前三个秘诀研究Web浏览器中持续存储数据的不同方法。然后研究在应用程序代码和数据量成长时，如何减少Ajax和JavaScript的使用。

17.1 使用客户端存储

17.1.1 问题

你打算编写一个富互联网应用程序，在Web浏览器中处理数量可观的用户数据。出于对缓存这些数据提高性能或者离线使用应用程序的目的，你需要在客户端存储数据。

17.1.2 解决方案

我们将用一个简单的任务列表来说明客户端数据存储。和本章的许多秘诀一样，将使用一个jQuery插件处理浏览器的不一致性：

```
<!DOCTYPE html>
<html><head>
    <title>17.1 - Using Client-Side Storage</title>
    <script type="text/javascript" src="../../jquery-1.3.2.min.js"></script>
    <script type="text/javascript" src="jquery.jstore-all.js"></script>
</head>
<body>
    <h1>17.1 - Using Client-Side Storage</h1>
    <p>Storage engine: <span id="storage-engine"></span></p>
    <input id="task-input"></input>
    <input id="task-add" type="submit" value="Add task"></input>
    <input id="list-clear" type="submit" value="Remove all tasks"></input>
    <ul id="task-list"></ul>
</body></html>
```

HTML由操纵任务列表的表单元素组成：一个输入任务的文本字段，一个添加任务的按钮和一个删除所有任务的按钮。当前任务将用无序列表列出：

```
(function($) {
    $.jStore.ready(function(ev,engine) {
        engine.ready(function(ev,engine) {
            $('#storage-engine').html($.jStore.CurrentEngine.type);
            $('#task-list').append($.store('task-list'));
        });
    });
```

jStore插件提供两个回调方法：jStore.ready()和engine.ready()。这两个方法和jQuery的ready()函数很类似，允许在jStore和当前存储引擎完成内部初始化后进行一些初始设置。这是在页面上显示当前使用的存储引擎和任何保存的任务列表项的一个机会：

```
$('document').ready(function() {
    $('#task-add').click(function() {
        var task = $('#task-input').val();
        var taskHtml = '<li><a href="#">done</a> ' + task + '</li>';
```

```
        $.store('task-list',$('#task-list').append(taskHtml).html());
        return false;
    });
```

一旦文档就绪，就为相关的控件绑定单击事件。当单击“Add task”（添加任务）按钮时，用任务文本字段的内容和标记该任务完成的一个链接构造列表项元素。然后，把该列表项附加到任务列表的内容中，任务列表用tasklist键码存储在本地存储器上。以后，该列表可以用这个键码读取，这一步在engine.ready()回调中完成：

```
$('#list-clear').click(function() {
    $('#task-list').empty();
    $.remove('task-list');
    return false;
});
```

当单击“Remove all tasks”（删除所有任务） 时，包含任务列表的元素将清空。task-list键码及其相关的值从本地存储中删除。

```
        $('#task-list a').live('click',function() {
            $(this).parent().remove();
            var taskList = $('#task-list').html();
            if( taskList ) { $.store('task-list',taskList); }
            else { $.remove('task-list'); }
            return false;
        });
    });
})(jQuery);
```

最后，任务列表中每个项目的done链接绑定一个live事件。通过使用live()函数代替bind()函数或者click()等快捷方式，所有匹配#task-list的元素都绑定到单击事件函数，即使这些元素在live()调用时尚不存在。这样就能够为每个新项插入“done”链接，不需要在每次插入时重新绑定单击事件。

当一个项标记为完成时，它就从列表中删除，更新的列表用task-list键码存储在本地存储中。当保存更新的列表时需要多加小心：

```
if( taskList ) { $.store('task-list',taskList); }
else { $.remove('task-list'); }
```

当列表中的最后一项删除时，taskList变量将为空。这会导致求store()函数的值时就像用单一参数（而不是两个参数）调用的情况。当store()接收单一参数时，读取具有该键码的项目，不修改保存列表。目标是保存一个空列表。else子句中的remove()函数删除task-list键码和相关值。这能满足为空白列表设置保存状态的目标。

17.1.3 讨论

传统上，在客户端保存数据的唯一选择是cookie。cookie所能保存的数据量有限。现在已经有了更好的替代品。下表包含目前可用的存储机制及其浏览器兼容性。

存 储 机 制	Firefox	Safari	Internet Explorer
DOM Storage	2.0+	no	8.0+
Gears	yes	yes	yes
Flash	yes	yes	yes
SQL Storage API	no	3.1+	no
userData behavior	no	no	5.0+

DOM Storage和SQL Storage API是新兴的HTML标准的一部分。它们本身还没有得到跨浏览器支持。Google Gears和Flash是可以用于客户端存储的浏览器插件。Internet Explorer包含用于客户端存储的userData behavior已经有一段时间了。如果需要一个支持所有主流浏览器的机制，基于Flash或者Google Gears的方法能够提供最广泛的支持。但是，这需要用户安装浏览器插件。

警告

jStore插件的1.0.3版本包含一个bug，需要改正一个录入错误。jquery.jstore-all.js的第403行应该如下：

```
return !!(jQuery.hasFlash('8.0.0'));
```

幸运的是，jStore（可以从*http://plugins.jquery.com/project/jStore*下载）提供了一个抽象层，使跨浏览器的客户端存储成为可能，在大部分情况下，它不用依赖浏览器插件。jStore为前面列出的存储机制提供统一的接口。jStore支持人工选择，但是这个例子展示了jStore自动为当前使用的浏览器选择合适存储机制的能力。当在不同浏览器中查看时，这个秘诀显示的是当前选择的存储机制。

