HZ BOOKS
华章科技

PEARSON

Web开发
技术丛书

jQuery应用开发实践指南

[德] Ralph Steyer 著　姚军 等译

Learning jQuery
A Hands-on Guide to Building Rich Interactive Web Front Ends

- 资深Web技术专家、Ajax技术权威撰写
- 全书以实例驱动，系统讲解jQuery各种功能组件的用法和技术细节，以及jQuery UI和jQuery Mobile等各种插件及扩展的使用方法，能帮助读者迅速掌握jQuery应用的开发

机械工业出版社
China Machine Press

jQuery JavaScript程序库大大简化了现代丰富Web应用程序的创建，同时能够无缝地与所有主要Web开发平台和框架集成。本书将指导你在自己的项目中使用jQuery、jQuery UI和jQuery Mobile。你将一步一步地学习如何完成所有的工作，从添加简单的特效直到构建完整的互联网应用。

书中有丰富的代码示例，是为每位Web开发人员设计的。在清晰地解释所有基本知识之后，Ralph Steyer展示了如何应用jQuery创建特效、动画、幻灯片、列表、可拖放元素和交互式表单等。

如果你是一位Web开发人员，即使只有基本的JavaScript经验，本书也是你利用jQuery取得成功的最快途径！

本书主要包含如下内容：

- 揭示jQuery的能力，以及它处理JavaScript和DOM的方式
- 选择组件以支持动态处理
- 操纵网页内容和结构
- 用CSS样式表，通过jQuery应用和修改格式
- 更有效和可靠地处理复杂事件
- 生成依赖时间和独立于时间的CSS特效
- 用插件扩展jQuery能力
- 使用jQuery创建更简单、更好、更强大的Ajax代码
- 掌握强大、灵活的jQuery UI插件，用于视觉控制和用户交互
- 用ThemeRoller简化jQuery UI界面的创建
- 掌握成功使用组件和窗口小部件的基本原则
- 用jQuery Mobile构造支持触摸的移动前端

PEARSON
www.pearson.com

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

上架指导：计算机/Web设计
ISBN 978-7-111-45952-1
9 787111 459521 >
定价：79.00元

jQuery应用开发实践指南

Learning jQuery
A Hands-on Guide to Building Rich Interactive Web Front Ends

[德] Ralph Steyer 著　姚军 等译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

jQuery 应用开发实践指南 /（德）斯泰尔（Steyer，R.）著；姚军等译 . —北京：机械工业出版社，2014.4
（Web 开发技术丛书）
书名原文：Learning jQuery: A Hands-on Guide to Building Rich Interactive Web Front Ends

ISBN 978-7-111-45952-1

I. j… II. ① 斯… ② 姚… III. JAVA 语言 – 程序设计 – 指南 IV. TP312-62

中国版本图书馆 CIP 数据核字（2014）第 035791 号

本书版权登记号：图字：01-2013-4810

jQuery 应用开发实践指南

（德） Ralph Steyer 著

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：关 敏
印 刷：藁城市京瑞印刷有限公司
开 本：186mm × 240mm 1/16
书 号：ISBN 978-7-111-45952-1
版 次：2014 年 4 月第 1 版第 1 次印刷
印 张：23.75
定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066
购书热线：（010）68326294 88379649 68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

译 者 序

在不断发生重大改变的互联网中，各种技术层出不穷，但在 Web 前端编程中，JavaScript 以其编程简易和各种平台均支持而长盛不衰，即便在 HTML 语言和 CSS 都得到很大扩充的今天，它仍是 Web 前端人员的首选语言。

Web 前端的编程意味着开发人员要面对各种纷繁复杂的平台和浏览器。浏览器历史遗留下来的实现不一致等问题虽然在万维网联盟和各厂商的努力下有所改进，但仍不时困扰着开发人员，而移动 Web 的兴起和用户对上网体验的高要求更加剧了这一问题。如何在不花费大量精力的情况下开发在各种平台上都能提供一致体验的 Web 应用呢？在这种情况下，各种 JavaScript 框架应运而生，jQuery 就是其中的佼佼者。

jQuery 不仅从核心的 DOM 操纵开始，结合 CSS 选择器，为 JavaScript 提供了大量方便的函数和方法，在所有主流平台上实现一致的效果，大大减轻了开发人员的负担和工作量，而且通过 jQuery UI 和 jQuery Mobile 子框架，更进一步延伸了开发的范围，既能在图形界面上为 RIA 提供良好的支持，又能够适应现代移动 Web 开发的要求。这些强大的能力使它成为许多网站的支柱，也备受 Microsoft 等大厂商的青睐。

本书通过大量实例，由浅入深地介绍了 jQuery 框架的方方面面，为对这一框架感兴趣的各类开发人员提供了一本完备的指南。书中精选了 jQuery 的核心组件，着重介绍了 jQuery 对传统 JavaScript 编程的增强，细心研读之下，能够对这一流行框架有全面的了解，相信也能给网站编程带来很大的好处。除了核心框架之外，本书还用独立的章节介绍了 jQuery UI、jQuery Mobile 以及插件的使用和开发。在翻译期间，我们也深深地为 jQuery 系列框架的强大功能和精巧的设计所打动，因此，我们向广大读者推荐本书，希望它能够伴随你们走上 Web 开发之路。

本书的翻译工作主要由姚军完成，徐锋、陈绍继、郑端、吴兰陟、施游、林起浪、刘建林、陈志勇、白龙、方翊、林耀成等人也为本书的翻译工作做出了贡献。由于译者水平所限，错漏在所难免，敬请广大读者批评指正。

译者

2014 年 2 月

目　录

第 1 章
导　言

富互联网应用（Rich Internet Application，RIA），以及它们所带来的表达有些模糊的“丰富”（Rich）机会，在过去几年内显著地改变了我们使用 Web 的方式。这种变化的速度还在不断加快。经典桌面应用程序的重要性已经被重新评估。我们传统上仅当成桌面应用程序使用的许多种程序突然出现在 Web 上，比如个人时间表、整套办公室软件、游戏、行程安排或者通信程序。而且，智能手机上的应用也越来越多地基于 Web 技术。这完全改变了用户行为和用户对互联网应用程序以及服务可用性的期望。作为经典且有一定额外价值的 Web 应用，一方面，只要有还过得去的互联网访问手段和一个现代的浏览器，RIA 就总是可用的。[⊖] 另一方面，从操作、性能和外观来看，它们和经典的桌面或者移动应用已经很难区分了。

确保拥有这些丰富机会的最有效手段之一通常涉及使用合适的 Web 框架。但是要注意，如果你使用一个框架，就会在很大程度上依赖一个制造商或者一个项目，也就不再对应用中的源代码有完全的控制。在任何情况下，使用框架要求充分熟悉相关的函数库和系统的工作方法。和某些框架（以及某些工具）浮夸的宣传形成对比的是，通常只有在理解 Web 编程概念，并且至少拥有底层技术的基本知识之后，才能有效地使用它们。严格地讲，到你不那么需要框架、掌握了更多的基本知识之后，才能从框架中得到最多的好处。

尽管有这些问题和不足，框架和工具包的使用仍然值得一提。它们无疑能够帮助我们更快、更高效地开发和维护复杂的网站；也可以用它们提供更丰富和更健壮的网站。

注意

上文中多次使用了“框架”和“工具包”等词语。框架及其与工具包之间的差别没有标准的定义。实际上，可靠的定义和差别并不是很直接。一般来说，“框架”一词指的是已经提供了某些功能的编程框架。框架本身不是一个完成的程序，但是程序员可以用它所提供的骨架创建应用程序。框架通常包含一个带有预定义代码结构的程序库，而且规定了对使用

⊖ 浏览器变成某个特定任务的多功能访问工具，从而代替了经典的应用类型。未来，用户可能只需要一个浏览器作为应用，或者操作系统和浏览器可能合并，无法分辨。

这些程序库的行为模式的一些控制（例如，某种语法），这与纯粹的程序库形成了对比。工具包的重点是一组程序（工具），但是这些程序也可以基于特定的程序库或者某个语法概念。框架和工具包都常常提供窗口小部件或者组件——换句话说，组成图形用户界面（GUI）的元素。

1.1 本书的内容

本书提供了使用 jQuery、jQuery UI 和 Mobile jQuery 进行 Web 编程的简单介绍。jQuery 是一个建立于 JavaScript 语言基础上的免费、全面的框架。它最初由 John Resig 开发，于 2006 年 1 月的 BarCamp（NYC）会议上发布。现在，jQuery 已经进一步发展为一个开放源码项目。jQuery UI 建立于 jQuery 之上，用 UI 专用组件扩展了 jQuery 框架。同样，jQuery Mobile 也构建于 jQuery 之上，用移动设备专用的组件扩展了 jQuery 框架。

jQuery 框架提供了一整组非常有用的功能（例如，易用的 DOM 操纵和浏览函数，以及基本的 AJAX 支持）。除此之外，该框架还提供了对层叠样式表（Cascading Style Sheet，CSS）、扩展的事件系统、令人印象深刻的特效和动画、多种辅助功能以及大量免费插件的支持。

但是，jQuery 特别出色的方面是与许多大型行业提供商的 Web 平台或者它们的官方支持的无缝集成。Microsoft 在 Visual Studio 开发环境中使用 jQuery 与 ASP.NET MVC 框架及 Microsoft 异步 JavaScript 和 XML(Asynchronous JavaScript and XML，AJAX）相结合。例如，如果创建一个新的 ASP.NET 项目，可以自动集成 jQuery（在最新版本中，这不是必需的）。

在 Visual Studio 中的 ASP.NET MVC3 或者更新版本的 Web 应用中，甚至可以选择集成某些 jQuery 插件（例如用于验证用户输入的 jquery.validate.js）。如图 1-1 和图 1-2 所示。

图 1-1 在 Visual Studio 2010 的一个 ASP.NET Web 应用中，也可以集成 jQuery

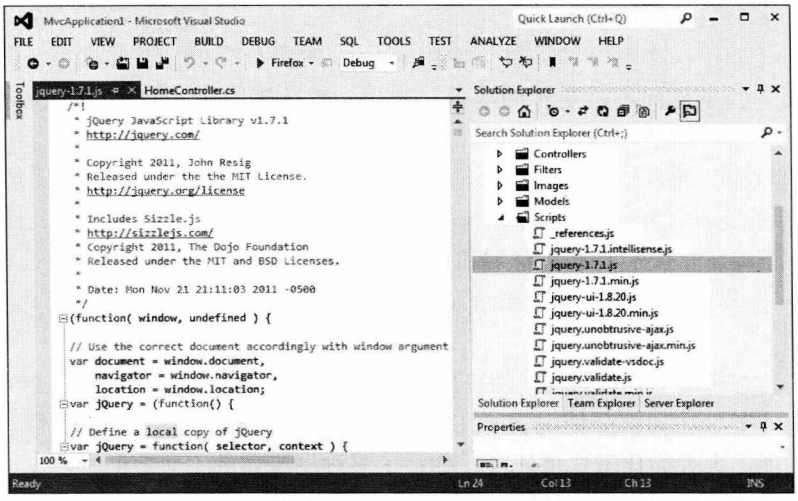

图 1-2 一个 ASP.NET MVC4 Web 应用引用了 jQuery 和几个 jQuery 插件

指向程序库的链接在预先生成的源文本中已经创建，可以通过删除注释字符简单地启用。如图 1-3 所示。

```
<head>
    <title>@View.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    @*
    <script src="@Url.Content("~/Scripts/jquery-1.4.1.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/jquery.unobtrusive-ajax.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/jquery.validate.min.js")" type="text/javascript"></script>
    <script src="@Url.Content("~/Scripts/jquery.validate.unobtrusive.min.js")" type="text/javascript"></script>
    *@
</head>

<body>
```

图 1-3 在生成的代码中，只需要启用脚本引用

除了 Visual Studio 之外，许多其他的 Web 开发工具也提供 jQuery。而且，一些不同方面的其他供应商也使用 jQuery（例如，手机制造商诺基亚的 Web 运行时平台，以及 Google、Dell、Mozilla、WordPress、Drupal 和 Digg）。流行用户的列表就像 Web 的名人录一样。正如许多统计数字所证明的那样，jQuery 框架的流传已经极其广泛。如果关注现代 RIA 就会发现，尽管 jQuery 框架也有强大和引人注目的竞争者，但超过 60% 的这类应用正在使用 jQuery 和 jQuery UI。

可以从本书学到什么

在本书中，你将学习到如何将 jQuery 用于自己的 Web 应用——从只想添加单独特效的简单网站，到复杂的 RIA。本书针对初学者，虽然不是从零开始，但是本书的目标读者不是狂热的爱好者和编程专家。你不一定要有 AJAX、框架或者工具包的大量经验。但是你应该对 Web 编程有一定的经验——你很快会在目标读者相关的内容中看到更多的解释。

本书的每一章都遵循相同的基本结构，首先是一个简单的介绍，然后是更为详细的主

题，最后是一个小结。

具体的路径如下：本章已经提供了使用jQuery的所有需求，在此之后，我们将直接深入叙述并讲解一些示例而不做更多的准备。这样做旨在提供对jQuery所能完成工作的感性认识。

接着，我们将介绍一些关于Web、JavaScript、AJAX、可扩展标记语言（XML）、JavaScript对象标记法（JSON）等的基本背景信息。然后，我们更详细地研究使用jQuery所涉及的知识。下一步，我们转向选择器和过滤器。我认为，在网站上下文中选择对象的这些选项是jQuery框架中最大的特色，形成了访问网站元素的基础。我们提供了许多例子帮助大家理解。

接下来是动态超文本标记语言（DHTML）的主题。DHTML的重点是根据某些事件修改网站。本质上（至少在大部分情况下），DHTML意味着动态地影响CSS属性。jQuery同样提供了许多选项，大大简化了这项任务，弥补了多种浏览器的不兼容问题。

在前一段中，我使用了“事件”（event）一词。Web上的事件处理是许多浏览器的“无底洞”。jQuery提供了一个解决方案。在本书中就可以学到。

对许多访问者来说，特效和动画是网站吸引人们眼球的地方。同样，jQuery有十八般兵器，无须担心这方面的工作。

然后，我们探索Web 2.0，转向AJAX，看看jQuery在这方面提供了什么。

这实际上就是jQuery的所有主题。但是等一等，还有什么其他的内容吗？jQuery UI!目前，我在描述本书包含的主题时还没怎么提到它。现在，大家可能还将jQuery UI看成jQuery世界中的丑小鸭，对它可能没有太大兴趣，事实并非如此。jQuery UI实际上是美丽的天鹅。纯粹从视觉外观来说，jQuery UI提供的功能远超过jQuery本身，而且使用更加简单——只要理解jQuery。在源代码和编程方面，jQuery是让人们更为轻松的基础，而jQuery UI是构建于这个基础之上的独立框架，在直观的高级界面组件和CSS主题框架上做得尤为出色。当然，在本书中我们也会仔细地研究jQuery UI，并且介绍它所提供的各种窗口组件的许多例子。此外，你将详细地学习如何使用选项、事件、方法和主题，并进一步对其进行调整。我们还将详细地介绍jQuery UI主题框架和ThemeRoller。

jQuery还有一些插件作为该框架的扩展。我们将学习如何使用外部插件，补充jQuery和jQuery UI核心程序库中所不具备的功能，还将学习如何创建和发布自己的插件。

最后（但并非不重要），本书还描述了创建基于jQuery的移动应用的方法。这涉及直接基于原生jQuery构建（正如jQuery UI）的移动框架的使用。

注意

为清晰起见，在本书中我们常常使用代码示例。大家应该自行输入完整的代码示例（当然，如果喜欢，也可以修改和试验）。然而，也可以在本书的配套网站[⊖]上找到程序清单。

⊖ 本书说的配套网站是指本书英文版的配套网站。——编辑注

1.2 本书使用的约定

我会在完整程序清单的源文本段落上添加特殊的注释。对于所有完整清单和一些较大的源代码片段，大家还会在源文本行上看到编号。这些编号当然不是源文本本身的一部分；它们只是用于更清晰地指出我所引用的是源代码的哪一部分。在少数情况下，由于技术原因而将某个源代码行分成几行可能是有必要的。在这种情况下，源代码行的编号说明在编辑器中哪些行应该被当成一行输入。这对于必须分为多行的较长字符串（引号中的文本）来说特别重要。

1.3 本书的目标读者

预测谁对某个特定主题感兴趣总是很难的。但是关于谁对学习用 jQuery 创建 RIA 感兴趣、他们感兴趣的潜在原因以及本书最可能的读者，我有一些想法，从 jQuery 的研讨会中也得到了很多的经验。我假定大家已经创建过网站，并且已经编写过某种形式的程序。JavaScript 是很重要的基础，但是其他编程技术也是受欢迎的，只是学习曲线将会略微陡峭一些。样式表对你来说也应该是个熟悉的概念。如果过去没有任何创建网站或者使用 HTML 及编程的经验，本书对你来说可能有一定的困难（但是这不应该成为阅读本书的障碍）。我还假定，你厌倦了静态 HTML 网站的局限性。你可能已经有某些动态网站的经验（至少作为用户），你可能想要找到一个简单的方法来创建这种现代化的网站。jQuery 是实现这一目标的有趣方法。

越来越多使用 Java 或者 .NET 等强大技术和环境的编程人员开始进入 Web 编程领域。相应地，我也希望考虑具有这类背景知识的读者。对于从这些强大而严格的环境转换过来的编程人员来说，Web 编程世界似乎微不足道（但是实际上相当与众不同），但人们却往往不得其门而入。

1.4 你需要什么

接下来我们介绍使用本书和 jQuery 所需要满足的条件。

1.4.1 硬件和操作系统

我们要和互联网打交道。所以，当然需要一台能够访问互联网的电脑。对于电脑本身没有特殊的要求，但是硬件应该具有合理的质量。现代化操作系统的需求已经确定了所需硬件的最低水平。只要进行了相当的更新，所有图形化操作系统如 Linux、Windows 或者 OS X 都能够工作得很好。正如 Web 上的大部分情况，使用的系统类型和我们的目的没有关系。

1.4.2 jQuery 和 jQuery UI

当然，需要有 jQuery 才能重现本书中的示例。在本书的后半部分还需要 jQuery UI。可以

从jQuery网站下载最新版本的jQuery和过去的版本：http://docs.jquery.com/Downloading_jQuery。

提示

还可以从jQuery项目的首页下载当前版本：http://jquery.com/。你将看到一个大型按钮，可以直接加载这个JavaScript程序库。在编写本书时，当前版本为1.9.1。

可以下载不同的版本，本质上，精简版本没有注释和多余的空格或者换行，主要用于生产环境，而非压缩版本的源代码中含有注释，容易理解但是较大。两个版本的功能相同：它们都包含通常名为jquery.js的一个JavaScript文件，文件名通常包含版本号以及对其类型的描述[⊖]，该文件是该框架整合到你的网站中的核心程序库。如果从互联网下载了zip档案文件，只要简单地解压即可。然后，在你的网站上按照一般的规则引用这个JavaScript文件（后面将做更详细的介绍）。

提示

如果点击下载jQuery文件的链接，大部分浏览器只会显示该文件，而不会提供保存它的选项。毕竟，这是一个JavaScript程序库，本身通常显示为纯文本。相反，如果点击一个zip文件，通常会在点击文件时弹出的浏览器下载对话框中得到保存的选项。点击jQuery程序库的时候，可以显示代码，然后点击浏览器的保存页面选项，将jQuery程序库保存到本地。如图1-4和图1-5所示。

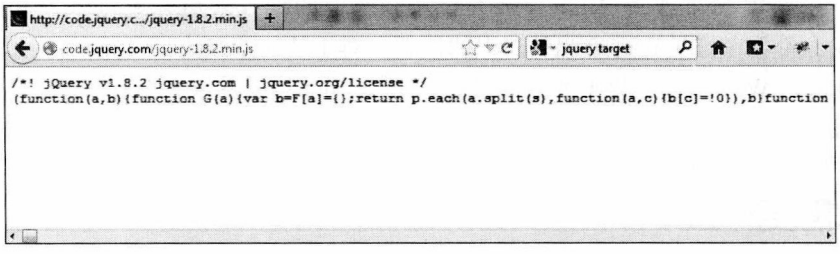

图1-4 显示jQuery文件

如果想使用jQuery UI，还需要下载该框架，因为jQuery UI并不包含在常规的jQuery JavaScript程序库中。jQuery UI是整个jQuery框架中的一个独立项目，除了JavaScript文件外，还包含CSS文件和图形等其他资源。可以在http://jqueryui.com找到该项目的首页。在那里，可以通过Download链接下载框架。下载完成后，你会获得一个zip压缩文件，可以解压并将其放到服务器上（和jQuery程序库一样），然后通过一个核心的jQuery UI JavaScript文件将其整合到你的网站。

⊖ 例如，jQuery-1.7.2.min.js表示版本1.7.2的精简版本，jQuery-1.7.2.js是1.7.2版本的非压缩版本。

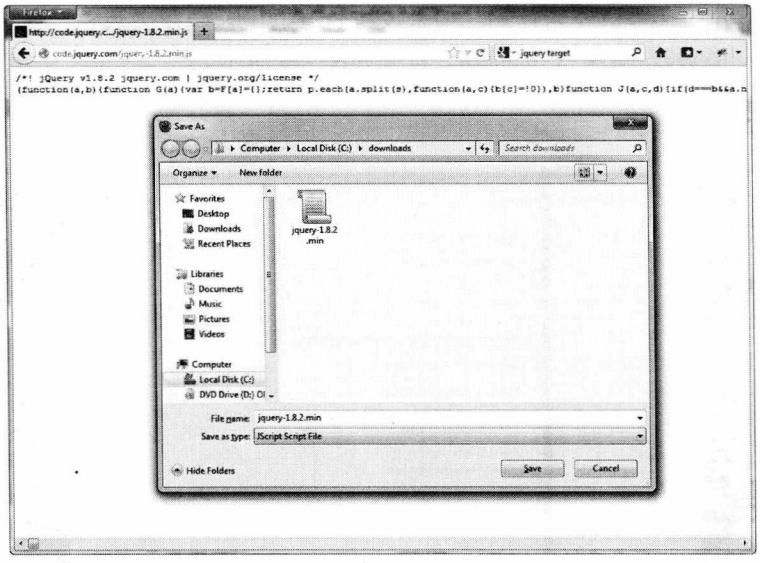

图 1-5 使用浏览器的保存对话框将 jQuery 程序库保存到本地

警告

对于版本需要注意的是，jQuery UI 版本本身总是使用特定的 jQuery 版本，如果两者的版本不匹配，可能出现不兼容的情况。但是 zip 文件总是包含某个版本的 jQuery，该版本满足最低的要求。关于 jQuery UI 的下载和使用还有一些需要注意的地方，我将在与之相关的章节中更详细地解释。

1.4.3 浏览器

用 jQuery 编程当然需要一个支持 jQuery 的浏览器。毕竟，应该能查看自己的网站，才能对其进行测试。使用 jQuery 时，还需要考虑网站的访问者必须符合某个最低标准。与大部分框架和工具包一样，除了浏览器之外，jQuery 对访问使用该程序库的网站的用户没有什么要求。每个 jQuery 新版本对浏览器的最低要求都会改变，但是目前对如下的浏览器有官方支持（可以在 http://docs.jquery.com/Brower_Compatibility 上查看这些支持是否仍然准确）：

- Firefox ⊖
- Internet Explorer
- Safari
- Opera
- Chrome ⊜

⊖ 适用于所有等价的浏览器，如 Netscape Navigator 和 Mozilla。

⊜ 包括安全且更加含蓄（和 Google 喜欢捕捉数据的习惯相比）的变种 Iron。

其他浏览器也可能能够正常使用，但是没有官方的保证。在文档中，有些浏览器正式列出为基本可用，但是有一些已知的问题。具体如图 1-6 所示。

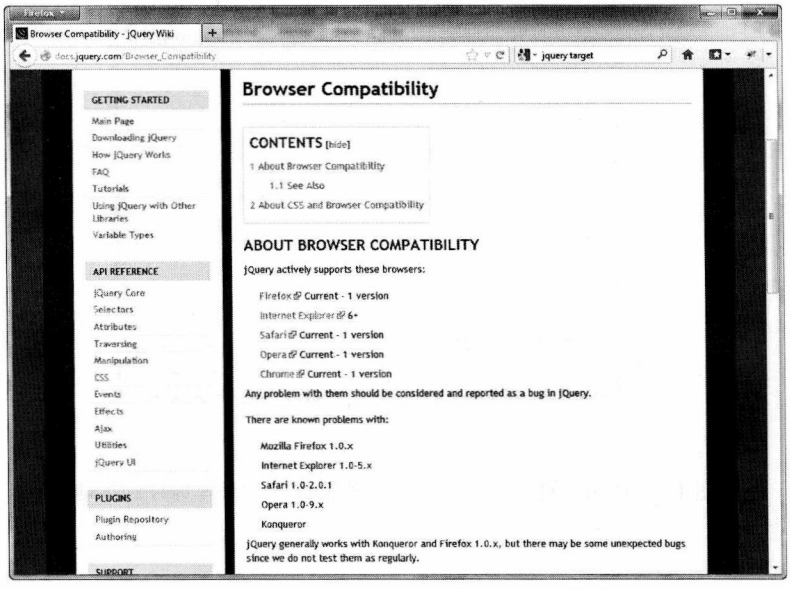

图 1-6 官方支持的浏览器

注意

对版本的要求相当高，但是即使不考虑 jQuery，使用旧浏览器的用户也没有多大希望体验当今的 Web。现在的情况和几年之前已经有所不同，越来越多的网站创建者不再支持尽可能多的浏览器，而是明确地设定最低级别。某些功能在旧浏览器上根本就无法实现，或者需要大量的努力，由于只有少数用户仍然使用这些过时的产品，没人愿意为此付出代价。有些开发人员甚至停止支持某些浏览器类型，以给人留下他们制作的网站特别现代化的印象。网站不支持 Internet Explorer 或者在其中不能正常显示，几乎是“高质量的标志”(据我所知，这在美国几乎成了全民运动)。这种趋势实际上与 10 年前正相反，当时许多网站都表明“为 Internet Explorer 优化”。现在，许多网站希望通过不再支持 Internet Explorer（至少是第 7 版之前）来表明自己优异的质量。它们的座右铭可能是：我们制作现代化的网站，为了使用它们，需要拥有现代化的强大浏览器。

我相信这是错误的想法。我认为，尽管 Internet Explorer 9 已经推出，而且是个真正的好浏览器，我们仍然应该为版本 7 和 8 提供某种“保护物种”的支持（在某种程度上，对版本 6 也应该支持）。毕竟，互联网上的许多用户被迫使用 Internet Explorer[⊖]，或者是因为公

⊖ 甚至古老的版本 6 也仍然时有使用，我在 2010/2011 的研讨会上吃惊地发现了这一点（参会的都是真正的大公司）。

司战略，或者是因为他们没有足够的知识使用替代浏览器，或者是他们就是喜欢使用某种特定的浏览器。我相信，jQuery 在对老版本的 Internet Explorer 的支持上采用了一种明智的方法，我们可以适应 jQuery UI 在 Internet Explorer 上的一些局限性，更多信息参见后面相关的章节（明确的 Microsoft 支持也反映了这一点）。

除非有上面列出的某个浏览器，否则无法可靠地测试 jQuery Web 应用程序。作为网站的创建者，你绝对应该有几个浏览器。因为即使使用 jQuery 这样可靠、制作精良的框架，也仍然需要在所有相关浏览器上测试应用程序。

通常，我们并不知道网站访问者使用的是什么浏览器。所以，在 jQuery 没有完全保证支持的浏览器上也进行测试是一个好主意。例如，尽管在 Firefox 1.0.x、Internet Explorer 1.0 ～ 5.x、Safari 1.0 ～ 2.0.1、Opera 1.0 ～ 9.x 以及 Konqueror 上有已知的问题，但是 jQuery 一般在 Konqueror 或者 Firefox 1.0.x 上都能正常工作，只是无法保证所有组件都正常工作。最终，是否出现问题取决于你所使用的功能，可以在相关的浏览器中进行测试。

1.4.4 用于测试的不同操作系统和虚拟机

如前所述，使用 jQuery 创建网站时操作系统的选择大体上是个人偏好或者其他条件的约束，与测试环境的选择无关。理想的情况下，应该有多个操作系统来测试应用程序，毕竟，网站的访问者也会使用不同的操作系统。

当然，Windows 本身是一个参考系统。大部分 Web 用户使用这种系统。但是 Linux 和 OS X 也得到了广泛的使用，Windows 也有不同的版本。有时候会发现一种有趣的现象，不同的 Web 应用在不同操作系统下会有不同的表现，但是差别应该不会很大。所以，在可能的情况下使用不同的操作系统测试应用程序。

你不必拥有多台电脑或者在操作系统之外安装另一个操作系统。特别是 Linux，可以从许多出色的引导 CD 或者引导 DVD 上直接启动而不需要对硬盘进行任何修改。对于硬件容量充足的读者，仔细考虑一下虚拟解决方案（虚拟机 [VM]）也是很有趣的，例如 VMware（http://www.vmware.com）、Microsoft 的 Virtual PC（http://www.microsoft.com/windows/virtual-pc/default.aspx）或者 VirtualBox（http://www.virtualbox.org/）。这些产品都是免费的，至少在个人使用的时候是这样的，它们在当前运行的操作系统中模拟另一个操作系统。例如，可以使用这些 VM 在 Windows 中启动一个 Linux 系统，反之亦然，也可以在与你的 Windows 安装并行的一个 Windows 系统中安装另一个版本的 Internet Explorer。利用 AJAX，客户系统（换句话说，就是运行于 VM 中的系统）可以作为服务器或者客户端，你也就因此在一台电脑上拥有两个完全独立的系统，可以测试客户端 / 服务器关系，就像在实际系统中一样。

1.4.5 实际测试用的 Web 服务器

使用 AJAX，浏览器从 Web 服务器请求数据并将其集成到网站，而无须重新加载网站。因此，为了实际使用和测试这样的应用程序，必须能够访问互联网上的一台 Web 服务器并

且在上面执行程序和脚本。最终，这对实践中的 AJAX 项目也是必需的。然而，在 Web 应用仍在开发中时，通常无法直接在互联网中的一台 Web 服务器上工作（特别是在只想测试少数功能时）。即使不使用 AJAX，为了正确地测试 Web 应用程序，也需要在 Web 服务器的实际条件下测试。

因此，应该创建一个测试环境，在本地电脑或者本地网络中安装 Web 服务器。Linux 的发行版本几乎总是包含一个或者多个 Web 服务器，Web 应用程序的不同开发环境也有一个集成的 Web 服务器，因此，这一点没有问题。即使没有自动获得一个 Web 服务器或者想要尽可能简单处置，也可以使用 XAMPP 等全包含式软件包——可以从互联网下载不同操作系统的版本（http://www.apachefriends.org）。

XAMPP 软件包是一组与 Apache Web 服务器相关的程序，包括数据管理系统 MySQL（以及管理数据库的 phpMyAdmin 软件）和 PHP 支持、FTP 服务器 FileZilla，以及多种其他 Web 技术。只要用一个简单的助手安装这个软件包，就能够使用一个处于基本配置的全功能 Web 服务器。

警告

注意，在默认设置中，XAMPP 的这些软件包仅用于本地测试目的。为了尽可能地简化工作，所有安全设置都处于最低的水平。

XAMPP 安装完成之后，立刻就可以手工启动 Apache 或者将其设置为你的操作系统的集成服务或者过程，甚至让它在电脑启动时自动启动。XAMPP 提供了一个易于使用的控制面板。如图 1-7 所示。

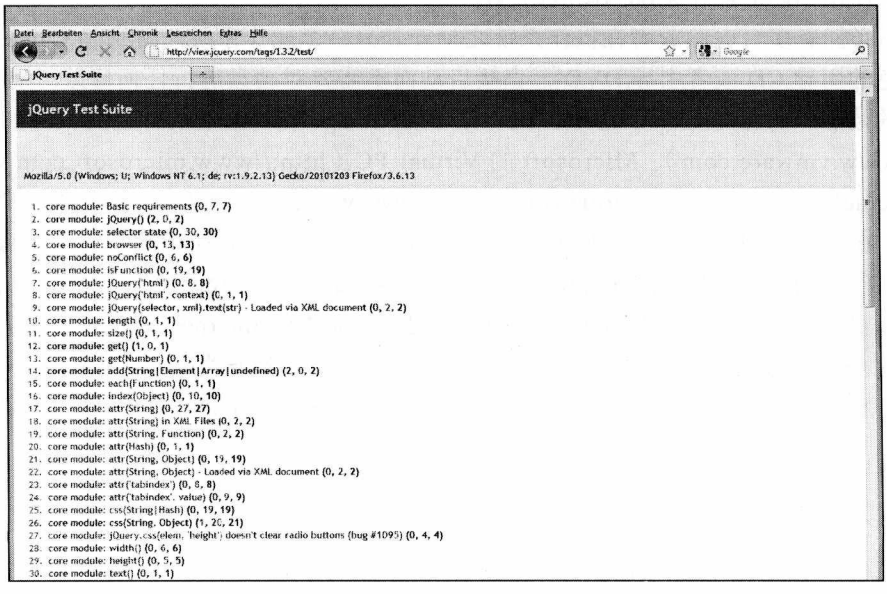

图 1-7 XAMPP 控制面板：Apache、MySQL 和 FileZilla 正在运行

提示

注意，在 XAMPP 下，指定路径时必须遵循互联网或者 Web 上常用的惯例。不能采用 Windows 下的方式（当然，这意味着从一开始就必须确保环境能够完全正常地工作，避免潜在的问题）。在 Apache 下通常是大小写敏感的。最佳的方法是在目录名和文件名上一致地使用小写。Windows 用户应该注意，不要在互联网上使用反斜杠分隔不同的级别，而应该使用正斜杠。

1.4.6 开发工具

我们几乎总是使用 HTML 或者 XHTML 框架作为 RIA 和任何网站的骨干部分。所以，为了创建 HTML 或者你使用的其他技术（如 CSS、JavaScript 等）的源文本，至少要有一个纯文本编辑器，任何操作系统都包含这种工具。

但是，在实践中可能使用更强大的编程工具来支持源文本的创建和分析。这些程序了解某些编程或者描述语言（例如 HTML、CSS 或者 JavaScript）的组成，支持简单（有时候甚至较为复杂）的标准处理，例如特殊字符的掩码（编码表现形式）、插入源代码模板或者通过对命令的颜色编码来提供更好的清晰度。有些编辑器还直接提供所使用语言的命令，这意味着编程人员可以使用菜单或者工具栏选择（有时候使用鼠标）。例如，Notepad++（http://notepad-plus-plus.org/）是一个出色的编辑器，它就提供上述的功能。

有些程序提供另一种功能——不同的文档视图。这常常出现在纯 HTML 编辑器上。然后，可以选择在网站的预览（就像在浏览器中看到的）、图形编辑模式以及 HTML 代码视图之间切换。

即使在 Web 编程中，现在也可以使用一些合适的集成开发环境（IDE）。这些开发环境可以从一个集成的公共界面中编程和执行。Aptana（http://aptana.com/）是一个免费且非常强大的 IDE。它直接基于强大的 Eclipse 开发环境（http://www.eclipse.org）。Aptana 提供了一个源代码编辑器，并且拥有许多直接支持 JavaScript、HTML 和 CSS 的特性，以及对 DOM 对象模型甚至 AJAX 的访问能力。代码辅助（Code Assist）功能试图自动完成各种用户输入，并用语法高亮显示提示语法（用不同颜色高亮显示关键词和语法结构）。显示对象属性和方法的选项特别有趣。该软件甚至提供了一个 JavaScript 调试器。Aptana 还包含自己的小型 Web 服务器（Jetty），可以通过它测试 AJAX 应用而无须安装独立的 Web 服务器。如果用 Aptana 创建项目，就会看到各种流行的 JavaScript 程序库已经直接集成，包括 jQuery，但是互联网上通常有更新的版本。如图 1-8 所示。

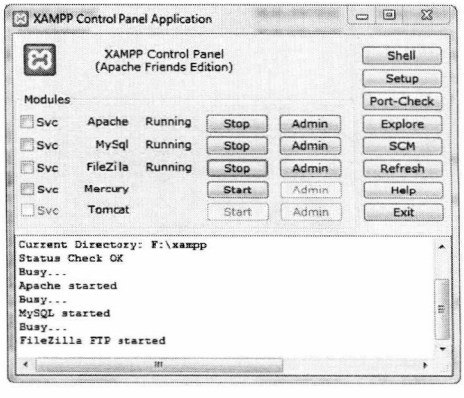

图 1-8 Aptana 直接支持 jQuery

提示

尽管 Aptana 中直接集成的 jQuery 版本不是最新的，也应该导入这个 jQuery 程序库（最近的版本，但是也可能已经过时）。甚至不需要使用这个过时的版本。相反，我建议单独下载最新的版本，在你的 Web 项目中使用。但是，导入 Aptana 自带的版本使你可以选择在 Aptana 中启用用于导入程序库的代码完成功能（如果只通过 HTML 将 JavaScript 文件集成到项目中，就不可能使用代码完成）。代码完成可能不是最新的，但是仍然很有帮助。

首先必须完成如下步骤，才能使用代码完成功能：

1. 通过 File（文件）> New（新建）创建一个新的 Web 项目。

2. 从可用 Web 项目类型列表中选择 Default（默认）。

3. 输入一个名称后，可以直接选择一个 jQuery 程序库（如果已经提供），也可以点击 Install JavaScript Libraries（安装 JavaScript 程序库）按钮。在下一个对话框中，选择 JavaScript 程序库下的 jQuery。在后续的相应对话框中，也要确保选中 jQuery。

4. 在 Window（窗口）> Preferences（首选项）之下，可以进入 Aptana Studio 类别，打开子类别 Editors（编辑器）然后选择 JavaScript。这里，可以在 Code Assist（代码辅助）之下选择 jQuery 代码完成。以后，这些选项在所有默认 Web 项目中都有效。换句话说，只需要完成这三步一次，而不需要对每个项目都这么做。如图 1-9 所示。

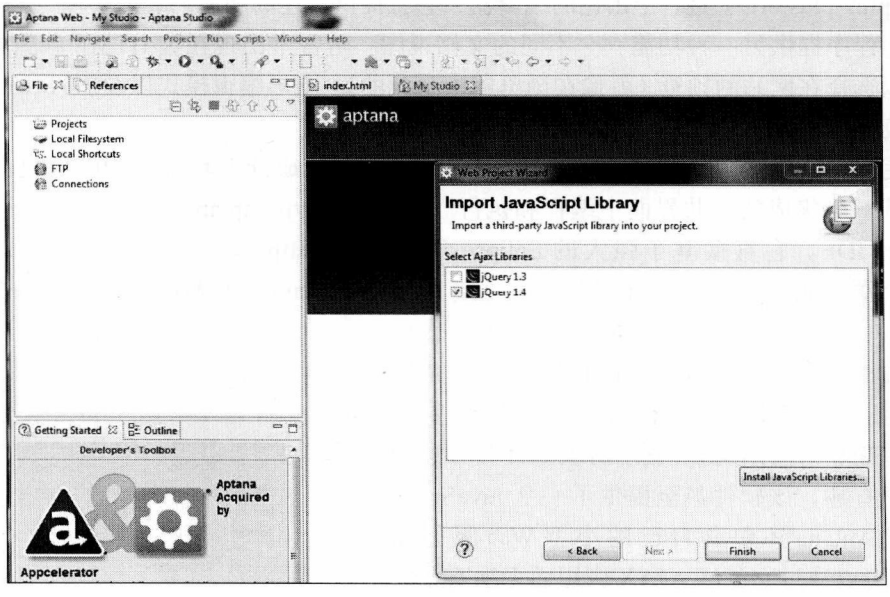

图 1-9 启用代码完成

注意

特定的编辑器或者 IDE 与本书无关，我主要使用 Aptana。

正如其他地方提到的那样，开发工具主要供应商也在它们的工具中集成 jQuery；例如，Microsoft 从版本 2008 开始就在 Visual Studio 中集成了它。对于 Web 应用的创建，集成到 Visual Studio 中的 Web Developer 特别有趣。如果你有 ASP.NET 的背景，Visual Studio（及其免费的 Express 版本，例如 http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products 下的 2012 版本）是出色的开发环境。

还可以用许多扩展程序（加载项）扩展 Firefox 浏览器，为 Web 开发提供许多有用的特性（例如，DOM 检查器、Firebug、Live HTTP headers 和 Web Developer）。安装之后，可以在 Extras 菜单中找到这些加载项。安装 Firefox 扩展的最简单方式是进入 Mozilla 项目的下载页面（https://addons.mozilla.org/），在搜索字段中输入想要的加载项，然后点击浏览器的安装超链接。

1.5 关于作者

在本章将要结束的时候，我做一下自我介绍。你已经从本书的封面或者前言中看到了我的名字，但是为了礼貌起见，请允许我再一次介绍自己：我的名字是 Ralph Steyer。我曾经就读于德国法兰克福 / 美因大学的数学专业（并获得学位）。然后，我在莱茵 - 美因地区的一所大型保险公司就职多年，担任程序员和概念性项目成员，开始时使用 Turbo Pascal，后使用 C 和 C++。4 年之后，我花费 1 年的时间进行 MVS 主机数据库的数据库设计工作。这段经历是我成为自由职业者的最大动因，因为我发现自己不想长期从事这项工作。

从 1995 年起，我成了自由职业者，在技术作家、专业记者、IT 讲师、顾问和程序员之间不断转换角色。除了这些岗位之外，我有时候在 Web 会议上发表演讲，在各种学院和大学授课，偶尔翻译专业书籍和录制在线培训视频。按照我的看法，这是个很好的组合，保持了我对专业的兴趣，并使我靠近实践和开发的第一线。特别是，我享受着不断更新 IT 开发知识的欢乐和压力，因为电脑知识的半衰期非常短。相应地，我的工作有时候很疲劳，但是始终激动人心。

1.6 小结

在本章中，你已经知道谁将通过本书为你提供指导，本书的内容是什么，目标读者又是谁（还有其潜在的结构）。现在，你知道需要满足什么条件，才能创建基于 jQuery 的 RIA，以及接下来我们要做什么。

第 2 章

jQuery 的第一批示例

本章中，我们和 jQuery 第一次接触，而不作任何进一步的准备。换句话说，我们直接深入 jQuery 之中。我渴望你能对 jQuery 可以做什么以及从这个框架中可以得到什么有感性认识。你只需要知道一点：源文本中的许多问题现阶段还不得不保留。但是不要担心，这些问题在随后的几章中就会得到解答。在这个阶段，对程序清单的解释也还比较肤浅，这是为了避免离题。我们希望尽快地进入 jQuery 的实际应用，进行一些有趣的尝试，这意味着创建示例。

注意

对于本章中的例子和后续章节的大部分例子，使用哪个版本的 jQuery 都没有关系。本书的例子用 jQuery 1.8.2 或者更新的版本创建，但是 1.3 或者 1.4.1 之后的版本往往就已经足够。

2.1 元素访问和 DOM 保护

如果对 Web 编程已经有了一些基础认识[⊖]，你已经知道，可以通过 JavaScript 或者其他脚本语言，在浏览器中通过名为文档对象模型（Document Object Model，DOM）的对象模型访问网页组件。对于这类访问有多种标准技术[⊜]，每种技术各有其弱点。

特别是，在访问网页的一个元素（或者一组）时，通常必须输入许多字符。这需要很大的工作量，也容易出错。因此，大部分框架提供了一个系统，使这种访问可以通过简短、统一的方法进行。底层机制的增加弥补了标准访问方法的各种弱点，首先是弥补了浏览器相关的特殊性，补充了纯 DOM 概念所缺失的各种功能。特别重要的是，这些补充功能通常都在所有官方支持的浏览器上进行过测试，因此能够相当可靠地工作。

⊖ 考虑到本书的目标读者，我假定你有这些知识。

⊜ 例如，getElementById() 和 getElementsByTagName() 方法可以通过对象字段或者名称访问。

下面的例子演示了 jQuery 另一个极其重要的功能——保护 DOM。后面的章节对此还有更多解释。现在，我们所要说明的是，不同浏览器在加载（解析）页面时对网页的访问方式不同，这可能在访问网页元素时造成许多问题（特别是如果你想在脚本中立即访问网页元素——换句话说，在浏览器正确构造 DOM 之前）。这里，jQuery 提供了一个可靠的方法来应对这些问题。

这个例子还顺便说明了如何使用 jQuery 作为访问带有文本的元素内容以及事件响应的标准手段。介绍已经够多了，下面看看我们的第一个程序清单（ch2_1.html）：[⊖]

程序清单 2.1 第一个 jQuery 示例

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="Content-Type"
     content="text/html; charset=utf-8" />
   <title>The first jQuery example</title>
   <script type="text/javascript"
       src="lib/jquery-1.8.min.js"></script>
   <script type="text/javascript">
     $(document).ready(function(){
       $("#a").click(function(){
         $("#output").html("Boring :-(");
       });
       $("#b").click(function(){
         $("#output").html("A nice game :-)");
       });
       $("#c").click(function(){
         $("#output").html("A strange game. " +
              "The only winning move " +
             "is not to play.");
       });
     });
   </script>
 </head>
 <body>
   <h1>Welcome to WOPR</h1>
   <h3>Shall we play a game</h3>
   <button id="a">Tic Tac Toe</button>
   <button id="b">Chess</button>
   <button id="c">
     Worldwide Thermonuclear War</button>
   <div id="output"></div>
 </body>
</html>
```

⊖ 这里引用了电影《战争游戏》——最早的黑客影片之一。强烈建议你在下次电视台播放的时候观看。

只需要在一个独立的目录中创建这个 HTML 文件，保存为上述的名称。

在实践中，通常将项目中的所有资源保存在一个独立的目录中。对于 Web 项目，最好的方案是在 Web 服务器上的一个共享文件夹中创建这些目录。在 Apache/XAMPP 中，这个目录通常是 htdocs。这样做的好处是——如果 Web 服务器正在运行——可以直接通过 HTTP 和对应的 Web 调用进行测试，而不是通过 FILE 协议将文件加载到浏览器中（换句话说，经典的“以文件方式打开”，或者简单地将文件拖到浏览器中）。后者不是符合实际的测试，因为以后这些页面还是必须由访问者通过 Web 服务器请求。

如果使用 Aptana 或者 Visual Studio Web Developer 等集成开发环境（IDE），通常可以直接从 IDE 通过集成的 Web 服务器显示网页。在 Aptana 中，这通过 Run 命令完成，在 Web Developer（Firefox 加载项）中，可以使用快捷键 Ctrl+F5。

注意

*在本书中，所有示例按照章节排序，并相应地在配套网站上列出（*http://jquery.safety-first-rock.de）。

在第 7 行和第 8 行中，可以看到对一个外部 JavaScript 文件的引用——在这个特殊的例子中，jQuery 程序库保存在网站所在项目目录的 lib 子目录中。如图 2-1 所示。这种结构在实践中已被广泛接受。这意味着，jQuery 程序库也必须在那个位置。当然，可以选择使用不同的路径结构。

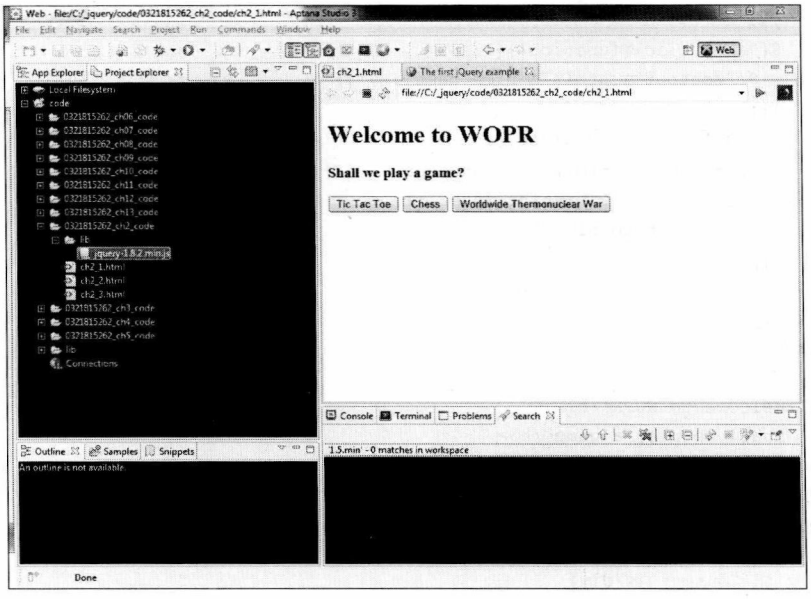

图 2-1 在这个项目中，从网站的角度看 jQuery 程序库位于 lib 目录

第 9 ～ 23 行包括一个常规的 JavaScript 容器。其中，网页用 $(document)（第 10 行）引

用。$() 函数是一个简写记法，用于引用网页的一个元素。你还将在第 11、12、14、15、17 和 18 行中看到这些简写访问标记。但是在这些行中，使用元素 ID 作为参数。

注意

作为 $() 参数的元素（从 jQuery 方面）没有使用引号，而 ID（或者另一个选择器）使用引号。

现在我们来简单地看看第 10 行开始、第 22 行结束的 ready() 方法。这个方法确保所包含的调用只在网页完全加载、DOM 正确构造时才执行。正如前面的提示所述，这是一个价值不可估量的功能，在此不再赘述。

注意

对于具有相应知识和经验的读者，ready() 方法是在网页主体中以 HTML 或者对应 DOM 对象的 JavaScript 中编写的 onload 事件处理器的一个替代品。但是，onload 事件处理器被认为极其不可靠，因为它在各种浏览器中的实现都有不足之处，最好是尽可能避免使用它。

在 ready() 方法中，3 个事件处理器各指定了点击所列元素时的反应。在我们的例子中，这些元素是 3 个用独特 ID 标记的按钮。

注意

click() 方法封装了 onclick 事件处理器的函数调用（你可能已经猜到）。

正确的函数分配通过 ID 和 click() 方法中函数的触发实现。注意，我们在这里使用的是匿名函数（没有标识符）。

用户点击某个按钮时也很有趣。这会在网页的某个部分上显示一个特定的文本输出。我们再次使用了 $() 和 ID，来选择这一部分（一个 div 块），用 html() 方法访问内容。如图 2-2 所示。

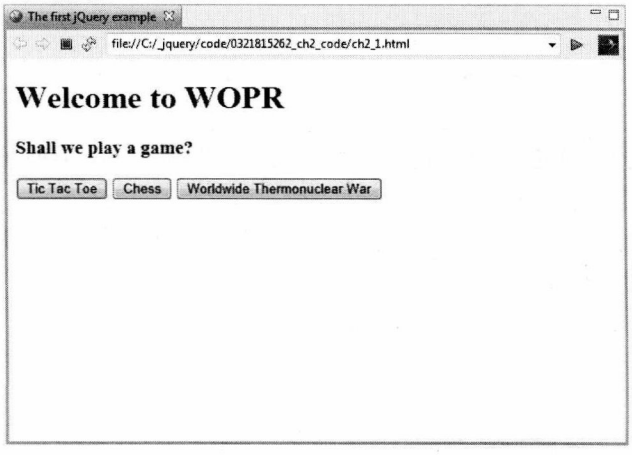

图 2-2 有 3 个按钮的网页；用户刚刚点击了第 3 个按钮

注意

在后面的所有示例中，我们省略了 DOCTYPE 语句的编写或者使用。为了完整起见，确实需要这条语句，但是省略它对我们没有任何影响，而且因为这条语句总是相同的，一次又一次写下它只是对本书篇幅的浪费。在配套网站上的例子中，这条语句被包含在内，因为它形成了正确标准的一部分。

2.2 用 jQuery 风格的 DHTML 编辑网页

一般来说，可以使用样式表而非纯 HTML 设计更美观、更高效的网页视觉外观。特别是，它们能够更简单地分离网站的布局和结构。这没有错，但是对你来说可能已经是老生常谈。

如果现在通过 JavaScript 动态地改变网站的样式表，我们所要讨论的就是动态超文本标记语言（Dynamic Hypertext Markup Language，DHTML）。但是，通过其他 JavaScript 技术显示和隐藏网页的一部分也是这种语言的一部分。在下面的例子中，我们来看看如何用 jQuery 快速、简单、方便且可靠地在不同浏览器中进行动画式的网页变动。在这个例子中，我们动态地改变一个元素的层叠样式表（Cascading Style Sheets，CSS）类。

首先，我们来看一个小型的 CSS 文件，这个文件应该整合到后面的网页中，保存在 lib 目录（ch2_2.css）：

程序清单 2.2 具有外部样式表的文件