17.2 为单个会话保存应用程序状态

17.2.1 问题

你打算在客户端存储数据直到当前会话结束——也就是窗口或者选项卡关闭的时候。

17.2.2 解决方案

在下面的例子中有两个HTML页面。每个页面包含一组可选择元素。当元素选中和反选时，保存页面状态。通过在两个页面之间导航，你可以看到在用户浏览网站时，指定页面的状态是如何维护的。sessionStorage对象用于用户后续访问期间不需要持续存储的数据：

```
<!DOCTYPE html>
<html><head>
    <title>17.2 Saving Application State for a Single Session</title>
    <style>
        .square {
            width: 100px; height: 100px; margin: 15px;
            background-color: gray; border: 3px solid white; }
        .selected {
            border: 3px solid orange; }
        </style>
        <script src="../../jquery-1.3.2.min.js"></script>
</head>
    <body>
    <h1>17.2 Saving Application State for a Single Session</h1>
    <a href="one.html">page one</a>
    <a href="two.html">page two</a>
    <div id="one" class="square"></div>
    <div id="two" class="square"></div>
    <div id="three" class="square"></div>
</body></html>
```

两个HTML页面（one.html和two.html）内容相同。下面的JavaScript代码负责管理每个页面的状态，每个页面反映过去的用户操作：

```
jQuery(document).ready(function() {
    $('.square').each(function(){
        if( sessionStorage[window.location + this.id] == 'true' ) {
            $(this).addClass('selected');
        }
    });

    $('.square').click(function() {
        $(this).toggleClass('selected');
        sessionStorage[window.location + this.id] = $(this).hasClass('selected');
        return false;
```

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

当文档加载时，查询sessionStorage对象，获得组成当前URL的键码以及每个可选择方块的id。每个方块都应用了合适的CSS类。当单击一个方块时，切换CSS类影响其显示，它的状态也相应存储。每个方块的状态用当前URL和当前元素id配对生成的键码保存。

17.2.3 讨论

前一个秘诀中使用的jStore插件有相似的会话限定客户端存储。通过使用jStore，可以得到跨浏览器兼容性。本秘诀只能在Internet Explorer 8.0和Firefox 2.0或更高版本中正常工作。Safari 3.1没有这一特性，但是未来的版本可能包含它。

DOM Storage API在广泛的浏览器兼容性不成为关注点时具有吸引力。为公司内联网开发的应用程序就属于这一类。DOM Storage API是HTML5规范的一部分，未来将会得到广泛采用。使用内置的存储API的好处是不需要附加的JavaScript代码的开销。精简的jStore插件和jStore.swf大小为20KB。

17.3 在会话之间保存应用程序状态

17.3.1 问题

你打算在会话之间，在客户端保存数据。秘诀17.1在会话之间保存任务列表状态。本秘诀阐述如何在不使用jStore插件的情况下启用类似的功能。

17.3.2 解决方案

本解决方案的HTML部分参见秘诀17.1（完全相同）。下面列出JavaScript：

```
(function($) {
    $('document').ready(function() {
        if( window.localStorage ) { appStorage = window.localStorage; }
        else { appStorage = globalStorage[location.hostname]; }

        var listHtml = appStorage['task-list'];
        $('#task-list').append(listHtml.value ? listHtml.value : listHtml);
```

开始的设置比基于jStore的解决方案要更冗长。Firefox有DOM Storage API长期存储部分的一个非标准实现。它使用globalStorage数组而不是localStorage对象在会话之间存储数据。globalStorage数组中的每个存储对象都以提供当前文档的域作为键码。下面的代码在localStorage可用时将使用它，否则回退到globalStorage。

在代码的下一部分中，用任何已有的任务填充无序列表。在基于jStore的例子中，这只需要一行代码。这里附加的复杂性是因为Firefox的特殊行为。localStorage返回一个字符串，但是访问globalStorage时返回一个具有两个属性value和secure的对象。当value属性存在时使用它，否则假定localStorage返回一个字符串：

```
        $('#task-add').click(function() {
            var task = $('#task-input').val();
            var taskHtml = '<li><a href="#">done</a> ' + task + '</li>';
            appStorage['task-list'] = $('#task-list').append(taskHtml).html();
            return false;
        });

        $('#list-clear').click(function() {
            $('#task-list').empty();
            appStorage['task-list'] = '';
            return false;
        });

        $('#task-list a').live('click',function() {
            $(this).parent().remove();
            appStorage['task-list'] = $('#task-list').html();
            return false;
```

```
        });
    });
})(jQuery);
```

剩下的代码添加新的任务，在任务标记为“done”时删除任务，和基于jStore的前一个秘诀一样，为DOM元素附加事件，清除任务列表。但是，这里没有使用基于jStore函数的接口操纵存储数据，而是直接为前面创建的appStorage对象赋值。这样就简化了删除任务的代码。

17.3.3 讨论

DOM Storage API由两个接口组成：sessionStorage和localStorage。

Firefox从2.0版本起包含这个特性，当时标准还在开发中。之后，该标准经过了修改。Internet Explorer 8.0有当前API的一个实现。Safari和Firefox即将推出的版本也将遵循当前的规范。但是，Firefox 2.0～3.0浏览器还将持续一段时间。编写支持globalStorage的应用程序也能为这些遗留的浏览器服务。

17.4 使用JavaScript模板引擎

17.4.1 问题

你打算使用JavaScript模板引擎显示JSON数据。

17.4.2 解决方案

本秘诀介绍的是一个书籍列表。它从一个服务器端脚本获取书籍的相关信息，将其添加到浏览器中显示的书籍列表中。从服务器返回的书籍详情是一个JSON字符串。Pure模板引擎（*http://plugins.jquery.com/project/pure*）用来格式化数据并插入网页中：

```
<!DOCTYPE html>
<html><head>
    <title>jQuery Cookbook - 17.4 Using a Javascript Template Engine</title>
    <style>.hidden { display: none }</style>
    <script type="text/javascript" src="../../jquery-1.3.2.min.js"></script>
    <script type="text/javascript" src="pure.js"></script>
</head>

<body>
    <h1>17.4 - Using a Javascript Template Engine</h1>
    <input type="button" id="add-book" value="Add book"></input>
    <input type="button" id="clear-list" value="Clear booklist"></input>
    <div id="book-list"></div>
```

网页上有两个按钮。当单击其中一个时从服务器读取书籍详情，另一个清除本地显示的书籍列表。书籍列表将在id为book-list的 元素中显示。这些元素在页面加载时可见：

```
    <div id="book-template" class="hidden book">
        <ul class="author-list"><li class="author"><span class="name"></span>
</li></ul>
        <p class="title"></p>
        <p class="year"></p>
        <div class='book-footer'>
            <div class="rating-div">Rating: <span class="rating"></span></div>
            <div>Location: <span class="location"></span></div>
        </div>
    </div>
</body></html>
```

id为book-template的 指定了hidden类。这个 不显示，它将作为从服务器接收数据的模板。Pure模板引擎将数据结构中的属性与具有相同类的HTML元素关联。因此，具有year类的段落元素的内容将反映数据结构中year属性的值：

```
{
    "title": "Democracy and the Post-Totalitarian Experience",
    "author": [
        {
            "name": "Leszek Koczanowicz"
        },
        {
            "name": "Beth J. Singer"
        }
    ],
    "year": "2005",
    "rating": "3",
    "location": "Mandalay"
}
```

上述代码是从服务器返回的JSON数据的一个例子。title、year、rating和location属性有单一值，可以直接映射到HTML模板中的一个元素。为了不只一次地重复这些值，就只能将相应的类赋予模板中的其他元素。

author属性包含一个对象数组。每个对象都有一个属性name。用这种方式表示多个作者，说明模板引擎的迭代能力。该模板包含类为author的一个列表项元素。列表项包含类为name的元素。对于数据结构中值为数组的属性，将为数组的每个元素创建一个关联的HTML元素。这样就能创建任意数量的列表项：

```
(function($) {
    $('document').ready(function() {
        $('#add-book').data('id',1);
```

一旦文档就绪，JavaScript以使用jQuery data()函数存储请求的书籍当前id开始。这个id在每次请求书籍数据时递增。data()函数允许在DOM元素中存储任意数据：

```
$('#add-book').click(function() {
    var curId = $(this).data('id');
    $.getJSON('server.php', {id: +curId}, function(data) {
        if( data.none ) { return false; }
        var divId = 'book-' + curId;
        $('#book-list').append($('#book-template').clone().attr('id',divId));
        $('#'+divId).autoRender(data).removeClass('hidden');
        $('#add-book').data('id', curId + 1);
    });
    return false;
});
```

当单击“Add book”按钮时，用jQuery getJSON()函数向服务器发出请求。模板进程从复制HTML中隐藏的 开始。在附加到书籍列表之前必须改变这个克隆元素的id。如果id不变化，则将引入一个没有独特id的DOM元素。接着以JSON数据作为参数调用来自Pure插件的autoRender()函数，这将使用提供的数据渲染模板。

最后删除hidden类，使书籍的细节可见：

```
        $('#clear-list').click(function() {
            $('#add-book').data('id',1);
            $('#book-list').empty();
            return false;
        });
    });
})(jQuery);
```

清除书籍列表的函数相当简单，清空相关的 元素，将书籍id计数器重置为1。

17.4.3 讨论

使用基于JavaScript的模板引擎有两个好处。一是它们允许将JSON数据结构转换为设置了样式和结构化的HTML，不需要人工操纵数据结构的每个元素。这一好处可以通过将模板引擎应用到常由Ajax调用的小块数据（正如本例所展示的）来了解。

使用JavaScript模板引擎的第二个好处是能够生成纯粹的HTML模板。这些模板不包含用来指出模板化数据和实现迭代之类功能的脚本语言的任何痕迹。正如本秘诀，在浏览器中使用模板引擎时很难利用这一优势。这个特点在网站对搜索引擎的吸引力上有负面的影响，使得大部分人都不愿意采用它。但是，jQuery和Pure模板引擎也可以运行于服务器端JavaScript环境。
Jaxer（*http://www.aptana.com/jaxer*）、
Rhino（*http://www.mozilla.org/rhino/*）和SpiderMonker
（*http://www.mozilla.org/js/spidermonkey/*）都是很著名的例子。