```
01 body {
02  background: lightgray;color: blue;
03 }
04 div {
05  background: white;font-size: 14px;
06 }
07 .mClass {
08  background: red; color: yellow; font-size: 24px;
09 }
```

这个 CSS 文件没有太多内容。它确定整个网页以及所有 div 类型元素的背景和前景颜色，以及 div 类型元素的字体大小。

有趣的地方主要是第 7 ～ 9 行描述的类，在加载如下网页时它还没有得到使用，但是在用户操作时被动态分配（ch2_2.html）。

程序清单 2.3 改变 CSS 类

```
01 <html xmlns="http://www.w3.org/1999/xhtml">
02  <head>
03  <meta http-equiv="Content-Type"
04    content="text/html; charset=utf-8" />
```

```
05 <title>The second jQuery example</title>
06 <link type="text/css" rel="stylesheet"
07   href="lib/ch2_2.css" />
08 <script type="text/javascript"
09   src="lib/jquery-1.8.2.min.js"></script>
10 <script type="text/javascript">
11   $(document).ready(function(){
12     $("#a").click(function(){
13       $("#c").addClass("mClass");
14     });
15     $("#b").click(function(){
16       $("#c").removeClass("mClass");
17     });
18   });
19 </script>
20 </style>
21 </head>
22 <body>
23  <h1>Editing Style Sheets with jQuery</h1>
24  <button id="a">Add CSS class</button>
25  <button id="b">Remove CSS class</button><hr/>
26  <div id="c">He who knows all the answers
27    has not been asked all the questions.
28  </div><hr/>
29  <div id="c">Be not afraid of going slowly,
30   be afraid only of standing still.</div>
31 </body>
32 </html>
```

在这个例子中，你可以看到一个标题下有两个按钮，在一个 div 元素中有由分隔符分开的两段文本。这是纯粹的 HTML。在第 6 ～ 7 行中加入了指向 CSS 文件的链接。加载后的网站如图 2-3 所示。

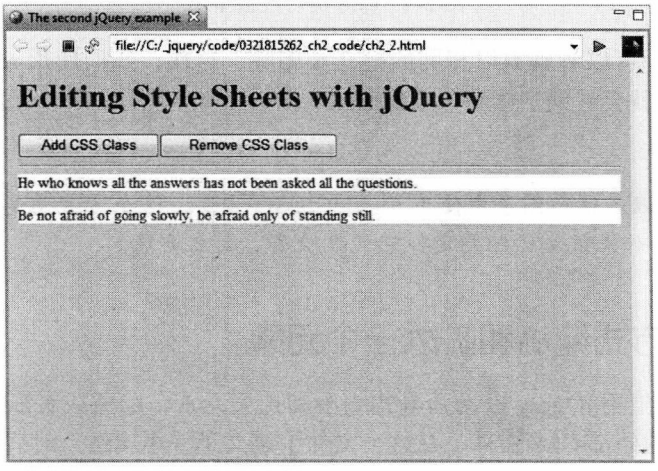

图 2-3 加载后的网站

但是现在我们打算用 jQuery 操纵按钮或者第一个 div 容器下方的文本。这就是 div 容器拥有一个 ID 的原因。容器下方的文本用于比较的目的。

为了访问网页的元素，这个例子使用第一个例子中提到过的 jQuery 机制。我们仍然使用 click() 方法响应按钮的点击，目前没有任何新的内容。

现在，你应该注意到我们尚未在加载网页时为元素指定链接的 CSS 文件中的 CSS 类。但是看看第 13 行。

程序清单 2.4 添加一个 CSS 类

```
$("#c").addClass("mClass");
```

方法 addClass() 的名称已经说明，调用这个方法会将样式表类分配给前述的元素。这是动态发生的，网页无须重新加载。该功能在用户点击对应按钮时触发，正如你在 click() 方法中所看到的那样。指定 CSS 类后如图 2-4 所示。

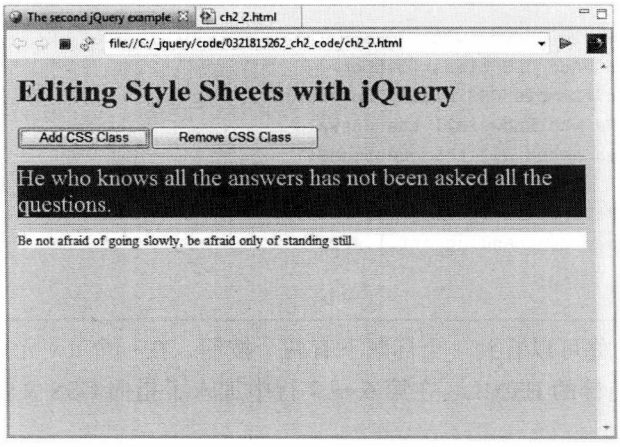

图 2-4 CSS 类已经被指定

在第 16 行中，可以看到类以相同的模式被删除。这次，我们使用 removeClass() 方法。如果测试该示例，就会看到字体和背景相应改变。

提示

作为替代，可以在这个例子中使用 toggleClass() 方法。这个方法总是根据状态删除或者添加一个 CSS 类。如果该类已经指定，它就删除类，反之亦然。

2.3 以动画方式缩小和扩大一个元素

现在，我们希望用 jQuery 以动画方式缩小和扩大元素，隐藏或者显示它。首先，我们看看 lib 子目录中的外部 CSS 文件，在这个文件中定义了一个属性，对后续的动画有特定的影响（ch2_3.css）：

程序清单 2.5 CSS 文件

```
01 body {
02  background: lightgray;color: blue;
03 }
04 #i1 {
05  width:300px; height:225px;
06 }
07 #i2 {
08  height:225px;
09 }
10 #h2{
11  background: white; color:#0000FF; font-size: 18px;
12 }
```

对于后续的示例，有趣的是第 5 行中的宽度数据。ID 用作引用一个图片的选择器。宽度规格影响后续的动画的类型。但是，我们首先要看看网页本身。它包含两个图片和下面的一些文本，如图 2-5 所示。我们希望生成所有 3 个元素的动画（ch2_3.html）。

程序清单 2.6 缩小和扩大两个图片和一些文本

```
...
06    <link type="text/css" rel="stylesheet"
07      href="lib/ch2_3.css" />
08    <script type="text/javascript"
09      src="lib/jquery-1.8.min.js"></script>
10    <script type="text/javascript">
11      $(document).ready(function(){
12        $("#toggle1").click(function(event){
13          $('#i1').slideToggle('slow');
14        });
15        $("#toggle2").click(function(event){
16          $('#i2').slideToggle('slow');
17        });
18        $("#toggle3").click(function(event){
19          $('#h2').slideToggle('slow');
20        });
21      });
22    </script>
23  </head>
24  <body>
25    <h1>Animated showing and hiding
26   of an image and text with jQuery</h1>
27    <button id="toggle1">Toggle Image 1</button>
28    <button id="toggle2">Toggle Image 2</button>
29    <button id="toggle3">Toggle Text</button><hr/>
30    <img src="images/i1.jpg" id="i1" />
31    <img src="images/i2.jpg" id="i2" /><hr/>
32    <h2 id="h2">A ski jump</h2>
33  </body>
34 </html>
```

这一动画的核心是 slideToggle() 方法。这个名称很能说明问题。可以使用这一特效根据当前状态显示或者隐藏对象，或者缩小和放大它们。换句话说，当前状态切换到相反的状态。可以在第 13、16 和 19 行中看到它的应用。

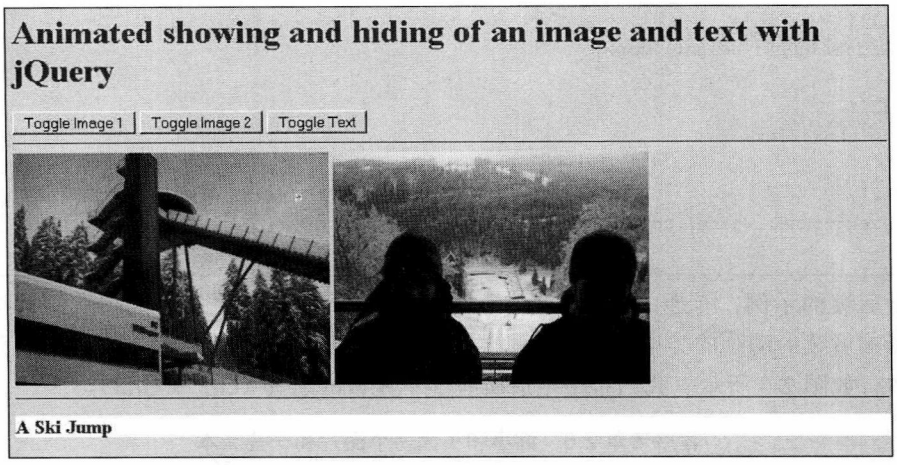

图 2-5 原来的外观

提示

你可能已经看到，方法的一个参数是临时的时间间隔。它确定动画花费的时长。在 jQuery 中，可以在大部分动画中传递这样的速度参数。允许的参数有 slow、normal、fast，也可以指定用毫秒表示的时间。以毫秒表示的时间不需要使用引号。

如果重新构造第一个图片的动画，你将会发现缩小图片导致图片高度减小，而后图片会完全消失。反之亦然，如果扩大图片，图片也以这种方式开始扩大。如图 2-6 所示。

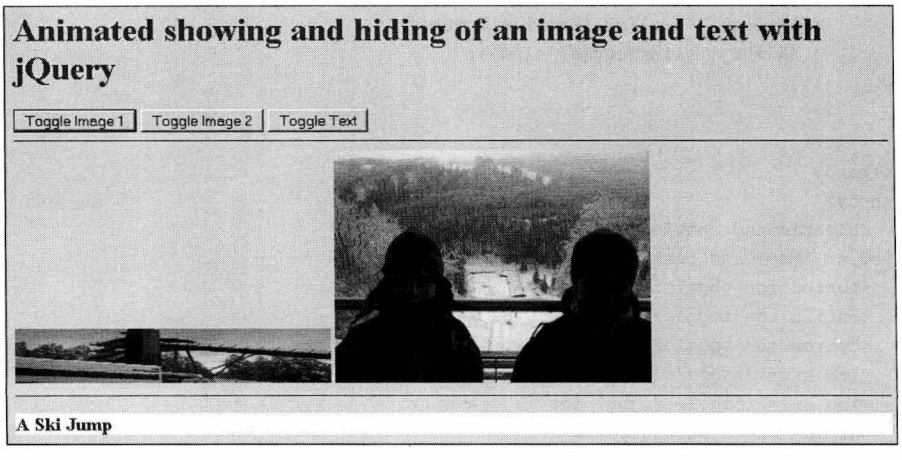

图 2-6 第一个图片被压扁

这种行为的重要原因是该图片的宽度是通过 ID i1 的 CSS 的规则制定的。这可以避免宽度减小，第二个图片没有指定宽度，就出现了这种情况。你将会看到在减小图片时，它缩小到图片的左下角然后完全消失。反之亦然，图片扩大时也以这种方式向右上角扩大。如图 2-7 所示。

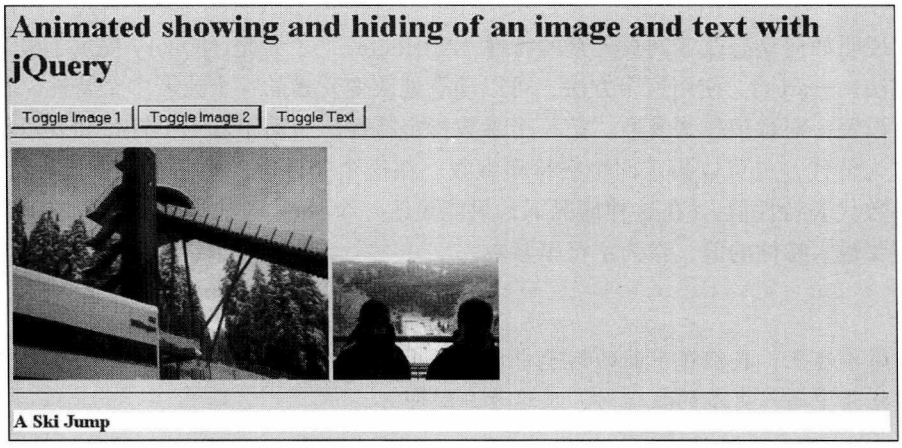

图 2-7 第二个图片以动画方式在宽度和高度两个方向缩小

现在观察一下，如果点击第三个按钮会发生什么。标题也将消失，但是只有高度发生变化。

至于 slideToggle() 的作用，重要的是应用动画技术的元素类型，过去应用到元素的 CSS 规则也扮演着某种角色。

上例中的动画实际上相互独立。如果设置的动画运行时间间隔足够长，可以让动画并行运行。3 个元素并行的动画如图 2-8 所示。

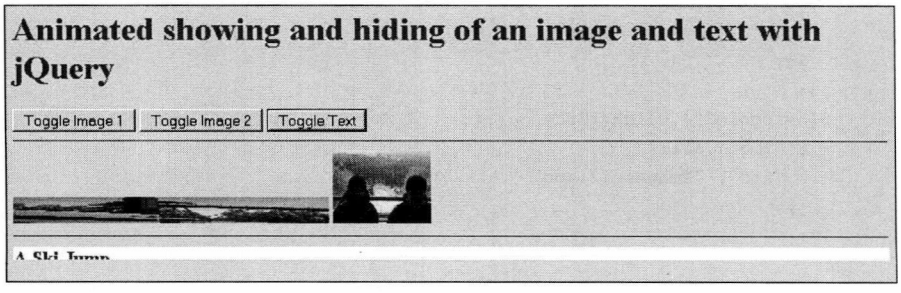

图 2-8 3 个元素并行动画

在那种情况下，要注意如果前面的元素完全消失（实际上，它被从文本流中删除），下方的内容会上移或者在水平方向重新定位。

但是，如果多次点击同一个按钮会发生什么呢？答案可能令人吃惊：这些事件被累积起来。这意味着，它们将连续执行，下一个事件只在前一个事件完全处理之后才执行。所以，

再次点击按钮不会导致当前动画停止，下一个动画立即出现。如果想实现这样的功能，必须明确地编程。

2.4 动态改变特性

本节说明如何动态改变网页元素的特性（Attribute）[⊖]。为此，jQuery 提供了极其灵活和实用的方法——attr()。使用这个方法，可以动态地改变元素的一个或者多个特性。在圆括号中，可以设置一对数值作为参数。第一个参数指定特性，后面跟着一个冒号和表示新值的字符串。作为替代，也可以指定两个字符串参数。在这个变种中，第一个参数代表特性名称，第二个参数代表特性值。（在这种情况下，只能更改一个特性。）如果只想要请求一个特性的值，只需要输入特性的值，作为字符串参数。

注意

为了简单起见，我们在下面的例子中只修改一个特性。如果想一次性修改多个特性，只需要在圆括号中写入更多的数值对，用逗号分隔即可。

在我们的例子中，我们希望修改一个 <img> 标记的 src 特性，替换网页中的第一个图片（ch2_4.html）。

程序清单 2.7 修改某个元素上的特性

```
...
06    <script type="text/javascript"
07      src="lib/jquery-1.8.min.js"></script>
08    <script type="text/javascript">
09      $(document).ready(function(){
10        $("#toggle1").click(function(){
11          $("img").attr({
12            src: "images/i1.jpg"
13          });
14        });
15        $("#toggle2").click(function(){
16          $("img").attr(
17            "src", "images/i2.jpg"
18          );
19        });
20      });
21    </script>
22  </head>
23  <body>
24    <h1>Replacing an image</h1>
25    <button id="toggle1">Image 1</button>
```

⊖ Attribute 一词许多书籍都译作属性，本书中为了防止与 Property 混淆，均翻译成特性。——译者注

```
26    <button id="toggle2">Image 2</button>
27    <hr/><img src="images/i1.jpg"/>
28 </html>
```

我们通过圆括号中的标记法修改特性值，然后通过两个字符串参数修改特性值。如前所述，我们在两种情况下都替换 src 值。

2.5 小结

本章只提供了几个例子，但是它们很好地展示了 jQuery 的核心关键因素。特别是，大家应该记住 $() 函数和 ready() 方法。此外，指定响应的技术如 click() 方法也很重要。addClass()、toggleClass()、removeClass() 和 slideToggle() 等动画技术在后面的 DHTML 特效中也很有帮助。在本章中，我们还学习了修改特性值的方法（attr()）。我们将在本书的其他章节中更全面地介绍这些技术，并且更加深入地研究 jQuery 的总体概念。

第 3 章

基本知识

在前一章中，我们完成了几个 jQuery 示例。现在是时候转向基本知识了——还不是 jQuery 本身的细节，而是 jQuery 的使用环境以及这个框架在环境中的切入方式。这个环境就是使用超文本标记语言（HTML）、扩展 HTML（XHTML）、层叠样式表（CSS）、JavaScript、可扩展标记语言（XML）、JavaScript 对象标记法（JSON）和异步 JavaScript 及 XML（AJAX）的 Web。毕竟，和这一领域的许多其他框架和工具包一样，jQuery 除了独立的语法之外，还是一个为网站提供某些 CSS 特性（在 jQuery UI 中甚至有自定义 CSS 主题）和 AJAX 支持的 JavaScript 扩展。如果对底层技术没有必要的基本知识，就很难有效和正确地使用 jQuery。在本章的所有说明中，我们的重点当然放在如何将 jQuery 集成到自己的网页中，以及需要警惕的方面。我们就从这里开始，以便以后能够更加深入地探索 jQuery。

注意

本章无法（也不应该）对相关技术进行完整介绍。在这里只能找到与 jQuery 绝对相关的信息。如果需要其他信息，可以使用其他的资源找到更多信息。附录包含了与作为 jQuery 核心技术的 JavaScript 相关的更多信息。

3.1 互联网上的 Web、Web 2.0 和客户 / 服务器准则

首先，我们先来简单地了解一下万维网（WWW）；毕竟，我们已经知道了 WWW 的基本事实。Web 的核心基于超文本传输协议（HTTP），文档描述语言 HTML 及其严格的基于 XML 双胞兄弟 XHTML，加上 Web 服务器和 Web 浏览器程序。

正如互联网上的所有服务一样，Web 是一个经典的客户 / 服务器系统，每个操作包括请求服务和提供服务的一个周期。更确切地说，这意味着 Web 实际上总是处于浏览器（客户端）请求一个文件（通常是一个网页或者集成到网页中的内容）的情况，在某些情况下发出进一步的请求，用于该网站中引用的外部资源（例如，图形、视频、Flash 动画或者外部

JavaScript 或者 CSS 文件）。然后，这些内容在浏览器中和网页一起显示，或者进行其他的处理。核心控制机制总是（X）HTML 文件。

3.1.1 Web 上的编程

经过多年的发展，Web 已经成为一个可以在服务器和客户端上编程提供内容的系统。近年以来，对于客户端编程危险性的神经质看法已经平息，几乎所有现代化网站都使用客户端编程实现应该在客户端进行更为明智的业务逻辑。经过这么多年以后，客户端领域的代表性技术中唯一仍然存在的是 JavaScript，但是现在它已经得到了广泛的使用和接受。看看互联网上流行的网站，没有一个网站能够在没有 JavaScript 的情况下进行管理。Web 上几乎所有用户都在浏览器中启用 JavaScript。毕竟，谁不希望完整和无限地使用 Google、Amazon、eBay、Facebook、Twitter、Wikipedia 或者 Yahoo! 等流行网站呢？作为网站的创建者，我们也就可以假定大部分客户端都使用了 JavaScript 及其相关技术[⊖]，这意味着，反对使用 jQuery 或者 Dojo Toolkit、Prototype、YUI 等 JavaScript 框架的大部分证据已经不再有效。

注意

正如本书前面所述，各制造商都试图建立专利的客户端技术（如 Silverlight），以突破 JavaScript 和浏览器客户端的限制。但是对它们的广泛支持和接受尚待时日。现在，这些专利技术是否将会成功尚不清楚。

3.1.2 Web 2.0

正如 Web 巧妙而简单的概念，HTTP、HTML 和经典 Web 浏览器的概念中也有一个简单而严重的问题。当浏览器从 Web 服务器上请求新数据时，后者总是必须发送一个完整的网页作为响应。更准确地说，浏览器解释响应的方式就是用新内容完全替代浏览器以前显示的网站。明显，这样做的效率很低。这就是 AJAX 登上历史舞台的原因。（你将很快学到更多有关这一技术背景的内容。）

一般来说，这是确保应用（几乎）实时响应的一个过程，尽管新数据从 Web 服务器请求。AJAX 数据请求不生成一个原则上已经在浏览器中显示的完整网页，实际上 Web 服务器只发送新数据，然后使用 DHTML 方法将其“嵌入”已经在客户端加载的网页中。这种请求在加载新数据时甚至不会打断用户和 Web 应用程序的常规交互。因此，现在可以在 Web 上创建以用户交互为中心的网站。AJAX 现在已经成了交互性网站的标准过程，这一点特别要感谢 Google。大约从 2005 年起，Web 2.0 这一宣传语成了最具交互性网站的统称。Web 2.0 往往还被用来指“参与性”和“交互性”的 Web，因为用户不再只是消费者，还能够自行贡献内容。只要想想博客、Tweet、Wiki 或者 Xing、Facebook、MySpace 等社区就知道了。

⊖ 根据我所掌握的统计数据，JavaScript 的支持率有很大波动。有些统计数字推测使用率达 99.9%，有些则认为“只有”99.1%。我希望你意识到我的陈述中的讽刺意味，你现在可以假定使用率几乎是无限的。

但是即使用户在一个在线日历中输入数据，也将造成网站的不同表现（例如，活动安排被显示，当然也被保存在服务器上）。这也是参与 Web 2.0 的一种形式。

3.2 JavaScript 及其与 jQuery 的关系

因为 jQuery 基本上就是一个 JavaScript 程序库，我们当然需要更仔细地研究这种语言。尽管 jQuery 和其他各种框架的市场宣传声称它们能够为你做很多关于 JavaScript 的事情，但是要使用 jQuery，你应该具备 JavaScript 的基础知识。如果想真正有效地使用 jQuery（例如，创建插件），甚至应该对 JavaScript 有深入的理解。

注意

接下来的几章会带着你深入 JavaScript 的世界，这与 jQuery 有着密切的联系。现在，关于 JavaScript 的基本信息就已经足够。关于 jQuery 的更多基本概念和信息，请参见附录。

总的来说，Web 上的脚本语言是对 HTML 或者 XHTML 最重要的扩展，它们实现了 Web 应用的客户端逻辑，这些脚本语言都是解释型语言，在宿主程序（浏览器）中于运行时翻译执行。这一点特别适用于 JavaScript。

注意

新的浏览器有 JavaScript 的即时编译器。即时编译器提供了在内存中始终保存翻译后代码的选项，在重复执行时高性能更佳，从而扩展了 JavaScript 的解释方法。这对于试图达到和桌面应用相同性能表现的富互联网应用程序（RIA）是一个关键的因素。

本质上，jQuery 是一个 JavaScript 程序库。换句话说，jQuery 仅基于每个现代浏览器都提供的一个功能。不需要为浏览器提供插件或者其他扩展，就可以使用这个程序库的功能。所以，不用将它添加到浏览器甚至操作系统（与之竞争的技术也是如此）。这一方面是个很大的优势，但是另一方面，用这个程序库只能实现 JavaScript 或者动态 HTML（DHTML）、层叠样式表加上文档对象模型（DOM）操纵所能达到的功能。而且，为了使用这些简单的技术完成 jQuery 所提供的激动人心和强大的效果，这些技术必须被运用到极致。结果是，不是每个浏览器都能完全支持它（特别是旧的浏览器）。

如果你确实擅长使用 JavaScript 编程，你可以自己重现 jQuery 的所有功能。但是，这需要花费一些精力。jQuery 团队已经投入数年时间开发这一程序库，你在工作中可以从这一程序库得益。

网站中 JavaScript 的通用集成

JavaScript 被看成 HTML 或者 XHTML 的直接补充和扩展，意图是用作对应网站框架的一个组成部分。Web 脚本直接用普通文本直接写入网站，或者在运行时集成和解释。将脚本连接到网站的技术多种多样。我们来简单地看看其中两种技术。

网站中的 <script> 容器

JavaScript 和网站的连接可以通过网站中直接的 JavaScript 标记进行。JavaScript 语句简单地用普通文本写入对应的（X）HTML 文件中。脚本的开始用仍为 HTML 一部分的单独控制语句标记，这条控制语句与相匹配的结束标记形成了脚本命令的一个容器。通过 <script> 元素，可以指定一个封闭容器内的所有内容为脚本。在这样的脚本中，可以执行 JavaScript。

看看程序清单 3.1。

程序清单 3.1 网站上直接标记 JavaScript 的一个代码片段

```
<body>
...
 <script>
 ... script statements
 </script>
...
</body>
```

如果在脚本标记中没有指定语言，所有已知的浏览器都默认使用 JavaScript，在 Internet Explorer 中会使用它克隆的 JScript。但是，不应该利用浏览器的高宽容度，因为实际上应该指定所要使用的脚本语言。可以使用 language 或者 type 参数指定脚本语言。[⊖]

下面可以看到两个可互换使用的例子。

程序清单 3.2 可互换的脚本语言说明

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

指定 language 值时大小写没有关系，但是在通过 type 参数指定 MIME 类型[⊖]时必须使用小写。对于 language，还可以指定 JavaScript 的版本——例如，JavaScript 1.3 或者 JavaScript 1.5。只要将正确的版本号加在 JavaScript 标记之后即可——不需要空格分隔。指定版本号，可以使不知晓这一版本的浏览器忽略脚本块。

警告

不要同时使用 type 和 language。如果两者都指定，language 数据会被忽略。这意味着，想要通过 language 指定的任何版本都没有作用。

过去，我们习惯于在脚本容器中（<!--…--> 甚至 <!--…//--> 写入 HTML 注释，避免不知道 JavaScript 的浏览器简单地显示命令。现在，这已经完全没有必要了，因为这种浏览器在 1996 年左右就不再进入市场了，因而实际上不再存在。注意，这里指的不是浏览器禁用 JavaScript 的情况。

⊖ 官方的规范要求使用 MIME 类型。

⊖ MIME 类型确定内容的类型。它首先指定主类别（如 text 或者 image），然后（用斜杠分隔）是子类别，如 html、css 或者 javascript。

在网站中集成 <script> 元素时，应该将它放在哪里就成了一个问题。这个问题没有明确的答案。基本上，这个元素属于网站首部的内容。

但是在实践中，你将会在网站的任何位置找到 <script> 元素。原则上，这种元素甚至可以出现在网站代码之后。脚本语句在网站加载时由浏览器从头到尾进行解析。所以，在网站后面写入的脚本语句（函数）将在网站加载到所在位置时才可用。因此，常用的方法是将 <script> 容器放在网站主体之前，嵌入重要函数的声明。然后，这些函数可以用于整个文档。

实际上，很明显有许多情况，用 JavaScript 编程时 <script> 元素的位置对执行有很大的影响。例如，许多网页迫使你将 <script> 放在首部之外的区域。Google 或者 Microsoft、Ajax 框架中提出的高专业性和现代化的概念利用这一位置弥补不同浏览器的某些行为。

注意

记得上一个例子中的 ready() 方法吗？它弥补了这些问题，所以不再需要使用其他技巧。

外部 JavaScript 文件的基本信息

使用前面描述的方法，可以在（X）HTML 文件中直接编写脚本，但是对于 JavaScript，将脚本直接放在一个外部文件中通常是最可取的方法，至少在打算处理子程序（函数）和变量声明时是如此。外部文件集合函数和变量，然后形成一个可以连接到任何网页的程序库。这是我们在引用 jQuery 程序库时看到的情景。这种方法实现了结构和功能性的分离，在实践中极为重要。在理想情况下，除了 JavaScript 函数的显式调用和少数几条语句之外，外部 JavaScript 文件（按照一般的规则，扩展名为 .js）包含你的全部 JavaScript 代码。

为了将外部 JavaScript 文件集成到网页中，只需要为 <script> 标记添加 src 特性。这个特性指定外部 JavaScript 数据的 URL。URL 的一般规则适用于单独 JavaScript 文件的引用。

引用 jQuery 程序库

jQuery 程序库是一个外部 JavaScript 文件，程序清单 3.3 展示了 3 个可以互换的代码片段，它们是应用 jQuery 程序库外部文件的两种变形。

程序清单 3.3 集成外部 jQuery JavaScript 文件

```
01 <script language="JavaScript"
     src="jquery-1.8.2.min.js"></script>
02 <script language="JavaScript"
     src="lib/jquery-1.8.2.min.js"></script>
03 <script type="text/javascript"
     src="lib/jquery-1.8.2.min.js"></script>
```

警告

在任何情况下，应该留空 <script> 容器（不能包含任何空格和换行），因为有些浏览器在容器中有内容时会出现问题。（正式的做法是不能包含任何东西；它是个空元素）其他浏

览器会按照容错准则忽略其中的内容。但是不要在空元素中使用 XML 语法 <script…/>，因为这样可能会在 Internet Explorer 等浏览器中遇到问题（但这种语法完全是合法的）。

在例子里的第一个变种中，JavaScript 文件位于引用网页的同一个目录。第二种情况下，位于相对于网站目录的 lib 子目录中。在这两种情况下，language 都指定引用脚本的类型。

在第三种情况下，jQuery 程序库也来自子目录 lib（和第二种情况一样），但是引用脚本的类型用 type 指定。

通常，应该在网页的首部写入对 jQuery 程序库的引用。但是 jQuery 的官方文档指出，在某些情况下应该避免这么做。同样，如前所述，你将会发现打破这些理论性的规则在某些时候是有必要的。

具体地说，这适用于引用 jQuery 的整个页面通过 PHP 函数 include() 或者 require() 整合到一个 PHP 脚本的情况。文档状态只是“由于某种原因，从 <head> 标记中无法调用 jQuery 脚本文件”。换句话说，即使 jQuery 团队也不知道为什么有时候这样做无效。但是，这就是 Web 编程。由于浏览器的行为极其多变，以及 PHP 解释程序等服务器端处理，有时候必须使用技巧，打破规则，使用变通的方法，等等。Google 是寻找这类往往罔顾正规学说的实用解决方案的大师。而其他项目也很重视这一实用功能。当然，我们无法预测将引用放在首部之外的技巧对于 jQuery/PHP 的所有版本或者每个浏览器是否都是必需的。然而，如果发生问题，应该将引用移到网页主体中，然后再次尝试。Web 编程有时就是个反复试验的过程。即使使用一个复杂的框架，也改变不了这一事实。

3.3 AJAX 和 XMLHttpRequest（XHR）

如前所述，AJAX 描述了确保 Web 应用在 Web 服务器请求新数据的情况下也能（几乎）实时反应的一种方法。具体地说，AJAX 只是一些建立已久的技术的相互作用，从 HTML、XHTML 和 HTTP，到 JavaScript、CSS 甚至 XML 或者 JSON。异步请求附加数据集成到网页的技术理论上在 1998 年左右就已经出现，只有 2005 年出现的 AJAX 这一术语以及流行词 Web 2.0 相对新颖。

实际上，AJAX 在推出时是作为 JavaScript 对象模型的扩展。但是我相信，如果使用这种简单的市场战略，就无法取得像 AJAX 这种流行语一样的成功。不过，纯粹从概念考虑，可以这样描述 AJAX。

为了支持这种异步通信，现代浏览器提供了一个内建接口，控制来自独立于浏览器“常规”数据请求运行的客户端编程语言（主要是 JavaScript）HTTP 事务。这个接口采用 XMLHttpRequest 对象的形式，作为 JavaScript 对象模型的一个扩展。这些 XHR 或者 XMLHttpRequest 对象直接面向 HTTP 的内部结构，形成每个 AJAX 请求的骨架。它们也以某种形式存在于带有“AJAX”头衔的所有框架和工具包中，当然，这也包括 jQuery。jQuery 提供了一些非常好用的方法，从 jQuery 1.5 开始，甚至对该对象本身进行了扩展。

为了在浏览器和 Web 服务器之间进行异步通信，XHR 对象使用函数引用，允许注册回

调函数，这些函数在每次事务状态变化时解释运行。

而且，可以使用 XMLHttpRequest 对象访问 AJAX 请求或者响应的所有 HTTP 头标字段。

除了普通文本之外，AJAX 请求的数据主要是 XML 和 JSON 数据。我不能肯定你了解这方面的所有知识，所以将对最重要的细节做简单的说明。

3.3.1 XML

XML 描述了一种基于 Unicode 的平台无关纯文本标准，用于创建机器和人类可理解的文档，交换任何类型的信息。XML 文档采用树型结构，允许浏览树的单个分支。正如 HTML 一样，XML 是一种提供信息（不仅是文本信息）结构的标记语言。文档中包含的信息通过标记构造，这些标记在 HTML 和 XML 中都包含在尖括号中。和 HTML 不同，XML 中的元素不是预先确定的，没有确定的有限元素词汇表，也没有像 HTML 标记那样的预定义 XML 标记。XML 只是元素语法和结构的描述，可以无限扩展。XML 规范只描述了定义标记时所遵循的规则。

和 HTML 相反，XML 在语法上是严格的，不存在容错原则（这是 HTML 的核心特征）。XML 规范很严格，不允许任何异常或者不清晰的结构。但是这也意味着，XML 很容易进行自动化校验和解释。所以，XML 注定是为数据交换而生的。

XML 只描述少数简单而严格且绝对清晰的规则，说明文档如何组合。

XML 文档的组成部分称为组件（Component）。XML 文档的基本结构由元素组成，如果没有受到限制，元素本身可以包含子元素，代表着组件的最重要形式。从 HTML 中，我们已经了解元素本身的结构。开始标记如下：

程序清单 3.4 XML 中的开始标记

```
<rjs>
```

XML 的开始标记总是需要结束标记——在斜杠之后重复标识符，除非标记被标识为空元素。开始标记中的任何特性在结束标记中都不再重复，在我们的例子中，结束标记如下所示。

程序清单 3.5 与开始标记匹配的 XML 结束标记

```
</rjs>
```

你应该从 HTML 中已经熟悉了所有这些标记，只是在 HTML 中不能自由地选择元素。

在元素里，可以写下任何内容——只要没有应用特殊的规则。这可能包含其他元素、文本或者两者的组合。

XML 允许声明空元素。这些元素主要与特性组合使用，空元素的定义如下。

程序清单 3.6 一个带有特性的空元素

```
<rjs url="www.rjs.de" />
```

也可以用下面的形式代替。

程序清单 3.7 空元素的另一种标记法

```
<rjs url="www.rjs.de"></rjs>
```

容器甚至不能包含空格。空格在 XML 中被看作正常内容，因此是树中的一个节点。遗憾的是，某些浏览器或者底层 XML 解析器没有考虑这一点。

XML 文档必须遵循的语法规则不多，但是非常严格。如果文档满足这些规则，就被称作“良好格式”(well-formed)。

XML 文档基本上由 Unicode 符号（16 位字符集）组成。但是可以在编辑器中创建 XML 文档时保存 ANSI 代码。然后，解析器可以将其翻译为 Unicode。

在 XML 中，大小写之间存在严格的区分。

XML 文档总是以一个序言（Prolog）开始。序言必须出现在 XML 文档的开始，最简单的序言如下所示。

程序清单 3.8 简单的 XML 序言

```
<?xml version="1.0" ?>
```

XML 文档可能只包含一个根元素（也称作根标记或者文档元素）。这个根元素在任何情况下都必须存在，紧跟在序言之后，与注释分开。

元素的嵌套必须清晰。

每个元素都有一个结束标记或者被写作空元素。

特性必须赋值，特性值必须包含在引号内。

程序清单 3.9 展示了一个典型的 XML 文件（rjs.xml）。

程序清单 3.9 一个典型的 XML 文件

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <rjs>
03   <business>
04     <name occupation="Dipl Math"
05       company="RJS EDV-KnowHow">Ralph Steyer</name>
06     <location>Eppstein</location>
07     <location>Bodenheim</location>
08   </business>
09   <websites>
10     <url>www.rjs.de</url>
11     <url>blog.rjs.de</url>
12     <url>www.ajax-net.de</url>
13   </websites>
14 </rjs>
```

任何浏览器都能显示 XML 文件。通常，浏览器以树形结构显示文件。通常可以点击树中的敏感元素，展开和收起分支。如图 3-1 所示。

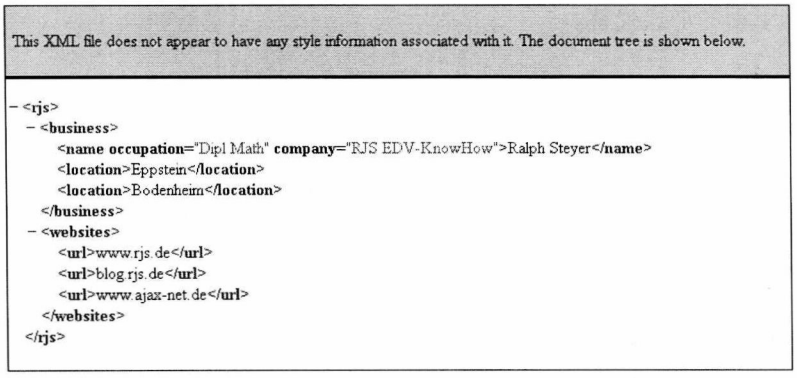

图 3-1 大部分浏览器将 XML 文件显示为交互式的树；下方的分支被收起

注意

为了简洁起见，任何 XML 文件都可以看作 DOM。这意味着，我们可以任意使用所有 DOM 方法和属性，包括 jQuery 方法。如果熟悉 DOM 的概念，就已经掌握了这一条。如果还不熟悉，可以在阅读 DOM 概念的小节时学到更多。

3.3.2 JSON

通过 AJAX 请求附加数据时，XML 能够将许多逻辑转移到传输格式中，但是对于 AJAX 请求来说往往过大，非常难以操纵。而且，在不同浏览器的处理中可能出现各种问题，但是 jQuery 已经弥补了大部分这类问题。不过，如果纯文本提供的逻辑不足，XML 又过于复杂和笨重，在两个极端之间还有一种替代方案。利用 JSON，我们就有了一种比 XML 更简单的传输格式结构，而且在不同 Web 浏览器中能够得到更加一致的处理。

和 XML 相同，JSON 是一种机器可以理解的纯文本数据交换格式，灵活性远低于 XML 但是更为紧凑。JSON 包含直接基于 JavaScript 的数据结构或者数据类型（对象、数组、字符串、数字、布尔值 true 和 false，以及 null 值），可以嵌套，在必要时可以用空格字符构造。

JSON 概念基于两个核心结构：

- 名称 – 值配对：这种组合可以在几乎所有现代编程语言中找到。它们通常被实现为对象、记录、结构、字典、哈希表、有键值列表或者关联数组。
- 和大部分语言中类似的有序值列表：根据编程语言，通过数组、向量、列表或者序列实现。

JSON 的基本概念是全局数据结构的描述，在所有现代编程语言中都有某种形式的支持。根据这些结构，JSON 提供了在编程语言之间交换数据的一种格式。特别是，JSON 可以直接由各种编程语言处理（例如，在 JavaScript 中通过 eval() 函数处理）。

JSON 中格式定义的具体语法是——正如名称所表示的——基于 JavaScript 的，但是对

象或者数据字段没有标识符：

- 对象定义以波形括号（{）开始，以右波形括号（}）结束。这样的对象块可能包含一个以逗号分隔的下级属性列表。
- 属性包含一个键和一个值。两者都以冒号（：）分隔。键是一个字符串，相关的值可以是任何数据类型（对象、数组、字符串、数字，或者表达式 true、false、null）。
- JSON 中的数组以左方括号（[）开始，以右方括号（]）结束。数组可以包含一系列以逗号分隔的值，它是一个有序列表。
- JSON 中的字符串和平常一样，以引号开始和结束。它可以包含任何 Unicode 字符和转义序列。JSON 中的字符串和 C 或者 Java 中非常相似。
- 数值就是数字 0 ～ 9 的一个序列。可以使用正负符号、小数点和幂次（e 或者 E）。JSON 中的数字和 C 或者 Java 中的数值很相似。但是有一个根本的不同。在 JSON 中，不能使用八进制或者十六进制数字形式。
- 布尔值和平常一样，以 true 或者 false 表达式代表。

下面是简单 JSON 文档的一个例子，信息内容和结构基础[⊖]对应于前面的 xml 文件（rjs.json）。

程序清单 3.10 JSON 结构

```
01 {
02   "business" : {
03     "name":"Ralph Steyer",
04     "company":"RJS EDV-KnowHow",
05     "occupation":"Dipl Math",
06     "location":["Eppstein","Bodenheim"]
07  } ,
08  "websites" : {
09    "url":["www.rjs.de","blog.rjs.de","www.ajax-net.de"]
10  }
11 }
```

JSON 的好处是这种语言确实简单，用上面的简短解释就能够说清楚。如果需要进一步的信息，可以查看 http://json.org/index.html。

3.3.3 关于 JavaScript 程序处理 JSON 的更多细节

对于了解 JavaScript 的读者来说，下面的例子将演示如何通过 JavaScript 解析 JSON。我们假定在 JavaScript 中已经有了一个 JSON 对象，我们获得一个包含 JSON 结构的字符串，但是必须首先将其转换为 JSON 对象。例如，如果我们通过 AJAX 请求 JSON 数据（在这个例子中没有这么做），情况就是如此。

⊖ 结构不是 100% 相同，但是大体如此。XML 文件中的属性必须采用不同的表现形式才能使结构 100% 相同。但是我们对这些细节不感兴趣。

程序清单 3.11 显示 JSON 数据的 HTML 页面（ch3_1.html）

```
...
06    <script type="text/javascript"
07       src="lib/ch3_1.js">
08    </script>
09  </head>
10  <body>
11    <h1>Processing JSON</h1>
12    <script type="text/javascript">
13      process_json_text();
14      process_json_object();
15    </script>
16  </body>
17 </html>
```

这个 HTML 页面是无害的。除了对外部 JavaScript 文件的引用和调用其中声明的函数——第 13 行和第 14 行的 process_json_text() 和 process_json_object()，就没有什么特别的了。我们简单地在网页中写入两个函数。

我们来看看引用的 JavaScript 文件 ch3_1.js。

程序清单 3.12 引用的 JavaScript 文件

```
01 function process_json_text(){
02   var JSONText = '{ "business" : {' +
03     ' "name" : "Ralph Steyer",' +
04     ' "company" : "RJS EDV-KnowHow",' +
05     ' "occupation" : "Dipl Math",' +
06     ' "location" : ["Eppstein","Bodenheim"] } ,' +
07     ' "websites" : { "url" : ["www.rjs.de",' +
08     ' "blog.rjs.de", "www.ajax-net.de"] }}';
09   var JSONObject = eval('(' + JSONText + ')');
10   output_all(JSONObject);
11   output_specific(JSONObject, name);
12 }
13
14 function process_json_object(){
15   var JSONObject = {
16     "business": {
17       "name": "Ralph Steyer",
18       "company": "RJS EDV-KnowHow",
19       "occupation": "Dipl Math",
20       "location": ["Eppstein", "Bodenheim"]
21     },
22     "websites": {
23       "url": ["www.rjs.de", "blog.rjs.de",
24               "www.ajax-net.de"]
25     }
26   };
27   output_all(JSONObject);
```

```
28   output_specific(JSONObject);
29 }
30
31 function output_all(JSONObject){
32   document.write("<table><tr><th>Key</th>" +
33        "<th>Value</th></tr>");
34   for (i in JSONObject.business)
35     document.write("<tr><td>" + i +
36          "</td><td>" + JSONObject.business[i] +
37          "</td></tr>");
38   for (i in JSONObject.websites)
39     document.write("<tr><td>" + i + "</td><td>" +
40          JSONObject.websites[i] + "</td></tr>");
41   document.write("</table>");
42 }
43
44 function output_specific(JSONObject){
45   document.write("<table><tr><th>Key</th>" +
46        "<th>Value</th></tr>");
47   document.write("<tr><td>Name</td><td>" +
48   JSONObject.business.name +
49   "</td></tr>");
50   document.write("<tr><td>Company Website</td><td>" +
51   JSONObject.websites.url[0] +
52   "</td></tr>");
53   document.write("<tr><td>Blog</td><td>" +
54   JSONObject.websites.url[1] +
55   "</td></tr>");
56   document.write("</table>");
57 }
```

在这个例子的一个函数中，使用集成的 JavaScript 函数 eval() 从一个包含对应于 rjs.json 的 JSON 结构的字符串中（第 2 行，注意该字符串必须是文件中的一行；否则，它必须拆分，并通过链接字符串连接起来）创建一个 JSON 对象（第 3 行——var JSONObject=eval('('+JSONText+')');)。在第二个函数中，你会发现第 18～26 行中通过数组记法直接声明 JSON 对象。两个函数的表现相同；调用两个输出函数进行 JSON 对象上的操作。JSON 对象通过 eval() 从一个字符串创建，这个对象与通过数组记法创建的对象完全相同。