17.5 Ajax请求队列

17.5.1 问题

你希望对不同Ajax请求的顺序有更好的控制。

17.5.2 解决方案

本秘诀阐述两种不同的Ajax请求排队方法。第一种用请求填充队列，在前一个请求返回响应之后发送后续请求。第二种并行发送一组请求。但是，在所有响应返回之前不执行每个请求的回调函数。

```
<!DOCTYPE html>
<html><head>
    <title>jQuery Cookbook - 17.5 - Queuing Ajax Requests</title>
    <script type="text/javascript" src="../../jquery-1.3.2.min.js"></script>
    <script type="text/javascript" src="jquery-ajax-queue_1.0.js"></script>
</head>

<body>
    <h1>17.5 - Queuing Ajax Requests</h1>
    <input type="button" id="unqueued-requests" value="Unqueued requests"></input>
    <input type="button" id="queued-requests" value="Queued requests"></input>
    <input type="button" id="synced-requests" value="Synced requests"></input>
    <p id="response"></p>
</body></html>
```

Ajaxqueue jQuery插件（在*http://plugins.jquery.com/project/ajaxqueue/*上可以下载）用于实现排队行为。三个按钮触发各组Ajax请求，在一个段落元素中显示响应日志：

```
(function($) {
    $('document').ready(function() {
        $('#unqueued-requests').click(function() {
            $('#response').empty();
            $.each([1,2,3,4,5,6,7,8,9,10], function() {
                $.get('server.php',{ data: this }, function(data) {
                    $('#response').append(data);
                });
            });
            return false;
        });
```

第一个按钮触发常规的Ajax请求。发送10个请求，每个请求都包含表示顺序的数字。server.php脚本通过随机时间的休眠之后返回一个响应，模拟具有负载的一台服务器。当响应到达后，把它附加到#response段落的内容中：

```
$('#queued-requests').click(function() {
    $('#response').empty();
    $.each([1,2,3,4,5,6,7,8,9,10], function() {
        $.ajaxQueue({url: 'server.php',
            data: { data: this },
            success: function(data) { $('#response').append(data); }
        });
    });
    $.dequeue( $.ajaxQueue, "ajax" );
    return false;
});
```

“Queued requests”（排队的请求）按钮调用ajaxQueue()函数将每个请求加入队列。在内部，每当请求离开队列时用提供的选项调用ajax()函数。在每个请求添加到队列之后，以ajaxQueue函数作为第一个参数调用dequeue()，触发第一个请求。后续的各个请求将按照顺序发送：

```
        $('#synced-requests').click(function() {
            $('#response').empty();
            $.each([1,2,3,4,5,6,7,8,9,10], function() {
                $.ajaxSync({url: 'server.php',
                    data: { data: this },
                    success: function(data) { $('#response').append(data); }
                });
            });
            return false;
        });
    });
})(jQuery);
```

最后一组请求使用ajaxSync()函数并行发送请求，但在响应返回时同步执行提供的回调。

17.5.3 讨论

未排队的请求将造成无序的返回响应。这种行为并不一定有害，在许多情况下甚至是首选方式。但是，在有些情况下，人们希望更好地控制Ajax请求及其响应。ajaxQueue()提供的功能适合于后续请求依赖于前一个请求的场合，而ajaxSync()支持操纵从不同服务器收集的数据的用例。在这种情况下，在所有服务器返回响应且数据集完整之前，处理无法开始。

17.6 处理Ajax和后退按钮

17.6.1 问题

使用Ajax填充网页创建了方便的交互式用户体验，这是传统HTTP请求所无法复制的。遗憾的是，每当用Ajax更新浏览器窗口的内容时，用浏览器的前进和后退按钮无法访问这些内容。大部分浏览器中的书签功能也不起作用。

17.6.2 解决方案

对这个问题的解决方案是将每个Ajax请求与一个独特的URL关联。这个URL可以加入书签，由浏览器的后退和前进按钮访问。方法之一是使用hash值。hash值通常用于链接到文档中的特定位置。*http://en.wikipedia.org/wiki/Apple#History*链接到维基百科上Apple词条的历史部分。在本秘诀中，hash值将引用Ajax请求加载的内容。

在这个例子中，样例项目是一个小的词汇表，表中有三个条目。当单击条目时，通过Ajax读取单词的定义并显示。当然，这样的内容可以一次性显示在单个页面上。但是，这里使用的方法可以用于更大、更多变的数据，例如，标签式界面中每个选项卡的内容：

```
<!DOCTYPE html>
<html><head>
    <title>17.6 Dealing with Ajax and the Back Button</title>
    <script src="../../jquery-1.3.2.min.js"></script>
    <script src="jquery.history.js"></script>
</head>

<body>
    <h1>17.6 Ajax and the Back Button</h1>
    <a href="#apples" class='word'>apples</a>
    <a href="#oranges" class='word'>oranges</a>
    <a href="#bananas" class='word'>bananas</a>
    <p id='definition'></p>
</body></html>
```

必要的JavaScript文件在文档头部包含。*jquery.history.js*文件包含jQuery历史插件（*http://plugins.jquery.com/project/history*）。词汇表中的三个条目各有一个锚元素。每个条目的定义将在id为definition的段落中显示：

```
(function($) {
    function historyLoad(hash) {
        if(hash) { $('#definition').load('server.php',{word: hash}); }
        else { $('#definition').empty(); }
    }

    $(document).ready(function() {
        $.history.init(historyLoad);
```

```
        $('a.word').click(function() {
            $.history.load($(this).html());
            return false;
        });
    });
})(jQuery);
```