在两种情况下，函数动态写入一个 HTML 表格。在 output_all() 函数中，在 JSON 对象上进行一个循环，通过组合句点标记法和数组标记法输出关键字和所包含的值。在 JavaScript 中处理 JSON 对象的绝妙之处在于，JSON 结构被表现为嵌套对象。如果在对象层次的一层上有同一类型的多个对象，这些对象被当成对象字段使用。很难用更简单和更符合逻辑的方法来使用这些结构。如图 3-2 所示。

在 output_specific() 函数中，处理来自结构的特定值。

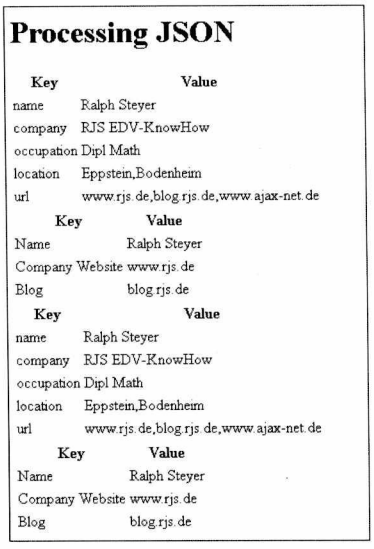

图 3-2 通过 JavaScript 处理 JSON 数据

3.4 DOM 和对象

在前一个 JavaScript 例子中，大家已经看到了对象。JavaScript 在一定程度上是一种面向对象语言。严格来说，它被称作基于对象的语言，因为缺乏某些真正面向对象语言所需要的特性。

对象（Object）这个术语在编程中指的是用于描述来自真实世界的对象及其所有属性和行为的模型。例如，打印机、显示器或者键盘都是对象。软件本身的各个部分也可以是一个对象：例如，浏览器或者它的组件——如框架、浏览器中的状态行或者浏览器窗口，或者文档的一部分——网页中的一个标题、段落、图形等。面向对象的方法将一切都看作对象，可以独立捕捉、描述和处理。

面向对象编程可以由这一事实描述：相关的语句和数据组成一个相互关联、完备的独立单元（对象），这个单元提供属性和方法（能力），并且可以拥有某种状态。属性和方法组成了对象的成员。

面向对象编程的核心是方法和属性（特性或者数据）总是分配给对象的。也就是说，没有不与对象相关的方法或者属性。

类元素是特殊的属性和方法，无法通过特定的对象访问，但是可以通过类访问。可以在没有首先创建对象的情况下使用它们。

DOM 和网页元素的访问

在 JavaScript 中创建一个对象或者使用类元素是很常见的事。实际上，我们总是会在自

己的 JavaScript 代码中使用浏览器自动提供的对象。这些对象基于某种对象模型，不被看作 JavaScript 或者（X）HTML 的一部分，而是用于描述几乎任何树形结构文档——这就是已经被反复提到的 DOM 概念。甚至可以更彻底地说，如果不使用 DOM，JavaScript 的编程就毫无意义！

可以通过各种不同的技术使用这个对象库，既可以通过编程或者脚本语言，也可以通过应用程序。有了 DOM，以这种对象特征访问网页基于一种跨平台、操作语言无关接口的概念。

在这一概念中，(X)HTML 网站（一般指的是具有树形结构的文档——例如，一个 XML 文档）不被当作一个静态构造的、完备和无法分辨的单元，而是被看作可以差别化的一个结构，其单独组件可以供程序和脚本动态访问。这种方法可以在网页已经加载到浏览器时操纵其单独组件。这种操纵超出了浏览器从头到尾加载文档时进行的简单解读。

DOM 的概念由几个方面组成。例如，它使浏览器将 (X)HTML 网站当成常规文本文件读入，并执行对应的 (X)HTML 语句。除此之外，浏览器还将在加载网页的这些组件时按照这一概念中（可以单独识别）的类型、相关属性和在网页中的位置进行索引。

这种树形结构在网页加载时在电脑的主内存中构建，在用户离开网站时删除。树中的元素被称作节点（Node）。

类似的元素在浏览器索引时在一个字段中一起管理。所以在加载网站时，浏览器对它在网页中能够单独处理得元素的所有相关数据有确切的认识。但是这些数据是什么，以及浏览器所能进行的处理在各种浏览器中可能有很大的不同。

每个可处理元素（例如，特定的 (X)HTML 标记）还可以在网页加载到浏览器时更新（例如，如果使用一个脚本编辑网页元素的位置，或者使用样式表在网页完全加载后动态地改变元素的布局）。这就是我们在第 2 章的例子中已经进行过的操作。

DOM 概念中的许多对象可以使用对象层次结构的形式访问。如果一个对象属于另一个对象，可以用句点（DOT）标记法书写：首先写下顶级对象的名称，然后写出下级对象的名称。

例如，以一个网页为例，它可以在 JavaScript 中通过 document 对象访问。因为网页位于浏览器窗口，该窗口可以通过 window 访问，所以该网页可以通过 window.document 访问。

DOM 对象的这些访问选项在对象嵌套中有严格的方向，但是有一些其他的语法选项总是引用相同的对象，提供独立于访问类型的属性和方法。例如，getElementById()、getElementByTagName() 和 getElementByName() 方法，也可以直接指定名称。但是正如刚才讨论的，所有访问 DOM 的标准方法都有某些问题，jQuery 通过 $() 和各种其他技术对这种高访问进行了标准化。

提示

附录中有一个可用 DOM 对象的列表。

3.5 样式表和 DHTML

对于许多网站来说，网页在加载到浏览器之后出现变化是核心的特征。这就是 DHTML 的准确定义。但是 DHTML 往往被描述为 HTML 或者 XHTML、JavaScript 和样式表的组合。

格式模板（或称样式表）由描述元素格式化规则的纯文本组成。现代网站将 HTML 和 XHTML 的意义降低为几乎全部用于网站的结构，而布局完全由样式表负责。特别是，样式表提供了去除布局命令与信息携带者混合的选项。可以实现结构和布局的清晰分离。通过 JavaScript 和样式表，还可以在网页加载到浏览器中之后，用针对性的方法修改网页中某个元素的布局或者位置属性。

3.5.1 CSS：Web 标准语言

样式表并不是由一种单独的语言组成的，只是一个概念。样式表语言不止一种，而是有各种不同的方法或者语言。样式表的规则和语言元素在各种语言中略有不同，但是看上去很相似。在 Web 上，我们看到的主要是 CSS。

一般来说，在应用样式表时，数据以原始形式存在，或者以我们想要改变的某种方式存在。然后，数据采用不同的方式表现，信息本身保持不变。在某些情况下，源数据可能被抑制，或者用输出文档中的附加数据补充。变换或者格式化的描述通常以外部文件的形式出现，但是在某些情况下也可以直接写入数据所在的文件（例如，一个网页）。所以，样式表只是为已经存在的信息提供新的外观。为此，数据和格式化信息由一个翻译系统处理，然后以不同的方式显示。

基本上，网页中的样式规则可以应用到任何 HTML 或者 XHTML 元素上。但是某些元素特别适合应用样式表，它们的全部好处就是通过样式表格式化。

样式表通常由各种元素格式化规则组成。为了在网页中使用 CSS 等样式表，必须将它们添加到 HTML、XHTML 或者 XML 网站。这可以通过在文档中嵌入样式表，或者导入一个外部文件来完成。样式表是否加载以及如何完成格式化，由我们想要应用 CSS 格式的数据类型决定。

在网页中嵌入内部样式表可以通过 <style> 标签完成，这是一个纯粹的 HTML 容器，在这个容器中定义所有样式规则。在 HTML 中，可以在网页的任何地方编写这样的样式区域。

HTML 中样式表的结构类似于程序清单 3.13 所示。

程序清单 3.13 HTML 中样式容器的结构

```
<style type="text/css">
  ... any CSS formattings ...
</style>
```

注意

我们常常在容器中发现 HTML 注释。但是这不是必要的（和通过 JavaScript 整合的情况

一样)。

在网页中，可以通过 <link> 标记添加对外部样式表的引用。和 JavaScript 一样，引用外部文件实际上总是首选方案。在样式表中，这是实现人们希望的结构和布局分离的唯一方法。

程序清单 3.14 展示了引用外部 CSS 文件的结构。

程序清单 3.14 集成一个外部 CSS 文件

```
<link type="text/css" rel="stylesheet"
  href="[URL of CSS file]" />
```

如果想要为网页的一个元素指定单独样式信息，可以使用样式表指令作为元素的嵌入式定义。这意味着通过标记中一个附加的 style 参数设定一个特性，这条样式规则只适用于定义的容器中，也可以选择导入样式表。

3.5.2 CSS 声明的具体语法

我们简短介绍一下 CSS 声明的具体语法和指定格式化规则的选项。CSS 声明的语法总是遵循相同的结构。指定一个名称、一个冒号和需要格式化的属性。

CSS 声明的结构如下所示：

程序清单 3.15 CSS 声明结构

```
[name] : [value]
```

3.5.3 选择器

需要进行格式化的元素（称作选择器）被放在规则前面。一个元素的多条格式化规则用分号分隔，通常包围在波形括号中。选择器有如下不同形式：

- 元素选择器
- 特性选择器
- 代选择器
- 全局选择器
- 伪类

在选择器领域中，jQuery 所提供的功能是其最实用的亮点之一。

3.6 小结

本章，我们用极其紧凑的形式介绍了现代互联网编程的核心基础知识和特殊性。高效的应用和对 jQuery 等框架及工具包的理解基于这一领域的基本认识。我们只是简单地介绍了一些特定的技术，因为这一领域的某种经验（或者和其他来源的并行工作）是成功使用本书的需求之一。但是，大家现在应该已经了解了 jQuery 程序库的工作方式，也能够看到 jQuery 提供的优势。特别地，本章介绍了如何将 jQuery 与网页组合，以及需要注意的事项。

第 4 章
jQuery 工作原理

本章介绍 jQuery 的一般功能，特别是与 JavaScript 及文档对象模型（DOM）之间的关系。还要学习 jQuery 如何扩展 JavaScript 和 DOM 的核心功能。有效地使用 jQuery 编程要求熟悉 JavaScript 语法和概念，也就意味着你应该理解如下这些基本知识：

- 变量和数据类型
- 字面语法
- 表达式
- 对象及其方法和属性
- 函数、函数调用和函数引用
- 赋值
- 操作符
- 关键字

在本章中，还要学习一般编程特别是 JavaScript 编程中其他的一些核心术语。

注意

本书不是一本 JavaScript 教科书，我们也绝对没有办法提供一个编程入门。但是附录中提供了使用和理解 jQuery 必备的 JavaScript 特性的一个简单总结。除此之外，JavaScript 的知识是使用本书的先决条件。

在本章中，我们讨论 jQuery 核心（“核心”对应于 jQuery 文档）。这些核心概念是该框架的基础，基于复杂的 JavaScript 和 DOM。然而，本章比 jQuery 文档更为全面，覆盖了与 jQuery 工作原理相关的更多主题。

刚开始阅读本书时，没有足够 JavaScript 知识的读者可能感到气馁。但是不要担心，可以在不完全理解 jQuery 工作原理的情况下，应用它的许多特征。在任何情况下，都可以通读本章。毕竟，它解释了如何有效访问网页元素的精华技术，能够帮助大家理解 jQuery 方法的应用方式。

当熟悉了应用 jQuery 的简单方法，了解为什么 jQuery 以特定方式实现某些功能之后，本章提供必要的背景知识，形成更加深入地使用和理解 jQuery 程序库及其功能的核心基础。这些背景知识将帮助大家理解大部分动画 / 特效应用程序。

4.1 访问网页元素

网页在浏览器中以 DOM 中的元素表现。这些元素通常具有特性（Attribute）、文本内容和子元素。如前所述，可以通过纯 JavaScript 和原生 DOM 方法，以不同方式访问网页的组件。但是这种方式通常不方便或者极度不可靠，因为不同浏览器以不同的方式构造网页的 DOM，不管是在构造 DOM 的时机（很快将会介绍）还是实际存在的节点上都存在差异。相应的，只遍历 DOM 树的访问方法可能在不同浏览器中造成不同的结果。

为了让读者更清楚，我们来举个例子：假定你想要处理网页中的一幅特定图片。现在你描述获得图片的路径。这意味着，转到 DOM 树的根元素，从那里顺着各个元素找到所要图片的节点。

我们以程序清单 4.1 中的简单网页作为例子（ch4_1.html）。

程序清单 4.1 包含两幅图片的简单网页

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
   <title>DOM Nodes</title>
 </head>
 <body>
   <img src="images/b1.jpg" /><br/>
   <img src="images/i2.jpg" />
 </body>
</html>
```

这里，你可以看到一个包含两幅图片的简单网页。现在试着描述 DOM 中指向第二幅图片的路径。这不像从视觉表现中挑出超文本标记语言（HTML）元素那么容易。视觉的表现相当明显。但是即使是源代码中引用图片的编排也不足以帮助你找到引用路径，因为源代码只有浏览器用于创建 DOM 的工作指令。必须描述对应于该 DOM 树中的路径。但是这棵树是什么样子的呢？

可以从这个简单的网页中看出 DOM 的内部结构，了解浏览器实际处理的内容。在 Firefox 中，DOM Inspector 加载项特别适合于这个目的。较新版本的 Internet Explorer 提供的开发人员工具中也有这个视图。[⊖]利用这两个工具，就可以比较生成的 DOM 树中的路径，

⊖ 大部分其他现代浏览器也有类似的工具。

看到它们的不同之处。如图 4-1 所示。

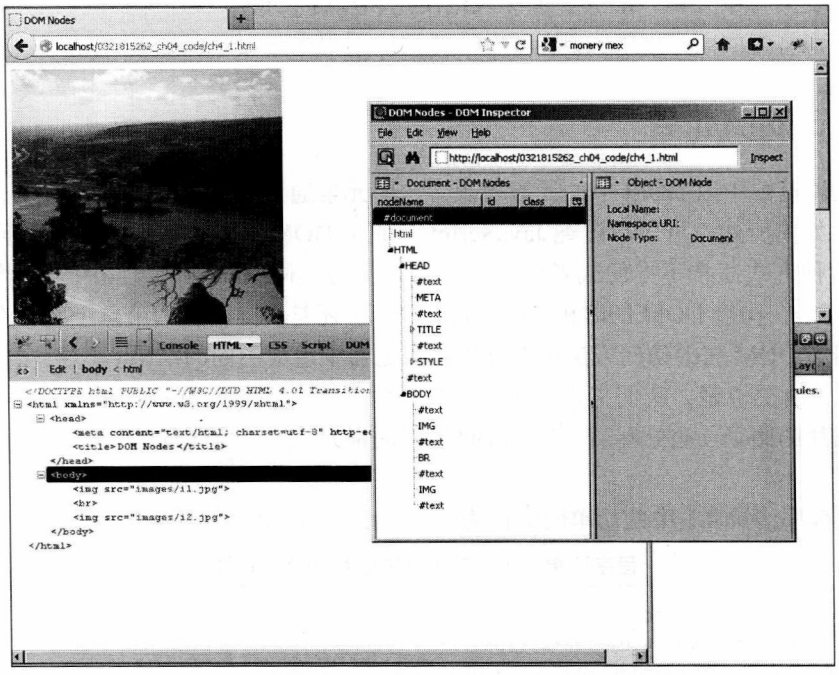

图 4-1 Firefox 实际上用的 DOM 结构可以在 DOM Inspector 中看到。开发人员工具显示 Internet Explorer 从网页生成的 DOM

只要所有浏览器用同样的方式构建 DOM，指向第二幅图片的路径描述就是全局可用和可转换的。但是如果在同一个页面上，浏览器或多或少地创建和其他浏览器不同的节点，该怎么办？如果这些树的分支数量不同怎么办？

遗憾的是，许多网站都确实是这种情况（和上面那个简单的例子一样）；结果是，在各种默认的 DOM 元素方法上出现了各种问题。这些默认方法基于浏览器在计算机内存中创建的 DOM 结构。

如你所见，我们的简单示例已经说明了这种尴尬的处境。如果在 DOM Inspector 或者开发人员工具中查看实际的 DOM 树，在 Firefox 中，指向第二幅图片的路径的（结构化）描述如下：

从根向下到达 HTML 元素，然后从这里到第二个子节点（BODY），从该节点前往第 6 个子节点。而在 Internet Explorer 中，需要从 BODY 前往第 4 个子节点。明显，节点的数量不同，所以这类路径描述会让我们在 Firefox 中准确地找到该节点，但无法直接转换到 Internet Explorer 中。

注意

jQuery 的访问技术缩短和标准化对 DOM 元素的访问，弥补了浏览器相关特殊性引发的

问题，并且添加了遗漏的功能。

4.2 jQuery 命名空间和 jQuery 对象

命名空间指明一个范围，其中特定的标识符都是唯一的。我们不仅会在编程中看到它，还会在可扩展标记语言（XML）（以及一个文件系统中）等处看到。目录也可以看作命名空间。在一个目录中，一个文件名只能出现一次，它必须是唯一的。但是在不同目录中，可以在一个文件系统中使同一个文件名重复出现。这是命名空间的好处。可以在一个上下文中用重复的标识符代表不同的含义，只要它们所处的命名空间不同即可。

命名空间在编程中也很有用，特别是想要合并来自不同程序库的函数和变量的时候。如何避免不同项目的程序员为某些函数或者变量取相同的名称，在合并的时候导致冲突？如果每个项目都有自己的命名空间，就可以避免这种冲突。

但是 JavaScript 实际上并不支持命名空间的概念（这点与 C# 或者 Java 不同，这两种语言中命名空间可以通过句点标记法简单地区分）。这种命名空间技术有效且可靠地集成到整体的概念中，而一种包结构确保了命名空间的安全隔离。

在 JavaScript 中，可以像在“大型”编程语言中那样做。为了实现命名空间概念，使用句点标记法分隔对象空间和属性，并以此为模板，将标识符中用句点分隔的组件当作不同的命名空间。[⊖]但这是“非强制”的规则，我们可以坚持或者打破不得体的语法结构。避免因为不谨慎的编程而浪费这一概念的作用，需要很多的训练。

jQuery 试图在 JavaScript 的世界和框架本身中引入正确的命名空间概念。在框架中，所有全局对象被指定在一个命名空间内，该命名空间用以 jQuery 标志开始、分隔符（句点）之后带上真正名称的标识符定义。以后，被标识的元素——从框架的角度——被指定到 jQuery 命名空间。可以通过 jQuery 标志或者较短的别名 $ 访问它。

警告

所以，在框架中的 jQuery 或者 $ 引用的既是命名空间也是一个对象（更准确地说，是一个具有元素的数组），它组成了一个正确的命名空间概念的一致实现（在整个框架中都保持不变）。

4.3 jQuery 中的特殊数据类型和结构

当然，jQuery 基于 JavaScript 的标准数据类型——String、Number、Boolean、Object 和 Function；数组；回调；XMLHttpRequest 对象；原型技术和许多其他的技术。如前所述，

⊖ 实际上，在 JavaScript 中，前置的标识符意味着一个对象的声明，附加的标识符表示一个属性。这和真正的命名空间概念很相近，但并不相同。

在我们的讨论中假定你对这一领域已经有了基本知识，所以不再详述（附录包含一个简短的摘要）。但是除了这些技术之外，jQuery中还有一些特殊的类型或者结构（内建结构），我们至少在某种程度上提醒自己这一点。

4.3.1 选项

最重要的事情应该立刻提及：在jQuery框架中，我们会在很多地方密集地使用选项（Option）。jQuery中的选项实际上指的只是纯JavaScript对象，但是使用特殊的标记法。这些选项通常在jQuery中通过波形括号表示的代码块声明使用。各个选项用逗号分隔，由一个选项标识符（可以用引号包含它们）、一个冒号和选项值（换句话说，一对值）组成。JavaScript中使用这种标记法的定义被称作对象字面语法（Object Literal）。

程序清单4.2 对象字面语法形式的选项

```
{
    border: "5px outset",  cursor: "move",
    opacity: 0.5, statusinfo: true
}
```

前面已经提到，可以将选项标识符放在引号中，如程序清单4.3所示。

程序清单4.3 对象字面语法形式的另一种选项标记法

```
{
    "border": "5px outset", "cursor": "move",
    "opacity": 0.5, "statusinfo": true
}
```

在某些情况下，必须为标识符加上引号。因此，应该始终使用引号，这样就可以安全地使用，而不会遇到需要花费很长时间才能发现的问题。

在任何情况下，选项都可以让你访问对应的属性和方法。

注意

还可以将回调应用或者匿名函数作为值赋给一个选项。这是jQuery中大部分事件处理的完成方法。

不同的值对可以写入多行（像例子中那样），也可以写在一行中。分隔符是逗号而不是换行符。

4.3.2 Map

jQuery添加了一种数据类型Map，作为特殊的选项。例如，这个类型用于框架的AJAX函数中。请求数据可以这种形式交付。这种类型可以是具有内部结构的一个字符串、包含表单元素的数组、包含表单元素的jQuery对象，或者包含键 – 值对的通用对象。在后一种情况中，可以为一个键赋多个值；更具体地讲，可以将一个数组赋给一个键。

4.3.3 Array< 类型 > 标记法

在 jQuery 应用程序编程接口（API）中，大家常常会发现 Array< 类型 > 这样的类型标记法，如程序清单 4.4 所示。

程序清单 4.4 泛型

```
dragPrevention Array<String>
```

这种标记法的动机是为了 JavaScript 中的泛型（Generic Type）。它表示一个方法不仅要求一个数组参数，还要求数组中元素的类型（例子中是字符串）。遗憾的是，JavaScript 是弱类型的，纯粹根据这种语言，我们无法满足这些需求。框架必须通过内部安全措施，费力地人工实现。在使用 jQuery 自行编程时，不可能使用这种标记法。

4.3.4 jqxHR

jQuery 1.5 中推出了一种与 AJAX 关联的新数据类型。$.ajax() 方法返回一个类型为 jqxHR 的新对象。这个对象基于经典的 XMLHTTPRequest 对象，并添加某些功能对其进行扩展。

4.4 jQuery() 函数和 $() 别名

jQuery() 绝对是整个 jQuery 框架中最重要的函数之一。可以使用各种参数，还可以通过 $() 别名使用它。该函数直接面对 jQuery 命名空间标志（或者 jQuery 对象）。利用这个函数，可以可靠地选择 DOM 中的一个或者多个对象，并将其赋值给 jQuery 命名空间。这样，就可以克隆 DOM 中的原始对象，创建一个 jQuery 对象，然后，我们就拥有了一个引用原始对象的新对象。这意味着，还可以在这些对象上使用框架的所有方法，通过框架将其应用到嵌入的元素上。

注意

通过集成这样一个强大而可靠的框架，jQuery 对象在后台允许使用许多函数，这些函数非常重要，却很容易被忽视。例如，它能识别浏览器或者操作系统，并且在后台提供适应性的措施。它还能在表示一组元素时[⊖]，确保被调用方法的某些步骤真正可靠地应用到所有元素上。框架的许多方法以某种循环的形式遍历一个集合的所有元素来实现这一点。此外，它还能确保遵循某些规则的扩展无缝地集成到框架（插件）中。

我们几乎可以向函数传递任何可以描述网页中任一个（或者一组）元素的数据，作为第一个参数。这包括字符串形式的 CSS 选择器、字符串形式的 HTML 标记、一个或者多个 DOM 元素，或者指向某个函数的回调（当然包括匿名函数）。

⊖ 例如，多个标题或者 div 元素。

程序清单 4.5 整个网页中所有类型为 div 的元素

```
$("div");
```

注意

我们稍后将更详细地讨论选择器。

我们来看一个完整的例子（ch4_2.html）。

程序清单 4.6 选择所有 1 级标题

```
...
08    <script type="text/javascript">
09      $(document).ready(function(){
10        jQuery("h1").css({
11          background: "red",  color: "white"
12        });
13      });
14    </script>
15  </head>
16  <body>
17  <h1>The hen is wisest of all the animal creation.
18    She never cackles until the egg is laid./h1>
19  <h1>Action speaks louder than words
20    but not nearly as often.</h1>
21  <div>
22    <h1>Do not go where the path may lead,
23      go instead where there is no path
24      and leave a trail.</h1>
25  </div>
26  <h1>No  great genius has ever existed
27    without some touch of madness.</h1>
28  </body>
29 </html>
```

在这个例子中，大家看到 jQuery() 两次，一次是第 9 行中通过简写的 $()。这里通过指定 DOM 对象 $(document) 选中整个网页。

而在第 10 行中，长版本的选择项还指定了字符串形式的一个 CSS 选择器。用 jQuery("h1")，可以得到网页中所有类型为 h1 的元素。这些元素被作为一个集合赋予 jQuery 命名空间。然后，可以使用框架的方法和属性。我们使用了 css() 方法设置网页中类型为 h1 的所有标题的某些 CSS 属性（通过对象字面语法设置选项），不过这一点在这里不是那么重要。但是要注意，其中一个标题在 div 块中。我们在下一个小节中使用它，在这个例子中，它是不相干的（我们还是想在这里演示）。已经格式化一级标题的网页如图 4-2 所示。

上下文

jQuery() 的第 2 个参数（可选的）指定上下文（Context）。上下文一词指的是元素可见的

环境。如果不指定它，则用整个 HTML 文档作为上下文。否则，可以指定一个 DOM 元素、文档或者 jQuery 对象作为上下文，然后，元素仅在这个上下文中可见（例如，一个段落或者一个表单中）。

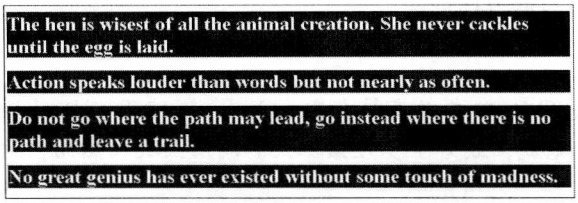

图 4-2 所有一级标题都已经格式化

程序清单 4.7 选择所有类型特性为 Radio 的输入元素（即单选按钮），但是仅在网页的第一个表单中寻找

```
$("input:radio", document.forms[0]);
```

我们来看看前一个例子稍作改变的版本（ch4_3.html——与 ch4_2.html 相同的代码行未列出）。

程序清单 4.8 选择以类型为 div 的元素为上下文的所有 h1 类型标题

```
...
09      $(document).ready(function(){
10        jQuery("h1","div").css({
11          background: "red",  color: "white"
12        });
13      });
...
```

注意，第二个参数限制了选择。我们只想选择在一个 div 块中的标题。这意味着，在这个特例中只有第三个标题被选中。如图 4-3 所示。

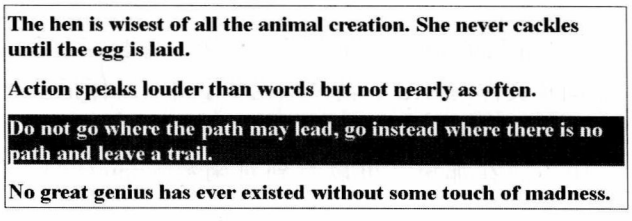

图 4-3 只有第三个标题被格式化

4.5 在 DOM 构建之后执行函数

我们用更基础的方法来探讨这个主题。前几章中我们已经学习了这个方法和整个主题

的重要性。无法在 DOM 树正确构造之前，可靠地通过 JavaScript 在浏览器中访问网页的组件。因为不同浏览器在构造 DOM 时表现不同，在试图访问网页元素时会发生各种问题——尤其是在试图过早地访问网页元素（在浏览器正确构造 DOM 之前[⊖]）的情况下。jQuery 提供了避免此类问题的一个可靠方法。

警告

原则上，可以在 HTML 中或者直接在 JavaScript 下使用 onload 事件处理器，在网页加载之后（等同于 DOM 完成，或者至少应该如此）调用函数。然而，这个事件处理器在不同浏览器中有着错误的实现。在 jQuery 中，我们能保证安全。

4.5.1 作为 jQuery() 参数的回调或者匿名函数

对 DOM 做出反应的方法之一是：如果传递指向一个函数的回调或者一个匿名函数，作为 jQuery() 或者 $() 的参数，这个函数在 DOM 完成之后会立刻执行——不会过早。

我们来看另一个例子（ch4_4.html）。

程序清单 4.9 函数回调

```
...
08    <script type="text/javascript">
09      $(output);
10      function output(){
11         $("#info").html($("img:first").attr("src"));
12      }
13    </script>
14  </head>
15  <body>
16    <div id="info"></div>
17    <img src="images/i1.jpg"/><br/>
18    <img src="images/i2.jpg"/>
19  </body>
20 </html>
```

注意，图片只在第 17 行和第 18 行中被集成。此外，输出所在的 div 区域只在访问后列出。输出的是第一个图片文件的 URL，清楚地说明了对应的标记已经用于构建 DOM 树。如图 4-4 所示。

关键的位置在第 10 行。在那里，可以看到对函数 output() 的引用，它的声明在第 10 ～ 12 行中。

我们再来看看匿名函数如何代替这一参数（ch4_5.html——与 ch4_4.html 相同的代码行不再列出）。

⊖ 这可能由从互联网加载网站的延迟引起。

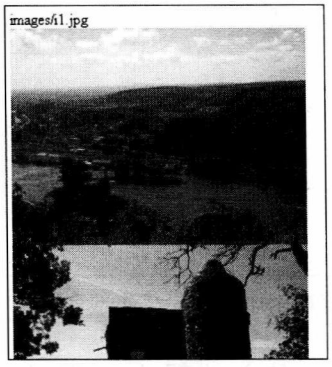

图 4-4 在函数被调用时，DOM 树已经构建完成

程序清单 4.10 作为参数的匿名函数

```
...
09      $(
10      function(){
11         $("#info").html($("img:first").attr("src"));
12      });
...
```

这两种标记法是 $(document).ready() 的简写形式，是对 DOM 加载事件的响应，该事件说明 DOM 树已经就绪，可以遍历和操纵。ready 事件处理器作为参数传递。

提示

可以在网站中使用任意数量的 $(document).ready() 事件（使用任何一种标记法）。绑定的函数按照添加时的顺序执行。

我们用一个完整的例子（ch4_6.html）来研究。

程序清单 4.11 ready() 方法的不同变种

```
...
06    <script type="text/javascript"
07      src="lib/jquery-1.8.2.min.js"></script>
08    <script type="text/javascript">
09      function extern(){
10        alert("Callback: " +
11          $("img:first").attr("src"));
12      }
13      $(function(){
14        alert("Anonymous: " +
15          $("img:first").attr("src"));
16      });
17      jQuery(extern);
18      $(document).ready(function(){
19        alert("Anonymous: " +
```

```
        $("img:first").attr("src"));
    });
    $().ready(extern);
  </script>
</head>
<body>
  <img src="images/i1.jpg" />
</body>
</html>
```

测试上述示例。大家将会看到输出窗口按照集成的顺序被调用。

4.5.2 将 document.ready() 放入外部 JavaScript 文件

如果注意我们迄今为止所使用例子的结构，就会发现jQuery程序库被作为一个外部JavaScript文件集成，但是函数和方法（包括document.ready()）的调用在内部的脚本容器中进行。对于我们的例子，这描述起来很方便，但是并不是强制的要求。可以完全分离结构和函数，将这些调用也放到外部文件中。实际上，这在实践中是推荐的通用原则。但是在引用jQuery程序库之后，必须提供对这些调用所在JavaScript文件的引用。

注意

网页中组合的所有JavaScript文件和内部脚本容器及嵌入脚本组成了一个公共的全局命名空间。这就是框架为了自己的功能，在其中组成jQuery命名空间的原因。这意味着，在一个脚本容器中，可以访问外部JavaScript文件的所有组件，而且可以从一个JavaScript文件访问另一个外部JavaScript文件中的组件（如果遵守顺序的话）。

这样的外部JavaScript文件只包含用任何允许使用的标记法表示的一个或者多个document.ready()调用。如果推断DOM完整，当然应该避免任何其他调用。但是，如果有必要的话，可以在外部JavaScript文件中写下额外的声明。在较大的项目中，我甚至将这些声明移到其他JavaScript文件中，使项目更加模块化；同样，加载网站时的所有函数调用也不应该（不管什么原因）打包到document.ready()的受保护区域中。

注意

如果在网页中使用样式表，所有样式表应该在脚本之前集成。特别是，集成应该在ready()方法调用之前进行。如果不这么做，可能在某些浏览器中造成问题。

4.5.3 为模块化 jQuery Web 应用创建基本结构的示例

我们来看一个完整的示例，它展示了以一致性的方式使用jQuery的实用Web应用程序结构（ch4_7.html）。

程序清单 4.12 模块化 jQuery 应用的 HTML 基本结构

```
01 <html xmlns="http://www.w3.org/1999/xhtml">
02  <head>
03    <meta http-equiv="Content-Type"
04      content="text/html; charset=utf-8" />
05    <title>Structuring a jQuery Application
06    </title>
07    <link type="text/css" rel="stylesheet"
08      href="lib/ch4_7.css" />
09    <script type="text/javascript"
10      src="lib/jquery-1.8.2.min.js"></script>
11    <script type="text/javascript"
12      src="lib/ch4_7_declarations.js"></script>
13    <script type="text/javascript"
14      src="lib/ch4_7_ready.js"></script>
15  </head>
16  <body>
17    <h1>Replacing an image</h1>
18    <button id="toggle1">Image 1</button>
19    <button id="toggle2">Image 2</button>
20    <hr/><img src="images/i1.jpg"/>
21  </body>
22 </html>
```

这个例子展示了刚才描述的完全模块化 jQuery 应用程序的 HTML 基本结构。首先，你可以看到对 CSS 文件的引用（第 7 行和第 8 行）。如果有必要，可以在这里包含更多的 CSS 引用。然后是对 jQuery 程序库的引用（第 9 行和第 10 行）。接下来，引用其他 JavaScript 文件。

在 jQuery 程序库之后引用的 ch4_7_declarations.js 文件只包含例子中的两个函数。

程序清单 4.13 两个函数的声明

```
01 function image1(){
02   $("img").attr({
03     src: "images/i1.jpg"
04   });
05  }
06  function image2(){
07   $("img").attr("src", "images/i2.jpg");
08  }
```

这两个函数更改引用对象的 src 特性。[⊖]（大家在第 2 章中了解过这个例子。注意，该文件明确地使用 jQuery 语法！）

不要误解：对图片元素的访问不会在网页中其描述之前和 document.ready()（该函数

⊖ 更准确地说，是该类型的所有元素，但是在这里没有关系。

在稍后引用的 JavaScript 文件中才能找到）之外进行。上述文件中，只有它的声明，而不是调用。

ch4_7_ready.js 文件如下所示。

程序清单 4.14 使用 document.ready()

```
01 $(document).ready(function(){
02  $("#toggle1").click(image1);
03 });
04 $(function(){
05  $("#toggle2").click(image2)
06 });
```

在这里可以看到顺序列出了两种[⊖]document.ready() 的用法。每种用法都描述了一个点击事件，该事件有一个指向在其他外部 JavaScript 文件 ch4_7_declarations.js 中声明的函数的回调。

现在，我们来试验引用的顺序。你会发现这可能造成错误。看看下面改变引用顺序的摘录（ch4_7_error.html）。

程序清单 4.15 集成顺序中的一个错误

```
...
09    <script type="text/javascript"
10      src="lib/ch4_7_declarations.js"></script>
11    <script type="text/javascript"
12      src="lib/ch4_7_ready.js"></script>
13    <script type="text/javascript"
14      src="lib/jquery-1.8.2.min.js"></script>
...
```

在这个变种中，对 jQuery 程序库的引用（第 13 行和第 14 行）出现得太晚。因此，jQuery 语法在前面的脚本中还不可用。如图 4-5 所示。

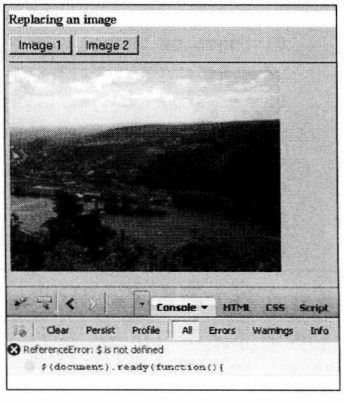

图 4-5 正如你在错误面板上所看到的，jQuery 语法此时不可用

然而，对文件 ch4_7_declarations.js 和 ch4_7_ready.js 引用顺序的改变没有关系，如果愿意，可以尝试伙伴网页上的 ch4_7_2.html 文件。

4.6 用 jQuery() 创建一个元素并将其插入网页

jQuery() 和 $() 的一个有趣的功能是即时创建 DOM 元素。然后，这些元素可以可靠地动态插入网页。从内部看，

⊖ 这是仅有的两个用于演示的方法调用。

使用 .html() 或者 innerHTML 属性，具体取决于元素的类型。但是不需要担心内部的工作原理。毕竟，jQuery 之类框架的精华之处是向程序员隐藏这些细节，最重要的是，它们能够在所有支持的浏览器上可靠工作，为你省去适应不同浏览器和不同元素类型的麻烦。

当然，不应该忽略一点：想要创建和插入的元素必须与上下文融合。例如，不能在一个 div 容器中插入一个 body 元素。而且，整个 HTML 内容都必须是格式良好的。特别是，必须确保正确地结束创建的元素。例如，$("") 这样的指令在某些浏览器中无法正常工作，而正确的 XHTML 语法 $("") 没有问题。

提示

可以指定第二个可选参数，指定创建新元素所在的文档（上下文）。

现在，创建一个元素并不意味着它已经被添加到网页中。但是可以这样，例如，使用 jQuery 方法 appendTo()。

我们来看一个完整的例子（ch4_8.html）。

程序清单 4.16 创建一个空白网页

```
...
06    <script type="text/javascript"
07      src="lib/jquery-1.8.2.min.js"></script>
08    <script type="text/javascript"
09      src="lib/ch4_8.js"></script>
10  </head>
11  <body></body>
12 </html>
```

显然，这个网页的主体部分是空白，链接的 JavaScript 文件 ch4_8.js 如程序清单 4.17 所示。

程序清单 4.17 动态创建元素并集成到网站中

```
01 $(document).ready(function(){
02  var block= $("<div>A Block</div>");
03  $("<h1>Dynamically Creating Elements</h1>").
04      appendTo("body");
05  block.appendTo("body");
06 });
```

在 DOM 创建之后，第 2 行中创建一个类型为 div 的 jQuery 元素。第 3 行创建一个类型为 h1 的 jQuery 元素并直接添加到网站（第 4 行）。然后，保存在变量 block 中的第一个创建元素被添加到网站。相应地，我们首先看到网站的标题，然后看到下面的文字块。如图 4-6 所示。

初始化特性的选项

从 jQuery 1.4 起，可以在创建元素中使用这种方法，指定选项为第二个参数，设置所创建元素的特性。下面的示例文件中 ch4_9.html 可以省略；除了引用的 JavaScript 文件是

ch4_9.js 之外，其他与 ch4_8.html 相同。下面是有趣的 ch4_9.js 文件。

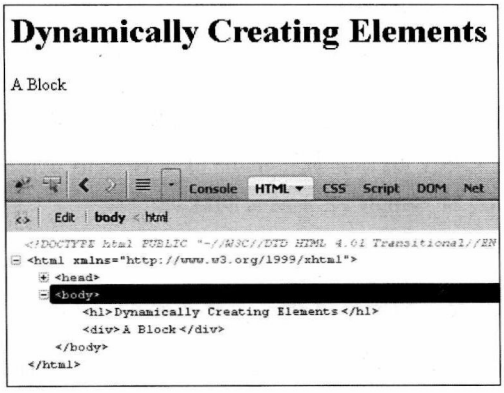

图 4-6 网站和 Firebug 清晰地显示了元素的添加方式

程序清单 4.18 创建具有选项的元素

```
01 $(document).ready(function(){
02  var block = $("<div/>", {
03    css: {
04      background: "red",
05      color: "white"
06    },
07    html: "A block with parameters<br />",
08    click: function(){
09      $(this).fadeOut("slow");
10    }
11  });
12  var image= $("<img />", {
13    src: "images/i1.jpg"
14  });
15  $("<h1/>", {
16    text: "Dynamically Creating Elements"
17  }).appendTo("body");
18  block.appendTo("body");
19  image.appendTo(block);
20 });
```

在这个例子中，即时创建 3 个元素，然后动态地添加到网站中：

- 一个类型为 div 的块
- 一幅图片
- 一个类型为 h1 的标题

上述三个元素都被创建为空白元素（第一个参数），但是通过第 2 个参数配置选项。我们首先来看看 div 元素。

程序清单 4.19 创建和初始化一个 div 元素

```
02  var block = $("<div/>", {
03    css: {
04      background: "red",
05      color: "white"
06    },
07    html: "A block with parameters<br />",
08    click: function(){
09      $(this).fadeOut("slow");
10    }
11  });
```

选项用对象字面语法编写。css 属性表示对应名称的 jQuery 方法，该方法被用来设置 HTML 的 style 属性。[⊖]利用 html 选项，可以使用 jQuery 的 html() 方法，该方法本质上对应于通过 innerHTML 访问，并将翻译过的指定文本写入网站中。click 代表事件助手方法 click()，它本质上对应于事件处理程序 onclick。这部分还确保了 div 块用 HTML 内容填充，用 CSS 格式化，并且具有事件处理。[⊖]

第 12 ～ 14 行很明显是用于创建图片。URL 被当作一个选项设置。第 15 ～ 17 行中，创建标题并直接指定给 body 元素。

第 19 行也很有趣。变量 image 中的元素被附加的对象本身通过变量 block 访问。这意味着，该图片被写入 div 区域。如图 4-7 所示。

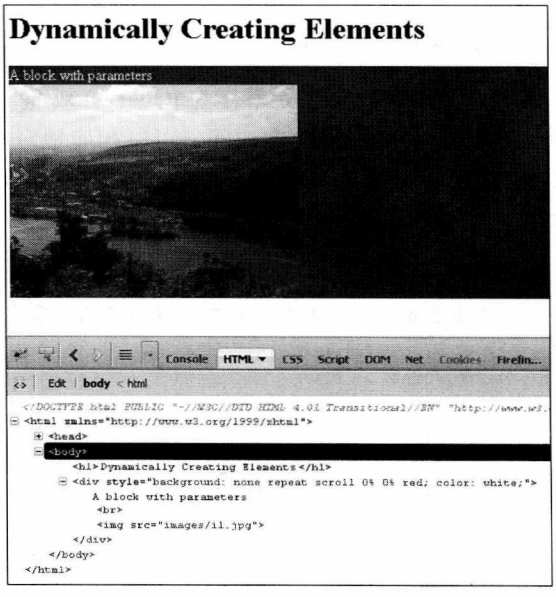

图 4-7 特性已经设置，元素被集成到网站中

⊖ 可以在 Firebug 中清楚地看到。

⊖ 如果点击这个区域，它会缓慢地淡出。

如果点击 div 区域，将会看到整个区域（包括图片）淡出。如图 4-8 所示。

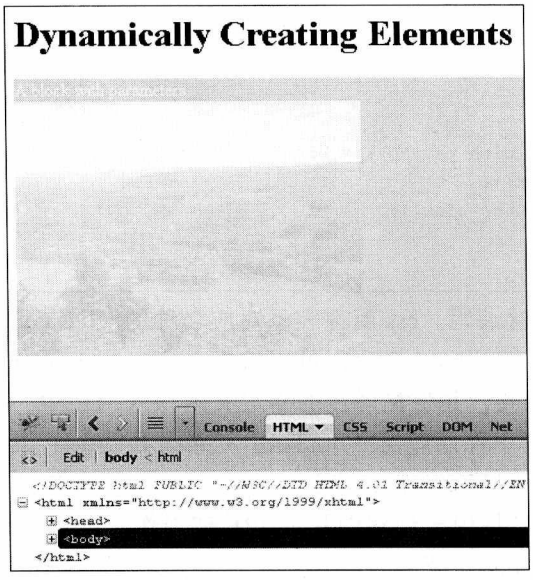

图 4-8 div 元素动态淡出

4.7 用 jQuery() 包装现有元素

我们再重申一次：当传递一个或者一组元素作为 jQuery() 或 $() 元素时，这些元素将被包装。因此，它们接着可以在 jQuery 上下文中访问，可以相应地对它们应用来自 jQuery 命名空间或者 jQuery 程序库的方法。这一点很了不起，因为该方法不仅接受 DOM 元素，还可以接受 XML 文档和 window 对象（即使它们不是 DOM 元素）。当然，也可以指定其他 jQuery 对象作为参数，我们在最后几个例子中已经展示了不同的方法。

程序清单 4.20 网站的背景颜色被设置为蓝色

```
$(document.body).css( "background", "blue" );
```

注意，元素没有包围在引号中。

通过 get() 直接访问 DOM 元素

如果使用 jQuery() 将元素放入 jQuery 上下文，就可以使用该框架的方法。但是要付出代价！因为在这个上下文中，不能直接访问经典的 DOM 属性和方法。此外，也不能将所有元素放入 jQuery 上下文中。

考虑下面的例子（ch4_10.html）。

程序清单 4.21 试图访问一个经典的 DOM 属性

```
...
06    <script type="text/javascript"
07      src="lib/jquery-1.8.2.js"></script>
08    <script type="text/javascript">
09      $(function(){
10        alert($("img:first").src);
11      });
12    </script>
13  </head>
14  <body>
15    <img src="images/i1.jpg" />
16  </body>
17 </html>
```

在第 10 行中，访问网站的第一幅图片并集成到 jQuery 上下文。通过刚生成的 jQuery 对象，我们试图访问经典的 DOM 属性 src。但是如果加载该网页，就会看到它是未定义的（undefined）。如图 4-9 所示。

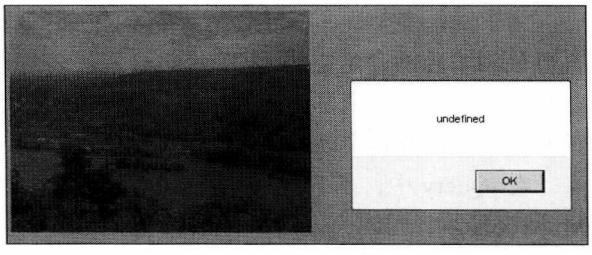

图 4-9 无法用这一方法访问经典的 DOM 属性

但是利用 get() 方法，就可以直接从 jQuery 上下文访问 DOM 树。特别是，这类访问还适用于通过 jQuery 对象无法直接访问的元素（如果没有合适的可用属性或者方法）。例如，如果你打算直接操作 DOM 元素而不需要任何 jQuery 内建函数（或者没有可用的函数），就可以使用这类访问。访问 DOM 属性成功后的页面如图 4-10 所示。

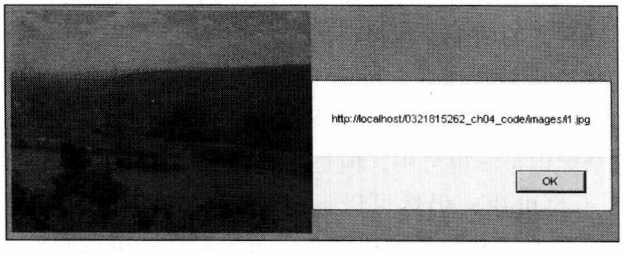

图 4-10 访问 DOM 属性成功

我们来看看对例子的如下修改（ch4_11.html）。

程序清单 4.22 通过 get() 访问经典 DOM 属性的另一种方法

```
...
10        alert($("img:first").get(0).src);
...
```

如果没有参数，get() 以一个数组的形式返回 $() 选择的所有 DOM 元素（和平常一样带

有所有操作选项)。如果像例子中一样指定一个索引，可以和平常一样选择一个 DOM 对象，处理所有可用的 DOM 属性。

4.8 使用 jQuery 和其他框架结合

jQuery 程序库提供了许多函数。一般来说，这些函数为 Web 编程中出现的大部分问题提供了合适的解决方案。如果它们对你来说还不够，还有基于 jQuery 的扩展（如 jQuery UI 或者 Mobile jQury），以及许多兼容性不受限制的 jQuery 插件。但是，另一个程序库或者另一个框架可能包含特定的函数，实现的方式与 jQuery 不同。你也可能更喜欢另一个框架中提供的解决方案。在这种情况下，你可能想要在一个网站中组合使用 jQuery 和其他程序库或者框架。根据前面描述的规则，你也可能想要在一个网页中集成这些框架。

为了确保多个程序库或者框架尽可能顺利地协同工作，如前所述，jQuery 程序库和所有插件使用单独的命名空间。[⊖]如果组合使用多个程序库或者框架，这能够确保一个框架中的标识符不会与另一个框架中的相同标识符发生潜在的冲突。但是，这当然只适用于所有程序库和框架支持或者实现命名空间技术的情况。

在 jQuery 中，所有全局对象保存在 jQuery 命名空间中。所以，如果将其与 Protype、MooTools 或者 YUI 等程序库组合使用，应该不会遇到问题。

noConfict() 函数

但是，在某些情况下，仍然可能有冲突。如前所述，$ 标识符是 jQuery 命名空间的简写（更准确地说，是 jQuery 变量或者对象）。注意，Prototype 框架也使用 $。所以，如果两个框架在一个网站中组合使用，我们就可能碰到 JavaScript 中存在的纯命名空间概念无法解决所有冲突的情况。如果两个框架组合使用，$ 标识符就不是唯一的，因为 JavaScript 中的命名空间的根将包含不同含义的两个 $ 标识符。

因此，jQuery 的核心区域提供了 jQuery.noConfict() 函数，用于在结合使用多个程序库时避免这类冲突。如果使用该函数，$ 变量被赋予网站中第一个实施的程序库所指定的含义。对于其他程序库，这个标识符不再可用，但是这能够解决歧义，从而消除冲突。

如果 jQuery 中 $ 不再可用，仍然可以通过同义的 jQuery 变量访问 jQuery 命名空间。jQuery.noConfict() 并没有关闭 $ 别名。利用一个布尔参数，可以删除所有 jQuery 变量，包括同义的变量 jQuery。

警告

jQuery.noConfict() 函数必须在 jQuery 程序库集成之后，集成和使用导致冲突的其他程序库之前调用。

⊖ 在 JavaScript 概念的限制之内。

jQuery.noConfict() 函数返回 jQuery 对象，所以可以将其赋予一个单独的变量。这个变量可以用作你自己的 $ 别名，如程序清单 4.23 所示（ch4_12.html）。用 $j 访问 jQuery 对象的页面如图 4-11 所示。

程序清单 4.23 为 $ 定义你自己的别名

```
...
06    <script type="text/javascript"
07      src="lib/jquery-1.8.2.js"></script>
08      <script type="text/javascript">
09        var $j = jQuery.noConflict();
10        $j(function(){
11          alert($j("img").get().length);
12        });
13      </script>
14      </head>
15      <body>
16        <img src="images/i1.jpg" />
17           <img src="images/i2.jpg" />
18  </body>
19 </html>
```

最终，可以将所有 jQuery 代码打包到 ready() 方法中。在这个区域中，有一个单独的命名空间，可以在其中使用 $。在这个方法之外，$ 将代表 Protype 功能。窍门在于，$ 作为一个参数传递给 ready() 调用的函数。

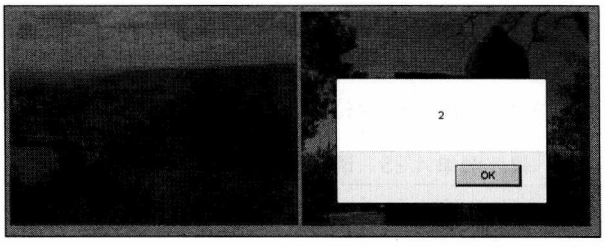

图 4-11 可以用 $j 访问 jQuery 对象

程序清单 4.24 将所有 jQuery 代码移入 ready() 方法

```
...
   <script src="prototype.js"></script>
   <script src="jquery.js"></script>
   <script>
     jQuery.noConflict();
     jQuery(document).ready(function($){
       // $ here stands for jQuery
      ...
     });

     // Here $ stands for the Prototype variation
```

```
    ...
  </script>
...
```

4.9 关于上下文的更多知识

我们回到上下文（Context）这个术语。这个术语在本章中反复出现。在JavaScript中，this变量总是引用当前上下文（换句话说，当前所在的对象）。

在基本设置中，this引用浏览器窗口（也就是DOM中的window对象）。但是如果在一个函数中，上下文也可能指向其他对象（window的子对象，如按钮或者表单字段）。这取决于函数调用时所处的情况。在jQuery中，所有时间处理器引用事件的触发元素作为上下文。

context、selector 和 nodeName

从版本1.3开始，jQuery提供context属性。这个属性提供已经传递给jQuery的DOM上下文。如果没有传递上下文，context将等同于文档。context属性本身也是一个对象，通过nodeName可以访问树节点的名称。

context属性通常与selector属性组合使用，确定操作的来源。这通常用于插件的开发。从jQuery1.3版本开始，selector属性代表jQuery()提交的选择器。利用它，可以直接确定请求的选择器，并进一步在脚本中使用。这一信息可以与context属性相连使用，确定请求的准确情况。

注意

选择器将在后面的章节中更详细地讨论。

程序清单4.25 请求上下文（ch4_13.html）

```
...