历史插件有两个值得考虑的函数：init()和load()。init()函数在ready函数中调用，参数是处理Ajax请求的一个回调函数。load()绑定到单词链接，参数是每个锚标记的内容。回调方法historyLoad()负责请求传入hash值所指向的内容，还必须能够处理没有hash值的情况。

17.6.3 讨论

historyLoad()回调方法在两个地方调用。首先，当页面加载的时候它在$.history.init()函数内调用。hash值从URL的尾部剥离，作为一个参数。如果不存在hash值，参数为空。load()函数也调用historyLoad()。在这种情况下，传递给$.history.load()的参数（单击的单词）将作为回调函数的参数。

在本解决方案中使用了一个jQuery插件。如果不使用插件，使用JavaScript的window.location.hash对象也很容易实现类似的功能。jQuery历史插件仅包含156行代码。选择它而不从头开始编写解决方案的原因是这个插件的大部分代码处理不同浏览器不一致的问题。当处理浏览器差异时，利用插件中积累的共有经验往往比自己考虑各种实现差异更有效率。

17.7 将JavaScript放在页面的最后

17.7.1 问题

随着项目规模的增大，包含的JavaScript量也会增大。这将造成页面加载速度的降低。将多个不同的JavaScript文件组合成一个庞大的文件，然后使用精简和压缩，有助于减小JavaScript的尺寸，并减少HTTP请求的数量。但是，总是必须加载一些代码。从感觉上减低这些代码对加载时间的影响是很好的事情。

17.7.2 解决方案

用户根据他们在屏幕上看到的东西来感觉加载的时间。当浏览器加载外部内容（如JavaScript、CSS样式表和图片）时可以使用的HTTP连接有限。当JavaScript放在文档的开头时，就会推迟其他可见资源的加载。解决方案是将JavaScript文件放在页面的最后：

```
<!DOCTYPE html>
<html><head>
    <title>17.7 Putting JavaScript at the End of a Page</title>
</head>

<body>
    <h1>17.7 Putting JavaScript at the End of a Page</h1>
    <p>Lorem ipsum dolor...</p>
    <script src="../../jquery-1.3.2.min.js"></script>
    <script type="text/javascript">
        jQuery(document).ready(function() {
            jQuery('p').after('<p>Ut ac dui ipsum...</p>').show();
        });
    </script>
</body></html>
```

17.7.3 讨论

通过在<body>结束标记之前放置JavaScript，文档中引用的所有图片或者CSS样式表会首先加载。这不会使页面更快加载，但是它将减少感觉上的加载时间。可见元素比JavaScript代码更优先，将JavaScript文件放在后面加载没有任何不良的影响，因为通常它们都不应该在整个页面加载完毕之前运行。

将内联JavaScript放在文档最后得不到任何好处。在这个例子中，内联JavaScript放在那里只是因为在jquery-1.3.2.min.js加载之前无法调用jQuery函数。如果将内联JavaScript放在<head>元素中，由于jQuery未定义，将会生成一个错误。

第18章　单元测试

*Scott González*和*Jörn Zaefferer*

18.0 导言

自动化软件测试是开发中不可或缺的工具。单元测试是自动化测试的基本组成部分：软件的每个组件（单元）都需要一个可以由测试运行器运行，而不需要人工干预的测试。换句话说，你可以一次编写测试，根据需要多次运行而没有任何附加成本。

除了好的测试覆盖这一好处之外，测试还能够驱动软件的设计，这称为测试驱动设计，在这种方法中测试在实现之前编写。你先编写一个非常简单的测试，证明测试失败（因为被测试的代码还不存在），然后编写必要的实现，直到测试通过。然后，你扩展测试覆盖更多想要的功能，再次实现。重复上述步骤，最终的代码通常与从实现开始编写的代码有很大的不同。

JavaScript中的单元测试与其他编程语言没有太大的不同。你需要一个小的框架提供测试运行器，以及编写实际测试的一些工具。

18.1 自动化单元测试

18.1.1 问题

你想要自动化测试应用程序和框架，甚至从测试驱动设计中获益。编写自己的测试框架很有诱惑力，但是需要花费许多精力去了解所有细节，以及在各种浏览器中测试JavaScript代码的特殊需求。

18.1.2 解决方案

虽然有用于JavaScript的其他单元测试框架，但我们将关注QUnit（*http://jquery-cookbook.com/go/qunit*）。QUnit是jQuery单元测试框架，广泛地用在各种项目中。

要使用QUnit，需要在网页中包含jQuery和两个QUnit文件。QUnit包含测试运行器和测试框架`testrunner.js`，以及设置测试套件页面样式，显示测试结果的`testsuite.css`：

```
<!DOCTYPE html>
<html>
<head>
  <title>QUnit basic example</title>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
  <link rel="stylesheet"
href="http://jqueryjs.googlecode.com/svn/trunk/qunit/testsuite.css" type="text/css"
media="screen" />
  <script type="text/javascript"
src="http://jqueryjs.googlecode.com/svn/trunk/qunit/testrunner.js"></script>

<script type="text/javascript">
   test("a basic test example", function() {
   ok( true, "this test is fine" );
   var value = "hello";
   equals( value, "hello", "We expect value to be hello" );
   });
</script>

</head>
<body>
  <div id="main"></div>
</body>
</html>
```