06    <style type="text/css">
07      #output {
08        width: 800px;
09        background: green;
10        color: white
11      }
12    </style>
13    <script type="text/javascript"
14      src="lib/jquery-1.8.2.min.js">
15    </script>
16    <script type="text/javascript">
17      $(document).ready(function(){
18        $("#output").append(
19          "<tr><td>$(\"div:first\").context</td><td> " +
20          $("div:first").context + "</td></tr>")
```

```
21          $("#output").append("<tr><td>$(\"div\", " +
22            "document.body).context.nodeName</td><td> " +
23            $("div", document.body).context.nodeName +
24            "</td></tr>")
25          $("#output").append("<tr><td>$(\"img\")" +
26            ".context.nodeName</td><td> " +
27            $("img").context.nodeName + "</td></tr>");
28          $("#output").append( "<tr><td>$(\"img\", " +
29            "document.getElementsByTagName(\"div\")[0])" +
30            ".context.nodeName</td><td> " +
31            $("img", document.getElementsByTagName("div")
32              [0]).context.nodeName +
33            "</td></tr>");
34          $("#output").append("<tr><td>$(\"img\", " +
35            "document.getElementsByTagName(\"div\")[0])" +
36            ".selector</td><td> " +
37            $("img", document.getElementsByTagName("div")
38              [0]).selector +
39            "</td></tr>");
40        });
41      </script>
42    </head>
43    <body>
44      <div>
45        <img src="images/i1.jpg" />
46      </div>
47      <hr/><table id="output" />
48    </body>
49  </html>
```

上例展示了 context 的不同使用方法。nodeName 属性返回节点的名称，其他输出显示上下文本身和选择器。如图 4-12 所示。

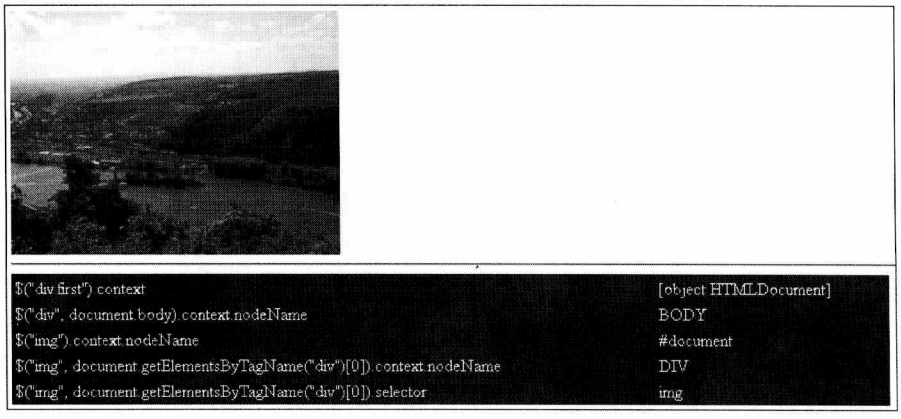

图 4-12 不同元素的上下文

4.10 链接 jQuery 对象

jQuery 的核心特征之一是框架的大部分方法本身的返回值是一个 jQuery 对象（或者一组 jQuery 对象）。因此，可以使用句点标记法实现方法调用的链接。我认为这是该框架的重点之一，可以用它实现极其紧凑的标记。我们将在动画技术中更清晰地看到这一点。但是，即使简单地在一个区域中附加内容，也可以用这种方法很紧凑而合乎逻辑地完成。

程序清单 4.26 链接 jQuery 方法（ch4_14.html）

```
...
06    <script type="text/javascript"
07       src="lib/jquery-1.7.2.js"></script>
08    <script type="text/javascript">
09      $(document).ready(function(){
10        $("#output").append("First this. ").
11         css({background:"red",color:"white"}).
12         append("<hr />").
13         append("And then this.");
14      });
15    </script>
16  </head>
17  <body>
18    <div id="output" />
19  </body>
20 </html>
```

在第 10 行中，可以看到文本通过 append() 方法添加到 div 区域。作为返回值，它再次返回一个 jQuery 对象。相应地，可以对这个对象应用 jQuery 方法。我们的例子中，第 11 行中应用了 css() 方法，用 CSS 格式化前述对象。正如面向对象编程中最常见的情况，链接符是句点（这在 JavaScript 下适用）。css() 方法的返回值也是一个 jQuery 对象，代表此时已经被处理的选中区域。对于这个对象，也可以依次引用 jQuery 方法（以此类推）。如图 4-13 所示。

图 4-13 链接 jQuery 方法

顺序执行函数调用 :jQuery 队列

与自调用、链接调用或者 jQuery 中通常会推迟的函数调用相关的延迟调用都保存在一个队列中。队列（Queue）这一术语也是该框架中负责队列任务的多个函数名称的一部分。

如果多个函数调用顺序执行，它们必须在一个队列中管理。严格地说，这种队列就是一个函数引用数组。jQuery 框架自动在后台管理这个队列（例如，在链式调用的情况下）。也可以使用各种方法人工访问这个队列（例如，queue() 和 dequeue() 或者 delay() 方法）。使用 queue() 方法，可以访问队列的第一个元素（第一个函数）。可以指定一个字符串参数，用它标识队列。默认标识为 fx。如果指定第二个参数，它是一个回调。可以在队列的最后添加一个新的函数，在以前的所有函数调用完成之后执行。如果指定另一个队列的名称作为第二个参数，现有队列被新队列所代替。也可以指定一个函数数组。

使用 dequeue() 方法，从队列的开始处取得一个函数并执行。方法有一个可选参数，是你想要取得函数的队列名称。delay() 方法是在 jQuery 1.4 中增加的，可以用它推迟队列中后面的方法的执行。版本 1.4 还添加了另一个方法 clearQueue()，用于删除队列中尚未执行的所有函数。和 queue() 一样，可以省略一个参数访问默认队列 fx，也可以指定字符串形式的队列名称。

4.11 版本 1.5 之后的新核心技术

在 1.5 版本中，[⊖]在框架的核心区域增加了少数新技术。为了完整起见，我们在此做简单的介绍，而不深入讨论。

注意

与 1.4.3 版本相比，在 1.5.x 找不到多少新特性。1.5.x 中的大部分更改都发生在后台（AJAX 是个例外）。在 1.3.x 到 1.4.x 的版本更改中，有较多方便不同编程的扩展。

4.11.1 jQuery.sub()

用 jQuery.sub() 方法创建 jQuery 对象的一个新拷贝，其属性和方法可以修改，而不会影响原始的 jQuery 对象。框架中引入这个方法有两个原因。其一，可以方便地用它重写 jQuery 中的方法，而不会损毁原始方法或者阻止访问。其二，可以更好地封装 jQuery 插件的命名空间。

4.11.2 jQuery.when()

jQuery.when() 很有趣。它让你描述一种 if 条件，如果出现该条件，可以执行一个或者多个对象的回调函数。这对于异步或者通常推迟发生的事件（例如 AJAX）特别有用。和 then() 方法链接使用，可以描述一个紧凑的响应系统。

我们不得不在一个示例中提前描述这种延迟事件。我们发送一个 AJAX 请求，在应答到达时响应。[⊖]

在官方文档中，你会找到如下的代码片段。

程序清单 4.27 在 AJAX 请求中使用 when() 的官方代码片段

```
$.when( $.ajax("test.aspx") ).then(function(ajaxArgs){
    alert(ajaxArgs[1]); /* ajaxArgs is [ "success", statusText, jqXHR ] */
 });
```

但是，这个例子在某些旧浏览器中会出现问题。原则上，程序清单 4.28 中所示的代码

⊖ 本书中的例子用版本 1.8.2 测试。

⊖ 如果你还不熟悉 AJAX，保持耐心；我们将在后面详细介绍。

应该能够正常工作。

程序清单 4.28 使用共同处理的多个延迟事件

```
$.when($.ajax("rjs.txt"), $.ajax("rjs2.txt")).
then(myFunction, myErrorMessage);
```

所以，可以创建多个延迟事件的公共处理。如果这些方法只传递唯一一个对象，说明没有延迟的响应，每个回调立即执行。

4.11.3 版本 1.6：有何新特性

前面已经提到，jQuery 中的修改和增加很重要。然而，这些修改主要是优化，对于框架的用户大部分不可见。但是程序员可以（也必须）注意和使用一些特殊的优化 / 修改。

如果在 http://api.jquery.com/category/version/1.6/ 上查看版本 1.6 的其他新特性，就会发现这些功能相当专业或者具有技术性，和“常规”的网站编程没有什么联系。例如，新的 deferred.always() 和 deferred.pipe() 方法用于处理延迟的对象和方法，jQuery.holdReady() 用于在 jQuery 中执行 ready 事件。此外还有一个 promise() 方法，它交付一个动态生成的对象，这个对象可以看作已经结束的所有前一个类型操作（例如动画或者内容更改）的承诺。触发这一对象时，可以模拟一个保证状态。这个方法的特殊性是相关的操作可以链接，也可以不链接。

attr()、prop() 和 removeProp()

attract() 方法的内部逻辑已经被重写。从 jQuery 1.6 起，DOM 特性（Attribute）和属性的处理被分开了。对于后者，jQuery 1.6 引入了 prop() 和 removeProp() 方法。新的 prop() 方法设置 DOM 元素的属性（Property，方法的名称即源自于此），或者返回对应的值。与此对应，removeProp() 删除属性。

背景是：在前面的版本中，jQuery 没有清晰地区分 DOM 特性和属性。但是这两者有小的区别。通常，DOM 特性代表着文档交付的某个 DOM 信息的状态（例如，<input type="text" value="abc"> 中 value 特性的值）。所以，特性是由 HTML 标记设置的静态值。与此相反，DOM 属性代表着文档的动态状态。这可能与静态预设值不同，如下面的例子所示。所以，如果用户在这种情况下在表单中输入一个值，prop("value") 方法返回用户输入而非预设值。到目前为止，你应该已经在表单元素中使用过 val()，但是你也可能已经使用过 attr ("value")。

在大部分情况下，浏览器返回 value 特性值，作为对应属性的起始值，但是对于 checked 或者 disabled 等布尔特性，这样做会带来一些问题（如果想要考虑 XHTML 逻辑，就必须避免这些布尔特性，这样就不会发生问题）。

在 jQuery 1.6 之前的版本中，attr("checked") 返回布尔属性值 true，但是从 jQuery 1.6 起，会得到一个空串（其他布尔属性也会发生同样的情况）。现在，可以用方法 prop() 处理，如程序清单 4.29 所示。

程序清单 4.29 适用于布尔属性值

```
$(this).prop("checked")
```

作为替代（和版本无关），对布尔属性值也可以使用 this。

程序清单 4.30 可以使用 is() 方法替代

```
$(this).is(":checked")
```

data()

另一项革新涉及 data() 方法。在版本 1.6 中，这个方法按照万维网联盟的 HTML5 规范进行了改编。更具体地说，用破折号连接的特性名称被转换为驼峰式大小写名称。这种标识符中的每个新词拼写中都有一个大写字母（正如 DOM 概念和 jQuery 的其他各个地方的用法那样）。这一事实与导入数据特性有关，因为在 jQuery 1.5 中，data() 方法引入了一个功能，自动用 JSON 语义将元素设置的所有数据特性转换为 JavaScript 值。jQuery1.6 中的新功能也有相同的效果。我们假定有一个特性 data-max-value="100"。

程序清单 4.31 jQuery 1.5.2 中的结果

```
{ max-value: 100 }
```

程序清单 4.32 jQuery 1.6 中的结果

```
{ maxValue: 100 }.
```

在 jQuery 1.6 做出的改变中，CSS 属性现在可以相对更改也值得一提。例如，可以使用 +、= 操作符。其他改进设计动画和特效，但是不特别引人注目。

4.12 小结

在本章中，我们学习了 jQuery 工作原理中最为重要的背景信息。这一背景基于 JavaScript 和 DOM 概念，因为 jQuery 大量利用它们的潜力。对用户的好处是，他们得到了这个框架（跨越许多种浏览器）中的强大技术。必须承认，不需要了解 JavaScript 就能够使用 jQuery 的陈述明显不正确。但是我们得到了网站编程的最简便方法，而纯粹的 JavaScript 或者原生 DOM 对象不能以相同的方式提供这些能力。当然，如果你是 JavaScript 专家，也可以自己编写程序，但这意味着重复劳动（需要巨大的工作量）。

本章介绍了 jQuery 中的特殊对象。访问网站的元素是重要的方面，这可以通过 jQuery 命名空间和 jQuery 对象，加上 jQuery() 和 $() 方法以及 jQuery 中的其他各种方法可靠地进行。

第 5 章

选择器和过滤器

本章的篇幅相当大，介绍如何使用 jQuery 选择网页的特定组件。这是网页中所有动态过程的基础，也是对单独网页区域的针对性格式化。经典的方法是通过名称、对象字段或者原生文档对象模型（DOM）方法选择，因为每个网页在浏览器中都被表现为 DOM。[⊖] 顾名思义，对象字段通过数组标记法（也只限于该对象）使用，只对少数元素有效。[⊜] 从 JavaScript 中通过名称访问也只在有限的范围内可能做到。[⊜] 至于可用的选项，原生 DOM 方法只允许很有限的选择选项（例如，只能选择一个 ID 或者一个类、几种关系规范，或者只能指定超文本标记语言 [HTML] 元素）。结论是，在经典的 DOM 访问中，许多可以想象的选择选项根本不存在、效果不同或者只得到少数浏览器的支持。

无疑，jQuery 框架的最大优势是，可以使用该框架，以简单、可靠、大量和统一的方式，在所有常见浏览器中选择元素。本质上，可以通过 jQuery() 方法及其别名 $() 和选择元素的参数，确定所选元素。

注意

这种针对性选择的重大意义在框架的名称中也有体现，这就是 Query（查询）一词。选择网页中的一个或者一组元素时，指定的是一个基于选择器和过滤器的查询。我认为，指定选择条件的大量选项，是整个框架中最重要的特性。特别是，它们组成了通过样式表动态操纵网页的基础。jQuery 还提供了一系列有趣的方法，我们在本章中使用它们进行视觉演示，但是更详细的讨论放到稍后的章节中。

⊖ 但是正如我们在前面的例子中说明的那样，DOM 在不同浏览器中可能不同。

⊜ 但是，在历史上，这是访问网页组件的最早的选择之一。这种方法的局限性是，仅对 Netscape1995 年左右第一次草拟的对象模型（DOM 概念的基础）包含的元素有效（例如，表单、小程序、链接或者图片）。有时候，某些信息来源也用 HTMLCollection 代替对象字段。

⊜ 实际上，这种方法也只在 Netscape 有意设置对象字段的那些元素上能够可靠使用。本质上，在经典的 DOM 访问中，通过名称访问仅限于图片、表单或者表单元素。

5.1 基础知识

我们首先来澄清选择器（Selector）的准确含义。本章围绕这一概念进行介绍。

注意

在本章和后续的章节中，我们多次使用 Firebug 展开说明。在安装这个 Firefox 附加程序之后，可以通过“工具”菜单或者在上下文菜单中选择“用 Firebug 检查元素”来打开 Firebug。

5.1.1 什么是选择器

在 jQuery 中，选择器指的是在网页树形结构中选择一个或者多个元素的规范。你可能从层叠样式表（CSS）中已经熟悉了这个术语，在可扩展标记语言（XML）中，也有多种选择一个元素或者一个节点的技术。

提示

如果在 jQuery 的任何函数或者方法中指定选择器为参数类型，可以写下 jQuery 构造器所能接受的任何类型，例如字符串、元素或者元素列表。

另一方面，jQuery 中的选择器很容易应用。加上读者已经有了 CSS 的知识，也就已经有了选择器的基础知识。不过，jQuery 中的特定选择器极其强大，这使它们更加复杂，更准确地说，是更加广泛。因此，用单独的一章讨论它们是合适的。

5.1.2 什么是过滤器

除了选择器，jQuery 还使用过滤器（Filter）。过滤器用进一步的条件限制已经选中的元素组。例如，如果你已经使用一个选择器选择了网页中的所有图片元素，可以对该集合进行过滤，只获得其中的第二幅图片。在 jQuery 框架的旧版本中，选择器和过滤器之间有明显的区别，但是在较新的版本中，几乎全都使用选择器这一术语，过滤器只以几个方法的形式出现。我们也按照这种方式，只在合适的时候使用特定的过滤器术语。

5.1.3 作为基础的 XPath

为了理解 jQuery 中选择器和过滤器的工作原理，理解描述选择器和过滤器的各种标志，有必要更仔细地看看浏览网页树形结构的方法。我们简单地介绍一下 XPath，对这种相当复杂的 XML 技术不做过于详细的讨论。

XML 路径语言（XML Path Language，XPath）是 W3C 为访问 XML 文档（或者树形结构 /DOM）组件开发的一种查询语言。XPath 对 XML 的作用往往被比作数据库中 SQL 的作用——可以用于许多情况下的通用查询语言。

注意

在jQuery中，选择器和过滤器由经典的CSS选择器和XPath组合而成。用附加的XPath子集扩展CSS（版本1.3）可能是个好的描述。真正的XPath选择器在jQuery中可以插件的形式使用。

本质上，可以使用XPath表达式[⊖]访问树型结构文档中的特定内容。这些选择的核心概念是树或者DOM。

树、节点和元素

树由发源自单一根的节点和轴（Axes）组成。树的节点都包含元素[⊖]。元素中包含的下级元素称为子元素。在树的层次结构中优先于子元素的元素被称作父元素。

所有其他节点与包含它们的节点属于树的同一个层次，这些节点被称为兄弟节点。

轴被看作树的主干或者分支，重视元素声明的顺序（从树根开始顺序查看）。相应地，树中有祖先和后代。

我们来看看以这种方式引用元素的一些XPath术语，见表5-1。

表5-1 选择的XPath术语

标　志	访问的节点
祖先	所有前驱节点。引用树中当前节点的所有祖先，直到树根
子女	直接从属于当前节点的所有节点
后代	直接或者非直接从属于当前节点的所有节点
后续	所有按照树的顺序在当前节点之后的节点
后续兄弟	与当前节点在同一个层次级别（同一个父节点的兄弟节点），而在顺序上在当前节点之后的所有节点
父	当前节点的直接前驱
前驱	所有按照树的顺序在当前节点之前的节点
前驱兄弟	与当前节点在同一个层次级别（同一个父节点的兄弟节点），而在顺序上在当前节点之前的所有节点

XPath还提供了名为节点测试的一种机制。节点测试由一个轴表达式加上两个冒号和一个测试节点组成。可以指定进一步限制一组选中节点的方法（过滤器）。在jQuery中可以找到相同的逻辑，但是只使用一个冒号作为分隔符，而且（如前所述），我们所谈到的新版本中的选择器通常指的是由一个（可选）前驱选择器、一个冒号再加上后续的过滤器组成的表达式。最终，重要的是它的效果，统一的术语也强调了这一点。

在jQuery中，选择器和过滤器由类型区分，但是这种区别在文档中也变得越来越不明显。我们在后续的小节中仍然按照主题单独介绍选择器，首先用一两张表格描述一组逻辑上属于同类的选择器，然后介绍示例。

注意

本书中的选择器分组不是标准化的，可能有重叠，某些选择器可能被分到其他类别中。

⊖ 以及类似的结构化选择技术。

⊖ 我们对XPath中关于特性、注释、命名空间等的细节不做详细介绍。这里的元素是通用意义上的概念。

我们的分组采用启发式规则，其中一部分也采用了我自己的分类方法。效果和应用于选择器所在的组别无关。

5.2 基本选择器和层次化选择器

注意

可能需要在 jQuery 选择器中使用特殊字符。在这种情况下，这些字符必须通过斜杠前缀（\）屏蔽，这和 JavaScript 中一样。

jQuery 中的基本选择器分类下的选择选项与经典的 CSS 选择器对应。通过一个元素、一个 ID 等，可以访问网页的组件，结果是获得一个元素或者一个元素数组。[⊖]

正如 XPath 的介绍中所述，也可以用父子关系、兄弟关系和祖先或者后代关系来描述在树中的位置。这些关系在 CSS 中也会出现。

这样的选择可以非常舒服地描述属于同一层次级别的一组元素。因此，有些信息来源将这些选择器称作层次化选择器，从功能上将它们与基本选择器区分开来。我们不想研究基本选择器和层次化选择器在大部分情况下的具体区别；这对于它们的应用毫无影响。具体的列表见表 5-2。

表 5-2 基本选择器和层次化选择器

名 称	描 述
`#id`	指定一个 ID 获取特定元素。注意 # 前缀。通常，只会得到一个元素，因为 ID 在一个文档中应该是唯一的。但是因为浏览器的容错功能，这种唯一性在文档中得不到保证。尽管 jQuery 能够应付这种情况，但是页面充满错误，无法正常编程。在那种情况下，可能必须更正文档
`element`	指定一个特定元素为选择器，或者所有与该名称匹配的元素组成的数组
`.class`	获得与指定类匹配（class 特性或者动态指定的类）的所有元素组成的数组。注意选择器的句点前缀
`*`	全局选择器，选择所有元素
`selector1, selector2, ..., selectorN`	不同选择器的组合。你将获得匹配其中某个指定条件的元素或者所有匹配元素组成的数组
`ancestor descendant`	这种层次化选择返回树中前导的祖先元素的一个后续元素，或者所有后续元素组成的数组。通过第二个选择器选择的元素必须是第一个选择器结果的后代。这也被称为上下文敏感的选择器，因为选中的元素处于另一个元素（上下文）之中。至于标记方法，后代元素简单地跟在祖先元素之后，用空格分隔

⊖ 严格地讲，数组可以只有一个元素，甚至为空。

（续）

名　称	描　述
.class1 .class2	这种选择返回与一组类匹配（class 特性或者动态指定的类）的一个元素，或者多个元素组成的一个数组。匹配列表同时包含被指定了两个（或者）多个类的元素。注意，句点前缀代表类的链接。但是，这一语句有些难度，因为它并不意味着一个元素有两个或者更多个被当作单独特性赋予的类，而是存在后代嵌套，父节点被赋予了前导的类。这是层次化后代规则的一个特例
parent > child	层次化选择的这一变种返回选中父元素的子元素或者所有子元素组成的数组
prev + next	第二个选择器指定的元素必须是第一个选择器指定元素的直接后续兄弟元素。这样，你将得到一个类型由 next 指定，在 prevous 指定的元素之后的元素。这也是一种层次化选择
prev ~ siblings	这种层次化选择返回第一个指定元素之后，对应于第二个选择器过滤的兄弟元素，或者所有这些元素组成的数组

5.2.1 示例

注意

在如下的例子中，我们主要设置网页元素的 CSS 属性，因为这样在打印出来的屏幕截图上效果最明显。但是，选择器和过滤器的应用绝不仅限于此！

在下面的每个例子中，我们使用如下的 HTML 文件，该文件包含我们可以演示基本选择器和层次化选择器的主要结构。为此，该 HTML 文件包含了类的指定、ID、子元素、后代等（ch5_1.html）。

程序清单 5.1　基本结构

```
...
05    <script type="text/javascript"
07       src="lib/jquery-1.8.2.min.js"></script>
08    <script type="text/javascript"
09       src="lib/ch5_1_ready.js"></script>
10  </head>
11  <body>
12    <h1 class="c2">Basic Selectors</h1>
13    <div class="c1">
14      <span class="c1">
15         The example shows the effect of
16         <span class="c2">basic selectors</span>
17         in jQuery</span>
18      <div>
19        To demonstrate this, different
20         <span class="c1" class="c2">structures</span>
21        are nested.
22      </div>
23      <div>
```

```
24        <img src="images/i1.jpg" id="i1" />
25      </div>
26    </div>
27    <p class="c1">
28      <span class="c1" class="c2">First variant</span>
29    </p>
30    <img src="images/i2.jpg" id="i2"/>
31  </body>
32 </html>
```

注意，我们在 HTML 标记中使用 ID、指定类，有意不使用任何样式表。CSS 格式化由 jQuery 动态完成，这样就能够看到不同选择器的效果。没有任何格式的网页如图 5-1 所示。

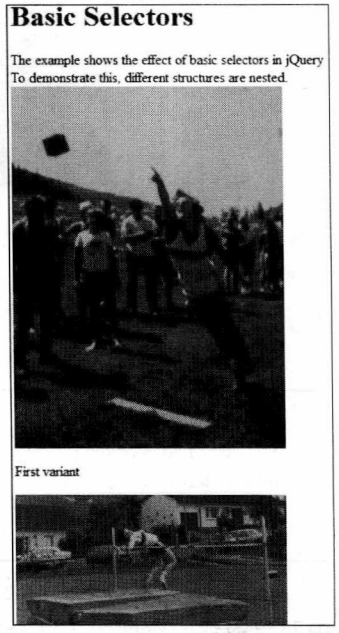

图 5-1 没有任何格式的网页

注意

两个标记重复使用了 class 特性。这一般不是好主意，大部分浏览器将造成 CSS 格式化[⊖]中只使用一个类（第二个类会被忽略）。我们使用双标记法是为了更清晰地演示选择器的效果。

注意

为了清晰起见，我们一次只测试一个选择器的效果。所以我们使用了许多小的示例。HTML 文件的修改只有引用的 JavaScript 文件名，这些文件名对应于 HTML 文件名，只有少量文本更改。因此，我们就不需要再关注 HTML 文件，或者明确地提及它。

下面是第一个 JavaScript 文件（ch5_1_ready.js）。

程序清单 5.2 选择一个 ID

```
01 $(function(){
02   $("#i2").css({
03     position: "absolute", top: "100px", left: "400px"
04   });
05 });
```

上述程序清单的含义不言自明。在这个例子中，我们通过 ID 选择网页中的第二幅图片（我们在本书中已经多次这么做了）。页面如图 5-2 所示。作为例子，jQuery 代码对该元素应用绝对定位。

⊖ 在这里特意避开了 CSS 格式化。

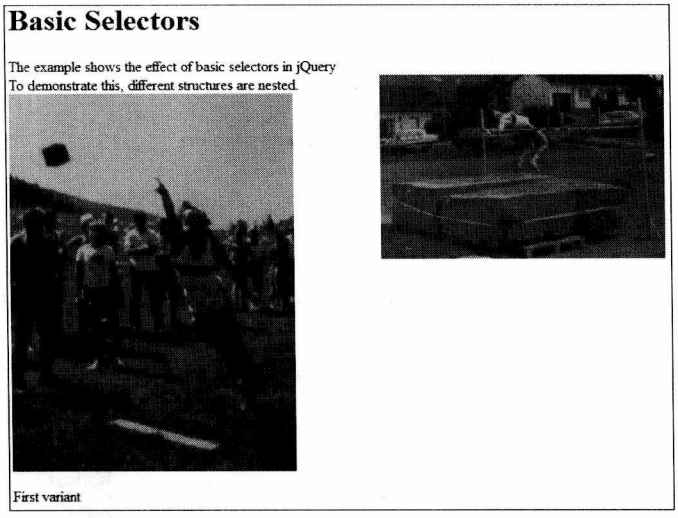

图 5-2 第二幅图片被选中并定位

现在，我们来简单地看一下元素类型的选择（ch5_2_ready.js）。

程序清单 5.3 选择一个元素

```
01 $(function(){
02  $("img").css({
03    width: "100px"
04  });
05 });
```

在第 2 行中，所有某个类型（这里是 img）的元素被选择。这里，我们精确地指定网页中两幅图片的宽度为 100 个像素。缩小后的图片如图 5-3 所示。

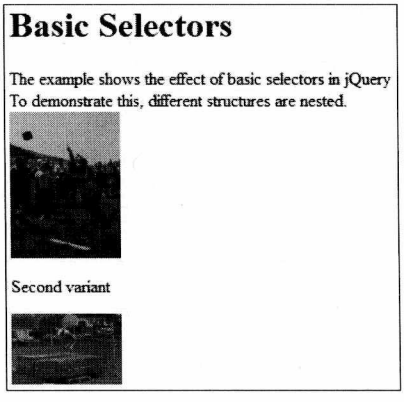

图 5-3 图片被缩小

我们转向通过类的选择（ch5_3_ready.js）。

程序清单 5.4 选择一个类

```
01 $(function(){
02  $(".c2").css({
03    background: "black", color:"white"
04  });
05 });
```

在这个例子中，HTML 文件中的所有标记为 c2 类的元素都被选中。在我们的 HTML 代码中，这些元素是标题和一个 span 容器。效果如图 5-4 所示。

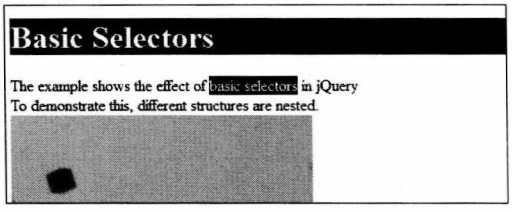

图 5-4 标记为 c2 类的元素背景变为黑色，文字变为白色

目前为止，一切都很简单。下一个例子使用两个类作为选择器，用空格分隔（ch5_4_ready.js）。

程序清单 5.5 指定两个类

```
01 $(function(){
02  $(".c1 .c2").css({
03    background: "black", color:"white"
04  });
05 });
```

乍一看，这个例子似乎很简单，因为 HTML 文件中有多个元素被指定了 c1 和 c2 类：

```
...
20      <span class="c1" class="c2">structures</span>
...
28      <span class="c1" class="c2">Fourth variant</span>
...
```

但是，格式却没有应用到这些元素上，你可能怀疑是因为标记中两次使用了同一个特性——如上所述，这是不建议的做法——因为这种做法没有得到支持。你也可能认为 jQuery 在这方面提供了其他的可能性。但是情况并非如此。因此，这个例子并不简单。例子中的代码选中了如下文本：

```
...
14      <span class="c1">
15         The example shows the effect of
16         <span class="c2">basic selectors</span>
...
```

你能理解为什么会出现这种情况吗？在前两种情况下（第 20 行和第 28 行），一个标记指定了两个类，但是它不是一个后代声明。在后一种情况下，span 元素是 P 元素的后代。P

元素指定了 c1 类，作为 P 元素后代的 span 具有 c2 类。这就是该区域被选中的原因。

我们通过指定元素再演示一次后代规则。为此，我们造成一个异常，而且应用第二个选择器——全局选择器。效果如图 5-5 所示。这很明显，但是为了完整性起见，例子中包含了它（ch5_5_ready.js）。

程序清单 5.6 全局选择器和后代

```
01 $(function(){
02  $("*").css({
03    background: "gray", color:"yellow"
04  });
05  $("div span").css({
06    background: "black", color:"white"
07  });
08 });
```

我们首先看到全局选择器被使用，没有什么需要解释的。在第 5 行中，我们看到对 div 区域中的所有 span 区域的选择。更确切地说，HTML 文件还包含了不作为 div 区域后代的 span 后代，这些区域不会被选中。效果如图 5-6 所示。

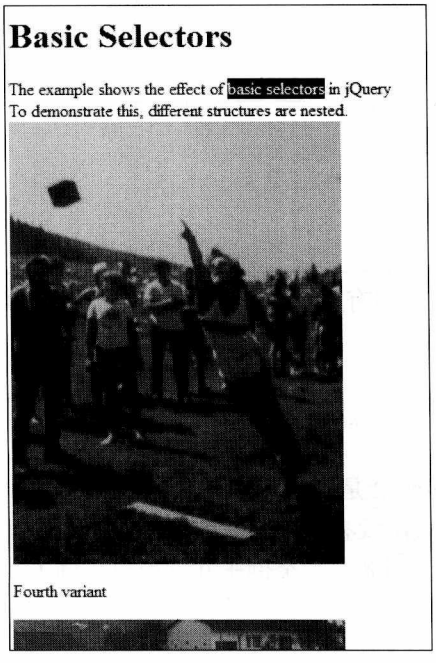

图 5-5 这个选择可能令人吃惊

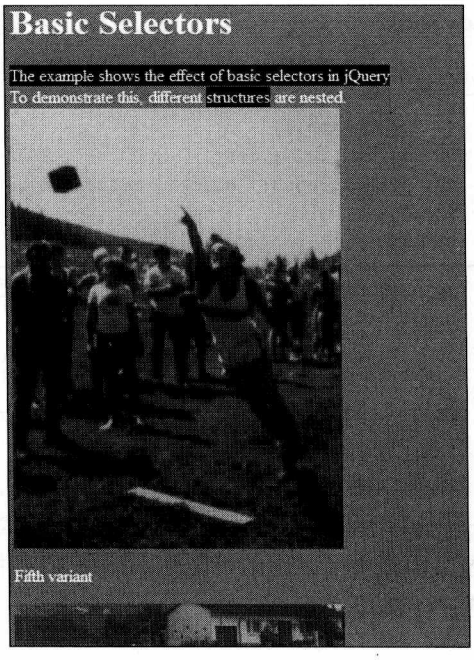

图 5-6 只有 div 区域中的 span 区域才会被格式化

现在，你将看到父子关系的选择（ch5_6_ready.js）。

程序清单 5.7 父子关系

```
01 $(function(){
02 $("div > img").css({
03   position: "absolute", top: "100px", left: "400px",
04    width:"50px"
05  });
06 });
```

我们指定一幅作为 div 区域子元素的图片。如图 5-7 所示。在我们的例子中，就是第二幅图片。

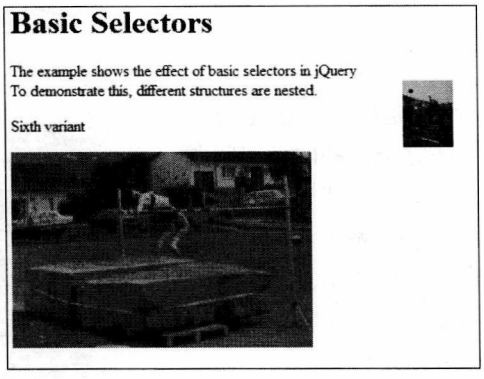

图 5-7 这幅图片是 div 区域的子元素

本小节的倒数第二个例子关注直接兄弟关系（ch5_7_ready.js）。

程序清单 5.8 div 区域必须跟在 span 区域之后

```
01 $(function(){
02  $("span + div").css({
03    background: "black", color: "white"
04  });
05 });
```

在上述选择中，指定 div 区域必须紧跟在同一级别的 span 区域（兄弟）之后。效果如图 5-8 所示。

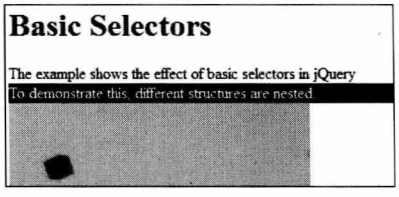

图 5-8 跟在 span 区域之后的 div 区域被格式化

在本节的最后一个例子中，可以看到另一种兄弟关系，它还选择非直接的兄弟（ch5_9_

ready.js)。

程序清单 5.9 另一种方式的兄弟关系

```
01 $(function(){
02  $("span ~ div").css({
03    background: "black", color: "white"
04  });
05 });
```

在上述选择中，指定选择与 span 区域同级别（兄弟）、在其之后的 div 区域。效果如图 5-9 所示。

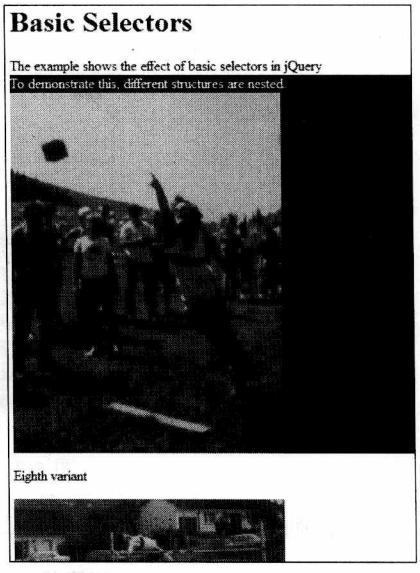

图 5-9 在两个 div 区域中的文本和图片被选中并格式化

5.2.2 潜在的问题

使用所有基本选择器（特别是所有层次化选择器）时，需要注意一些 jQuery 无法完全弥补的潜在问题。

例如，指定选择器时，要记得源代码中标记的顺序可能是有影响的（特别是与 CSS 赋值相关时，两条规则如相互冲突，后面的规则可能覆盖前一条规则）。另外，如果预期在后面才声明的对象，document.ready() 方法中标记的顺序也是有影响的。

更严重的是，在使用层次化选择器时必须小心，因为某些规则的效果可能与预期的不同。在某些情况下，这些差异的原因并不清楚。例如，可以指定选择某个段落中的一个 div 元素，也可以指定用于某个 div 元素中的一个段落的规则。但是在后一种情况下，选择器可能无效。严格地说，这并不是选择器或者 jQuery 受到限制或者出现错误；相反，必须注意浏览器渲染 DOM 的方法[⊖]。可以在 Firebug 中非常清晰地看到。

看看下面的文件（ch5_9_error.html）。

程序清单 5.10 某个段落中的一个 div 元素

```
...
08    <script type="text/javascript">
09      $(function(){
10        $("p div").css("background", "red");
11      });
12    </script>
13  </head>
14  <body>
15    <p>
16      Many thanks.
```

⊖ 最终，这意味着你在纯粹的 CSS 中也可能遇到问题。

```
17      <div>
18        I look forward to hearing from you.
19      </div>
20    </p>
21  </body>
22 </html>
```

如果尝试这个示例，CSS 规则的赋值无法正常工作。原因是，段落和 div 容器在实际渲染的 DOM 中不再是嵌套的（即使在 HTML 中嵌套它们，标准也不允许这样）。在段落中只能有文本，而 div 容器在树中被安排在段落之外。结果是，整条后代规则都不再正确。效果如图 5-10 所示。

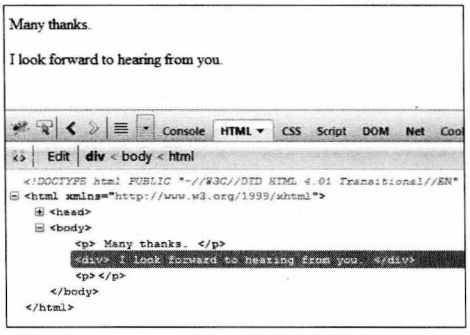

图 5-10 div 区域在浏览器渲染时被放在段落之外，选择器不再起作用

这种行为不是随机发生的，而是严格基于 HTML 规则。但是，由于容错原则，很少有人坚持这些规则的严格解读。但是正如你从我们的例子中可以看到的那样，在某些情况下这种宽松可能产生错误。

5.3 过滤选择器

jQuery 中的过滤选择器大约相当于 XPath 中的节点测试。利用这些选择器，可以为前面的元素组规定进一步的条件，两者之间用冒号分隔。在本节中，我们按照效果将这些过滤器分为几个类别。

提示

可以编写一个没有前导选择器的过滤器，仅从冒号开始。这会将过滤器应用到网页的所有元素，也可以链接任何过滤器。只要一个接着一个写下它们，用冒号分隔即可。

5.3.1 基本过滤器

jQuery 中有如下基本过滤选择器。

表 5-3 所有基本过滤器

名　称	描　述
:first	前导选择器描述的元素组中的第一个选中元素
:last	前导选择器描述的元素组中的最后一个选中元素
:not(选择器)	不匹配选择器的所有元素组成的数组（当然可以为空）。匹配选择器的元素被过滤掉
:even	索引结构（如表格和列表）中索引值为偶数（从 0 开始）的所有元素
:odd	索引结构（如表格和列表）中索引值为奇数的所有元素

（续）

名　称	描　述
:eq(索引值)	通过索引指定一个元素
:gt(索引值)	索引值大于（gt—greater than）指定值的所有元素组成的数组
:lt(索引值)	索引值小于（lt—less than）指定值的所有元素组成的数组
:header	所有标题（h1、h2、h3 等）组成的一个数组
:animated	所有当前正在动画的元素组成的数组
:focus	选择当前处于焦点上的元素

程序清单 5.11　各种过滤器示例所用的基本文件

```
...
08    <script type="text/javascript"
09       src="lib/ch5_10_ready.js"></script>
10  </head>
11  <body>
12    <h1>Robert Lee Frost</h1>
13    <h3>(1874-1963)</h3>
14    <h4>US poet</h4>
15    <div>The brain is a wonderful organ.</div>
16    <div>It starts working</div>
17    <div>the moment you get up in the morning</div>
18    <div>and does not stop</div>
19    <div>until you get into the office.</div>
20    <h1>Franz Kafka</h1>
21    <h3>(1883-1924)</h3>
22    <h4>German author</h4>
23    <h2>Life</h2>
24    <div>A continual distraction</div>
25    <div>that does not even allow us to reflect</div>
26    <div>on that from which we are distracted.</div>
27  </body>
28 </html>
```

在上述例子中，可以看到一个网页，其中有几个标题和一个由多个 div 区域组成的结构。没有格式的页面如图 5-11 所示。

为保持简单，我们没有在一个例子中应用所有过滤器，而是同时最多应用三个过滤器。

我们首先看看选择动画元素的过滤器。原则上，这是我们在本节中讨论的最困难的情况。首先，我们需要在网页中为某些元素产生动画，才能使用这个过滤器。这也由 jQuery 自身完成（ch5_10_ready.js）。

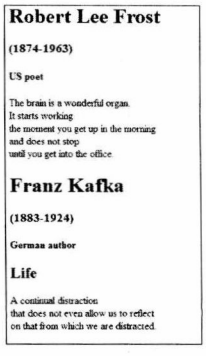

图 5-11　没有格式化的页面

程序清单 5.12 选择动画元素

```
01 $(function(){
02  animation();
03  $(":animated").css({
04    background: "black", color: "white",
05    textAlign: "center"
06  });
07 });
08 function animation(){
09  $("h2").slideToggle("slow", animation);
10 }
```

这个例子说明了几个有趣的现象。其一是第 3 行中的过滤器没有使用前导选择器选择网页上的所有动画元素。

具体地说，我们为所有类型为 h2 的元素播放动画，在我们的例子中只有一个这样的元素。我们将这个元素居中并改变它的背景和前景颜色。效果如图 5-12 所示。

为了播放元素的动画，我们使用 jQuery 的 slideToggle() 方法。注意，我们在这里使用递归调用，所以动画永久运行。这导致标题不断隐藏和显示，但是现在这并不重要。

重要的是，动画函数的第一次调用必须在过滤器应用之前发生。注意第 2 行。否则，标题在应用过滤器时不会被当作动画元素，尽管它在网页中是动画的。[⊖]

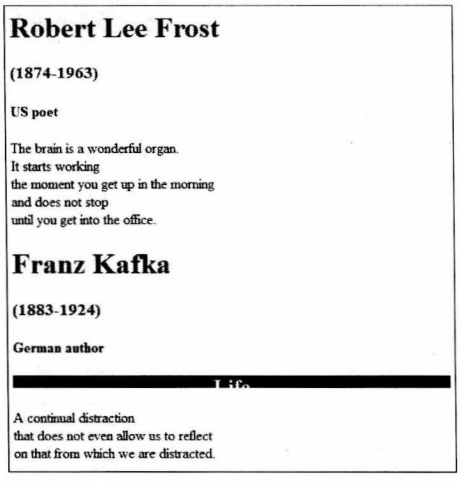

图 5-12 第 2 级标题被动画和格式化

现在，我们来看看选择第一个和最后一个元素以及所有标题的选择器（ch5_11_ready.js）。

程序清单 5.13 第一个和最后一个 div 类型的元素以及所有标题

```
01 $(function(){
02  $("div:first").css({
03    background: "black", color: "white",
04    textAlign: "center"
05  });
06  $("div:last").css({
07    background: "yellow", color: "blue",
08    textAlign: "right"
09  });
10  $(":header").css({
```

⊖ 这种情况在前面关于选择器和过滤器的潜在问题的小节中曾经提及。

```
    background: "lightgray"
});
});
```

这三个过滤器的效果可能很明显。第一个和最后一个 div 类型的元素和所有标题被选中，每个元素都被应用不同的格式。效果如图 5-13 所示。

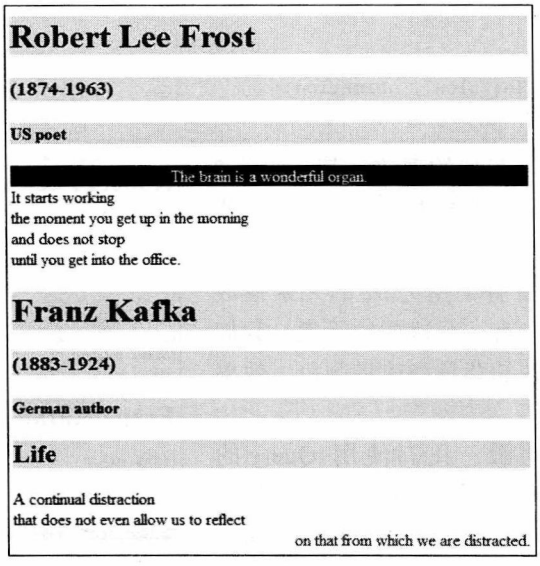

图 5-13 效果很明显

我们来看看如何选择索引值为奇数和偶数的元素（ch5_12_ready.js）。

程序清单 5.14 选择奇数和偶数索引

```
$(function(){
 $("div:odd").css({
   background: "black", color: "white",
   textAlign: "right"
 });
   $("div:even").css({
   background: "yellow", color: "blue",
   textAlign: "left"
 });
});
```

在前导选择器中，我们选择了网页中所有类型为 div 的元素，然后使用过滤器区分奇数和偶数索引。[⊖]效果如图 5-14 所示。

⊖ 注意，索引从 0 开始。

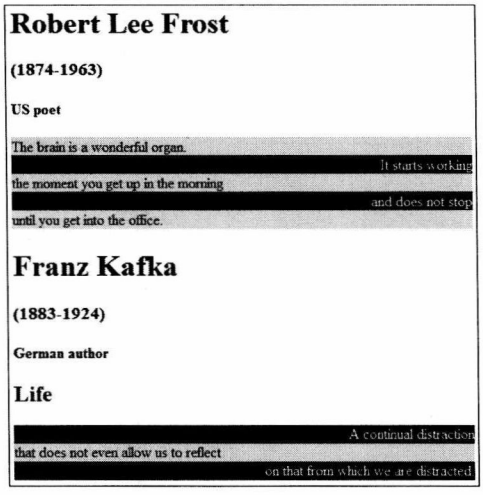

图 5-14 奇数和偶数索引交替格式

现在，我们明确地指定一个索引值或者索引区间，选择相应的元素（ch5_13_ready.js）。

程序清单 5.15 指定一个索引区间或者索引

```
01 $(function(){
02  $("div:gt(1)").css({
03    fontStyle: "italic", textAlign: "center"
04  });
05  $("div:lt(3)").css({
06    textDecoration: "underline"
07  });
08  $("div:eq(3)").css({
09    background: "#9999ff", textAlign: "right"
10  });
11 });
```

如你所见，从 1 开始（不含）的所有 div 元素以斜体显示并居中（换句话说，从第 3 个元素开始）。到索引值 2 为止（不含）的元素被加上下划线。对于第 3 个元素，两种效果都应用，它们只指定附加的格式化效果。

第 4 个元素设置了一个背景颜色并向右对齐。作为最后一条规则，右对齐覆盖了 gt() 过滤器应用的居中规则。效果如图 5-15 所示。

最后，我们来看通过 not() 进行的反选，这导致元素的排除（ch5_14_ready.js）。

程序清单 5.16 排除元素

```
01 $(function(){
02  $(":header:not(h2)").css({
03    fontStyle: "italic", textAlign: "center"
04  });
05 });
```

如前所述，也可以链接任何过滤器。只要一个接一个地写下不同过滤器，用冒号分隔，正如这里的演示。首先，我们选择所有标题，然后排除类型为 h2 的标题。效果如图 5-16 所示。

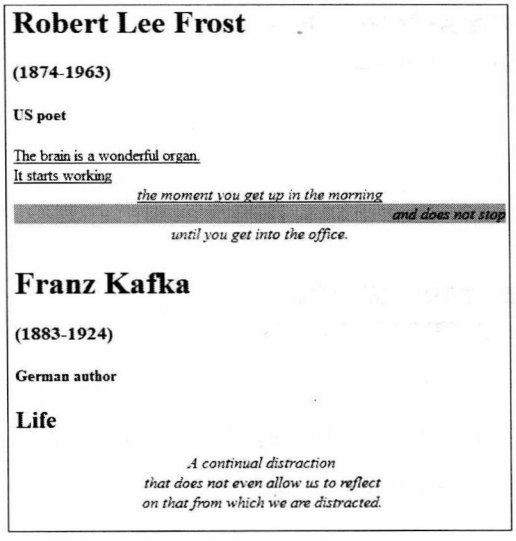

图 5-15 通过一个索引或者索引区间选择

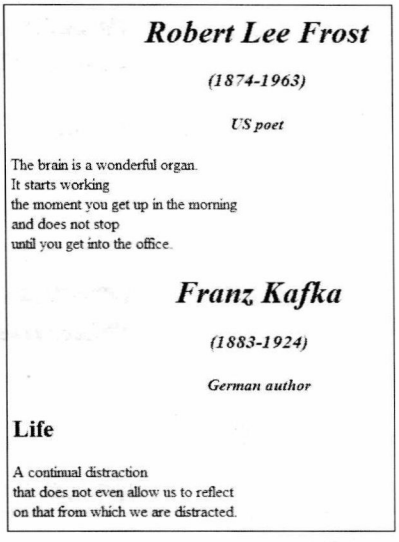

图 5-16 除了类型为 h2 的标题之外，所有标题都以斜体居中显示

5.3.2 内容过滤器

对于请求，元素的内容可能非常有趣。jQuery 中的内容过滤器可以通过评估内容选择元素，但是只有相当普通的功能。内容过滤器如表 5-4 所示。

表 5-4 内容过滤器

名 称	描 述
:contains(文本)	包含指定文本的所有元素组成的数组
:empty	没有子元素的所有元素组成的数组，包括文本节点。这些元素被称为空元素
:has(选择器)	包含至少一个指定选择器类型所属类型元素的所有元素。这个元素不一定是直接子元素，也可以出现在更深的结构中
:parent	所有父元素。这意味着这些元素有子元素，包括纯文本节点。它们是空元素的反面

我们为前几个例子中的 HTML 页面应用内容过滤器。

程序清单 5.17 根据“You”一词进行选择

```
01 $(function(){
02   $("div:contains(you)").css({ background:'cyan' });
03 });
```

可以看到，上述代码选择包含“You”一词的 div 元素。效果如图 5-17 所示。

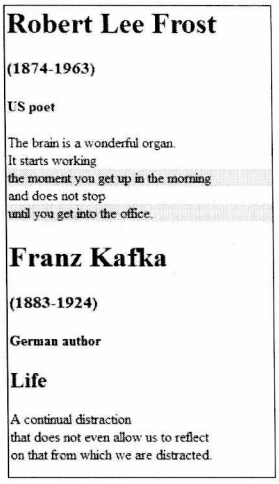

图 5-17 第 3 和第 4 个 div 区域包含“You”一词

对于其他三种过滤器，我们需要稍微不同的网页结构，如程序清单 5.18 所示（ch5_16.html）。

程序清单 5.18 新的基本文件

```
...
08    <script type="text/javascript"
09       src="lib/ch5_16_ready.js"></script>
10  </head>
11  <body>
12    <div>I love deadlines. I like the
13       <span>whooshing</span>sound they make
14    </div>
15    <div>as they fly by.</div><hr/>
16    <div>Douglas Adams</div>
17    <div><span>(1952-2001)</span></div>
18    <span>English writer and dramatist </span>
19  </body>
20  </html>
```

现在，我们的 div 区域中有 span 区域，可以演示 has()。加上第 15 行包含一个类型为 hr 的空元素，可以展示 empty 和 parent 的效果。我们用 JavaScript 文件 ch5_16_ready.js 完成。

程序清单 5.19 不同的内容过滤器

```
01 $(function(){
02  $("div:has(span)").css({
03    background: 'cyan'
04  });
05  $(":empty").css({
06    width: '200px'
07  });
```

```
08  $(":not(:parent)").css({
09    height: '5px'
10  });
11 });
```

在第 2 行中，我们选择所有包含 span 区域的 div 区域。这样的 div 有两个。在第 5 行中选择所有空元素。在我们的例子中，只有分隔线符合，代码指定了它的宽度。在第 8 行中，我们再次选择分隔线。当然，如果我们已经用 empty 过滤元素，这次附加选择就没有必要。但是可以看到 parent 的效果。我们选择所有不是父节点的元素（以及空元素）。这里的格式化代码在宽度规格的基础上，为分隔线添加了一个高度规格。父元素和子元素如图 5-18 所示。

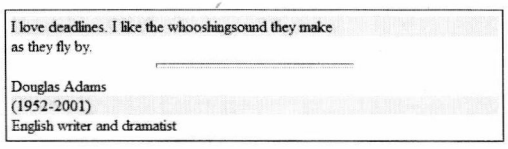

图 5-18 父元素和子元素

5.3.3 可见性过滤器

可见性过滤器如表 5-5 所示。

表 5-5 可见性过滤器

名 称	类 型
:hidden	选择网页中所有隐藏的元素。乍一看，这是相当简单的描述，但是要注意某些细节。网页中的元素可能因为各种原因隐藏： • CSS 属性 display 被设置为 none • 它们是隐藏的表单字段（type="hidden"） • 宽度和高度被设置为 0 • 某个祖先元素可能被隐藏，所有包含的子元素也就不会显示 此外还有 CSS 属性 visibility:hidden，和通过 opacity:0 设置透明度。注意，这些属性在 jQuery 中不被看作隐藏（可能与你的期望不同），因为它们处于可视状态时，仍然在网页中占据某些空间。这个过滤器或者 jQuery 只将网页中不占据任何空间的元素视为隐藏元素。⊖ 还需要考虑，如果元素通过动画隐藏，它们如何分类。只要动画还没有结束，它们就被看作是可见的。相反，在动画中显示隐藏元素时，元素在动画开始时立刻被视为可见元素，而不再是隐藏元素
:visible	所有可见元素，和 hidden 的解释完全相反

我们来看一个例子（ch5_17.html）。

程序清单 5.20 4 幅图片和一个按钮

```
...
08    <script type="text/javascript"
09       src="lib/ch5_17_ready.js">
10    </script>
11  </head>
12  <body>
13    <img src="images\i1.jpg" />
14    <img src="images\i2.jpg" />
```

⊖ 在 jQuery 1.3.2 之前的版本中，处理方式不同。

```
15    <img src="images\i3.jpg" />
16    <img src="images\i4.jpg" />
17    <br/>
18    <button>Switch</button>
19  </body>
20 </html>
```

这个例子显示一个有 4 幅图片和一个按钮的网页。所有有趣的事情都在 JavaScript 文件 ch5_17_ready.js 中发生。

程序清单 5.21 使用 :hidden 和 :visible

```
01 $(function(){
02  $("img:odd").css({
03    display: 'none'
04  });
05  $("button:eq(0)").click(function(){
06    $("img:visible").hide('slow');
07    $("img:hidden").css({
08      display: 'inline'
09    });
10  });
11 });
```

在第 2 行中，所有索引为奇数的图片都被隐藏，在下面的测试中，它们仍然被 :hidden 所选择。

在点击按钮的响应方法中（注意 eq 过滤器），当前被隐藏的图片现在应该可见，而可见的图片被隐藏（换句话说，切换效果）。

现在要记住，通过动画隐藏的元素如何理解。只要动画还未结束，它们就被看成是可见的，可以使用这种行为与可见性过滤器结合，节约许多编程工作。

决定性的因素是时间。在第 6 行中，过去可见的元素——通过 :visible 识别——变为不可见（但是采用动画形式，需要花费一些时间）。如果脚本流到达第 7 行，选择隐藏的元素（通过 :hidden 识别），点击事件中触发的可见元素仍然被 jQuery 看作可见的！这样，如果希望切换多个元素的隐藏和可见状态，就没有必要使用查询和决策结构。[⊖]页面的原始状态如图 5-19 所示。

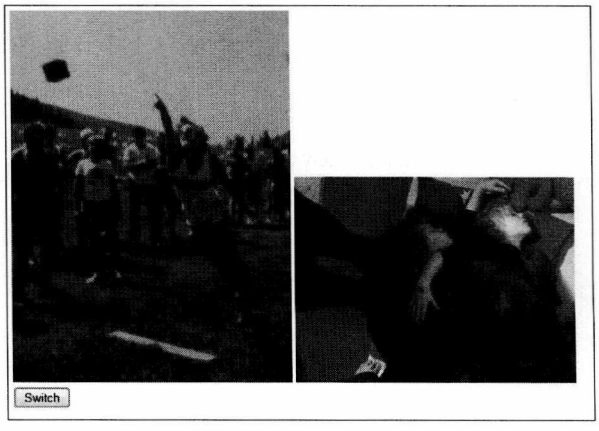

图 5-19 页面的原始状态

⊖ 这也是一种可能的选择。

如果点击按钮，就会看到之前不可见的元素立刻显示出来。[⊖]可见的图片慢慢地淡出。一旦它们完全隐藏，其空间被相邻的图片占据（只有到那时才会如此）。效果如图 5-20 所示。

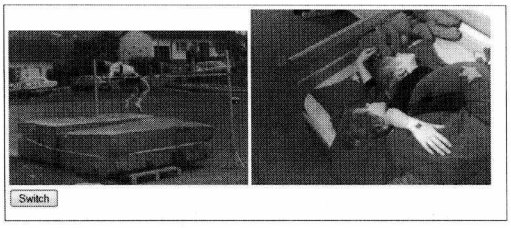

图 5-20 现在其他两幅图片可见

5.3.4 子过滤器

子过滤器（见表 5-6）始终根据父元素进行操作，过滤匹配某些因素的子元素。

表 5-6 子过滤器

名 称	描 述
:nth-child(索引 /even/odd/comparison)	父元素的某个子元素，或者索引为奇数或者偶数的元素。注意，索引不是从 0（像数组那样），而是从 1 开始
:first-child	由所有的第一个子元素组成的数组。和 first 不同，这个过滤器可以选择多个子元素
:last-child	由所有的最后子元素组成的数组。和 last 不同，这个过滤器可以选择多个子元素
:only-child	一个由所有“唯一子元素”组成的数组。父元素除了该元素之外没有其他子元素

我们来看看在实践中如何选择子元素（ch5_18.html）。

程序清单 5.22 嵌套容器

```
...
08    <script type="text/javascript"
09       src="lib/ch5_18_ready.js"></script>
10  </head>
11  <body>
12    <div>
13      <span>I love deadlines. </span>
14      <span>I like the whooshing sound
15        they make as they fly by.</span>
16    </div>
17    <div>
18      <span>Douglas Adams </span>
19      <span>(1952-2001) </span>
20      <span>English writer and dramatist </span>
21    </div>
22  </body>
23 </html>
```

在这个例子中，可以看到两个由 span 元素构造而成的 div 区域。这意味着，div 区域中

⊖ 你也可以在不损害本例功能的情况下实现动画效果。

有子元素，可以选择这些子元素。在第一个 div 区域中，有两个作为子元素的 span 区域，而在第二个 div 区域中有三个子元素。我们希望选择一个包含所有第一子元素的数组，和一个包含所有最后元素的数组。通过链接更多的过滤器，还可以从中选择单独元素，但是我们只希望为所有第一子元素设置和最后子元素不同的格式。其他子元素不受影响。例如，可以用如下代码进行（ch5_18_ready.js）。

程序清单 5.23 第一个和最后一个子元素

```
01 $(function(){
02  $("div span:first-child").css({
03    background: 'red',  color: 'white',
04  });
05  $("div span:last-child").css({
06    background: '#ddd',  color: '#111'
07  });
08 });
```

不同 div 区域的所有第一个 span 区域（第 2 行）或者所有的最后 span 元素（第 5 行）被选中，在第二个 div 区域中有 3 个 span 区域。第二个 span 区域没有被选中，因为它既不是第一个子元素，也不是最后一个子元素。效果如图 5-21 所示。

I love deadlines. I like the whooshing sound they make as they fly by.
Douglas Adams (1952-2001) English writer and dramatist

图 5-21 第一个和最后一个子元素被选中

为了演示子过滤器 :nth-child() 和 :only-child，我们稍微改变了 HTML 文件结构（ch5_19.html）。

程序清单 5.24 嵌套容器

```
...
12 <div>
13   <span>I <span>love</span> deadlines. </span>
14   <span>I like the <span>whooshing</span
15     sound they make as they fly by.</span>
16 </div>
...
```

在例子中，可以看到和前一个例子中基本相同的结构。唯一的不同是第一个 div 区域的前两个 span 区域中包含的一个词。那意味着，包含该词的 span 区域确实只有一个元素，因此可以通过 only-child 选择（ch5_19_ready.js）。

程序清单 5.25 选择子元素和唯一子元素

```
01 $(function(){
02  $("div span:nth-child(2)").css({
03    background: '#000', color: '#fff'
```

```
04 });
05   $("span:only-child").css({
06     letterSpacing:'0.8em'
07 });
08 });
```

在第 2 行中，我们选择底层数组中的第二个子元素。注意，如前所述，索引值不像以往一样从 0 开始，而是从 1 开始！在第 5 行，可以看到选择所有类型为 span，只包含一个子元素的元素。效果如图 5-22 所示。

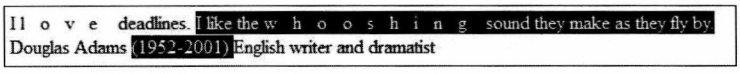

图 5-22 每种情况下的第二个子元素和唯一的子元素

5.3.5 特性过滤器

特性过滤器（见表 5-7）是 jQuery 中最有用的过滤器。可以使用它们过滤 HTML 标记或者元素特性的存在或者内容。对于大部分过滤器来说，容易使用是它们的突出特点。[⊖]特性过滤器包含在方括号中。

表 5-7 特性过滤器

名 称	描 述
[attribute]	所有指定特性的元素，与特性值无关
[attribute=*value*]	所有具有指定特性，且值与指定值匹配的元素
[attribute!=*value*]	所有具有指定特性，且值与指定值不匹配的元素
[attribute^=*value*]	所有具有指定特性，且值以指定测试值开始的元素
[attribute$=*value*]	所有具有指定特性，且值以指定测试值结束的元素
[attribute*=*value*]	所有具有指定特性，且值中任何位置包含指定测试值的元素
[attribute~=*value*]	所有具有指定特性，且值匹配指定测试值，或者指定特性以单词形式出现的元素。单词是一个用空格与其他字符分隔的文本段落
[attribute|=*value*]	所有具有指定特性，且值匹配指定测试值，或者以该值加上连字号开头的元素
[AttributeFilter1] [AttributeFilter2] ... [AttributeFilterN]	过滤后的值匹配各个过滤器的所有条件

注意

在 1.3 版本之前的 jQuery 版本中，@ 标志写在特性名称之前，或者至少允许这么做。这样做的动机来源于 XPath，但是现在这种写法不再有效。

我们用两个例子来演示效果（ch5_20.html）。

⊖ 和导致某些人头痛的正则表达式形成对比。但是，即使是 XPath 总体来说也不容易。

程序清单 5.26 这个网站包含具有特性的标记

```
...
08    <script type="text/javascript"
09       src="lib/ch5_20_ready.js"></script>
10  </head>
11  <body>
12    <a href="http://www.rjs.de" target="new">
13       Homepage</a><br/>
14    <a href="http://blog.rjs.de">Blog</a><br/>
15    <a href="http://www.ajax-net.de" target="new-2">
16       AJAX-Portal</a> <br/>
17    <a href="http://fliegerblog.rjs.de" target="new3">
18       Blog 2</a>
19  </body>
20 </html>
```

如你所见，这个网站包含 4 个具有特性的标记。具体地说，我们想要搜索超链接和 target 特性。ch5_20_ready.js 中的选择如程序清单 5.27 所示。

程序清单 5.27 检查特性

```
01 $(function(){
02  $("[target]").css({
03    textDecoration: 'none'
04  });
05  $("[target|='new']").css({
06    background: 'gray', color: 'yellow'
07  });
08 });
```

在第 2 行中，我们只检查 target 是否存在。所有这类超链接（这里指的是第二个超链接）不再具有下划线。

在第 5 行中，我们检查 target 值是否以 new 开始，该字符串是否以 new 结束或者后面只跟着一个连字号（这个过滤器也被称作前缀过滤器）。这明确地排除了第 17 行的 HTML 文件链接（<a href=http://filegerblog.rjs.de target="new3">）。该链接的 target 值确实以 new 开始，但是下一个字符不是连字号。效果如图 5-23 所示。

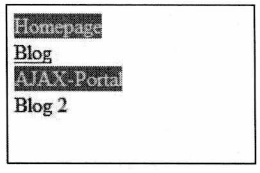

图 5-23 检查 target 特性的不同特性过滤器

我们用一个新的网页来测试其他过滤器。我们引用图片并检查其 alt 特性（ch5_21.html）。

程序清单 5.28 网页中有 5 幅图片指定了代用文本

```
...
08    <script type="text/javascript"
09       src="lib/ch5_21_ready.js"></script>
10  </head>
```

```
<body>
 <img src="images/i5.jpg"
    alt="Sheep in the meadow" />
 <img src="images/i6.jpg"
    alt="Different perspective - Sheep in the meadow" />
 <img src="images/i7.jpg" alt="Football" />
 <img src="images/i8.jpg"
    alt="Flying kites on the meadow" />
 <img src="images/i9.jpg"
    alt="I love my kite" />
</body>
</html>
```

在这个网页中，引用了 5 幅大小相同的图片，没有经过格式化，它们简单地一个接一个显示。效果如图 5-24 所示。

图 5-24 没有格式化时，页面如图中所示

我们现在看看 ch5_21_read.js 中发生了什么。

程序清单 5.29 根据 alt 特性值选择

```
$(function(){
 $("img[alt^='Sheep']").css({
   border: 'solid', borderWidth: '10px'
 });
 $("img[alt$='meadow']").css({
   position: 'relative', top: '100px'
 });
 $("img[alt*='Sheep']").css({
   width: '100px'
 });
 $("img[alt~='kite']").css({
   position: 'relative', left: '-100px'
 });
});
```

在第 2 行中，我们检查所有 alt 特性——仅限于类型为 img 的元素，在这里实际上没有必要——寻找 Sheep 标志。这一选择发生两次。但是通过使用 ^ 符号，我们指定该标志必须出现在特性值的开始，这只出现在第一幅图片中。

第 5 行中的搜索在 meadow 标志出现在特性值字符串最后时成功：这是第 1、2 和 4 幅图片中的情况。

在第 8 行的搜索中，只要求 Sheep 出现在特性值字符串的某处：这是图片 1 和 2 的

情况。

第 11 行中的选择要求匹配以单一字符出现的 kite 标志。只有第 5 幅图片出现这种情况。在第 4 幅图片中，这个标志是较长单词的一部分，因此没有被选中。效果如图 5-25 所示。

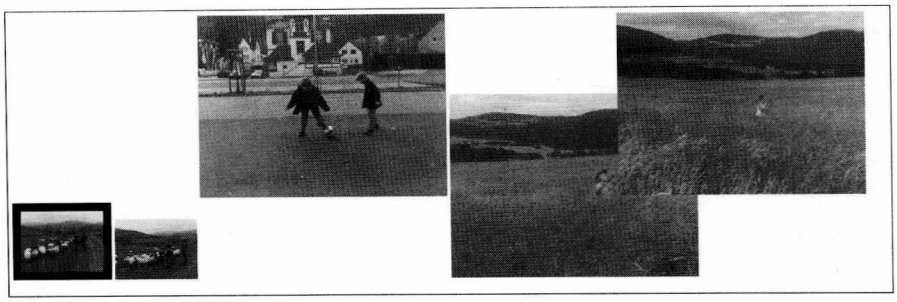

图 5-25 选择的效果和格式化的结果

5.3.6 表单元素过滤器和表单过滤器

这些过滤器的名称说明了我们下一步所要讨论的：利用这些过滤器，可以为表单元素设定特殊选择条件。在 jQuery 的早期版本中，这两类过滤器被单独列出。这种区别在当前版本的文档中已经被去掉，但是为了清晰起见，我们仍然在不同的标题下面讨论它们。

表单元素过滤器用于根据表单元素的类型选择一个或者一组元素，而表单过滤器通过状态过滤元素。

表单元素过滤器

要选择一个表单元素，当然可以在许多情况下使用 input 元素上的过滤器，以及通过 type 特性和特性过滤器更精确地选择。但是在 jQuery 中，特殊的过滤器可以使用更紧凑的标记法。表单元素过滤器如表 5-8 所示。

表 5-8 表单元素过滤器

名 称	类 型
:button	由所有按钮组成的数组（button 元素和类型为 button 的输入元素）
:checkbox	由类型为 checkbox 的所有输入元素组成的数组
:file	由类型为 file 的所有输入元素组成的数组
:image	由类型为 image 的所有输入元素组成的数组
:input	由所有输入元素组成的数组（所有类型为 input、textarea、select 和 button 的表单元素）
:password	由类型为 password 的所有输入元素组成的数组
:radio	由类型为 radio 的所有输入元素组成的数组
:reset	由类型为 reset 的所有输入元素组成的数组
:submit	由类型为 submit 的所有输入元素组成的数组
:text	由类型为 text 的所有输入元素组成的数组

警告

注意浏览器 DOM 中表单元素的准确表现形式。例如，为了格式化一个复选框或者一个选项字段，通常需要找到父元素，因为描述性文本是另一个节点。

我们再用一个例子来测试一些过滤器（ch5_22.html）。

程序清单 5.30 与表单有关联的过滤器

```
...
08    <script type="text/javascript"
09       src="lib/ch5_22_ready.js"></script>
10  </head>
11  <body>
12    <form>
13      <table width="500">
14        <tr>
15          <td>First name</td>
16          <td><input type="text" name="fn" /></td>
17          <td>Surname</td>
18          <td><input type="text" name="sn" /></td>
19        </tr>
20        <tr>
21          <td>User ID</td>
22          <td><input type="text" name="user" /></td>
23          <td>Password</td>
24          <td><input type="password" name="pw" /></td>
25        </tr>
26        <tr>
27          <td></td>
28          <td><input type="checkbox" name="t&cs" /></td>
29          <td></td>
30          <td><input type="checkbox" name="nl" /></td>
31        </tr>
32      </table>
33       <input type="submit" /><input type="reset" />
34    </form>
35  </body>
36 </html>
```

在这个例子中，可以看到由一些文本输入字段、一个密码字段、两个复选框、两个按钮组成的表单，它用一个表格构造。下面是外部 JavaScript 文件 ch5_22_ready.js。

程序清单 5.31 具体的表单过滤器

```
01 $(function(){
02  $(":reset").css({
03    background: 'red',
04    color: 'yellow'
05  });
06  $(":submit").css({
```

```
07    background: 'green',
08    color: 'white'
09  });
10  $(":text").css({
11    background: 'green',
12    color: 'yellow'
13  });
14  $(":password").css({
15    background: 'blue',
16    color: 'white'
17  });
18  $(":checkbox:first").parent().append(
19     "Accept Terms and Conditions").css({
20    background: "yellow", border: "1px red solid"
21  });
22  $(":checkbox:last").parent().append("Newsletter").
23     css({ background: "yellow", border: "1px red solid"
24  });
25  });
```

关于两个按钮、文本输入字段和密码字段的过滤现在可能不需要任何进一步的解释。但是让我们更仔细地观察 18 ～ 22 行中对复选框的过滤。

如你所见，这些过滤器链接起来；一个表单元素过滤器后面跟着一个基本过滤器（:checkbox:first 或者 :checkbox:last）。

上述过滤器选择每种情况下所需要的复选框。它表现得还不错。但是对于格式化这个元素没有意义，因为我们只选择了小复选框本身。这里，CSS 格式没有任何可见的效果。所以我们通过 parent() 方法转移到父元素。⊖

理论上，我们现在已经可以对这个元素应用 CSS 格式。但是这只能为复选框之后的空白位置添加一个框架和背景。到目前为止，复选框还没有标签。所以，我们使用 append() 在这个区域插入一个文本节点。现在，整个区域都得到了 CSS 格式。效果如图 5-26 所示。

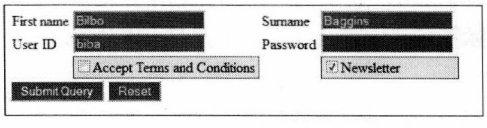

图 5-26 格式化的表单

表单过滤器

下面，我们转向另一些过滤器，可以通过它们，根据状态选择表单元素。

表 5-9 表单过滤器

名 称	描 述
:checked	所有被选中的元素
:disabled	所有禁用的元素
:enabled	所有启用的元素
:focus	所有当前获得焦点的输入元素组成的数组
:selected	所有被选择的元素

⊖ 我们当然也可以使用过滤器实现这样的选择。

警告

这些过滤器只部分适合于用CSS格式化的元素。相反，如果要根据过滤器检查具有特定状态的元素数量，或者想要根据状态编写响应，:checked或者:selected很实用，不需要在编程上花费很大的精力。

程序清单5.32演示了这些过滤器（ch5_23.html）。

程序清单5.32 两个复选框

```
...
08    <script type="text/javascript"
09       src="lib/ch5_23_ready.js"></script>
10  </head>
11  <body>
12    <form>
13      <table width="500">
14        <tr>
15          <td>Accept Terms and Conditions</td>
16          <td><input type="checkbox" name="agb" /></td>
17          <td>Newsletter</td>
18          <td><input type="checkbox" name="nl" /></td>
19        </tr>
20      </table>
21        <input type="submit" /><input type="reset" />
22    </form>
23  </body>
24 </html>
```

这里，我们有一个包含两个复选框的小表单。注意，和前一个例子不一样，标签已经通过HTML在复选框前的表格单元中写入。我们现在希望根据用户是否选中复选框改变对应标签的背景颜色。下面就是我们所做的（ch5_23_ready.js）。

程序清单5.33 根据选择状态过滤

```
01 $(function(){
02  $(":checkbox:first").click(function(){
03    if ($(":checkbox:first:checked").length == 1)
04      $("td:first").css({
05        background: "yellow",
06      })
07    else
08      $("td:first").css({
09        background: "white",
10      })
11  });
12  $(":checkbox:last").click(function(){
13    if ($(":checkbox:last:checked").length == 1)
14      $("td:eq(2)").css({
15        background: "yellow",
```

```
16      })
17    else
18      $("td:eq(2)").css({
19        background: "white",
20      })
21  });
22 });
```

在源代码中，我们为每个复选框注册一个响应机制，对用户的点击做出响应（第 2 和 12 行）。因为每次点击将复选框从选中变成未选中（或者相反），所以这样做是有意义的。但是我们如何知道用户切换到了哪个状态？

解决方案是：我们检查有多少元素具备 checked 状态。如果观察第 3 行中的选择 :checkbox:first:checked，可能只有一个或者完全没有元素（因为 first，第 13 行中的 last 也是一样）。我们可以将其与值 1 比较来检查。结果就是我们想要的。如果复选框被点击选中，标签就设置背景颜色。如果复选框被点击反选，标签的背景颜色被重置。效果如图 5-27 所示。

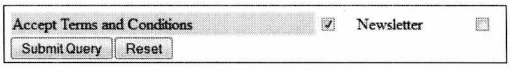

图 5-27 第一个复选框被选中

5.4 过滤器方法

在与作为 jQuery() 参数的过滤选择器的直接联系中，框架还提供了各种过滤器方法。过滤器 / 选择器的基本差别是它们应用的方式。过滤器方法通过句点标记法和前导的对象调用，而不是在选择器中作为参数传递给 jQuery()。除此之外，过滤器方法的效果通常很清晰地从上面列出的过滤器继承而来。所以，我们对大部分方法不提供示例。

5.4.1 eq()

通过 eq() 方法，可以通过一个索引从底层对象中选择一个元素。

5.4.2 not()

not() 方法从一组元素中过滤出与选择器相对应的元素。换句话说，这些元素不包含在找到的元素集中。

5.4.3 first() 和 last()

和过滤选择器类似，使用 first() 可选择一组元素中第一个匹配的元素，使用 last() 可选择最后一个匹配的元素。这些方法实际上很简单，但是在 jQuery 1.4 中才推出。

5.4.4 slice()

slice() 方法没有同名的过滤器。它过滤所有从起始值（索引）之后的所有元素。还可以

指定一个可选的结束值。换句话说，用这个过滤器方法指定一个索引范围。我们用一个例子来阐明（ch5_24.html）。

程序清单 5.34 过滤一个索引范围

```
...
08     <script type="text/javascript"
09        src="lib/ch5_24_ready.js"></script>
10  </head>
11  <body>
12    <div>When I was young,</div>
13     <div>I thought that money</div>
14     <div>was the most important thing in life.</div>
15     <div>Now that I am old,</div>
16     <div> I know:</div>
17     <div>It is.</div>
18  </body>
19 </html>
```

在上例中，我们打算根据如下的过滤器过滤 div 区域。

程序清单 5.35 应用 slice()

```
01 $(function(){
02  $("div:contains('I')").slice(2).css({
03    "color": "yellow", background:"red"
04  });
05 });
```

在第 2 行中，所有包含“I”的 div 区域被选中。然后，从索引 2 开始选择（所以，从选择结果集的第 3 个 div 元素开始）。效果如图 5-28 所示。

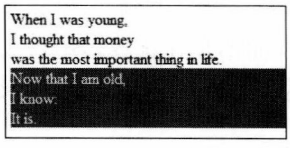

图 5-28 通过 slice() 过滤对应于一个索引范围的元素

警告

你是否理解第 3 个匹配元素？如你所见，标志字符串（在我们的例子里是“I”）不仅是一个单词，也可以是单词的一部分。

5.4.5 filter()

filter() 方法的功能不言自明。它删除所有不对应于参数指定的过滤器的所有元素。除了选择器之外，可以指定一个回调函数或者匿名函数形式的参数。

filter() 方法也没有同名的过滤器（ch5_25.html）。

程序清单 5.36 过滤基础

```
...
08     <script type="text/javascript"
09        src="lib/ch5_25_ready.js"></script>
```

```
10  </head>
11  <body>
12    <div>
13      One <span>Ring</span> to
14      <span class="yes">rule</span> them all,
15      One Ring to <span>find</span> them,
16      One Ring to <span class="yes">bring</span>
17      them <span>all</span>
18      and in the <span>darkness</span>
19      bind them
20    </div>
21  </body>
22 </html>
```

在这个例子中，我们根据指定的类过滤 span 区域，具体代码如下（ch5_25_ready.html）。

程序清单 5.37 使用 filter()

```
01 $(function(){
02  $("span").filter(".yes").css("background", "gray");
03 });
```

在这个例子中，我们从所有 span 区域中过滤两个类为 yes 的。下面的格式只应用到这些类上。效果如图 5-29 所示。

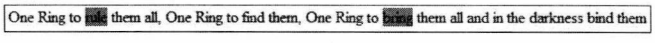

图 5-29 匹配过滤器的两个 span 区域

5.4.6 is()

利用布尔方法 is()，可以测试作为参数的指定表达式是否匹配前导选择。如果匹配则返回 true，否则返回 false。

我们再来看一个例子，改用稍有不同的标记法——直接集成 document.ready()（ch5_26.html）。[⊖]

程序清单 5.38 用 is() 过滤

```
...
08    <script type="text/javascript">
09      $(document).ready(function(){
10        $("button").click(function(){
11          if ($(this).is(":first-child")) {
12            alert("First button");
13          }
14        });
15      });
```

⊖ 不要忘记，在 jQuery 中也可以这样做。

```
16    </script>
17  </head>
18  <body>
19    <button>OK</button>
20    <button>Not OK</button>
21  </body>
22 </html>
```

在上述例子中，有两个按钮，都触发一个点击事件。在第 11 行的响应方法中，我们通过 is(":first-child") 检查触发元素是不是第一个子元素。只有这种情况下，才显示消息框。效果如图 5-30 所示。

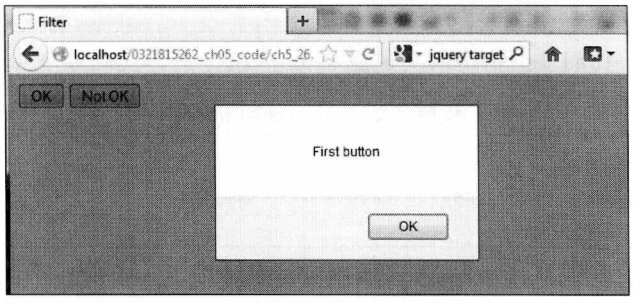

图 5-30 第一个按钮被点击

5.4.7 map()

可以用 map() 方法将 jQuery 对象中的一组元素转换为 jQuery 数组中的一组值。[⊖]然后，可以对数组应用 get() 等特定方法。作为参数，可以指定一个回调或者匿名函数。

注意

该方法直接与 jQuery 数据类型 Map 相关。

程序清单 5.39 将 jQuery 对象映射到一个 jQuery 数组（ch5_27.html）

```
...
08    <script type="text/javascript">
09      $(document).ready(function(){
10        $("button").click(function(){
11          var data = $("input").map(function(){
12            return $(this).val();
13          });
14          $("#output").text(data.get().join(" "));
15        });
16      });
17    </script>
```

⊖ 在编程中，这被称作映射（mapping），方法名即得于此。

```
18 </head>
19 <body>
20   <table>
21     <form>
22       <tr>
23         <td>Surname:</td>
24         <td><input/></td>
25       </tr>
26       <tr>
27         <td>First name:</td>
28         <td><input/></td>
29       </tr>
30       <tr>
31         <td>Age:</td>
32         <td><input/></td>
33       </tr>
34     </form>
35   </table>
36   <button>OK</button>
37   <h2>Data you have entered:</h2>
38   <div id="output" />
39 </body>
40 </html>
```

上例描述包含一个表单的网页。如果用户点击按钮，代表三个输入字段的元素被映射到一个 jQuery 数组。严格地讲，输入字段包含的值用 val() 方法映射。

程序清单 5.40　返回输入字段的值

```
11 var data = $("input").map(function(){
12     return $(this).val();
13 });
```

通过 get()，我们在第 14 行中从 data 数组获取元素（值）。然后，这些元素用 join() 连接成一个字符串输出。效果如图 5-31 所示。

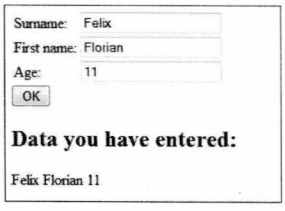

图 5-31　用 Array() 方法进一步处理数据

5.5 小结

在篇幅较大的本章中，我们了解了选择器和过滤器的使用方法，以及 jQuery 下的各种对应的过滤器方法。利用这些，框架大大地扩展了 DOM 选择及历史变种（如对象字段和名称）的潜力。而且，过滤器和选择器保持浏览器一致性、容易应用。在 jQuery 中，选择器和过滤器的语法可以看作经典 CSS 选择器和 XPath 表达式的一种组合。在我看来，jQuery 的方法相当智能，原因很多（因为许多用户已经了解 CSS 选择器，来自 XPath 的思路已经和谐地集成到这一概念之中并进行了简化）。除此之外，过滤器方法可以更加灵活地选择元素。

第 6 章

访问网页的元素

jQuery 为访问网页或者操纵其内容 / 结构提供了多个方法。在本章的语境中，操纵意味着在网站中插入或者删除节点，或者修改节点的内容 / 结构。此外，可以用 jQuery 提供的多种方法遍历一个网页，找到特定的节点和内容。本章很长且有些理论性，但是非常重要，介绍的这些技术很大程度上基于已经讨论过的树形结构浏览和过滤器及选择器。毕竟，访问网页元素是在网页中编写所有动态特效的基础，可以用它来制作动画、动态改变内容或者处理用户输入。

6.1 检查、修改、添加和删除节点的一般信息

我们在好几处已经提到过，文档对象模型（DOM）的概念将网页解释为一棵树，节点和轴由单一的根发展而来。根据节点的类型，DOM 概念提供了不同的方法和属性，以便将节点插入树或者删除及修改节点（特别是节点的内容）。然而，这些访问技术有时候不太直观，在某些浏览器中可能不可靠。jQuery 使这些过程更加抽象或者更加常规，确保它们能够在所有浏览器中安全地使用。

警告

如果通过 jQuery 对象访问网页中的元素，[⊖]不能直接访问常规的 DOM 属性和方法。可以用 get() 获得原生 DOM 节点，但是这样就明确地放弃了 jQuery 命名空间及其相关的语法以及对浏览器错误的保护。

6.2 检查和修改节点内容：html() 和 text()

我们从两个最重要的网页访问方法开始。当然，访问节点内容属于修改节点的类别，因

⊖ 换句话说，通过 jQuery 命名空间。

为在 DOM 概念中，内容（文本）也是独立的节点，它们是文本节点，在 DOM Inspector 中很容易看到。可以使用 html() 方法，检查通常要使用 innerHTML 访问经典 DOM 才能看到的文本节点。该方法返回第一个匹配元素的内容。

注意

为了使说明更加清楚，我们还要谈谈元素的文本内容（例如，标题或者 div 区域），因为我们通常将这些容器元素的内容文本视为在 DOM 中是下属的独立节点（换句话说，是一个文本节点）。在标题或者 div 区域中看到的文本必须通过这个下属文本节点读取。这就是 html()（或者 DOM 概念中的 innerHTML）通过封装的访问所完成的工作。但是，如果直接操作节点对象，首先必须选择文本节点（例如，通过 firstChild，然后用 data 等属性访问内容）。

text() 方法的工作方式相同（更准确地说，是几乎相同）。和 html() 不同，返回的内容没有被转换。换句话说，包含的 HTML 标记没有返回。这对应于经典 DOM 编程中的 innerText。

如果为两个方法指定一个字符串参数，内容被传递给所有[⊖]匹配的文本节点。所以，这是对相关属性的复制，与对应 DOM 属性的通常效果相同。对于 html()，传递的内容在网页中得到解释。这意味着 HTML 标记得到处理。但是对于 text()，尖括号在内部被屏蔽，所以标记被显示在网页中。

我们来看看这两个方法的一个例子，用这两种方法获取和设置内容（ch6_1.html）。

程序清单 6.1 获取和设置文本节点的内容

```
...
06    <script type="text/javascript"
07       src="lib/jquery-1.8.2.min.js"></script>
08    <script type="text/javascript"
09       src="lib/ch6_1_ready.js"></script>
10  </head>
11  <body>
12    <h1>html() and text()</h1>
13    <p> Tiger, tiger, burning <i>bright</i> in the
14      <u>forests</u> of the <b>night</b></p>
15    <button>Get content via html()
16    </button>
17    <button>Get content via text()
18    </button>
19    <button>Set content via html()
20    </button>
21    <button>Set content via text()
22    </button>
23    <div id="output" /></div><div/>
```

⊖ 注意，你可以很方便地将相同的内容赋予多个节点。你可以灵巧地使用正确的选择器在一条语句中完成，而不必苦心孤诣地使用一个循环。

```
24  </body>
25 </html>
```

在这个例子中，可以看到 4 个按钮。用其中两个按钮可以获得一个段落的内容，另两个按钮可以设置 div 容器的内容。下面是引用的 JavaScript 文件 ch6_1_ready.js。

程序清单 6.2 使用 html() 和 text()

```
01 $(function(){
02  $("button:first").click(function(){
03    $("#output").html($("p:first").html());
04  });
05  $("button:eq(1)").click(function(){
06    $("#output").text($("p:first").text());
07  });
08  $("button:eq(2)").click(function(){
09    $("div:gt(0)").html(
10   "<i>What immortal hand or eye\
11    could frame thy fearful symmetry?</i>");
12  });
13  $("button:last").click(function(){
14    $("div:gt(0)").text(
15    "<i>What immortal hand or eye\
16    could frame thy fearful symmetry?</i>");
17  });
18 });
```

如果通过 html() 方法（第 3 行）显示段落内容，会看到包含的 HTML 标记得到解释。效果如图 6-1 所示。

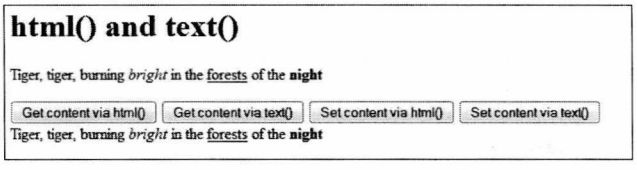

图 6-1 这里使用了 html()

但是，如果使用 text() 方法（第 6 行），只能得到纯文本。效果如图 6-2 所示。

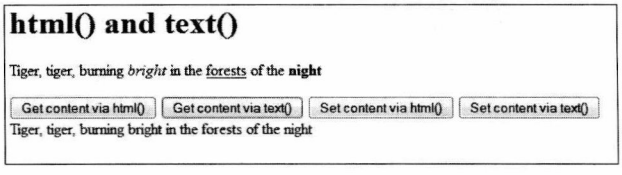

图 6-2 如果使用 text()，没有任何格式化

如果使用字符串参数调用方法，内容被传递给所有匹配的节点，为了演示这一效果，我们使用通过过滤器 :gt(0)（换句话说，从第 2 个 div 容器之后）访问的一个 div 容器。所以，

在这个例子中，只选择一个元素。

如果通过 html() 方法（第 9 ～ 11 行）为 div 容器赋值“What immortal hand or eye could frame thy fearful symmetry?”，你将看到输出区域中这些文本以斜体显示。这意味着所包含的 HTML 标记已经得到解释。效果如图 6-3 所示。

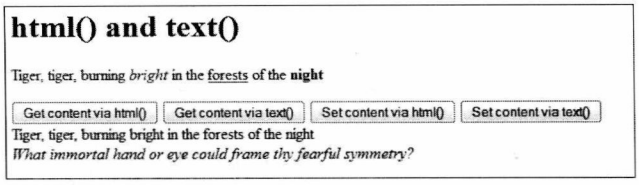

图 6-3 包含的 HTML 标记已经得到解释；文本以斜体显示

但是，如果使用 text() 代替，HTML 标记被显示在网页中，如图 6-4 所示。

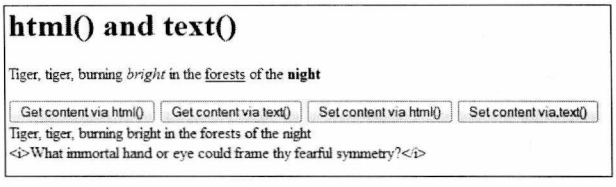

图 6-4 HTML 标记被显示在网页中

如果用 Firebug 分析元素，就会看到尖括号被用实体屏蔽，如图 6-5 所示。

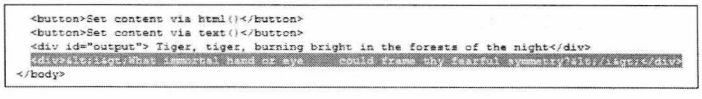

图 6-5 尖括号通过 text() 被屏蔽

6.3 表单字段的内容：val()

在实践中，访问表单仍然是 JavaScript 最重要的应用之一。当然，jQuery 也提供支持。利用 html() 和 text()，可以获取或者设置通过 innerHTML 和 innerText 得到的节点内容。但是表单字段不在此列。而且，在 jQuery 中也必须使用单独的方法获取表单字段内容：val()。向这个方法传递一个参数，将设置该字段的值。如果没有传递参数给 val()，则返回该字段的值。

程序清单 6.3 访问表单字段（ch6_2.html）

```
...
08    <script type="text/javascript"
09       src="lib/ch6_2_ready.js"></script>
10  </head>
```

```
11  <body>
12    <button>Get content</button>
13    <button>Set content</button><hr />
14    <form><input type="text" />
15    </form><hr />
16    <div id="output">
17    </div>
18  </body>
19 </html>
```

这里，我们有一个非常简单的表单，包含一个输入字段。对于通过 JavaScript 进行的访问，我们不需要设置方法和 action 属性。可以在引用的 JavaScript 文件 ch6_2_ready.js 中看到这种访问。

程序清单 6.4 使用 val()

```
01 $(function(){
02  $("button:first").click(function(){
03    var str = $("input:first").val();
04    $("#output").text(str);
05  });
06  $("button:eq(1)").click(function(){
07    var str = $("input:first").val("new");
08  });
09 });
```

在第 3 行中，我们用 val() 方法获得第一个输入字段的内容，并在输出区域中显示（通过 ID 访问）。在第 7 行中，我们通过一个预定义字符串设置输入字段中的值。如图 6-6 和图 6-7 所示。

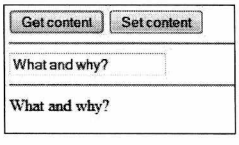

图 6-6 表单字段中输入的值被读取并在输出区域中显示

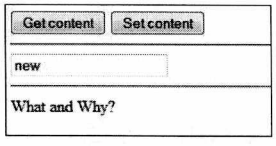

图 6-7 表单字段中的值通过点击按钮 2 预设

提示

作为 val() 的替代品，还可以使用方法 attr() 读出或者设置表单字段的内容。但是这一方法更为通用，因为可以用它设置或者读取任何 HTML 元素特性的值。在表单字段中，这是对特性 value 的访问。但是我发现这一方法比 val() 更麻烦。你还应该知道，到 jQuery 1.6 时，attr() 方法的工作方法稍有不同。静态的 DOM 特性与动态的 DOM 属性被区分开来。所以，如果使用 jQuery 1.6 或者更新的版本，使用 prop(“属性值”) 是更好的选择。但是我仍然更喜欢使用专用的 val() 方法。

6.4 通过 attr() 访问特性

如果想要动态更改网页，从 jQuery 中使用经典的 HTML 特性值非常实用。如前所述，访问网页元素特性的最通用方法是 attr()。只要以一个字符串（换句话说，一个参数）的形式输入特性，就能返回一个特性值。

程序清单 6.5 src 特性值