在浏览器中打开以上文件可以看到如图18-1的结果。

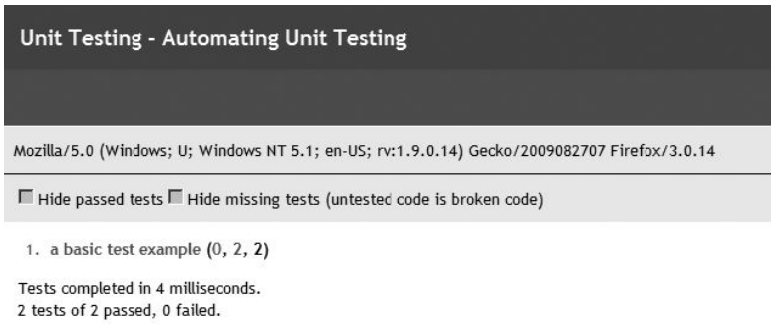


图18-1　浏览器中的测试结果

<body>元素中唯一必需的标记是具有id="main"属性的 。这对于所有QUnit测试都是必需的，即使元素本身为空也不例外。该元素提供了测试夹具，这将在秘诀18.6中解释。

有趣的部分是testrunner.js包含代码之后的<script>元素。它由对test函数的调用组成，使用了两个参数：以后用于显示测试结果的测试名称字符串，以及一个函数。该函数包含实际的测试代码，包括一个或者多个断言。该示例使用了两个断言ok()和equals()，秘诀18.2将介绍这些断言。

注意，代码中没有document-ready块。测试运行器会处理相关的工作：调用test()只是将测试添加到队列，它的执行被推迟，由测试运行器控制。

18.1.3　讨论

测试套件的首部显示了页面标题、所有测试通过时的绿条（至少一个测试失败时的红条）、带有navigator.userAgent字符串的灰条（便于不同浏览器中测试结果的屏幕截图）以及带有过滤测试结果的几个复选框的横条。

“Hide passed tests”（隐藏通过的测试）在运行许多测试且只有少数失败的情况下有用。选中该复选框将隐藏所有通过的测试，更易于关注失败的测试。

“Hide missing tests”（隐藏丢失的测试）在有许多只是作为占位符的测试时有用，占位符用测试名称“missing test—untested code is broken code”表示。当你有很大的未测试代码库并且为每个仍然需要编写的测试添加占位符时，这一选项很有用。为了关注已经实现的测试，可以用这个复选框暂时隐藏占位测试。

页面的实际内容是测试结果。编号列表中的每个条目都从测试名称开始，随后的圆括号中是失败、通过和总断言数。单击条目将显示每个断言的结果，通常有预期结果和实际结果的细节。双击将运行该测试（详见秘诀18.8）。

在测试结果下面是一个摘要，显示运行所有测试花费的时间和总断言数及失败断言的数量。

18.2 断言结果

18.2.1 问题

断言是任何单元测试必不可少的要素。测试的作者需要表达预期的结果，由单元测试框架将它们与实现的测试产生的实际值比较。

18.2.2 解决方案

QUnit提供三个断言。

ok(boolean[, message])

最基本的断言是ok()，它只有一个布尔参数。当参数为真时，断言通过；否则，失败。此外，它还接受一个字符串作为测试结果中显示的消息：

```
test("ok test", function() {
    ok(true, "ok succeeds");
    ok(false, "ok fails");
});
```

equals(actual, expected[, message])

equals断言使用简单比较运算符（==）比较实际和预期参数。当两者相等时，断言通过；否则，失败。当该断言失败时，在测试结果中显示预期和实际值，以及指定的消息：

```
test("equals test", function() {
    equals("", 0, "equals succeeds");
    equals("three", 3, "equals fails");
});
```

和ok()相比，equals()更易于调试失败的测试，因为导致测试失败的值很明显。

same(actual, expected[, message])

same()断言可以像equals()一样使用，在大部分情况下前者是更好的选择。它用更精确的比较运算符（===）代替简单比较运算符（==）。这样，undefined不会等于null、0或者空串（""）。它还比较对象的内容，所以{key: value}等于{key: value}，即使比较的两个对象标识不同。

same()还能处理NaN、日期、正则表达式、数组和函数，而equals()只能检查对象标识：

```
test("same test", function() {
    same(undefined, undefined, "same succeeds");
    same("", 0, "same fails");
});
```

当确定不比较两个值的内容时，仍然可以使用equals()。一般来说，same()是更好的选择。

18.3 测试同步回调

18.3.1 问题

当测试具有许多回调的代码时，偶尔会出现应该失败的测试却通过的情况，在测试结果中从不出现问题中的断言。当断言所在的回调从未调用时，断言也就不会调用，测试也就悄无声息地通过了。

18.3.2 解决方案

QUnit提供了一个特殊的断言，用于定义测试包含的断言数量。当测试完成而没有通过正确数量的断言时，不管其他断言产生的结果如何，该断言都将失败。

这个断言的用法很简单；只要在测试开始调用expect()，唯一的参数是预期的断言数：

```
test("a test", function() {
    expect(1);
    $("input").myPlugin({
        initialized: function() {
            ok(true, "plugin initialized");
        }
    });
});
```

18.3.3 讨论

expect()在实际测试断言的时候最有价值。当所有代码运行于测试函数作用域时，expect()没有任何价值——任何阻止断言运行的错误都会导致测试失败，因为测试运行器捕捉到该错误，认定测试失败。

18.4 测试异步回调

18.4.1 问题

expect()对测试同步回调有用（见秘诀18.3），但是它在测试异步回调时达不到效果。异步回调与测试运行器排队和执行测试的手段冲突。当正在测试的代码启动超时、时间间隔或者Ajax请求时，测试运行器将继续运行其余测试以及后续的其他测试，而不是等待异步操作的结果。

18.4.2 解决方案

有两个函数能够人工同步异步操作。在任何异步操作之前调用stop()，在所有断言结束后调用start()，让测试运行器能够继续其他测试：

```
test("a test", function() {
    stop();
    $.getJSON("/someurl", function(result) {
        equals(result.value, "someExpectedValue");
        start();
    });
});
```

18.4.3 讨论

人工同步方法的缺点是当测试中的代码在别处失败时，有可能永远不会调用start()。在这种情况下，测试运行器不会继续，从而永远不会结束和显示最终结果，甚至不能显示当前测试的结果，而只显示之前的测试结果。当发生这种情况时，首先需要搜索前面结束的测试，找出没有结束的测试，然后找到代码中的测试，跳到下一个测试。完成了这一步，可以为stop()的调用添加一个超时参数来简化调试：

```
test("a test", function() {
    stop(500);
    $.getJSON("/someurl", function(result) {
        equals(result.value, "someExpectedValue");
        start();
    });
});
```

在这个例子中，测试运行器将等待500毫秒让测试结束（使用setTimeout）；如果测试没有结束则将其声明为失败并继续运行。通过查看其他测试结果，就能够更简单地找出实际问题并加以修复。

但是，对于常规测试不应该使用超时参数。如果为了调试而添加该参数，在测试正常工作之后删除它。

这是为什么？超时的缺点是使测试变得不确定。在慢速或者负载沉重的机器上运行测试时，超时可能太短，导致其他完全没有问题的测试失败。寻找一个根本不存在的缺陷是非常费时而令人沮丧的经历——一定要避免。

18.5 测试用户操作

18.5.1 问题

依赖用户操作的代码无法仅靠调用函数来测试。通常必须模拟绑定到元素事件（如单击）的一个匿名函数。

18.5.2 解决方案

可以用jQuery的`trigger()`方法触发事件并测试预期行为的发生。如果你不希望触发浏览器原生事件，可以使用`triggerHandler()`执行绑定的事件处理程序。这在测试链接上的单击事件时有用，因为`trigger()`会导致浏览器打开另一个位置，这是测试中不希望出现的。

假定要测试一个简单的击键记录程序：

```
var keylogger = {
    log: null,
    init: function() {
        keylogger.log = [];
        $(document).unbind("keydown").keydown(function(event) {
            keylogger.log.push(event.keyCode);
        });
    }
};
```

可以人工触发按键事件，查看记录程序是否正常工作：

```
test("basic keylogger behavior", function() {
    // 初始化
    keylogger.init();

    //触发事件
    var event = $.Event("keydown");
    event.keyCode = 9;
    $(document).trigger(event);

    // 验证预期行为
    same(keylogger.log.length, 1, "a key was logged");
    same(keylogger.log[0], 9, "correct key was logged");
});
```

18.5.3 讨论

如果事件处理程序不依赖事件的任何特殊属性，可以调用`.trigger(eventType)`。但是如果事件处理程序依赖事件的特殊属性，就需要像前面说明的那样，用`$.Event`创建一个事件对象，并且设置必要的属性。

触发复杂行为的所有相关事件也很重要，如拖动，它包含mousedown，至少一次的mousemove和mouseup事件。记住，有些看似简单的事件实际上也是复合式的，例如，单击实际上是一次mousedown、一次mouseup，以及之后的单击动作组成的。是否真的需要触发这三个事件取决于测试中的代码，在大部分情况下触发单击事件就可以了。

18.6 保持测试的原子性

18.6.1 问题

当所有测试合在一起时，可能有些应该通过的测试却失败了，或者应该失败的测试通过了。这是因为前一测试的副作用造成当前测试的结果无效：

```
test("2 asserts", function() {
    $("#main").append("<div>Click here for <span class='bold'>messages</span>.</div>");
    same($("#main div").length, 1, "added message link successfully");
    $("#main").append("<span>You have a message!</span>");
    same($("#main span").length, 1, "added notification successfully");
});
```

注意，第一个append()添加了一个，这在第二个断言中未加考虑。

18.6.2 解决方案

使用test()方法保持测试的原子性，小心地保持每个断言不会产生副作用。你应该只依赖#main元素中的夹具标记。修改和依赖其他标记都可能产生副作用：

```
test("test 1", function() {
    $("#main").append("<div>Click here for <span class='bold'>messages
</span>.</div>");
    same($("#main div").length, 1, "added message link successfully");
});
test("test 2", function() {
    $("#main").append("<span>You have a message!</span>");
    same($("#main span").length, 1, "added notification successfully");
});
```

QUnit在每次测试之后将复位#main元素中的夹具标记，删除现有的任何事件。只要仅使用夹具中的元素，就没有必要在测试之后手动清理以保持原子性。

18.6.3 讨论

除了#main夹具元素，QUnit还自行清理jQuery属性$.event.global和$.ajaxSettings。任何全局事件（如$().ajaxStart()）都由jQuery在$.event.global中管理——如果测试绑定了许多这类事件，在运行多项测试时它们将明显降低运行器的速度。通过清理这些属性，QUnit确保测试不会受到全局事件的影响。