```
var path = $("img:first").attr("src");
```

如果输入第二个参数，则设置特性值。

程序清单 6.6 设置 src 特性值

```
$("img:first").attr("src","http://rjs.de/bilder/devil.gif");
```

注意

这个方法很容易使用，所以我们没有提供完整的例子。除此之外，我们在其他例子中反复用到这个方法。

6.5 在网页中插入节点

使用 html() 和 text() 方法，可以为文本节点提供新内容。但是，也可以在现有内容上附加内容。这种方法更灵活，因为不限于纯文本内容。我们将相当紧密地与 DOM 节点结构相配合。

注意

如果最终结构在网页创建时还不清晰，节点的这种动态操纵就很方便。例如，想象一个在线商店，我们无法知道客户将要选择多少件产品。

6.5.1 append() 和 prepend()

首先，我们在本书中已经使用过 append() 方法，这一方法也或多或少地以相同的形式存在于原生 DOM 编程中（appendChild() 方法）。作为字符串参数传递的内容简单地添加在现有内容的最后。所以，在选中的元素中，我们得到另一个子节点。prepend() 方法的工作方式完全相同，但是内容被添加在前面。我们将再次用一个表单字段演示这两个方法，在点击一个按钮之后将内容添加到原来的文本上（ch6_3.html）。

程序清单 6.7 表单和输出区域

```
...
08    <script type="text/javascript"
```

```
09        src="lib/ch6_3_ready.js"></script>
10  </head>
11  <body>
12    <button>Append content</button>
13    <button>Prepend content</button><hr />
14    <form><input type="text" />
15    </form><hr />
16    <div id="output">| Original text |
17    </div>
18  </body>
19 </html>
```

我们使用了以前用过的表单，有一个输入字段（加上两个按钮，通过它们可以触发操作，还有一个输出区域，添加的节点将成为它的子元素）。一开始，网站的原始结构对我们来说很有趣，我们可以在 Firebug 中清楚地看到它。如图 6-8 所示。

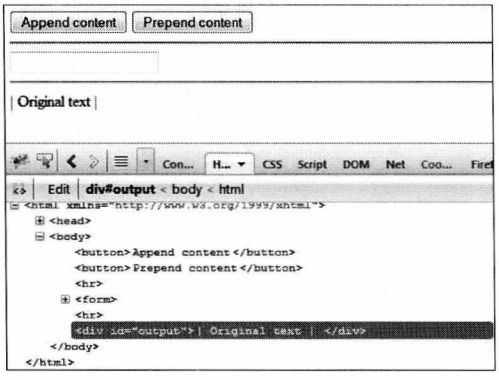

图 6-8 div 元素只包含原始文本

大家可以看到输出区域已经包含了某些原始文本。在这种情况下，DOM Inspector 的视图更为醒目。如图 6-9 所示。

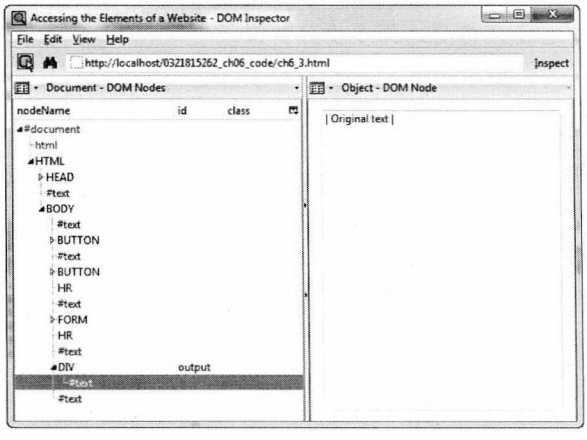

图 6-9 在 DOM Inspector 中，可以看到 div 容器的实际节点

在 DOM Inspector 中，可以看到输出区域的内容是一个下属文本节点（div 元素的子元素），正如 html() 和 text() 方法介绍中已经强调过的那样。还可以看到，该区域没有其他节点。可以在下面引用的 JavaScript 文件 ch6_3_ready.js 中看到对输出区域的访问。

程序清单 6.8　append() 和 prepend() 实用示例

```
01 $(function(){
02  $("button:first").click(function(){
03    var str = " < " + $("input:first").val();
04    $("#output").append(str);
05  });
06  $("button:eq(1)").click(function(){
07    var str = $("input:first").val() + " > ";
08    $("#output").prepend(str);
09  });
10 });
```

第一个按钮被点击时，第 4 行的语句将表单字段的内容附加到输出区域现有内容之后。在第 8 行中，内容被添加到现有内容之前。如图 6-10 所示。

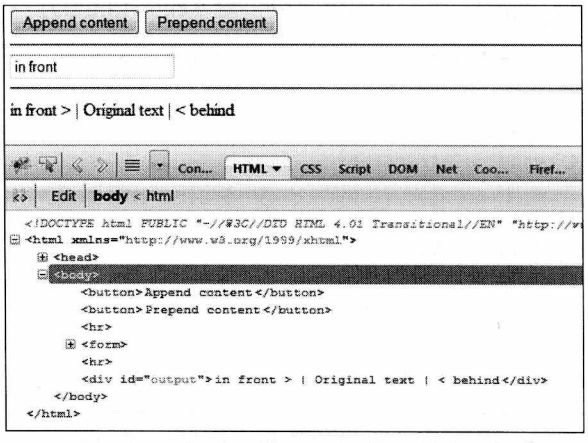

图 6-10　不同的文本被插入到原始文本之前和之后

如果在 DOM Inspector 中观察，就能清晰地看到 div 容器中的节点现在是什么结构。你将会看到，div 容器现在对每个插入的内容都有单独的文本节点。每个通过 append() 插入的文本节点在原始文本节点之后，而用 perpend() 插入的每个文本节点在原始文本节点之前。如图 6-11 所示。

提示

还可以为 append() 和大部分其他节点操纵方法指定函数调用作为参数。但是，应该首先直接创建这些元素（例如，用 document.createElement()）。否则，函数返回一个 HTML 字符串，代替匹配的元素组。

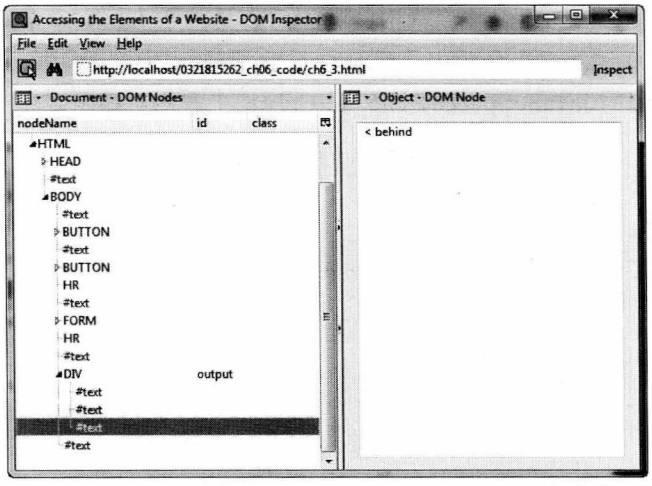

图 6-11 在 div 容器中，现在有单独的文本节点

利用刚才描述的方法，还可以将对象移动到网页的其他位置。注意，实际上是从 DOM 中的某个位置删除对象，然后将其插入到另一个位置（ch6_4.html）。

程序清单 6.9 将一个元素移到另一个位置

```
...
08 <script type="text/javascript"
09   src="lib/ch6_4_ready.js"></script>
10  </head>
11  <body>
12    <button>Move object</button><hr />
13    <h2>Text in Heading</h2><hr />
14    <div id="output">| Original text |
15    </div>
16  </body>
17 </html>
```

在例子中，我们想将 h2 级别的标题移动到输出区域。这很容易（ch6_4_ready.js）。

程序清单 6.10 移动一个元素

```
01 $(function(){
02  $("button:first").click(function(){
03    $("#output").append($("h2"));
04  });
05 });
```

在第 3 行中，我们选择标题 h2（严格地说是所有这类元素，但是在例子中只有一个）。如图 6-12 所示。

我们使用 append() 将其移动到输出区域，在 Firebug 中可以清楚地看到。如图 6-13 所示。

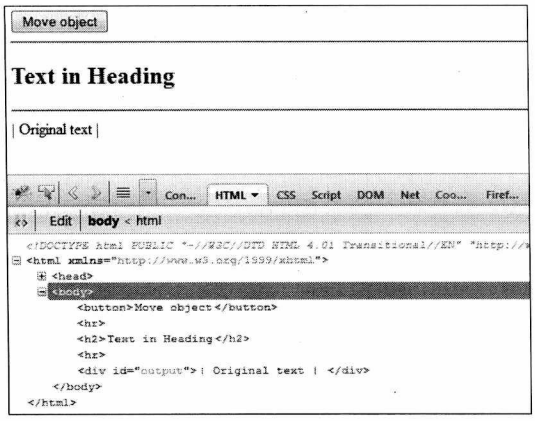

图 6-12 网页的原始结构

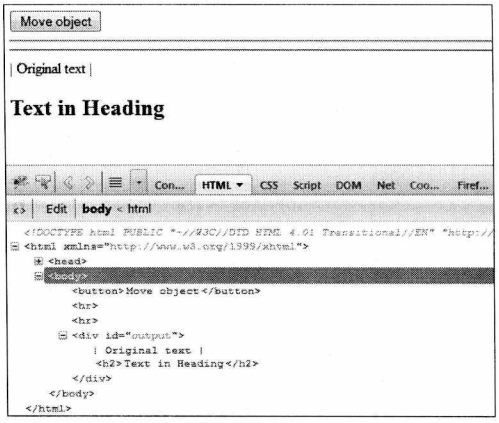

图 6-13 在 Firebug 中，可以特别清楚地看到标题如何被移到 div 容器中

注意

许多 jQuery 节点操纵方法（但不是所有）有这种表现。如果在网页中获得一个现有节点并将其插入网页的另一个位置，原始节点将被删除。但是，对于 jQuery 的某些方法需要注意一些特殊性，我们将在后面介绍。

6.5.2 appendTo() 和 PrependTo()

要附加或者移动节点，除了 append() 和 prepend() 之外还有其他选择。appendTo() 是一个替代方法。但是必须注意；这个方法的语法和基于经典 DOM 方法 appendChild() 应用的其他两个方法有所不同。

这个替代方法将前导选择器选择的节点附加到作为该方法参数的选择器指定的现有内容之后。换一种不同的说法，调用该方法的对象被附加到作为参数的对象。这和 append() 正好相反。prependTo() 与此完全一致，但将节点插入到现有内容之前。

但是，我们需要区分这些方法是否应用到现有元素，以及我们关注的是不是需要事先创建的元素。因为如果我们将这些方法应用到网页中现有的节点，这些节点再次被移动。但是移动节点的问题在某些情况下可能有些难以处理。为了澄清这一点，我们来看看 appendTo() 在一个虚构但是具有表现力的示例应用程序中的运行情况。

我们将方法应用到网页中已经存在的元素，并且移动树中附加的节点。换句话说，附加的内容不再位于树中应用 apeendTo() 之前相同的位置上，也不是被删除后附加到新位置的文本内容，而是整个节点（包括层叠样式单 [CSS] 格式等特殊节点属性）。这种行为可以在 Firebug 中进行分析，在前一个 append() 示例中也曾介绍（ch6_5.html）。

程序清单 6.11 用 appendTo() 将元素移到另一个位置

```
...
08 <script type="text/javascript"
09   src="lib/ch6_5_ready.js"></script>
10  </head>
11  <body>
12    <button>append()</button>
13    <button>appendTo() one node</button>
14    <button>appendTo() several nodes</button>
15    <hr/><span id="a" style="color:cyan;background:red">
16     - 1st Node - </span>
17    <span id="b" style="color:white;background:cyan">
18     - 2nd Node - </span>
19    <span id="c" style="color:green;background:yellow">
20     - 3rd Node - </span>
21    <div id="output" style=
22       "color:red;background:lightgray">Target area</div>
23  </body>
24 </html>
```

在这个例子中，我们要将 span 类型的现有元素移到输出区域中（ch6_5_ready.js）。

程序清单 6.12 append() 和 appendTo() 的对比

```
01 $(function(){
02  $("button:first").click(function(){
03    $("#output").append(" - append() - ");
04  });
05  $("button:eq(1)").click(function(){
06    $("#a").appendTo("#output");
07  });
08  $("button:last").click(function(){
09    $("span").appendTo("#output");
10  });
11 });
```

在上例中仍然有 3 个按钮，我们通过它们调用方法。和往常一样，我们使用一个输出区域。新内容被附加到该区域。如果自己尝试这个例子，就会看到 append() 和 appendTo() 的表现不同。

观察一下网页中具有特殊格式的 span 容器。如果加载该页面，你会看到这个 span 容器紧跟在分隔线之后，在我们附加内容的 div 容器之前。再看看 Firebug 中的 DOM 结构。你会看到树中对应的元素按照你在 HTML 中创建的顺序排列。如图 6-14 所示。

如果现在用 append() 方法将字符串参数中指定的文本附加到目标区域，就会看到它被简单地添加到现有内容上。在 Firebug 中，可以清晰地看到除了新的文本节点之外，树的节点结构没有变化。只有目标容器的内容被扩展。如图 6-15 所示。

但是，如果点击第二个按钮，ID 为 a 的 span 容器内容被附加到目标区域（div 容器）。更准确地讲，它被作为子元素插入到类型为 div 的父元素中。

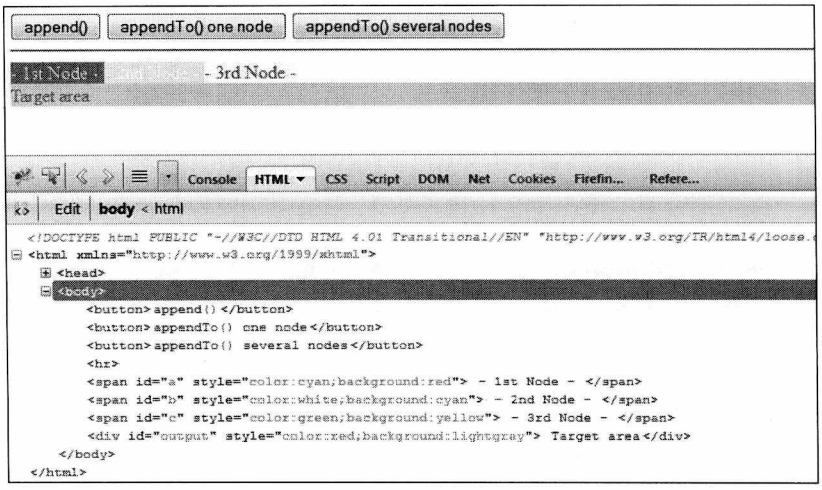

图 6-14 加载之后的网页

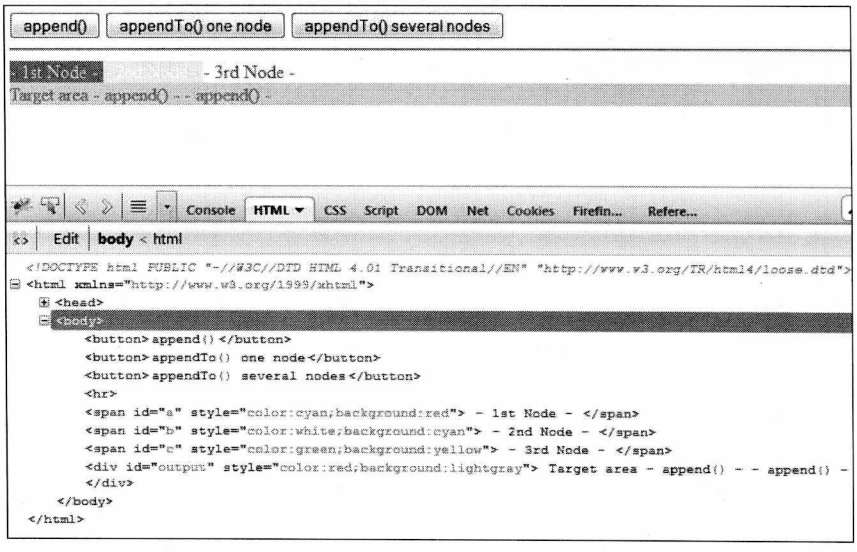

图 6-15 使用 append()，树的节点结构不会变化（除了新文本节点之外）

程序清单 6.13 通过 ID 指定的节点被附加到目标区域

```
06 $("#a").appendTo("#output");
```

过程中，这个节点从树中的原始位置删除。同样，可以在 Firebug 中看到。在我们解释了 append() 之后，你应该不会对这一行为感到吃惊。如图 6-16 所示。

如果在这个状态下再一次应用 append()，结果也很有趣。你将会看到指定的字符串再一次附加到 div 容器内容之后。这也适用于前面的解释。如图 6-17 所示。

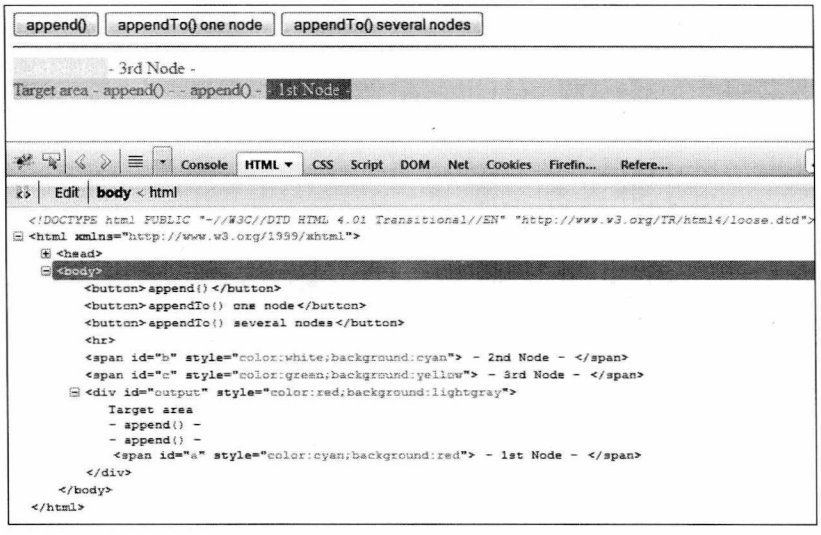

图 6-16 ID 为 a 的 span 容器现在成为 div 容器的子元素

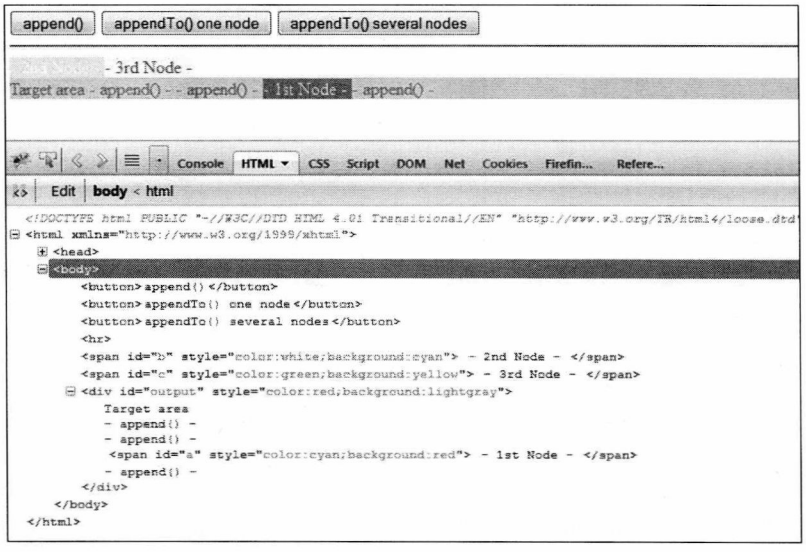

图 6-17 用 append() 在最后一个子元素之后再次插入内容

如果现在用选择器为 appendTo() 选择多个节点，这些节点会被一个接一个地从树中的原始位置删除，并添加到目标区域成为子元素，即使这些元素已经出现在目标区域的另一个位置中。[⊖]然后，这些元素此时在 DOM 中的位置对于再次插入的顺序变得非常重要。如图 6-18 所示。

⊖ 例如，作为前一个操作的结果。

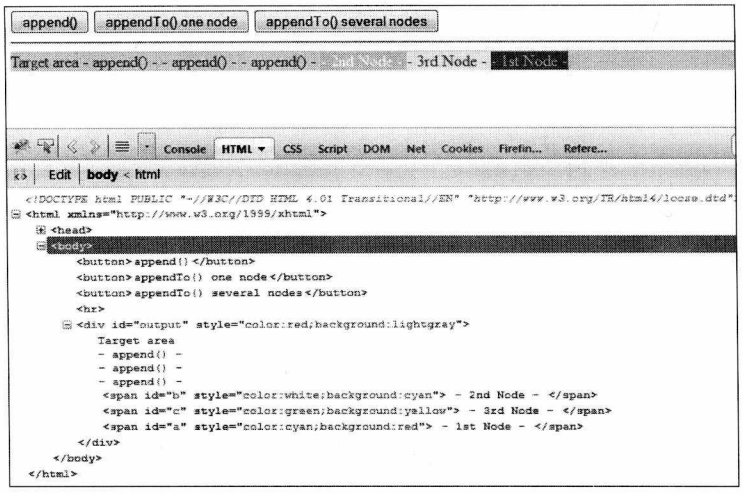

图 6-18 将 appendTo() 应用到多个元素

注意

我们没有展示 prependTo() 的实际应用情况；它和 appendTo() 相同。

现在，我们将把 appendTo() 应用到网页上尚不存在的元素。我们创建一个 jQuery 对象，然后将其添加到页面（ch6_6.html）。

程序清单 6.14 用 appendTo() 将元素移动到另一个位置

```
...
08    <script type="text/javascript"
09      src="lib/ch6_6_ready.js"></script>
10  </head>
11  <body>
12   <button>appendTo() a new node</button><hr/>
13    <div id="output" />
14   </body>
15 </html>
```

在本例中，我们创建新元素，并将其附加到输出区域（ch6_6_ready.js）。

程序清单 6.15 appendTo() 和新创建的元素

```
01 $(function(){
02  $("button:first").click(function(){
03    $("<div>New</div>").appendTo("#output");
04  });
05 });
```

在第 3 行中，可以看到创建了一个新的 div 元素，然后添加到输出区域。如图 6-19 所示。

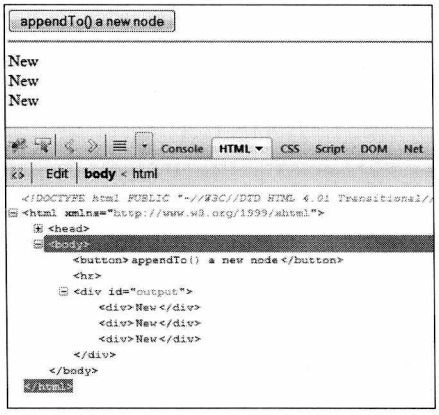

图 6-19 3 个新节点被插入输出区域

6.6 在前面或者后面插入节点

通过 append() 或者 prepend()，以及 appendTo() 或者 prependTo() 可以插入现有节点。但是，jQuery 还提供了在选择器指定的某个节点前后插入节点的方法。

6.6.1 after() 和 before()

可以用 after() 方法在匹配选择器的每个元素之后添加内容。before() 方法也有相应的表现——内容被添加到匹配选择器的每个元素之前（ch6_7.html）。

程序清单 6.16 在特定位置插入元素

```
...
08   <script type="text/javascript"
09     src="lib/ch6_7_ready.js"></script>
10  </head>
11  <body>
12    <button>after()</button>
13    <button>before()</button><hr/>
14    <div id="output" style=
15       "color:red;background:lightgray">Target area</div>
16  </body>
17 </html>
```

在本例中，我们创建新元素，并将它们添加到输出区域（ch6_7_ready.js）。

程序清单 6.17 after() 和 before()

```
01 $(function(){
02  $("button:first").click(function(){
03    $("#output").after(
04       '<span style="color:white;background:red">
```

```
        After</span>');
05  });
06  $("button:eq(1)").click(function(){
07    $("#output").before(
        '<span style="color:red;background:yellow">
08      Before</span>');
09  });
10 });
```

如果尝试这个例子，就会看到具有特定内容的新节点和对应的属性（CSS 格式）分别被插入目标节点前后。如图 6-20 所示。

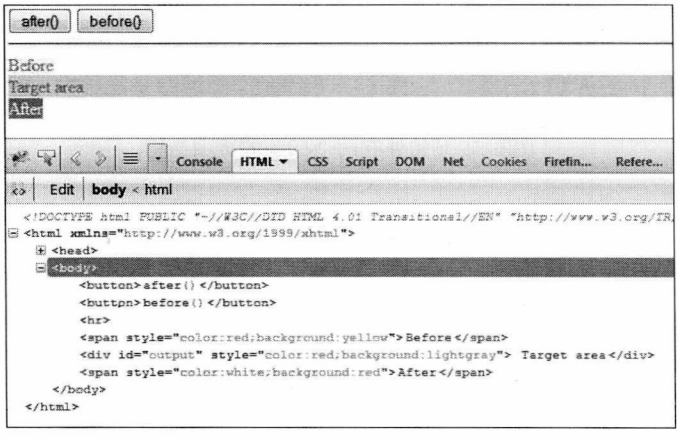

图 6-20 实际节点被插入到目标节点前后

我们再来看看这些方法应用到网页中已有元素时的效果（ch6_8.html）。

程序清单 6.18 将元素插入特定位置

```
...
08   <script type="text/javascript"
09     src="lib/ch6_8_ready.js"></script>
10  </head>
11  <body>
12        <h2>after()</h2>
13      <h2>before()</h2>
14    <button>after()</button>
15    <button>before()</button><hr/>
16    <div id="output" style=
17      "color:red;background:lightgray">Target area</div>
18  </body>
19 </html>
```

本质上，你看到的页面和以前一样。但是现在我们有两个类型为 h2 的标题。如图 6-21 所示。

我们在例子中用 after() 和 before() 处理这些标题（ch6_8_ready.js）。

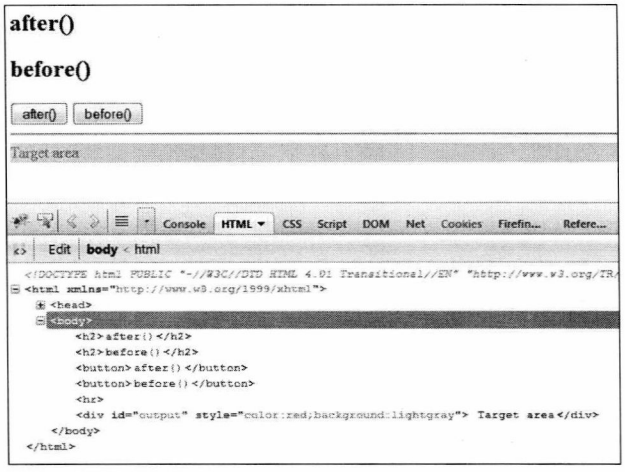

图 6-21 开始处有两个标题的页面结构

程序清单 6.19 after() 和 before()

```
01 $(function(){
02   $("button:first").click(function(){
03     $("#output").after($("h2:first"));
04   });
05   $("button:eq(1)").click(function(){
06     $("#output").before($("h2:last"));
07   });
08 });
```

如果尝试这个示例，就会看到第一个标题被移到目标区域之后，第二个标题被移到目标区域之前。所以，在现有元素的情况下，该方法和其他相关方法的效果相同。如图 6-22 所示。

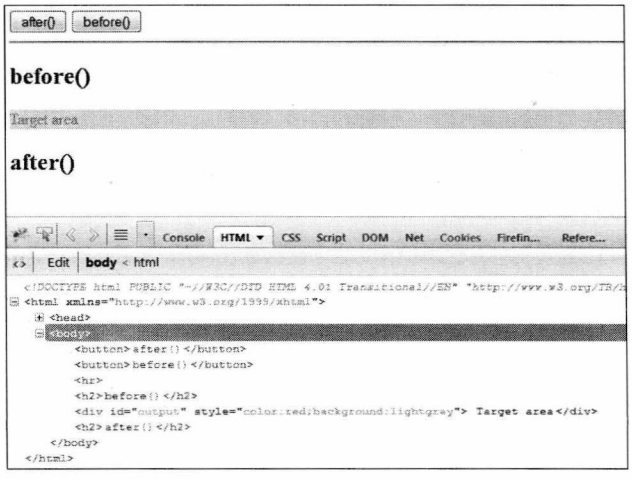

图 6-22 移动标题

6.6.2 insertAfter() 和 insertBefore()

还有两个方法：insertAfter() 和 insertBefore()。它们类似于 appendTo() 和 prependTo()。参数是指定元素添加的目标（ch6_9.html）。

程序清单 6.20 基本网页

```
...
08   <script type="text/javascript"
09     src="lib/ch6_9_ready.js"></script>
10  </head>
11  <body>
12    <button>insertAfter()</button>
13    <button>insertBefore()</button><hr/>
14    <div id="element" style=
15        "color:red;background:lightgray">Text</div>
16  </body>
17 </html>
```

可以看到两个按钮和一个 div 区域，我们将把这个区域移到第一个按钮之前和之后。如图 6-23 所示。

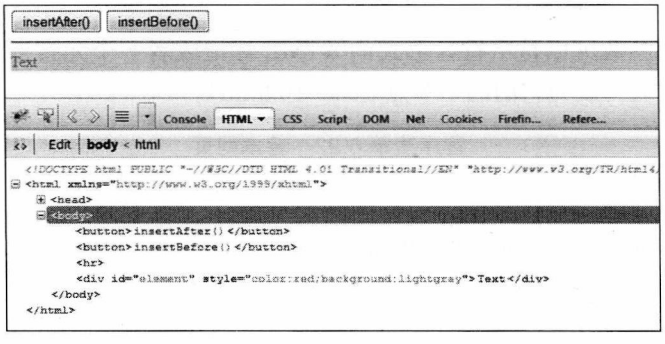

图 6-23 初始网站结构

下面是 JavaScript 函数（ch6_9_ready.js）。

程序清单 6.21 insertAfter() 和 insertBefore()

```
01 $(function(){
02   $("button:first").click(function(){
03     $("#element").insertAfter($('button:first'));
04   });
05   $("button:eq(1)").click(function(){
06     $("#element").insertBefore($('button:first'));
07   });
08 });
```

如果尝试这个例子，就会看到 ID 为 #element 的 div 区域被直接移到第一个按钮之前或者之后。如图 6-24 所示。

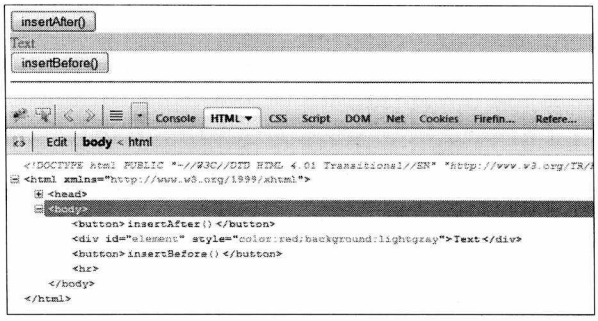

图 6-24 div 容器已经被插到第一个按钮之后

6.7 包装

在 jQuery 中，可以用某些结构包装现有内容。被包装的节点本质上成了所处结构的子节点。

6.7.1 用 wrap() 单独包装

wrap() 方法的参数可以是字符串形式的 HTML 标记或者一个元素，所有前面选择的元素用对应的结构包装，成为它们的子节点（ch6_10.html）。

程序清单 6.22 基本网页

```
...
08   <script type="text/javascript"
09     src="lib/ch6_10_ready.js"></script>
10  </head>
11  <body>
12    <button>wrap() with HTML</button>
13      <button>wrap() with element</button>
14    <hr/><span>Wrapping</span>
15    <span>things</span>
16    <span>up</span>
17  </body>
18 </html>
```

在这个网页中有 3 个处于相同层次的 span 区域，我们将在各种情况下包装它们。如图 6-25 所示。

下面是我们用于包装的 JavaScript 函数（ch6_10_ready.js）。

程序清单 6.23 用 HTML 结构包装网页内容

```
01 $(function(){
02  $("button:first").click(function(){
03    $("span").wrap("<h3>");
04  });
```

```
05  $("button:eq(1)").click(function(){
06    $("span").wrap($("<div/>"));
07  });
08 });
```

在例子中，如果用户点击按钮，HTML 结构将围绕网页中的所有 span 元素进行包装。在第 3 行，可以看到一个 HTML 标记作为参数，实现这种效果。如果用户点击第一个按钮，事件助手被触发，所有 span 区域被各自包围在 h3 级别标题内。如图 6-26 所示。

在第 6 行中，我们明确地创建一个 div 元素，将其用作参数。[⊖]结果是，它围绕 span 区域进行包装。这是在 jQuery 中包装元素的另一个选项。如图 6-27 所示。

注意，这个例子没有考虑用户是否重复点击按钮。这不是一个大问题。span 区域从内到外一次又一次地重复包装，结果可能是一个嵌套很深的层次结构。它有可能成为无效的 HTML，在实践中，不应该允许这种情况发生。如图 6-28 所示。

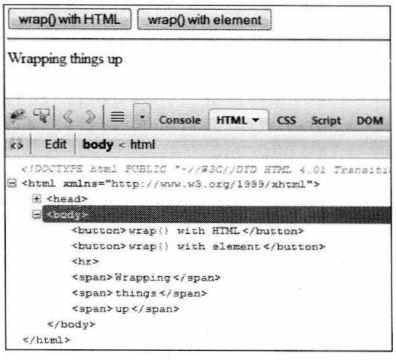

图 6-25 开始时的网站结构

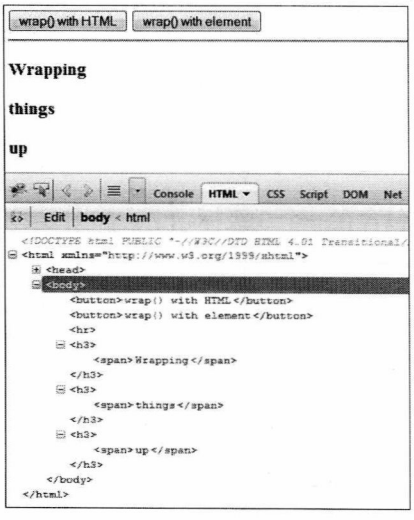

图 6-26 第 3 级标题围绕所有 span 区域进行包装

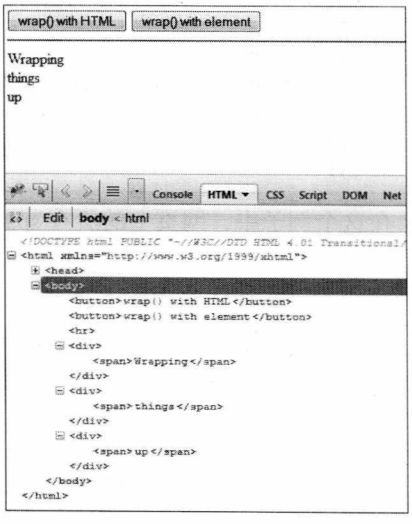

图 6-27 现在 span 区域被包装在 div 容器中

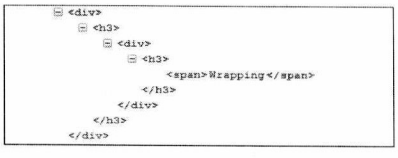

图 6-28 越来越多的包装和嵌套

⊖ 你也可以编写一个返回对象的函数调用，或者再次使用 document.CreateElement()。

6.7.2 用 wrapAll() 包装所有元素

jQuery 还提供了进一步的包装方法。通过 wrapAll() 方法（与 wrap() 相关），可将所有匹配元素都被包含在一个包装元素内（ch6_11.html）。

程序清单 6.24 基本网页

```
...
08   <script type="text/javascript"
09     src="lib/ch6_11_ready.js"></script>
10  </head>
11  <body>
12    <button>wrapAll()</button>
13    <hr/><span>Wrapping</span>
14    <span>things</span>
15    <span>up</span>
16  </body>
17 </html>
```

本例中同样有 3 个 span 区域，但是我们想要将它们完全包含在一个包装元素中（ch6_11_ready.js）。

程序清单 6.25 用一个元素包装

```
01 $(function(){
02  $("button:first").click(function(){
03    $("span").wrapAll("<h3>");
04  });
05 });
```

在这个例子中，网页上的所有 span 元素完全被包含在一个 3 级标题中。如图 6-29 所示。

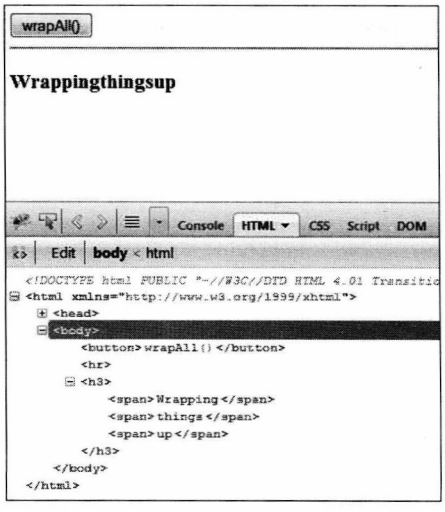

图 6-29 可以清楚地看到，所有 span 区域现在都在类型为 h3 的一个标题中

6.7.3 用 wrapInner() 包装内部区域

jQuery 还提供 wrapInner() 方法。它和 wrap() 工作方式类似，但是将所有匹配元素（包括文本节点）的内部子元素包装在一个 HTML 结构中（ch6_12.html）。

程序清单 6.26 基本网页

```
...
08   <script type="text/javascript"
09     src="lib/ch6_12_ready.js"></script>
10  </head>
11  <body>
12    <button>wrapInner()</button>
13    <hr/><span>Wrapping</span>
14    <span>things</span>
15    <span>up</span>
16  </body>
17 </html>
```

我们同样有 3 个 span 区域，但是这次我们不用结构包装它们，而是包装其内容（ch6_12_ready.js）。

程序清单 6.27 用一个元素包装内容

```
01 $(function(){
02  $("button:first").click(function(){
03    $("span").wrapInner("<h3>");
04  });
05 });
```

在上例中，网页中的所有 span 元素被选中，它们的内容被包装在一个 h3 级别标题中。如图 6-30 所示。

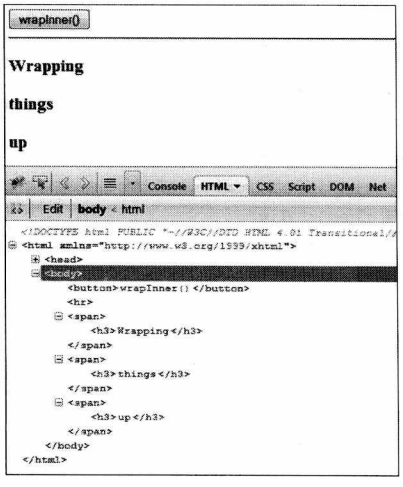

图 6-30 这可能不是明智的 HTML，但是 span 区域的所有内容用 h3 标题包装

6.7.4 用 unwrap() 解除包装

用 wrap() 方法可以包装一个元素。这意味着，将目标元素放在父元素内部。利用 unwrap() 方法，可以删除父元素，这是 wrap() 的逆操作。该方法的使用极其简单；甚至没有必要指定参数。

程序清单 6.28 删除所有段落的父元素

```
$("p").unwrap();
```

6.8 用 replaceWith() 和 replaceAll() 替换

本章的主题是访问网页的元素。当然，这也包括网页组件的替换。利用 jQuery 的 replaceWith() 和 replaceAll() 方法，可以简单、方便、可靠地替换网页中的元素。

6.8.1 用 replaceWith() 替换

利用 replaceWith() 方法，可以用参数指定的 HTML 或者 DOM 元素替换前导 jQuery 对象选择的所有元素。

警告

替换的元素明确地从现有 DOM 中删除。[⊖]然后，以 jQuery 元素的形式获得这些元素，作为方法的返回值，可以进一步使用它们。

我们再来看一个不同的群体（ch6_13.html）。

程序清单 6.29 替换网页的一部分

```
...
08   <script type="text/javascript"
09     src="lib/ch6_13_ready.js"></script>
10  </head>
11  <body>
12    <button>replaceWith()</button><hr/>
13    <div>One Ring to rule them all,</div>
14    <div>One Ring to find them,</div>
15    <div>One Ring to bring them all</div>
16    <div>and in the darkness bind them.</div><hr />
17  </body>
18 </html>
```

这里，我们有 4 个 div 元素，包围在两条分隔线内。替换过程在用户点击一个按钮时触发。

⊖ 在某种程度上，这种情况类似于我们移动网页上的现有元素。

我们打算在用户点击按钮时，用 5 级标题连续替换 div 元素。每次点击用一个标题替换当时的第一个 div 元素（ch6_13_ready.js）。如图 6-31 所示。

程序清单 6.30 替换 div 区域

```
01 $(function(){
02  $("button:first").click(function(){
03    $("div:first").replaceWith("<h5>" +
04      $("div:first").text() + "</h5>");
05  });
06 });
```

可以看到，事件助手方法中，每次用户点击按钮都选中当时的第一个 div 元素。然后，这个元素在 DOM 中被用 replaceWith() 的参数替换。在例子中，方法的参数来源于我们替换的 div 区域的文本内容（这时，因为替换还没有进行，这些内容仍然可用）和 HTML 标签。旧的内容被包围在一个 5 级标题中。在第 3 行和第 4 行的语句执行之后，第一个 div 已经被从 DOM 中删除，被具有相同文本内容的 h5 元素替换。如图 6-32 所示。

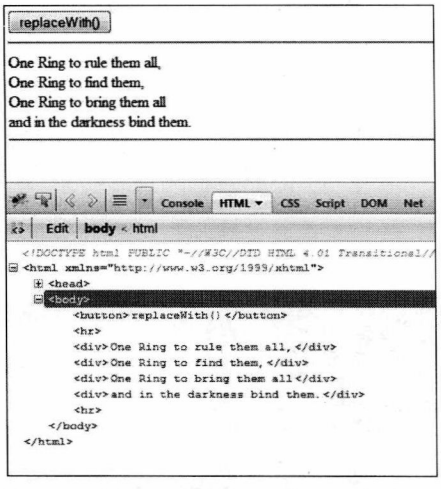

图 6-31 网页原始状态下的 4 个 div 区域

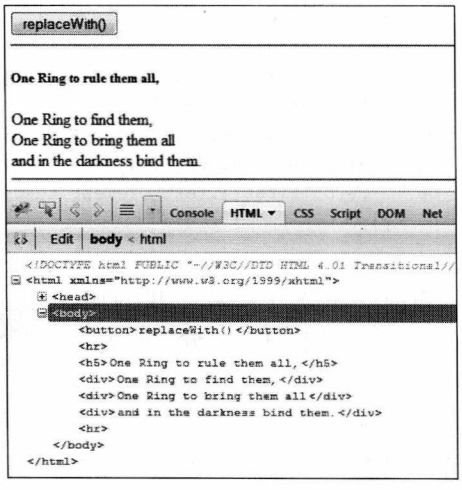

图 6-32 第一个 div 已经被变成 5 级标题

如果用户现在再次点击，当前的第一个 div[⊖]被选中，并被替换。以此类推，在 4 次点击之后，所有 div 区域都被转换为 5 级标题。如图 6-33 所示。

在最后一个例子中，我们用方法 text() 读出内容，在对象被替换之前使用了它。但是 replaceWith() 方法将替换的节点当作 jQuery 对象返回。所以，我们还可以使用它。我们来看看上例的一个变种，了解如何进一步处理返回的元素。底层 HTML 文件 ch6_14.html 与文件 ch6_13.html 相同（除了对 JavaScript 文件的引用之外）。因此，我们只关注 JavaScript 文件 ch6_14_ready.html。

⊖ 这是网站加载时的第二个 div。

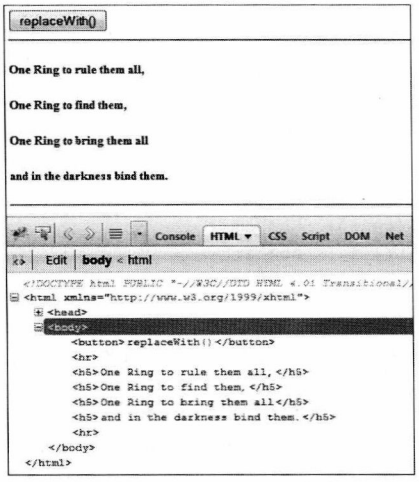

图 6-33 div 元素是历史

程序清单 6.31 处理返回值

```
01 $(function(){
02  $("button:first").click(function(){
03    var element =
04       $("div:first").replaceWith("<h5></h5>");
05    alert(element.text());
06    $("h5:first").text(element.text());
07  });
 });
```

和前一个例子一样，我们用 5 级标题连续替换 div 区域。但是这些标题都是空白的。在第 3 行和第 4 行中，我们给从网页 DOM 中删除并已替换的 div 元素赋予一个局部变量 element。这个变量因此引用代表上次删除的 div 元素的一个 jQuery 对象。相应地，我们可以在后续的步骤中使用这个对象——只要 element 变量可用。例如，在第 5 行中，我们用 alert() 输出文本内容。如图 6-34 所示。

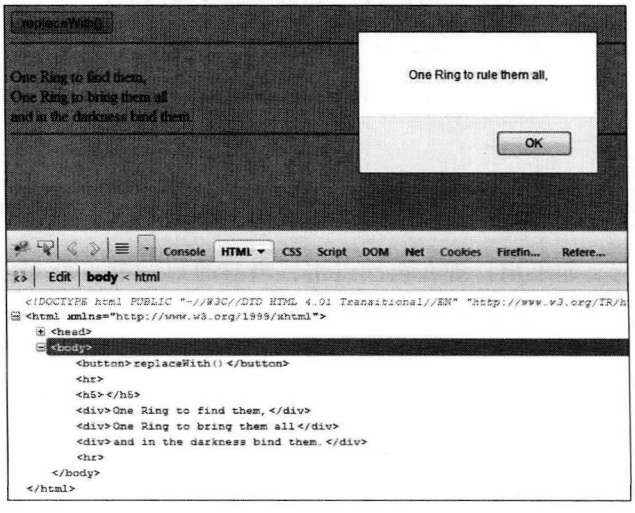

图 6-34 从网页删除的 div 元素内容仍然可用，可以通过 alert() 显示

在第 6 行中，我们通过变量 element 再次访问文本内容，将其写入网页，作为 h5 类型的第一个标题

的内容。如图 6-35 所示。

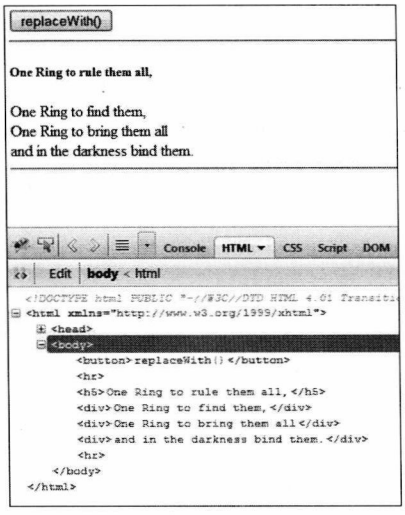

图 6-35 用被替换的 div 元素的内容填充第一个 h5 类型标题

如果我们现在再次点击按钮，发生的过程相同。但是要注意，在前一个例子的这个变种中，内容始终被写入第一个 h5 类型标题。结果是，这个变种中的网站只有类型为 h5 的空白标题。如图 6-36 所示。

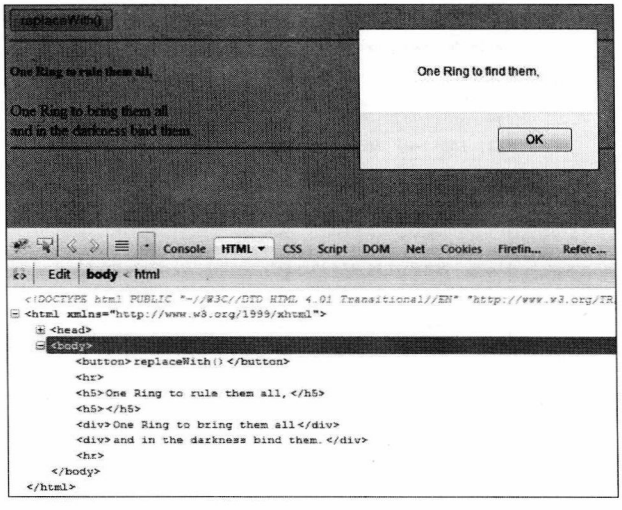

图 6-36 填充的标题数量减少

6.8.2 用 replaceAll() 替换所有元素

replaceAll() 以一个选择器作为参数。所有匹配该选择器的元素都被方法所应用的元素

替换。作为返回值，可以得到所有插入的元素。因此，它的语法和 replaceWith() 相反；但是在使用它时，必须注意是否仍然想要使用原始元素。如前所述，返回值包含了插入的元素。如果需要原始元素或者内容，就必须在替换之前将它们保存到变量中。这通常比 repleceWith() 的使用更复杂。replaceAll() 在想要用相同的结构完全替换一个元素，或者用不同的固定结构替换同一个段落时很实用（ch6_15.html）。效果如图 6-37 所示。

程序清单 6.32 替换网页中的元素

```
...
08   <script type="text/javascript"
09     src="lib/ch6_15_ready.js"></script>
10  </head>
11  <body>
12     <button>replaceAll()</button>
13    <hr />One Ring to rule them all,
14    <hr />One Ring to find them,
15    <hr />One Ring to bring them all
16    <hr />and in the darkness bind them.
17  </body>
18 </html>
```

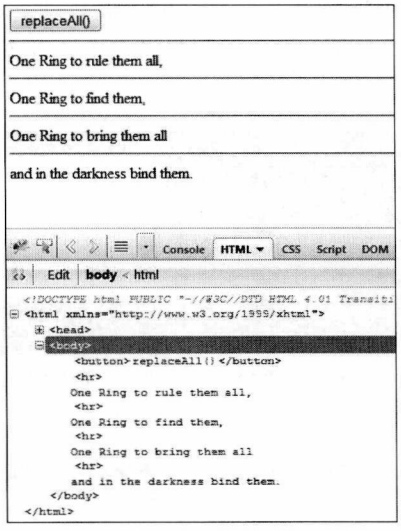

图 6-37 4条分隔线

我们使用4条分隔线分组输出，在用户点击按钮时，我们要用换行代替这些分隔线（ch6_15_ready.js）。

程序清单 6.33 替换分隔线

```
01 $(function(){
02  $("button:first").click(function(){
```

```
03    $("<br />").replaceAll("hr");
04  });
05 });;
```

请确保想要替换的参数被指定为参数，要用来替换的元素通过前导的 jQuery 对象指定。还要确保所替换的元素没有按照标签语法编写。效果如图 6-38 所示。

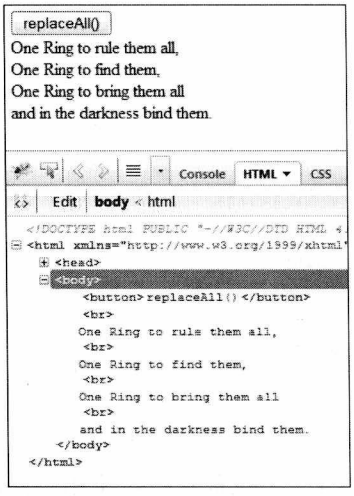

图 6-38 操作后的网站。新结构在 Firebug 中可以很清晰地看到

6.9 用 empty() 和 remove()/detach 以及 removeAttr() 删除

根本上，empty() 和 remove() 都提供了从网页中删除节点的选项。利用 empty() 方法，可以简单地删除匹配的一组元素的所有子元素（包括文本节点）。而使用 remove()，可以从 DOM 中删除所有匹配的元素（和子元素）。

但是两个方法有一些深层的差异。例如，remove() 不从 jQuery 对象中删除元素，仍然可以进一步使用它们，但是所有处理程序和内部缓存数据都被删除。remove() 的优点在于将 DOM 删除的元素作为返回值，可以进一步利用它们。而且，可以指定一个过滤器表达式，作为 remove() 的可选参数。

警告

如果结合一个过滤表达式[⊖]进行删除，使用通过 remove() 删除的元素就有些困难（因为 remove() 的返回值总是应用 remove() 的元素的完整集合）。换句话说，虽然过滤表达式避免不匹配的元素被从 DOM 中删除，返回的 jQuery 对象仍然包含[⊖]所有元素，甚至包含没有被

⊖ 首要的是，有些以类似方式工作的 jQuery 方法不完全一致。

⊖ 或者引用。

删除的元素！所以，如果打算在后续步骤中进一步使用 jQuery 对象，需要操作完整的元素集合，并且可能需要再一次直接选择单独元素。下面的例子说明了这种行为（ch6_16.html）。

程序清单 6.34 删除元素

```
...
08   <script type="text/javascript"
09     src="lib/ch6_16_ready.js"></script>
10  </head>
11  <body>
12    <h1>Ash nazg durbatulûk,</h1>
13    <h1 id="i1">ash nazg gimbatul,</h1>
14    <h1>ash nazg thrakatulûk</h1>
15    <h1>agh burzum-ishi krimpatul</h1>
16    <button>empty()</button>
17    <button>remove() - all</button>
18    <button>remove() - only once</button>
19    <button>Add one element</button>
20    <button>Add all elements</button>
21    <hr id="i2">
22  </body>
23 </html>
```

在上面的网页中，我们使用 4 个 1 级标题，希望用不同的方式删除。通过 remove() 删除的元素被插入到树中的另一个位置（ch6_16_ready.js）。原始效果如图 6-39 所示。

程序清单 6.35 删除节点的技术

```
01 var element = null;
02 $(document).ready(function(){
03  $("button:first").click(function(){
04    $("h1").empty();
05  });
06  $("button:eq(1)").click(function(){
07    element = $("h1").remove();
08  });
09  $("button:eq(2)").click(function(){
10    element = $("h1").remove("#i1");
11  });
12  $("button:eq(3)").click(function(){
13    $("#i2").after(element[1]);
14  });
15  $("button:eq(4)").click(function(){
16    $("#i2").after(element);
17  });
18 });
```

如果用户点击第一个按钮，调用 empty() 方法（第 4 行），所有 1 级标题被清空。这意味着子节点（文本节点）从树中删除。如图 6-40 所示。

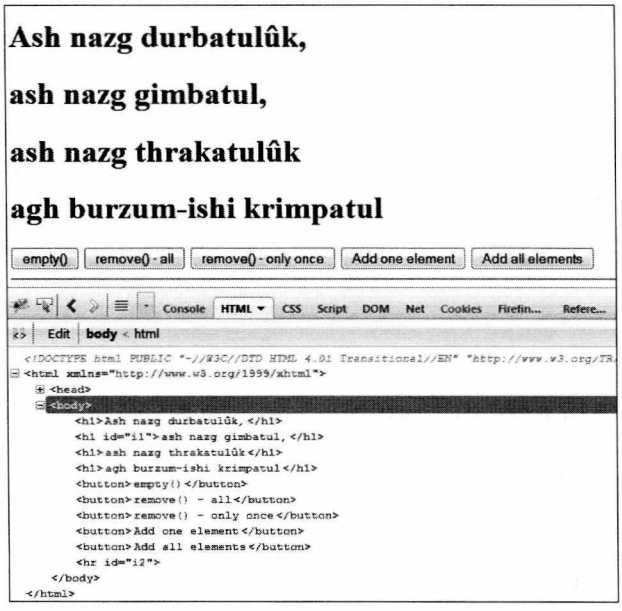

图 6-39 包含树相关部分的未修改网页

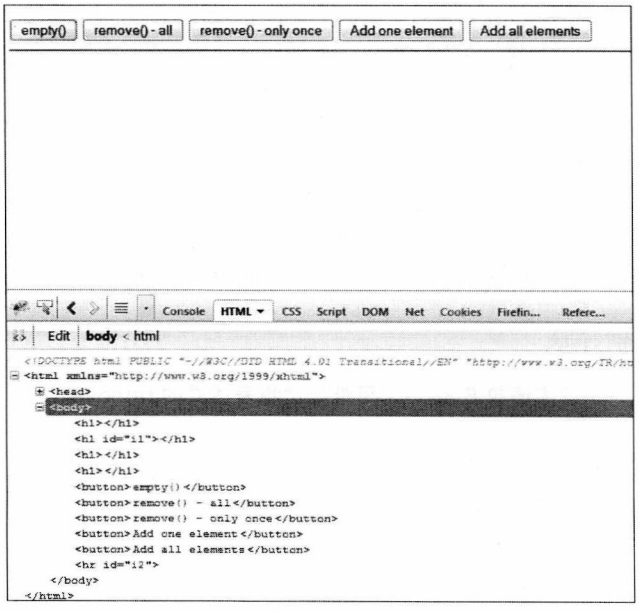

图 6-40 所有标题的内容在第一个按钮被点击时被删除

注意，我们在第 1 行创建了一个全局变量 element，它将接受通过 remove() 删除的元素。在第 7 行中，可以看到我们使用没有参数的 remove()。

程序清单 6.36 使用没有参数的 remove()：删除所有 1 级标题

```
element=$("h1").remove();
```

被删除的 1 级标题被保存在 element 变量中，从树中被删除。如果点击第 4 个按钮，它们通过 after() 方法[⊖]被重新插入 ID 为 i2 的分隔线之后。效果如图 6-41 所示。

程序清单 6.37 将保存 element 变量中的元素插入另一个位置

```
23 $("#i2").after(element);
```

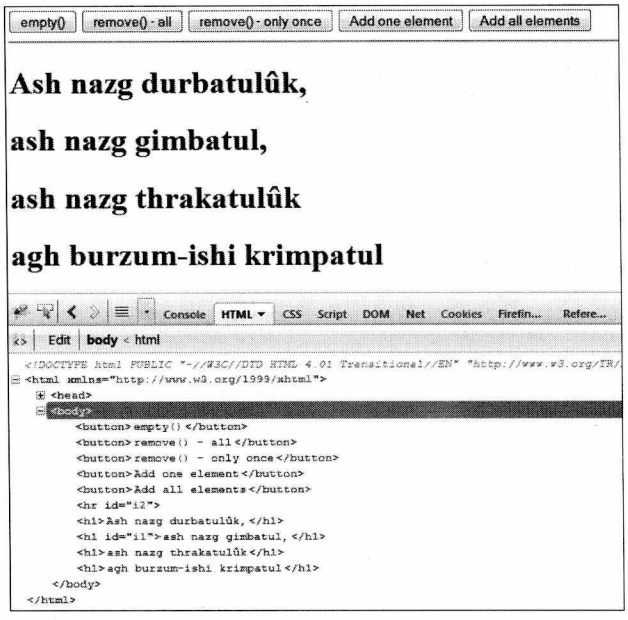

图 6-41 用 after() 将通过 remove() 删除的元素插入另一个位置

第 10 行现在特别有趣。

程序清单 6.38 使用带有过滤表达式的 remove()

```
element=$("h1").remove("#i1");
```

在这里可以看到，我们是如何使用带有一个过滤表达式的 remove() 的。我们只想删除 ID 为 i1 的标题。但是如果接着点击第 4 个按钮，4 个 1 级标题仍然会全部插入到新位置。而且，其余 3 个未删除的 1 级标题也从树的原始位置删除。[⊖]所以不管有无过滤器，element 都包含所有 1 级标题。

所以，如果只想将 element 数组中过滤的元素插入另一个位置，必须在数组中特别处

⊖ 前面已经介绍过。

⊖ 这是合乎逻辑的结果，因为我们不想复制这些元素。

理。如果用户点击第 3 个按钮，我们就会进行这种处理，注意第 13 行。

程序清单 6.39 插入元素数组中的第 2 个元素

```
$("#i2").after(element[1]);
```

和 remove() 方法的过滤器同步，现在只有数组中匹配的元素被插入并相应地从原始位置中删除。其他标题保持不变。效果如图 6-42 所示。

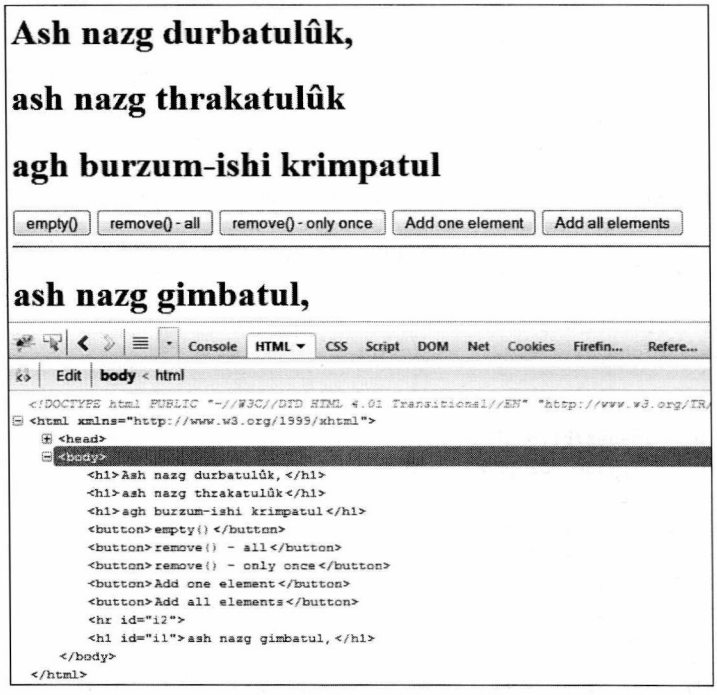

图 6-42 一个元素从树中删除并被插入另一个位置；其他元素保持在原来的位置

6.9.1 remove() 的替代品 :detach()

也可以用 detach() 代替 remove()。该方法几乎和 remove() 相同，但是 detach() 保留所有与被删除元素相关的 jQuery 数据。这在重新将它们插入 DOM 树时很关键。

6.9.2 删除特性

为了删除特性，jQuery 提供了 removeAttr() 方法。它的使用相当简单；只需要指定想要删除的属性作为参数即可。这里不需要一个完整的例子。

程序清单 6.40 删除 disable 特性

```
$("input:first").removeAttr("disabled");
```

注意

removeAttr() 方法直接使用 JavaScript 函数 removeAttribute()，但是通过 jQuery 命名空间封装，避免了与不同浏览器中特性的不同命名相关的问题。

6.10 用 clone() 进行克隆

现在，我们必须应对一种情况，用 after() 插入的元素从原始位置删除节点。这种情况和其他插入元素的方法也相关。毕竟，在大部分情况下，我们不想复制或者克隆元素。但是，用 jQuery 进行克隆当然也是可能的（如果需要，甚至包括事件处理器或者分配给元素的事件处理器）。可以通过 clone() 方法完成这项工作。它有一个可选的布尔参数。如果该参数为 true，还能克隆事件处理器或者事件辅助程序。该参数默认为 false（ch6_17.html）。

程序清单 6.41 克隆元素

```
...
08     <script type="text/javascript"
09       src="lib/ch6_17_ready.js"></script>
10   </head>
11   <body>
12     <img src="images/b1.png" />
13       <img src="images/b2.png" /><hr/>
14    <button>clone() Image 1</button>
15    <button>clone(true) Image 2</button>
16    <button>clone(false) Image 2</button>
17    <hr/><div id="info"></div><hr id="i2">
18  </body>
19 </html>
```

在上述网页中，我们有两张图片需要克隆。克隆之后，它们被重新插入树中（ch6_17_ready.js）。

程序清单 6.42 克隆元素

```
01 $(function(){
02  $("img:last").click(function(){
03    $("#info").html("Number of images: "
04       + $("img").length);
05  });
06  $("button:first").click(function(){
07    element = $("img:first").clone();
08    $("#i2").after(element);
09  });
10  $("button:eq(1)").click(function(){
11    element = $("img:eq(1)").clone(true);
12    element.css({
13      border: "solid 1pt"
```

```
14    });
15    $("#i2").after(element);
16  });
17  $("button:eq(2)").click(function(){
18    element = $("img:eq(1)").clone(false);
19    $("#i2").after(element);
20  });
21 });
```

在第 2 ～ 5 行中，可以看到网页的最后一幅图片分配了一个事件助手。如果用户点击网页中的最后一幅图片（加载时的第二幅图片），显示网页的图片数量。注意，我们明确地使用了 :last 过滤器。这在后面克隆图像并将其插入网页时很重要。[⊖]当我们这么做时，出现了一个问题——事件辅助程序如何分配（因为加载网页时的最后一幅图片在以后不再是网页中的最后一幅图片）。下一个问题是，在我们克隆事件辅助程序时发生了什么。如图 6-43 所示。

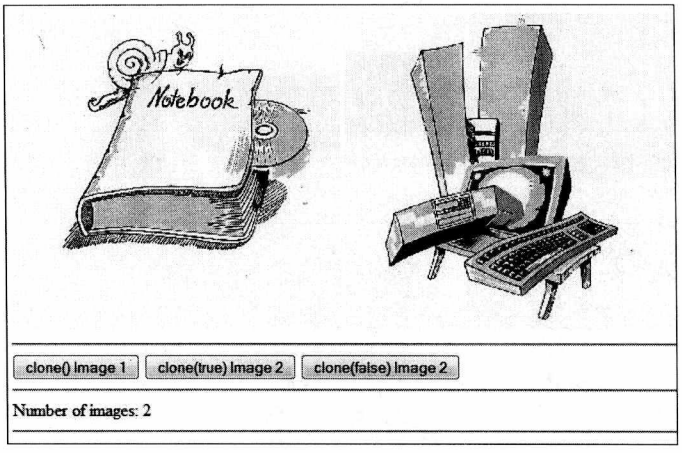

图 6-43 网页中的第二幅图片在加载之后是鼠标点击敏感的；点击在网页中产生一条消息

在第 7 行中，可以看到第一幅图片是如何克隆的。副本被保存在 element 变量中，以后可以进一步利用。在例子中，我们简单地将该元素重新插入树中的另一个位置（在 ID 为 i2 的分隔线之后）。如图 6-44 所示。

每次点击按钮都创建一个新的克隆，并将其插入网页。如图 6-45 所示。

现在，我们将看到分配事件处理时发生的情况。因为我们在第 2 行的源文本中使用 $("img:last")，最后一幅图片应该是敏感的，在点击时写入网页。但是如果点击它，你将会看到情况并非如此。但是，如果点击第二幅图片，开始时指定的操作再次被触发。如图 6-46 所示。

⊖ 例如，在实践中，可以为使用购物车的在线商店采用这种方法，选择由图标代表的一种产品。如果该产品被放入购物车，则创建在线商店中该图标的一个副本，并在购物车中显示。

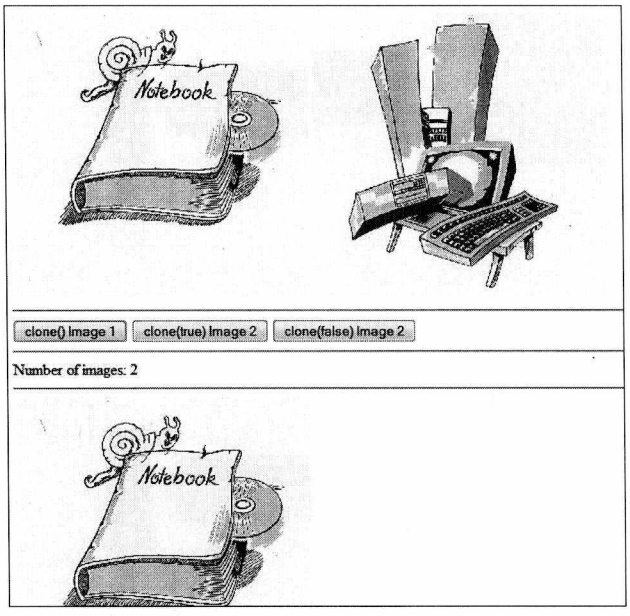

图 6-44 第一幅图片被克隆并插入分隔线之后；显示的图片数量还没有更新

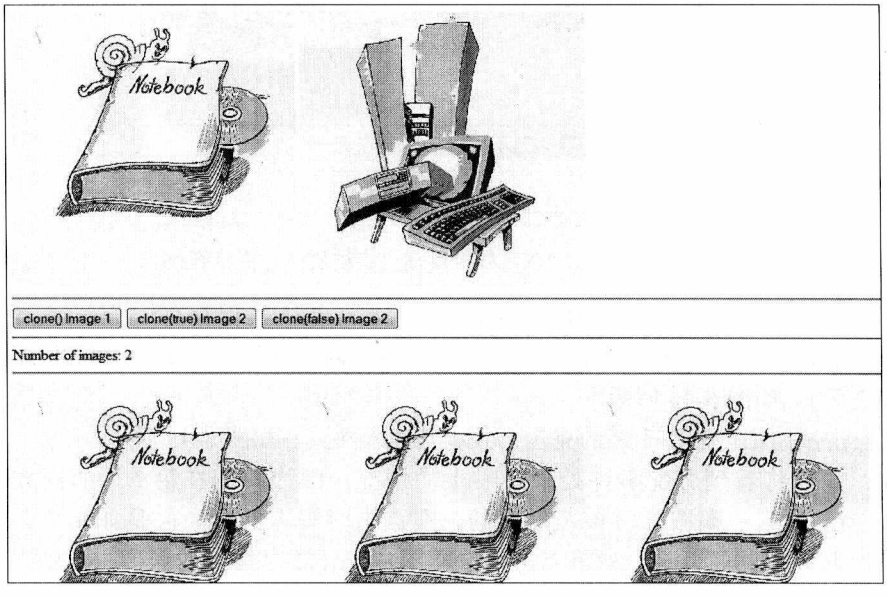

图 6-45 现在，图片 1 有了 3 个克隆

显然，框架通过 jQuery 选择器和回溯性动态添加的元素解决了原始赋值意图问题。框架在内部赋予和运行开始时一样的值——尽管现在这和源代码中的语法已经不相适应。一方面，这是很棒的缓解手段，我们再次看到了 jQuery 框架的优势；另一方面，我们当然要考

虑这种行为，因为可能确实想要重新定位事件处理。

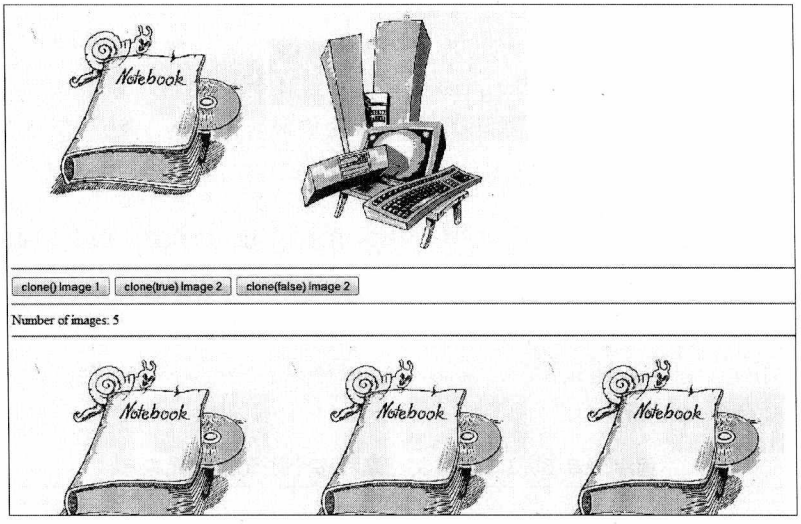

图 6-46 当点击第二幅图片，操作被触发

我们还要看看，如果克隆事件处理，会发生什么。可以在第 11 行中看到，在该行中我们指定布尔参数为 true。后续添加的一个画框只是为了从视觉上将敏感克隆的元素与非敏感克隆的元素区分开来，没有其他效果。

如果尝试这个例子，就会看到利用这种方法，事件助手实际用于克隆的元素。而第二个元素仍然保持敏感。框架在后台负责克隆和正确的赋值。如图 6-47 所示。

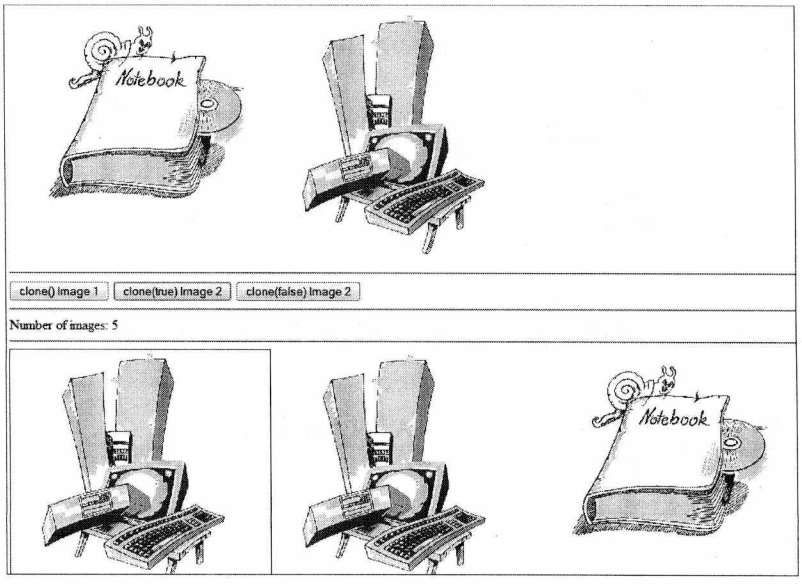

图 6-47 加框的图片是敏感的，和第二幅图片一样

6.11 搜索和查找

jQuery 为网页中的搜索相关结构提供了一些强大的方法。与其他各种不同 Web 编程相比，这些方法的特征是在纯粹 JavaScript 或者动态超文本标记语言（DHTML）中没有任何替代方法。在这个领域，使用 jQuery 能够增添特殊的价值。jQuery 方法当然还是基于典型的树导航，就像在 XPath 中那样。

6.11.1 子节点和父节点：children() 和 parent() 加上 parents()/parentsUntil()

网页树形结构中非常重要的关系之一是父节点和子节点之间的关系。利用 children() 方法，可以确定一个或者多个父元素的所有直接子元素，并将它们当作 jQuery 对象返回。该方法也可以使用一个过滤表达式参数。类似地，parents() 方法返回除了根之外，包括一个元素的所有祖先（ch6_18.html）。

程序清单 6.43 具有父元素和多个子元素的元素

```
...
08     <script type="text/javascript"
09       src="lib/ch6_18_ready.js"></script>
10   </head>
11   <body>
12     <div>
13       <img src="images/b1.png" />
14       <div class="c1">
15         <img src="images/b2.png" /><br/>
16         <img src="images/b3.png" class="c1"/>
17       </div><hr/>
18     </div>
19     <button>All children of the first DIV</button>
20     <button>All children in a DIV that has
21        the class c1</button>
22     <button>All children of the parent of the
23        3rd image</button>
24     <button>All parents of the 2nd image</button>
25     <div id="output"></div>
26   </body>
27 </html>
```

在网页中，我们有两个嵌套的 div 元素，我们将搜索其父元素和子元素结构（ch6_18_ready.js）。

程序清单 6.44 确定父元素和子元素

```
01 $(function(){
02   $("button:first").click(function(){
03     element = $("div:first").children();
04     $("#output").text("Number of child elements: " +
05     element.length);
```

```
06 });
07 $("button:eq(1)").click(function(){
08   element = $("div").children(".c1");
09   $("#output").text("Number of child elements: " +
10   element.length);
11 });
12 $("button:eq(2)").click(function(){
13   element = $("img:eq(2)").parent();
14   $("#output").text("Number of child elements: " +
15   element.children().length);
16 });
17 $("button:eq(3)").click(function(){
18   elements = $("img:eq(1)").parents();
19   $("#output").text("Number of parent elements: " +
20   elements.length);
21    for (i = 0; i < elements.length; i++)
22     $("#output").append("<br />" + i + ": " +
23        elements[i].nodeName);
24 });
25 $("img").css("width", "150px");
26 });
```

注意，网页中的图片由换行（
）和水平线（<hr />）分隔。这些元素在树中也是独立的节点，被视为子元素。相应地，第一个 div 容器中子元素的数量为 3（两幅图片和一个换行）。如图 6-48 所示。

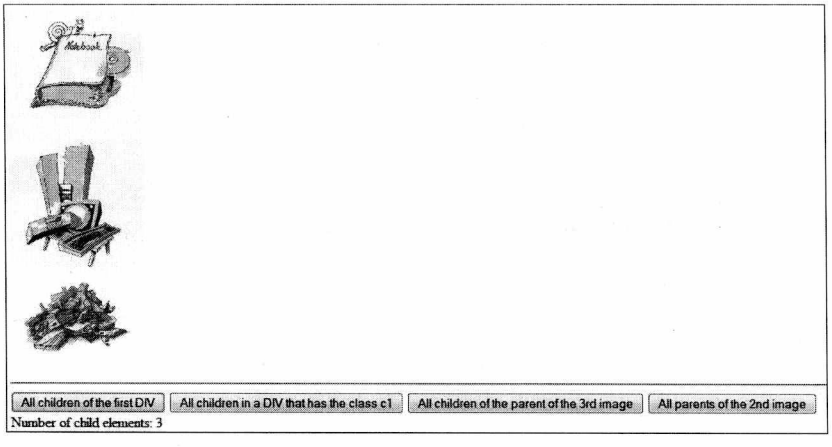

图 6-48 在第一个 div 容器中有三个子元素

使用 children()，我们在第 3 行中确定 div 容器的所有子元素，并将其保存到 element 变量中。显示的文本证明有 3 个子元素。因为变量 element 现在包含一个应用这些元素的 jQuery 对象，我们可以通过 length 确定这一数量，就像正常数组的操作一样。

在第 8 行中，我们使用一个过滤表达式作为 children() 的参数。这个过滤表达式确保只有 jQuery 对象 element 中的元素与过滤器匹配。这样的元素有两个，你很容易自己尝试。

在第 13 行中，我们选择第三幅图片的父元素，然后确定它的子元素数量。这样，我们就有了这个元素的兄弟元素的信息。我们再来看看 HTML 文件中的一个代码块。

程序清单 6.45 从第三幅图片到父元素，然后计算所有子元素

```
<div class="c1">
  <img src="images/b2.png" /><br/>
  <img src="images/b3.png" class="c1"/>
 </div>
```

我们从第三幅图片开始，通过 parent() 转到包围它的 div 元素。这个元素包含三个子元素，因为换行也被计算在内。

第四种情况选择第二个图片元素，从这里返回除了根之外的所有父元素（由内向外，与数据字段的索引相关）。

如果我们在 firebug 中查看 DOM 树，就可以从第二幅图片看到直接的父元素是一个 div 元素。这个 div 元素的父元素是另一个 div 元素，该元素的父元素是 body。这样，我们已经有了 3 个父元素。body 是 html 元素的子元素，所以第二个图片元素有 4 个（非直接）父元素。如图 6-49 所法。

图 6-49 从内向外有 4 个父元素

警告

在通过 parents() 确定父元素时，有一个相当奇怪的现象。[⊖]至少，我认为使用这个方法时，框架的行为有些不一致或者危险。如果用第 21 行中的 for(i in elements) 循环读取 parents() 中返回的数组，会得到 root#document 及其所有的 DOM 属性。可以用配套网站上的示例文件 ch6_18_1.html 测试。

⊖ 至少在本书所用的 jQuery 版本中是如此。

从 jQuery1.4 起，parents() 有一个 parentsUntil() 形式的变种。这个方法的工作方式与 parents() 相同，但是可以指定一个参数作为选择器，指定父元素选择的停止点。通过选择器指定的元素不再被选择。如果完全没有应用该选择器，所有父元素都将被选择，就像 parents() 一样。

6.11.2 offsetParent() 和 closest()

对于父元素和子元素的关系，还可以使用 closest()[⊖] 和 offsetParent() 方法。利用 closest()，可以得到一组直接父元素与指定选择器相匹配的元素（包括起始元素）。这个方法首先检查当前元素是否匹配指定表达式。如果匹配，当前元素将被返回。如果不匹配，该方法向上一个接一个地遍历父元素，直到找到匹配指定表达式的元素。如果没有一个元素匹配，则什么都不返回。

offsetParent() 方法返回一个 jQuery 集合，包含第一个匹配元素的被定位[⊜]父元素，但是该方法只对可见元素有效（ch6_19.html）。

程序清单 6.46 寻找元素的子元素和父元素

```
...
08      <script type="text/javascript"
09        src="lib/ch6_19_ready.js"></script>
10    </head>
11    <body>
12      <div>
13       <div>
14         One Ring to rule them all,
15               <h6>One Ring to find them, </h6>
16         <h5>One Ring to bring them all </h5>
17         <h5>and in the darkness bind them.</h5>
18       </div><hr/>
19       <table border="1" style="position:absolute">
20         <tr><td>closest("div") von $("h5:eq(0)")</td>
21           <td id="output1"></td></tr>
22         <tr><td>offsetParent() von $("td:eq(1)"</td>
23           <td id="output2"></td></tr>
24       </table>
25     </div>
26   </body>
27 </html>
```

下面是 JavaScript 文件 ch6_19_ready.js。

⊖ jQuery 1.3 时。

⊜ 通过 CSS 的 position 属性，不管定位是绝对的还是相对的。

程序清单 6.47 closest() 和 offsetParent()

```
01 $(function(){
02   $("#output1").text($("h5:eq(0)").
03     closest("div").html());
04  $("#output2").text($("td:eq(1)").
05     offsetParent().html());
06 });
```

在第 3 行中，我们用 $("h5:eq(0)").closest("div") 搜索类型为 div，最接近第一个 h5 类型标题的父元素。如果查看网页正文，就会看到这个父元素是 HTML 文件中从第 13 到 18 行的内部 div 容器。我们通过 html() 获得其内容，并用 text() 不加解释地显示。

在第 5 行中，我们确定第二个表格单元的偏移父元素。当然，表格单元包围在一个 <table> 元素中，它具有 position 属性，正如在 HTML 文件第 19 行中所看见的那样。相应地，我们用 html() 获得了表格[⊖]的完整内容（表格被当作一个 jQuery 对象），我们再次用 text() 输出这个对象。效果如图 6-50 所示。

One Ring to rule them all,

One Ring to find them,

One Ring to bring them all

and in the darkness bind them.

closest("div") of $("h5:eq(0)")	One Ring to rule them all, One Ring to find them, One Ring to bring them all and in the darkness bind them.
offsetParent() of $("td:eq(1)"	<tbody><tr><td>closest("div") of $("h5:eq(0)")</td> <td id="output1"> One Ring to rule them all, One Ring to find them, One Ring to bring them all and in the darkness bind them. </td></tr> <tr><td>offsetParent() of $("td:eq(1)"</td> <td id="output2"></td></tr> </tbody>

图 6-50 closest() 和 offsetParent() 的效果

6.11.3 兄弟元素

通过树中的关系寻找元素可以用除父子关系之外的几个其他方法完成。这些方法的应用方法和 children() 或者 parent() 方法相同，所以我们在此只做简单的讨论。这些方法利用的是兄弟关系；换句话说，我们搜索的是和选择器元素同级别的元素。

next() 方法返回一个 jQuery 对象，指向一个或者一组元素之后的直接兄弟元素。nextAll() 方法的工作方式几乎相同，但是当前元素的所有兄弟元素都将返回。

还可以用 prev() 获得当前元素的直接前驱兄弟，用 prevAll() 得到元素之前的所有兄弟元素。

利用 siblings()，可以获得当前元素或者一组元素前后的所有兄弟元素。

⊖ 以浏览器内部处理内容的形式。

提示

利用可选的过滤表达式作为参数，可以限制所有方法的选择，描述同前。

程序清单 6.48 兄弟关系（ch6_20.html）

```
...
08     <script type="text/javascript"
09       src="lib/ch6_20_ready.js"></script>
10   </head>
11   <body>
12     <div>
13      <h3>Tiger, tiger, burning bright</h3>
14      <h3>In the forests of the night</h3>
15      <h3>What immortal hand or eye</h3>
16      <h3>Could frame thy fearful symmetry?</h3>
17      <h3>In what distant deeps or skies</h3>
18      <h3>Burnt the fire of thyne eyes?/h3>
19      <h3>On what wings dare he aspire?</h3>
20      <h3>What the hand dare seize the fire?</h3>
21    </div><hr />
22     <h2>Relationships of the third
23       heading of the type h1</h2>
24     <table border="1">
25      <tr><td>next()</td><td id="output1"></td></tr>
26      <tr><td>nextAll()</td><td id="output2"></td></tr>
27      <tr><td>prev()</td><td id="output3"></td></tr>
28      <tr><td>prevAll()</td><td id="output4"></td></tr>
29      <tr><td>siblings()</td><td id="output5"></td></tr>
30     </table>
31   </body>
32 </html>
```

这里我们有一个示例网页，包含了几个同级别的 h3 类型标题。下面是 JavaScript 文件 ch6_20_ready.js。

程序清单 6.49 兄弟元素

```
01 $(function(){
02  $("#output1").text($("h3:eq(2)").next().text());
03  $("#output2").text($("h3:eq(2)").nextAll().text());
04  $("#output3").text($("h3:eq(2)").prev().text());
05  $("#output4").text($("h3:eq(2)").prevAll().text());
06  $("#output5").text($("h3:eq(2)").siblings().text());
07   $("h3:eq(2)").css("background","lightgray");
08 });
```

在上述例子中，我们确定第 3 个 h3 标题的兄弟元素，也就是内容为“What immortal hand or eye”的标题。这个标题在第 7 行中高亮显示，只是为了增加视觉上的强调。脚本的输出如前所述。如图 6-51 所示。

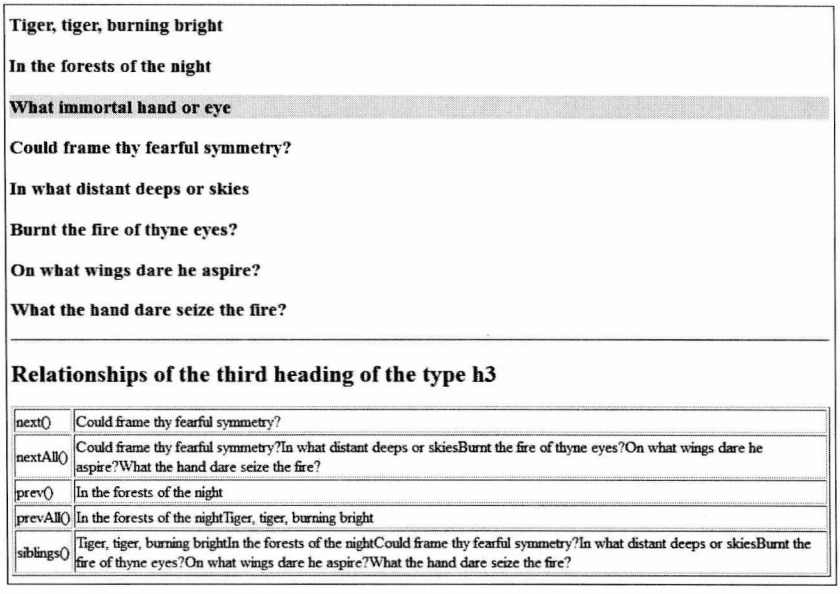

图 6-51 第 3 个 h3 类型标题的各个兄弟元素

用 next()，我们得到了直接后续的 h3 类型标题。通过 nextAll()，选择所有后续的 h3 类型标题，相应地，prev() 可以获得直接前驱的标题。用 prevAll()，可以获得选中的元素之前的两个标题。而 siblings() 可以选择除了所操作的选中元素之外的所有 h3 类型标题。

Until 变种

从 jQuery 1.4 起，nextAll() 和 prevAll() 有了形式为 nextUntil() 和 prevUntil() 形式的变种。这些方法的工作方式与两个兄弟方法完全一样，但是可以指定一个选择器参数，表示兄弟选择的停止点。通过选择器指定的元素不会被选择。

程序清单 6.50 兄弟元素的选择直到类“k1”为止

```
$('div').nextUntil('.k1')
```

如果选择器没有匹配项，指定方向的所有兄弟都被选中。

6.11.4 用 has() 搜索后代

从 jQuery1.4 起就有了 has() 方法，可以指定一个选择器或者 DOM 元素作为参数。这个元素必须是前导元素在树中的后代，符合这一条件的前导元素才会被选中。

程序清单 6.51 后代规则

```
$('div').has('p').css('background-color', 'red');
```

在这个例子中，只有具有类型为 p 的后代的 div 元素会被选中。

6.12 用 find() 和 contents() 寻找

以过滤表达式为参数的 find() 和 contents() 是寻找网页元素及其内容的两个有趣方法。用 find()，可以搜索一个元素的后代元素，了解它们是否匹配过滤表达式。

警告

当前元素不会被搜索！这就是 filter() 方法的目的。

使用 contents()，可以获得所关心的元素所有的子节点（包括文本节点）。如果将这个方法应用到内联框架，就可以获得包含的文档（ch6_21.html）。

程序清单 6.52 基本网站

```
...
08     <script type="text/javascript"
09       src="lib/ch6_21_ready.js"></script>
10   </head>
11   <body>
12     <div>
13       <h5>One Ring to rule them all,</h5>
14       <h6>One Ring to find them, </h6>
15       <h5>One Ring to bring them all </h5>
16       <h4>and in the darkness bind them.</h4>
17     </div>
18     <hr>
19     <table border="1">
20       <tr><td>find("h5") in $("div:eq(0)"</td>
21         <td id="output1"></td></tr>
22       <tr><td>contents() in $("h5")</td>
23         <td id="output2"></td></tr>
24     </table>
25   </body>
26 </html>
```

我们的示例网页在一个 div 区域中包含多个标题。下面是 JavaScript 文件 ch6_21_ready.js。

程序清单 6.53 find() 和 contents()

```
01 $(function(){
02   $("#output1").text($("div:eq(0)").find("h5").text());
03   $("#output2").text($("h5").contents().text());
04 });
```

在这个例子中，可以看到 find() 返回过滤表达式中指定的 5 级标题的内容，contents() 返回我们通过 $(“h5”) 选择的所有子元素内容。在我们的例子中，两次返回的结果相同。如图 6-52 所示。

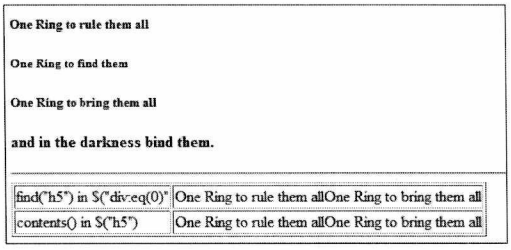

图 6-52 find() 和 contents() 的结果

6.13 在数组和对象中循环的 jQuery 方法 each()

现在，我们转向一个非常基础而又十分重要的部分。我们已经知道，在 JavaScript 中通常使用句点标记法访问对象的组件。替代方法是数组标记法。这包括通过数据字段标记法写入和读取属性或者方法，其中可以使用有意义的索引。在 JavaScript 中，对象的实现与数组完全相同——就是包含属性和相关值配对的列表。对于方法来说，值就是一个函数引用。

我们来看一个原生 JavaScript 的简单例子，例中创建一个对象，并通过数组标记法访问。

程序清单 6.54 通过数组标记法访问一个对象

```
var x = {
  name: "Felix",  age: 11,  occupation: "Student"
};
var index = "name";
document.write(x[index]); // "Felix"
document.write(x[age]); // 11
```

这个过程也被称作关联数组（Associative Array）。在 JavaScript 中（特别是循环读取一个对象的属性和方法时），可以完全一致地采用数组方法。由于对象中的索引不是数字[⊖]，而是属性和方法名称，只需要在 for-in 循环中使用适合的关联数组技术即可。

为澄清概念，我们来看一个完整的例子，这个例子完全没有使用 jQuery，使用内部 JavaScript（ch6_22.html）。

程序清单 6.55 在对象中循环

```
...
07  <body>
08    <table width="300" border="1">
09      <tr>
10        <th>Property</th>
11        <th>Value</th>
12      </tr>
13      <script type="text/javascript">
```

⊖ 严格地说，JavaScript 中的数字索引页是字符串，只是内容为数字。

```
14        var x = {
15          name: "Felix",
16          age: 11,
17          occupation: "Student"
18        };
19        for (i in x) {
20          document.write("<tr><td>" + i +
21             "</td><td>" + x[i] + "</td></tr>");
22        }
23      </script>
24    </table>
25  </body>
26 </html>
```

我们根据以上的描述创建一个对象，并显示在一个表格中动态显示其属性及属性的前导标识符，这个表格的基本结构用纯 HTML 建立。我们访问对象 x 的属性，以索引 i 为循环计数。如图 6-53 所示。

Property	Value
name	Felix
age	11
occupation	Student

图 6-53 在表格中输出属性

目前，纯 JavaScript 工作得很好。但是作为常规循环的补充，jQuery 提供了一个通用函数 jQuery.each()，循环读取对象和数组元素的所有属性。这个函数比经典的 JavaScript 技术更灵活，而且更加稳定可靠，在 jQuery 中的使用很频繁。[⊖]还有一个 each() 方法，可以应用到前导的数组或者对象中。

6.13.1 jQuery.each()

jQuery.each() 的第一个参数指定所要循环读取的数组或者对象。如果指定一个（匿名）函数[⊖]作为第二个参数，就可以通过第一个参数得到数组或者对象的索引，从第二个参数获得相关的值（ch6_23.html）。[⊜]

程序清单 6.56 基本站点

```
...
08    <script type="text/javascript"
09      src="lib/ch6_23_ready.js"></script>
10  </head>
11  <body>
12    <table width="300" border="1" id="tab">
13     <tr>
14       <th>Property</th>
15       <th>Value</th>
16     </tr>
17   </table>
```

⊖ 特别是，几乎在所有插件中都使用了。

⊖ 通常，你可以在这个函数中写下一个回调。

⊜ 注意，我们再次从标准结构开始，源代码的前 7 行集成了 jQuery。

```
18  </body>
19 </html>
```

这里，我们的网页结构和前一个不使用 jQuery 的例子相同。但是表格内容缺失，更确切地说是我们用内部 JavaScript 容器创建的部分和 for-in 循环，因为我们用 jQuery.each() 生成这些部分（ch6_23_ready.js）。

程序清单 6.57 使用 jQuery.each()

```
01 $(function(){
02  var x = {
03    name: "Felix",
04    age: 11,
05    occupation: "Student"
06  };
07  jQuery.each(x, function(index, value){
08    $("#tab").append("<tr><td>" + index
09       + "</td><td>" + value + "</td></tr>");
10  });
11 });
```

现在我们创建的仍然是同一个对象。在第 7 行中，可以看到循环的开始部分，对象 x 是第一个参数，第二个参数是一个匿名函数。这个函数由两个参数，第一个参数代表循环的索引，而第二个参数是此时对象或者数组中的值。

警告

我们已经为函数参数选择了有意义的名称，但是这些名称没有关系，它们不是关键词或者具有特定意义的标志。可以选择不同的名称，但是这不是个好的决定，因为它可能导致误解，使代码难以理解。

我们来看另一个例子，这里我们使用具有一个回调的 jQuery.each()，所要循环读取的是一个对象集合。换句话说，在这一背景中，我们感兴趣的不是集合中单独对象的属性，而是集合本身。

提示

在回调函数中，this 代表循环中目前指定的 DOM 元素。可以用 $(this) 获得对应的 jQuery 对象。

程序清单 6.58 基本站点（ch6_24.html）

```
...
08     <script type="text/javascript"
09       src="lib/ch6_24_ready.js"></script>
10   </head>
11   <body>
12     <div>An answer for you?</div>
```

```
13      <div>Yes. I have.</div>
14    <p>There really is one?</p>
15    <div>There really is one.</div>
16    <p>To Everything?</p>
17      <p>To the great Question of Life,
18      the Universe, and Everything?</p>
19    <div>Yes.</div>
20    <p>And you're ready to give it to us?</p>
21    <div>I am.</div>
22    <p>Now?</p>
23    <div>Now.</div>
24    <div>Though I don't think
25      that you are going to like it.</div>
26    <button>Deep Thought</button>
27    <button>Fool</button>
28  </body>
29 </html>
```

这里，可以看到一个网页，它的结构包括交替的 div 和 p 代码块。我们希望根据用户点击第一个或者第二个按钮，对它们应用不同的格式。为了选择所有从属元素，我们使用 jQuery.each()（ch6_24_ready.js）。

程序清单 6.59 以回调函数为参数的 jQuery.each()

```
01 $(function(){
02  $("button:first").click(function(){
03    jQuery.each($("div"),function(){
04      this.style.color = "blue";
05      this.style.background = "yellow";
06      this.style.fontSize = "22px";
07    });
08    jQuery.each($("p"),function(){
09      this.style.color = "black";
10      this.style.background = "white";
11      this.style.fontSize = "12px";
12    });
13  });
14  $("button:last").click(function(){
15    jQuery.each($("p"),format);
16    jQuery.each($("div"),function(){
17      $(this).css({
18        color: "black",
19        background: "white",
20        fontSize: "12px"
21      });
22    });
23  });
24 });
25 function format(){
26  $(this).css({
```