上述原则对于$.ajaxSettings也适用，$.ajaxSettings通常用来通过$.ajaxSetup()为$.ajax()调用配置常见属性。

除了秘诀18.8介绍的过滤器之外，QUnit还提供一个?noglobals标记。考虑如下的测试：

```
test("global pollution", function(){
    window.pollute = true;
    same(pollute, true);
});
```

在正常测试运行中，这将是一个有效的结果。用noglobals（*http://jquery-cookbook.com/examples/18/06-keeping-tests-atomic/globals.html?noglobals*）标志运行同样的测试会导致测试失败，因为QUnit发现代码污染了窗口对象。

没有必要总是使用这个标志，但是它对于检测全局命名空间污染来说很方便，这种污染在组合使用第三方程序库时将会成为问题。这个标记还有助于检测测试中副作用引起的缺陷。

18.7 分组测试

18.7.1 问题

你已经分解了所有测试，保持它们的原子性并消除副作用，但是你打算用合乎逻辑的方式组织它们，并且能够独自运行一组特定的测试。

18.7.2 解决方案

可以用module()函数对测试分组：

```
module("group a");
test("a basic test example", function() {
    ok( true, "this test is fine" );
});
test("a basic test example 2", function() {
    ok( true, "this test is fine" );
});

module("group b");
test("a basic test example 3", function() {
    ok( true, "this test is fine" );
});
test("a basic test example 4", function() {
    ok( true, "this test is fine" );
});
```

所有在调用module()之后发生的测试将集中到一个模块。在测试结果中的测试名称前面加上模块名称。以后，可以使用模块名称选择运行的测试（见秘诀18.8）。

18.7.3 讨论

除了分组测试之外，module()还可用于从该模块中的测试提取公共代码。module()函数的第二个参数是可选的，定义在模块中每个测试前后运行的函数：

```
module("module", {
    setup: function() {
        ok(true, "one extra assert per test");
    }, teardown: function() {
        ok(true, "and one extra assert after each test");
    }
});
test("test with setup and teardown", function() {
    expect(2);
});
```

可以指定设置和卸载属性，或者其中之一。

再次调用module()且不带附加参数将会重置前一个模块定义的设置/卸载函数。

18.8 选择运行的测试

18.8.1 问题

当调试失败的测试时，对代码进行小修改之后只需要查看某个测试是否通过，重新运行整个测试套件是巨大的浪费。

18.8.2 解决方案

QUnit提供URL过滤来选择运行的测试，这在与模块组合时效果最好。可以在测试套件URL后附加带有模块名称的查询字符串，只运行指定模块中的测试。例如，test.html?validation将运行模块validation中的所有测试：

```
// test.html?validation - 只运行验证模块
// test.html?validation&tooltip - 验证和工具提示模块
// test.html?!validation - 排除验证模块
// test.html?test 3 - 只运行"test 3",URL将显示为test.html?test%203
module("validation");
test("test 1", function () {
    ok(true, "bool succeeds");
});
test("test 2", function () {
    equals(5, 5.0, "equals succeeds");
});
module("tooltip");
test("test 3", function () {
    same(true, 3 == 3, "same succeeds");
});

test("test 4", function () {
    ok(false, "bool fails");
});
module("other");
test("test 5", function () {
    equals(3, 5, "equals fails");
});
```

18.8.3 讨论

可以一次指定多个模块，组合各个模块中的测试，模块之间用&符号分隔，例如，test.html?validation&tooltip将运行其中包含validation或者tooltip的测试。

可以用感叹号排除测试；例如，test.html?!validation将运行除了validation模块之外的所有测试。

也可以双击测试结果重新运行该项测试，而不是手动修改URL。QUnit将使用相同的过滤机制，把测试名称附加到当前位置。

后记

本书封面上的动物是貂（ermine），又称白鼬（stoat）。“ermine”有时候指的是冬天时这种动物的白色毛皮，而“stoat”一词则用来指在其他季节时它的褐色毛皮。貂属于黄鼬科，这一科包括貂鼠、雪貂、水貂、水獭和臭鼬，但是貂因其带有黑点的尾巴而不同于其他同科动物。

貂生活于欧洲、亚洲、北美洲北部的林地，主要在夜间活动，用树根在岩石下及隧道中筑巢。它是一种独居的动物，能够在一晚上行走多达10英里寻找食物。它的天敌包括狐狸、獾、猫科动物和食肉禽。

貂细长的身体使其即使在穿越雪地时跑得也非常迅速，攀爬和游泳时也很敏捷，尽管这种外形有优势，但是也导致它身体的热量很快散失。厚实的毛皮和快速的新陈代谢有助于补偿热量的散失，而且貂每天都必须进食才能满足能量的要求。它的食物包括小型哺乳动物、鸟类、鱼和昆虫。当发现猎物时，它快速潜近，抓住猎物的颈部，用反复地撕咬将其杀死。

白色的貂皮价格很高，用于缝制外套，但是近年来需求已经下降。通常，将多张貂皮缝在一起形成白色区域上的黑点图案，这种图案早在12世纪就被盾徽所效仿，最有名的是布列塔尼岛的徽章。貂皮也是高贵和纯洁的象征，这可能是英国女王伊丽莎白一世（“童真女王”）使用貂皮作为装饰的原因。

看完了

如果您对本书内容有疑问，可发邮件至contact@epubit.com.cn，会有编辑或作译者协助答疑。也可访问异步社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@epubit.com.cn。

在这里可以找到我们：

- 微博：@人邮异步社区
- QQ群：368449889