```
27     color: "red",
28     background: "lightgray",
29     fontSize: "22px"
30   });
31 }
```

在 JavaScript 文件中，可以看到两个事件助手，通过它们可以实现对用户点击的响应。利用选择器 butto:first，我们选择了网页中第一个按钮，通过 button:last 选择第二个按钮。这对于我们的特殊情况是有效的，因为我们只有两个按钮。根据用户点击的按钮，所有 div 区域或者所有段落被高亮显示。

源代码的第 2 ～ 13 行指定对第一个按钮的响应。点击该按钮触发使用 jQuery.each() 的循环。jQuery.each() 的第二个参数是一个匿名回调函数，在这个函数中改变所选元素的各种 CSS 属性。对于第一个按钮，我们为了方便演示，明确地用 this 代表 DOM 元素。如图 6-54 所示。

对于第二个按钮的响应，我们采用了和按钮 1 略有不同的处理。我们这样做只是为了演示，因为在实践中，应该尽可能避免为相同的问题采用不同的实现方法。

在第 14 ～ 23 行中，可以看到每次 jQuery.each() 循环中，相关段落的格式化通过由 $(this) 代表的 jQuery 对象和 css() 方法完成。这样做不仅更为紧凑，而且也更不容易导致浏览器错误。还要注意，我们使用对外部回调函数（第 25 ～ 30 行）的引用一次（第 15 行），而其他时候采用第 16 ～ 22 行的匿名回调函数。[⊖]注意，我们在外部回调函数中也使 this 或者 $(this) 可用，以便访问循环中当前选择的对象。如图 6-55 所示。

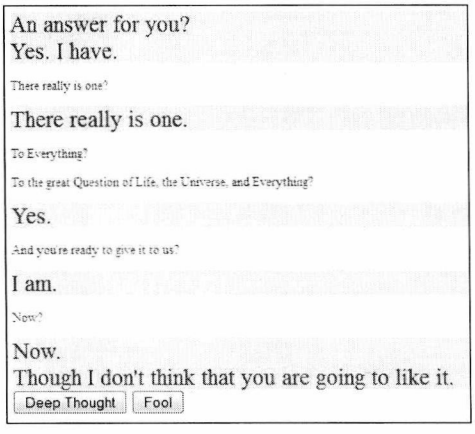

图 6-54 单击按钮 1 之后

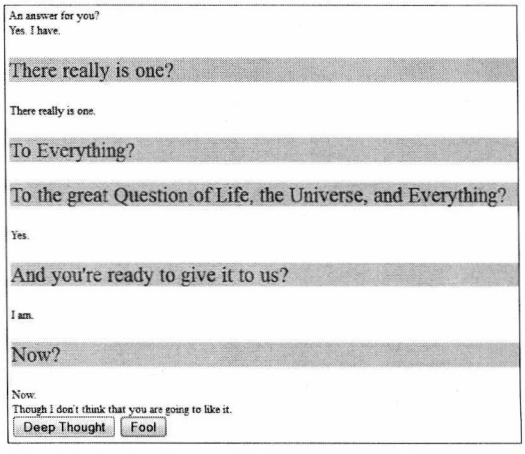

图 6-55 点击按钮 2 之后

提示

也可以在回调中提交一个返回值。值 false 立即停止循环，对应于常规循环中的 break

⊖ 如前所述，我们在这里只是为了阐述不同的方法。

语句。返回值 true 对应于 continue，立即启动下一次循环。

6.13.2 each() 方法

除了通用函数 jQuery.each() 之外，jQuery 框架还提供一个功能相同的 each() 方法。这个方法的使用甚至更简单，因为它应用到一个前导数组或者对象，指定一个匿名函数作为参数。该函数的第一个参数是循环索引，也可以实现回调函数。在匿名函数或者回调函数中，this 仍然代表一个集合中的当前 DOM 元素。所以，我们可以直接使用通用的 jQuery.each() 函数的所有说明，只需要对语法稍作简化。对于示例网站 ch6_25.html，我们用与 ch6_24.html 相同的结构，只删除了按钮。我们来看看 JavaScript 文件 ch6_25_ready.js。

程序清单 6.60 使用 each() 方法

```
01 $(function(){
02  $("div").each(function(index){
03       $(this).prepend((index + 1) + ": ");
04    $(this).css({
05          background:"lightgray"
06      });
07  });
08 });
```

在第 2 行中，我们选择网页中的所有 div 区域，用 each() 循环读取它们。在匿名函数中，循环的索引可以用参数访问。this 是当前 div 元素，我们用 $(this).css() 对其进行格式化。索引略作编辑之后附加到 div 元素的文本之前。如图 6-56 所示。

警告

函数参数名 index 仍然是我们选择的一个有意义的名称，但是它并不重要。

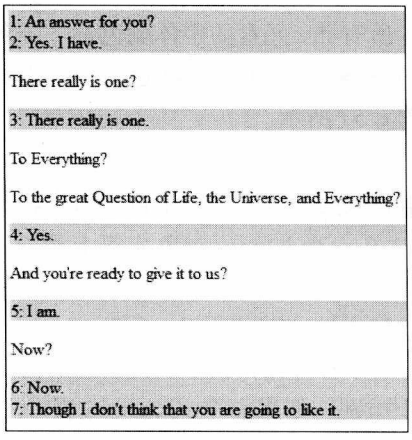

图 6-56 div 被格式化和编号

6.14 add() 方法

我认为，add() 方法很容易引起误解，因为它的名称暗示其功能是为网页或者 DOM 添加节点或者元素。但是该方法的作用并不是这样的。可以用 add() 方法在一组已经选中的元素中添加元素。换句话说，只能扩展之前做出的选择。这一方法使用链接表达式时最为实用。

可以指定一个附加元素必须匹配的选择器、DOM 元素、jQuery 对象或者 HTML 片段作为参数。第二个参数是可选的，可以指定一个上下文。

看看下面的例子，我们使用与 ch6_25.html 结构相同的 ch6_26.html。下面是 JavaScript 文件 ch6_26_ready.js。

程序清单 6.61 使用 add() 方法

```
01 $(function(){
02  $("div").css("background", "yellow").add("p").css({
03    "color": "red"
04  });
05 });
```

在第 2 行中，我们开始只选择所有 div 元素，格式化它们的背景。通过 css() 方法，背景颜色被设置为黄色。该方法提供一个返回值——包含所有选中的 div 元素（现在已经被格式化）的 jQuery 对象。我们可以对该对象应用进一步的 jQuery 方法（换句话说，链接多个方法调用）。这就是 add() 的切入点。调用这个方法，我们可以用所有类型为 p 的元素扩展选择的元素集合。对于扩展后的集合，我们再次应用 css() 方法将文本颜色设置为红色。

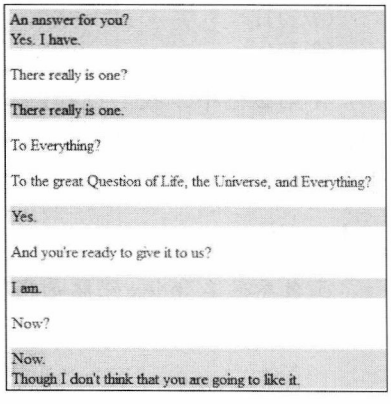

图 6-57 背景颜色只应用到 div 元素

在语句的链接顺序之后，所有 div 元素的字体颜色都为红色，背景为黄色，所有 p 元素的字体都为红色，但是没有黄色的背景。效果如图 6-57 所示。

end() 和 andSelf() 方法

为了完整起见，我打算用其他两个方法结束本章，但是我不认为它们很重要。jQuery 中大部分选择或过滤元素的方法在一个 jQuery 对象实例上操作，如果遇到一组发散的 DOM 元素，则产生一个新的实例。在这种情况下，就像把一组新元素写入一个堆栈中，栈中的现有元素下移。所以，如果仍然需要这些旧的元素，可以使用 end() 从堆栈尾部取回它们。但是正如我曾经说过的，这种方法非常特殊，我不认为会经常需要它。

andSelf() 方法属于同一个类别。如果一个 jQuery 方法将一组新元素写入堆栈，可以用这个方法取回前一组元素。

我们假定有如下的结构。

程序清单 6.62 枚举列表

```
<ul>
   <li>Ice Cream</li>
   <li>Beer</li>
   <li>Candy</li>
   <li>Chocolate</li>
   <li>Peanuts</li>
 </ul>
```

现在我们打算格式化从第 3 个列表项起的所有列表项。我们按照如下的方法进行。

程序清单 6.63 格式化从第 3 个列表项开始的项目

```
$(function(){
  $('li:eq(1)').nextAll().css({
    'background-color': 'red',
        'color':'yellow'
  });
 });
```

也可以采用如下的替代方法。效果如图 6-58 所示。

程序清单 6.64 用 andSelf() 格式化从第 3 个列表项开始的项目

```
$(function(){
  $('li:eq(2)').nextAll().andSelf().css({
    'background-color': 'red',
        'color':'yellow'
  });
 });
```

在第一种情况下，我们选择第二个列表项作为起点，然后选择所有后代。但是在第二种情况下，我们选择第三个列表项，然后选择所有后代，之后明确地选择第三个列表项。

可以在配套网站上看到完整的 ch6_27.html 示例。我不认为这个方法很重要，因为从这两个示例中可以看出，通常可以选择合适的选择器，建立一种不使用这个方法，又能很好地管理元素的方案。

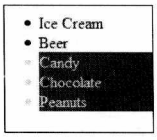

图 6-58 从第 3 个列表项起的项目都被格式化

6.15 更为全面的一个例子：日期组件

为了结束本章，我们创建了一个更复杂的示例，实践选择器和用于循环读取的方法的使用，在更为复杂的环境中操纵元素。首先，我们必须跳过一些知识的介绍，使用在下一章中才会详细介绍的 CSS 技术。但是，这些知识现在不会造成任何严重的问题，而涉及的其他

技术我们都已经使用过。

更准确地说，我们创建一个自动化的日期组件。这在 Web 上不是一个新功能，[⊖]但是在可以应用于许多其他情况的实用性应用程序上实践各种技术是很好的。

我们在创建这个组件时打算完成如下任务：

1. 自动确定并显示当前月份。
2. 根据当前日期，日历应该从本周中合适的日期开始，或者高亮显示每个月的第一天。
3. 如果需要显示前后几个月填充日历空间，为当前月份的日期设置不同于前月和下月日期的格式。
4. 为星期日应用特殊格式。
5. 高亮显示当前日期。
6. 为不同的日历行使用斑马纹图案。
7. 显示月份和年份。
8. 通过模板改变布局。

如你所见，需要做的事情很多。首先看看我们的 HTML 文件 ch6_28.html，但是其中的内容不多。

程序清单 6.65 几乎为空白的网页

```
...
06    <link href="lib/ch6_28_ready.css"
07       type="text/css" rel="stylesheet" />
08    <script type="text/javascript"
09       src="lib/jquery-1.8.2.min.js"></script>
10    <script type="text/javascript"
11       src="lib/ch6_28_ready.js"></script>
12  </head>
13  <body>
14    <h1>RJS Date Component V 1.0</h1>
15    <table/><div/>
16  </body>
17 </html>
```

我们打算基于表格结构创建日历组件。这样做很有意义。只要看看所知道的经典日历就知道了。现在网页明显只是一个雏形。我们要动态地生成整个结构。空白的表格将被扩展，我们打算用 div 区域显示月份和年份。

在第 6 行和第 7 行中可以看到，我们使用一个外部样式表文件 ch6_28_ready.css。这是满足组件可通过模板调整这一需求的基础。在这个例子中，我们所说的模板就意味着一个 CSS 文件。由此，我们已经前进到 jQuery UI 组件所应用的技术和思想（当然，jQuery UI 的方式更高级、更复杂）。在 jQuery UI 等高级框架中，模板也是一个 CSS 库。替换 CSS 文件已经能够完全地改变布局，改变的程度达到了组件的意图。这意味着，我们将日期组件中的

⊖ 已经有了一两个这方面的解决方案（在 jQuery UI 中就有）。

某些组件与 CSS 类联系起来（就像 jQuery UI 中所做的那样）。这里有 CSS 文件的一个变种，显示和我们的组件相关的结构。

程序清单 6.66 默认模板文件

```
table {
      font-size:12px;
}
tr{
      background: yellow;
}
.currentDay{
      color: red;
      border:solid 1pt;
}
.odd{
      background:lightgray
}
.headercells{
      background:gray
}
.sunday{
      background:cyan
}
```

特别地，我们在这里使用的类非常重要。如果改写它们（加上表格元素所用的规则），就能够在不改变 HTML 或者 JavaScript 文件的情况下编辑组件布局。但是相关的部分在 JavaScript 中进行（ch6_28_ready.js）。

程序清单 6.67 组件的功能用 JavaScript 实现

```
/**
* Global variables
*/

var currentDay; // Day
var currentMonth; // Month
var currentYear; // Year
var startMonthDay // Start day of the current month

/**
* Initialization function
* @param {Object} - a date object
*/
function init(tag){
 var today;
 // If no date is passed,
 // use current system date
 if (day == null)
   today = new Date();
```

```
20  else
21    today = day;
22  currentDay = today.getDate();
23  currentMonth = today.getMonth();
24  currentYear = today.getFullYear();
25  var startMonth =
26      new Date(currentYear, currentMonth, 1);
27  startMonthDay = startMonth.getDay();
28  if (startMonthDay == 0)
29    startMonthDay = 7;  // Sunday
30 }
31
32 /**
33 * Generate 6 rows with 7 columns each
34 * Add class to last row
35 */
36 function row(){
37  var z = $("<tr></tr>");
38  for (var i = 0; i < 7; i++) {
39    if (i == 6)
40      z.append($("<td></td>").addClass("sunday"));
41    else
42      z.append($("<td></td>"));
43  }
44  return z;
45 }
46
47 /**
48 * Generate adapted days in table
49 * @param {Object} startMonthDay
50 */
51 function days(startMonthDay){
52  for (var i = 0; i < 42; i++) {
53    $("td").eq(i).append(
54       new Date(currentYear, currentMonth,
55         i - startMonthDay + 2).getDate());
56  }
57 }
58
59 /**
60 * The ready method
61 */
62 $(function(){
63  // Either pass current day date
64  // or any date
65  init(new Date());
66  // Table header rows
67  $("table").append(
68     $("<tr><th>Mon</th><th>Tue</th><th>Wed</th>" +
69       "<th>Thu</th><th>Fri</th>" +
```

```
70        "<th>Sat</th><th>Sun</th></tr>"));
71  // Append 6 rows with 7 columns each
72  for (var j = 0; j < 6; j++) {
73    $("table").append(row());
74  }
75  // Fill cells depending on start day
76  days(startMonthDay);
77  // Format TH cells
78  $("th").addClass("headercells");
79  // Format odd line numbers
80  $("tr:odd").addClass("odd");
81  // All TD cells before start of month transparent
82  $("td").slice(0, startMonthDay - 1).css(
83      "opacity", "0.3");
84  // All TD cells after end of month transparent
85  for (i = 28; i < 42; i++) {
86    if ($("td").eq(i).html() < 15)
87      $("td").eq(i).css("opacity", "0.3");
88  }
89  // Format the current day
90  $("td").eq(currentDay + startMonthDay - 2).
91      addClass("currentDay");
92  // Output month and year
93  $("div:first").append(
94      (currentMonth + 1) + "/" + currentYear);
95 });
```

本书是一本jQuery书籍而不是JavaScript书籍。所以，我对第1～30行中的纯JavaScript的作用仅作简短的描述。如图6-59所示。

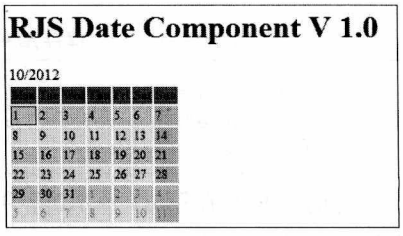

图6-59 高亮显示不同结构的日期组件

日期组件的基本问题是必须确定当前日期，这决定了显示的月份。实际上，只有月份是关键，但是我们还打算高亮显示当前日期。

我们使用new date()确定当前日期。如果不想使用当前系统日期，也可以将任何其他日期传递给构造函数。[⊖]关键的部分在第65行。如果为函数调用指定参数，例如new Date(2012,12,2)，就明确地选择了2012年12月2日。[⊖]在这一行中，调用第14～30行的一个初始化函数，确定日期信息的各个部分，将它们当作全局变量使用。有了这些信息，我们能够确定本月的当前日期，在以后格式化它。我们还可以确定本月的第一天是星期几（当然，它并不总是周一）。这样，我们必须在表格第一行合适的列中写入数字1。而且，如果本月不是从周一开始的，这行必须用前一个月的最后几天填充。这些日期将用特别的方式高亮显示。

同样，如果本月的最后一天不是星期天，最后一行可能需要用下一个月的日期填充。

⊖ 这是纯JavaScript。

⊖ 记住，月份的索引是从0开始的。

对于月份，还必须考虑天数的不同。想象一下平年且二月一日是星期一的情况，对于该月我们的表格中只需要4行。

然而，如果我们显示的月份有31天，且第一天是周六或者周日，当前月将占据6行。如果不打算在日期组件中使用动态行数，[⊖]可以使用6行来显示日期，并用特殊的高亮显示来表示当月的日期。这样做的好处是组件的大小总是保持相同。

在第36～45行的row()函数中，我们创建了一个7列的表格行，目前仍然为空。tr和td元素的这一结构以字符串形式返回。我们在tr元素的循环中使用append()附加单独的单元格。然后，我们通过addClass()方法为最后一列添加一个class特性。

程序清单6.68 用于当月日期的空白行

```
36 function row(){
37  var z = $("<tr></tr>");
38  for (var i = 0; i < 7; i++) {
39    if (i == 6)
40      z.append($("<td></td>").addClass("sunday"));
41    else
42      z.append($("<td></td>"));
43  }
44  return z;
45 }
```

这个函数在ready方法的循环中被调用6次（第73行），返回值通过append()附加到我们已经以HTML标签形式写入网页的表格元素中。

程序清单6.69 空白单元格被附加到表格

```
72  for (var j = 0; j < 6; j++) {
73    $("table").append(row());
74  }
```

之前，我们在第67～70行中通过th元素附加了表格的列标题。这种技巧现在被用来创建空白的表格单元的内容。

程序清单6.70 日期被写入表格

```
51 function days(startMonthDay){
52  for (var i = 0; i < 42; i++) {
53    $("td").eq(i).append(
54       new Date(currentYear, currentMonth,
55         i - startMonthDay + 2).getDate());
56  }
57 }
```

td类型的单元格共有42个。我们使用eq()过滤器函数在循环中处理这些单元格。作为表格填充的起始值，我们使用了目标月份第一天之前的周一。我们能够计算它的日期，因为

⊖ 这种做法不太常见。

我们已经确定了本月第一天的周日。然后，我们可以倒数日期，将其指定为构造函数的第三个参数。

例如，如果本月第一天是星期四，我们必须倒数，以求得前一个周一的日期。这取决于它所在的月份。在这种情况下，我们取本月的第一天，然后用周一和周四的差值倒数，求得前一个月中的日期。表达式中的 2 是校正因子，是不同位置选择的算法中所必需的。⊖

提示

我们明确地使用了 Date 构造函数的行为以防止溢出。例如，可能指定 new Date(2011,2,-8)（这在平年会得出 2 月 20 日）或者 new Date(2011,15,66)。天数简单地从最后一天倒数。

现在，我们更仔细地检查格式，因为选择器在这里使用得很有效。

程序清单 6.71 使用智能选择器设置不同的格式

```
77  // Format TH cells
78  $("th").addClass("headercells");
79  // Format odd line numbers
80  $("tr:odd").addClass("odd");
81  // All TD cells before start of month transparent
82  $("td").slice(0, startMonthDay - 1).css(
83     "opacity", "0.3");
84  // All TD cells after end of month transparent
85  for (i = 28; i < 42; i++) {
86    if ($("td").eq(i).html() < 15)
87      $("td").eq(i).css("opacity", "0.3");
88  }
89  // Format the current day
90  $("td").eq(currentDay + startMonthDay - 2).
91     addClass("currentDay");
```

在第 78 行中，格式化了列标题。这很简单，因为我们只要使用类型为 th 的元素。在第 80 行，可以看到实现斑马纹图案有多么简单。在这两种情况中，我们都分配了一个特殊类（对于当前日期也是如此，见第 90 行和第 91 行）。所以，可以简单地改变类定义，甚至用另一个 CSS 文件改变布局，而不需要修改 HTML 或者 JavaScript 代码。如图 6-60 所示。

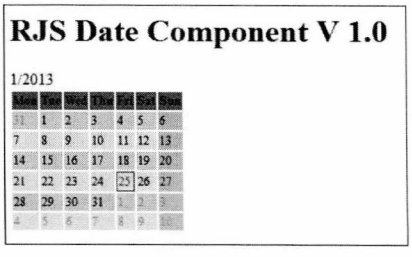

图 6-60 不同的日期和不同的“模板”

注意，我们在源代码中为前一个月和下一个月中的日期设置了透明度。这种做法有意地避免了适应性（为了演示目的）。需要注意（特别是在与透明度的联系中）的是，Internet Explorer 不支持默认属性 opacity。如果想要在 CSS 文

⊖ 当然，我们也可以重构这一算法，但是它的工作原理是唯一的重点。

件中设置这个属性，必须为 Internet Explorer 额外使用 filter() 函数。但是，如果使用 jQuery 提供的 css() 方法，就可以使用 opacity，因为框架已经在后台弥补了这些问题。在第 93 行和第 9 行中，我们输出月份和年份。

6.16 小结

在篇幅巨大的本章中，大家看到了如何用 jQuery 遍历网页并对其进行操纵，以及如何在网页中插入、编辑或者删除节点。我们还遭遇了寻找特殊节点及其内容的进一步选项。特别应该记住 html()、text()、append()、prepend()、appendTo()、prependTo() 以及 after()、before()、insertAfter()、insertBefore()。用 wrap() 进行包装也很重要。此外，大家现在还知道了替换内容和节点以及删除和克隆、循环读取对象和对象集合的方法。最后，大家还了解了父元素和子元素、兄弟元素之间的关系，以及 jQuery 对这些关系的处理方法。

第 7 章

在 jQuery 中使用样式表格式化

选择器和过滤器加上对应的过滤器方法，是根据样式表动态创建和修改网页的基础。在 Web 上，层叠样式表（Cascading Style Sheet，CSS）是样式表的标准。纯粹对网页而言，其他样式表语言确实起不到任何重要的作用。现在，使用超文本标记语言（HTML）命令从视觉上增强网页的时代已经一去不复返了。使用 <font> 之类的标记，或者通过 HTML 属性指定颜色或者对齐方式现在只适合于不需要满足任何特殊期望的应急站点。除了这种方法在 10 多年就已经得到应用之外，其他的都不值一提。

然而，Web 设计人员直到最近还在继续使用 HTML 标签进行网页的视觉设计。在某些情况下，这种坚持有其优点，因为尽管 CSS 的标准化已经进行了许多年，在某些浏览器上，一些特效即使在今天仍有某些问题。可以断定 CSS 1 得到所有浏览器的支持，但是 CSS2 中的某些规则仍然不能在所有浏览器中都正常工作——即使在 W3C 对其标准化之后 12 年仍然如此。更不要说 CSS3 和某些出现多年的扩展属性了。只要想一想，CSS 属性 opacity 仍然没有得到 Internet Explorer 的支持，在这个浏览器中需要一个不遵循标准的标记。[⊖]还有，Internet Explorer 坚持拒绝支持块元素的圆角。[⊖]

在常规 jQuery 编程中，不可能经常需要这个功能。

在 jQuery 下使用样式表格式化并不意味着不应该按照惯例集成 style 特性、内部样式容器，或者像通常建议的那样，在使用 jQuery 时在 HTML 网站中引用外部 CSS 文件。相反，几乎总是要这么做。但是，本章的重点在于从 JavaScript 中动态地访问 CSS 格式（换句话说，结合某些编程逻辑）。

用纯粹的 DHTML，可以轻松地通过样式对象访问 CSS 属性，但是如果选择这种方法，就必须应对前面描述的问题。加之，我们必须面对 style 对象的属性与 CSS 属性标记法在大小写字母拼写和破折号使用上的差异。这些都是小的困难，但是它们会积累起来。

⊖ Microsoft 使用专利方法 filter()，但是 jQuery 在后台自动使用它。

⊖ 甚至 jQuery 也无法成功地为 Internet Explorer 提供圆角支持。更确切地说，jQuery 一致性地使用 CSS3，任何选择不实施这一标准的人都必须承担后果。

通过使用 jQuery，可以规范化属性，然后它们（包括 opacity）可以在所有支持的浏览器中以相同的形式使用。我相信，这是 jQuery 的主要好处。采用 jQuery 中的 CSS 方法可以轻松地访问样式表，并将这种方法整合到框架的整体思想中（包括链接选项）。

注意

当然，jQuery 方法直接基于经典的 CSS 或者 style 对象应用。

7.1 css() 方法

jQuery 中访问样式表的最基本、最通用也可能是最重要的方法是 css()。[⊖]

7.1.1 获取样式属性

通过 css() 方法，可以轻松地获得一个 CSS 属性的值。只要简单地指定一个属性，作为字符串参数即可。例如，可以用如下的方法获得某个属性的特定值。

程序清单 7.1 获得背景颜色

```
var color = $(this).css("background-color");
```

对于 CSS 属性的名称，也可以使用 style 对象提供的标记方法。jQuery 框架很好地接受两种标记法；例如，也可以这样写：backgroundColor。（注意，几个单词必须写在一起，第二个单词用一个大写字母开头；这是必需的。）

提示

这适用于在 jQuery 中指定样式属性的任何位置。

程序清单 7.2 样式属性的另一种命名

```
var color = $(this).css("backgroundColor");
```

警告

有些 CSS 属性（如 border）常用的简写标记没有得到 css() 和其他类似方法的支持！这些方法的支持程度在 jQuery 框架的不同版本中各有不同。例如，旧版本不理解 background，必须写下 CSS 属性的全名。例如，可以使用 background-color 而不是 background。在 1.5 版本中，简写形式 background 是可行的。到 1.7.2 版本，简写的 CSS 属性不再受到支持。从安全的角度出发，最好是始终使用完整的版本而不是简写版本。

⊖ 我们已经使用这个方法好几次了。

7.1.2 设置属性

当然，也可以通过 css() 方法设置 CSS 属性值。CSS 属性通过向 css() 传递 name、value 参数，或者传递一个具有多个 name:value 对的对象字面标记设置。通常，属性的名称和值包含在引号中，如程序清单 7.3 所示。

程序清单 7.3 设置一个属性值

```
$(this).css("color","red");
```

这种标记法用于设置一个属性。另一种选择是只指定一个参数。这个参数不是一个字符串，而是一个选项对象，可以用包含在尖括号中，以逗号分隔的键 – 值配对组成属性列表，利用这种标记法，还可以一次设置多个属性。

程序清单 7.4 用一个属性列表设置多个属性

```
$(this).css({
    'background-color' : 'blue', 'color' : 'white',
    'opacity' : '0.7'
  });
```

警告

利用属性名称，在某些情况下可以不采用字符串形式，名称不用包含在引号中。这通常是为 style 对象的属性使用对象字面标记法的情况（例如，backgroundColor）。但是，我建议不要使用这种方式，而推荐使用引号中的统一标记形式。这种形式更一致、更易于管理，也更为稳妥。对于有两个参数的变种，总是必须以字符串的形式写下属性名，并将其放在引号中，即使使用 style 对象的属性标记也是如此。

css() 方法非常简单，可以应用到任何可以用 CSS 格式化的情况下，不需要进一步的解释。但是为了完整性，我们在程序清单 7.5 中提供了一个完整的例子（ch7_1.html）。

程序清单 7.5 基本文件

```
...
06    <script type="text/javascript"
07        src="lib/jquery-1.8.2.min.js"></script>
08    <script type="text/javascript"
09        src="lib/ch7_1_ready.js"></script>
10  </head>
11  <body>
12    <div>A child of five</div>
13    <div>would understand this.</div>
14    <div>Fetch me a child of five.</div><hr/>
15    <button>Set text color</button>
16    <button>Set text color and
17       background color</button>
```

```
18    <button>Get text color</button>
19    <hr/><div id="output"></div>
20  </body>
21 </html>
```

下面是 JavaScript 文件 ch7_1_ready.js。

程序清单 7.6 获取和设置 CSS 属性

```
01 $(function(){
02  $("button:first").click(function(){
03    $('div:lt(3)').css("color", "red");
04  });
05  $("button:eq(1)").click(function(){
06    $('div:lt(3)').css({
07      "color": "blue",
08      "background-color": "lightgray"
09    });
10  });
11  $("button:last").click(function(){
12    $("#output").text($("div:first").css("color"));
13  });
14 });
```

在这个例子中，我们设置了网页中前三个 div 元素的字体颜色。可以在第 3 行中看到。我们使用具有两个参数的变种。如图 7-1 所示。

在第 6 ～ 9 行中，可以看到具有选项形式参数的变种。字体颜色被设置为蓝色，背景被设置为浅灰色。如图 7-2 所示。

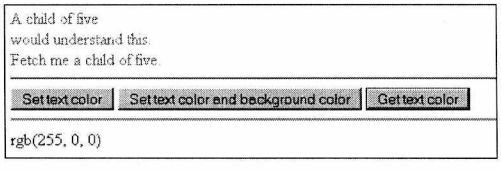

图 7-1 将字体颜色设置为红色并获取

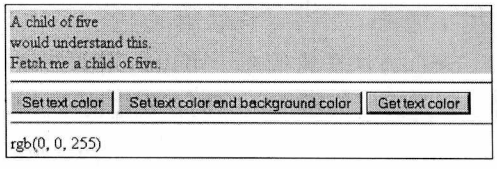

图 7-2 将字体颜色设置为蓝色并获取

最后，在第 12 行中可以看到我们获取了 color 属性。

链接的 CSS 格式也很容易在 jQuery 框架中实现。可以使用句点标记法链接多个 css() 方法。但是这不太必要，因为几乎总是可以用以选项为参数的方法调用完成。

7.2 修改元素的类

jQuery 框架提供一些有趣的方法访问 CSS 类。这些方法使动态处理变得更加容易。本质上，这些方法操纵元素 class 特性的值。

7.2.1 添加类：addClass()

顾名思义，调用 addClass() 方法为前面指定的元素指定一个或者多个样式表类。这一操作动态发生，但是网页不一定要重新加载或者进行任何方式的更新。当然，CSS 类一定可以某种形式在网页中使用。

提示

通过这个方法和访问 CSS 类的其他方法，可以确保布局和功能的分离。布局在 CSS 文件中通过静态指定类，以经典的方式准备，JavaScript 中只进行具体的赋值（逻辑）。

直接指定类名

被分配的类可以直接指定为参数。

程序清单 7.7 添加一个 CSS 类

```
$("#e").addClass("myClass1");
```

字符串中的多个类用空格分隔。

程序清单 7.8 添加 3 个 CSS 类

```
$("#e").addClass("myClass1 myClass2 myClass3");
```

使用一个回调函数

到 jQuery1.4 时，还可以编写一个回调函数，作为操纵类的所有方法的参数。这个函数应该（或者说必须[⊖]）返回一个或者多个类名，以空格分隔，这些类名被分配给前面制定的 jQuery 对象中的元素。

程序清单 7.9 指定一个返回一个或者多个类的回调函数

```
$("div").addClass(function(){});
```

回调函数有两个可选的标准参数。第一个可选参数代表当前访问元素在前面指定的一组元素中的索引位置。当用此方法处理不同元素时，可以根据索引决定对每个特定元素的操作（例如，返回一个单独的类，就像我们在下一个例子中所要做的那样）。

第二个可选参数代表 class 属性在相关处理阶段的当前值——换句话说，在赋值之前的值。

程序清单 7.10 指定一个回调函数，两个参数代表索引和旧的类

```
$("div").addClass(function(index, Class){});
```

⊖ 如果这个函数要发挥作用的话。

注意

我们用下面的例子详细地分析 addClass() 方法的工作原理。我们使用 Firebug 反复分析情况——特别是回调函数的行为，因为这相当有趣，从根本上说也很重要。以后，可以将这些解释应用到其他操纵 class 属性的方法，以及许多其他元素的操纵方法中。所以，对其他方法的解释就没有那么详细了。请不要为对这个相当简单的方法做详细解释感到惊奇。

我们转向完整的例子。我们用它测试 addClass() 的不同变种（ch7_2.html）。

程序清单 7.11 基本文件

```
...
06    <link rel="stylesheet" type="text/css"
07       href="lib/ch7_2.css" />
08    <script type="text/javascript"
09        src="lib/jquery-1.8.2.min.js"></script>
10    <script type="text/javascript"
11        src="lib/ch7_2_ready.js"></script>
12  </head>
13  <body>
14    <div>A child of five</div>
15    <div>would understand this.</div>
16    <div>Fetch me a child of five.</div><hr/>
17    <button>Assign class 1</button>
18    <button>Assign class 2 and 3</button>
19    <button>Assign class 4</button>
20    <button>Assign classes by index</button>
21  </body>
22 </html>
```

这个网页包含 3 个 div 区域，我们将动态地为它们指定类。我们希望测试 4 种操作，所以有 4 个按钮来触发这些操作。如图 7-3 所示。

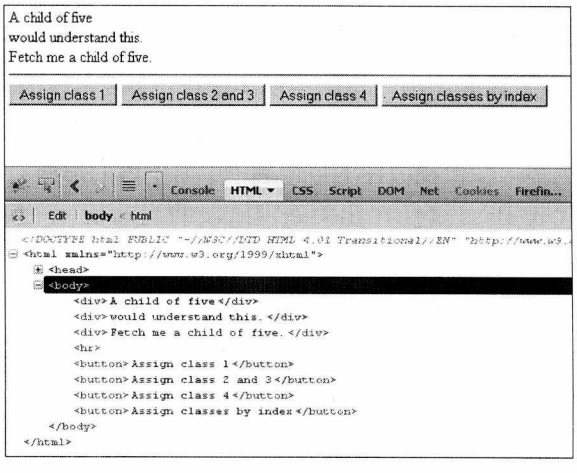

图 7-3 应用样式表之前的网页

如前所述，因为我们打算使用CSS类，所以必须使其在网页中可用。最好的方法当然是通过外部CSS文件。在第6行和第7行中，可以看到对外部CSS文件ch7_2.css的引用，该文件如程序清单7.12所示。

程序清单7.12 CSS文件

```
01 .c1 {
02  background: red;
03 }
04 .c2 {
05  color: yellow;
06 }
07 .c3 {
08  font-size: x-large;
09 }
10 .c4 {
11  word-spacing: 2em
12 }
```

该文件不是很激动人心。在CSS文件中，你可能只发现了4个简单的类，它们将以不同的方式应用到div元素，设置颜色、字体大小和单词间距。

注意

我们已经在jQuery基础知识的章节中提到过，但是在这里还是要注意，对CSS文件的引用在网页中必须先于对脚本的引用。

下面是JavaScript文件ch7_2_ready.js，在该文件中，我们对div元素进行特定的赋值。

程序清单7.13 指定CSS类

```
01 $(function(){
02  $("button:first").click(function(){
03    $("div").addClass("c1");
04  });
05  $("button:eq(1)").click(function(){
06    $("div").addClass("c2 c3");
07  });
08  $("button:eq(2)").click(function(){
09    $("div").addClass(function(){
10      return "c4";
11    });
12  });
13  $("button:last").click(function(){
14    $("div").addClass(function(index, Class){
15      alert("Value of index: " + index +
16         ", Value of class: " + Class);
17      return "c" + (index + 1);
18    });
19  });
20 });
```

单击第一个按钮，我们简单地以字符串方式指定类名，将类 c1 分配给所有 div 元素；参见第 3 行。可以在网页上看到结果，但是在 Firebug 中更好。如图 7-4 所示。

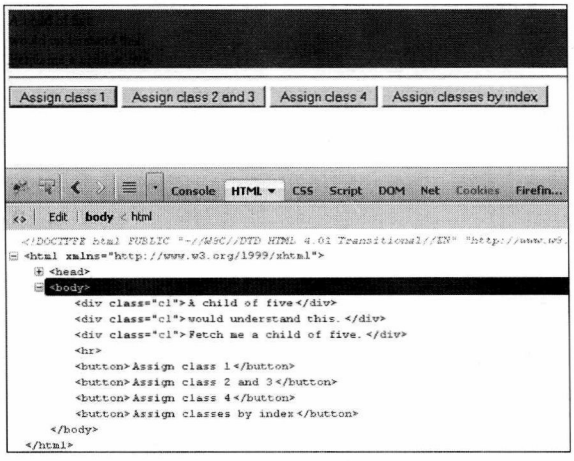

图 7-4 在 Firebug 中，可以看到 class 属性的值被设置

单击第二个按钮，我们在第 6 行中一次性指定多个类。字符串中的类简单地用空格分隔。如果在此之前已经指定了另一个类，这些类将累加使用，就像平常的 CSS 一样。[⊖] 如图 7-5 所示。

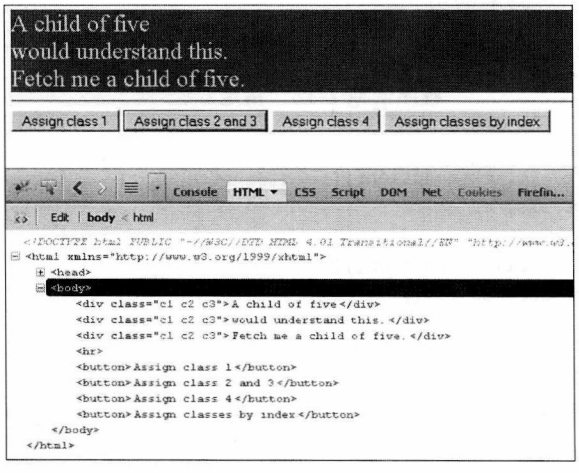

图 7-5 元素被指定多个类（以空格分隔）

对于第 3 个按钮，你看到的情况是在第 10 行有一个回调函数，简单地返回一个类。当然，它也可能返回多个类。这些类在字符串中必须以空格分隔。如图 7-6 所示。

⊖ 会产生和 CSS 下一样的后果。来自不同类的属性会组合或者出现冲突。在那种情况下，class 特性中最右边的类生效（但是这将把我们带入 CSS，而不再是 jQuery）。

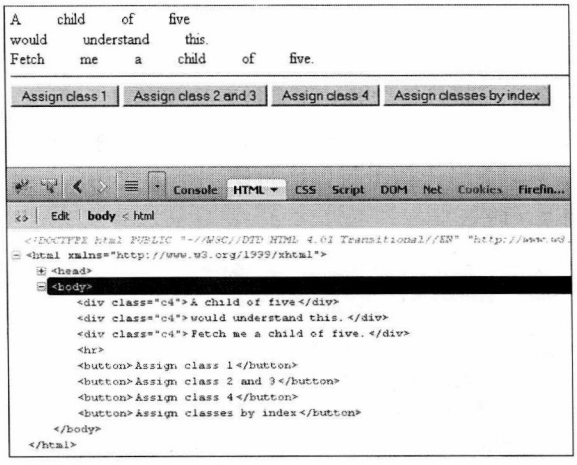

图 7-6 类已经通过一个函数的返回值指定

在实践中，使用回调函数所需的努力通常只有在实现了合理的逻辑之后才有意义。可以根据各种情况构造这一逻辑。但是有两种情况和 CSS 类的操纵特别相关：

- 应该使用一组被操纵元素中的元素索引。
- 以前指定的类起到某种作用。

addClass() 方法[⊖]的回调函数中有两个可选的默认参数能够满足以上需求。换句话说，在 jQuery 框架的类操纵方法中，回调函数总是有两部分可用的信息。可以在第 4 个按钮的响应中看到这些信息是如何使用的。

在第 14 ～ 19 行中，可以看到我们打算更详细分析的特殊应用。先让我们来仔细地看看它。

程序清单 7.14 使用带有参数的回调函数

```
14    $("div").addClass(function(index, Class){
15      alert("Value of index: " + index +
16         ", Value of class: " + Class);
17      return "c" + (index + 1);
18    });
```

第一个参数 index 指定选中集合中当前被处理的前导元素。框架单独地为每个项目处理该方法。例如，可以据此为每个特定元素指定一个特殊的类。通过我们用数字方式索引 CSS 文件中的类名，并冠以同一个字母的前缀，我们可以用一个字母和 index 的值组合出返回的类。可以在第 17 行中看到这一点。

第二个参数代表目前已经指定给元素的类。在我们的例子中，这些信息没有进一步处理为类的选择[⊖]，但是，我们在指定类之前，将其与 index 的值一起在一个 alert() 窗口中输

⊖ 以及后面介绍的操纵类的方法。

⊖ 就像其他可以想象的逻辑一样，这很容易实现。

出。[⊖]为了对 addClass() 以及 jQuery 框架中许多其他方法的工作原理进行分析和阐述，在对话窗口关闭之前中断脚本的处理有额外的好处。这样，我们可以更轻松地按照单独的步骤，了解它们的工作方式。你会看到，每个 div 元素被单独格式化。

在单击第 4 个按钮之后，首先会看到 alert() 窗口。第一遍时的索引值当然是 0。如果正在访问的元素已经指定了一个类，该类可以通过第二个参数访问，并将其显示出来。注意，新类还没有指定。如图 7-7 所示。

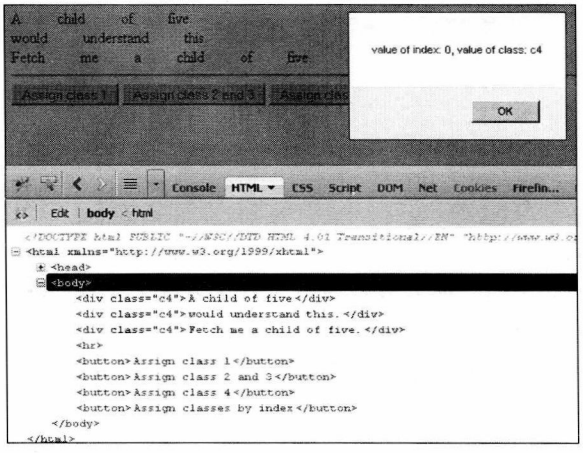

图 7-7 单击第 4 个按钮之后，还要注意 Firebug

如果确认 alert() 消息，该类被指定，我们用 return 返回类名。由于我们用标志“ c”和 index+1 的值生成类名，所以第一个 div 元素在所有现有类之外还被指定了 c1 类。对于其他 div 元素，格式保持不变。如图 7-8 所示。

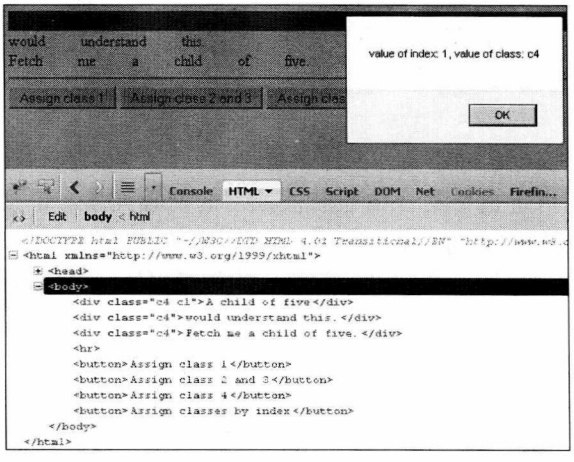

图 7-8 确认第一个 alert() 之后，指定 c1 类

⊖ 这至少说明这一信息可用，尽管显示它非常简单，但也是处理信息的一种方法。

在按照这种模式循环处理全部三个 div 元素之后，将 c1 类指定给第一个 div 元素，c2 指定给第二个，c3 指定给第三个。如图 7-9 所示。

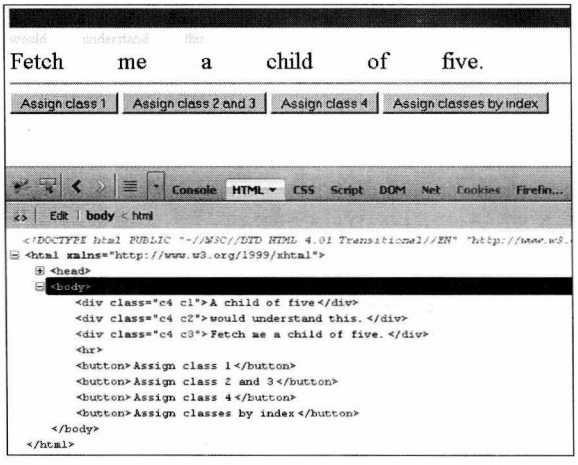

图 7-9 每个 div 元素被单独格式化

警告

虽然这一点应该已经很清楚，但是我还是要再次说明回调函数的工作方式。不要被这种回调函数的 return 所迷惑。该语句是一条跳转语句，表示函数或者方法的结束。看上去似乎是方法返回一个元素的类，然后继续运行，为下一个元素指定一个类。但是情况并非如此，尽管效果可以这样描述。return 语句已经结束了回调函数，和其他函数和方法一样。但是如果其他元素也注册了这个回调函数，它就会在下一个元素中被框架直接再次调用。所以，在我们的例子中，这个方法被调用三次，这些调用由框架管理。

如果现在在回调函数中再次处理 class 参数，就会看到已经有多个可用的类。这是一个简单的字符串，其中的类以空格分隔。如图 7-10 所示。

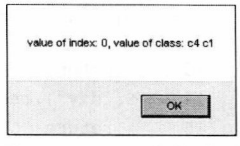

图 7-10 class 参数也可以访问多个类

7.2.2 删除类：removeClass()

类的删除和类的指定方法一样。我们使用 removeClass() 方法并以字符串方式传递想要删除的类。该方法的工作原理和 addClass() 完全一样，当然，也可以同时删除多个类。

程序清单 7.15 删除一个 CSS 类

```
$("#e").removeClass("myClass1");
```

到 jQuery 1.4 时，还可以使用一个回调函数作为参数。在下一个例子中，我们或多或少地使用演示 addClass() 时的同一个基本网页。CSS 文件保持不变，只是使用新名称 ch7_3.

css（ch7_3.html）。

程序清单 7.16 基本文件

```
...
10    <script type="text/javascript"
11        src="lib/ch7_3_ready.js"></script>
12  </head>
13  <body>
14    <div>A child of five</div>
15    <div>would understand this.</div>
16    <div>Fetch me a child of five.</div><hr/>
17    <button>Assign class 1</button>
18    <button>Assign class 2 and 3</button>
19    <button>Remove class 1</button>
20    <button>Remove class 2 and 3</button>
21  </body>
22 </html>
```

下面是 JavaScript 文件 ch7_3_ready.js。

程序清单 7.17 指定 CSS 类

```
01 $(function(){
02   $("button:first").click(function(){
03     $("div").addClass("c1");
04   });
05   $("button:eq(1)").click(function(){
06     $("div").addClass(function(){
07       return "c2 c3";
08     });
09   });
10   $("button:eq(2)").click(function(){
11     $("div").removeClass("c1");
12   });
13   $("button:last").click(function(){
14     $("div").removeClass(function(){
15       return "c2 c3";
16     });
17   });
18 });
```

单击前两个按钮，我们再次指定类。利用完全相同的语法，但是使用 removeClass() 方法，可以再次删除该类。

7.2.3 切换类：toggleClass()

使用 toggleClass() 方法，我们可以根据状态删除或者添加一个 CSS 类。如果作为参数指定的类已经指定，则将其删除。如果还没有指定，该类将被指定。和前面讨论过的类操纵方法一样，可以字符串方式指定一个或者多个类，或者一个合适的回调函数。

提示

在 jQuery 1.3 时，该方法有第二个可选的参数。这是一个布尔参数，指定该类应该添加还是删除。我的意见是，这种用法与方法名称的含义相抵触，毕竟，已经有了 addClass() 和 removeClass() 这些替代方法了。

程序清单 7.18 基本文件（ch7_4.html）

```
...
10    <script type="text/javascript"
11        src="lib/ch7_4_ready.js"></script>
12  </head>
13  <body>
14    <div class="c1">A child of five</div>
15    <div class="c3">would understand this.</div>
16    <div>Fetch me a child of five.</div><hr/>
17    <button>Toggle class DIV 1</button>
18    <button>Toggle class DIV 2</button>
19  </body>
20 </html>
```

在网页中，我们还是有 3 个 div。第一个 div 元素已经在 HTML 中指定了类 c1，第二个 div 区域已经指定了 c2 类。第 3 个 div 区域没有通过 CSS 类明确地格式化。使用不同的条件，我们现在来仔细地看看切换的工作原理。如图 7-11 所示。

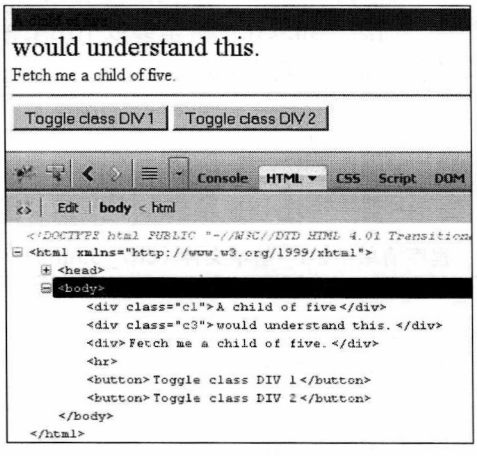

图 7-11 加载时的原始结构

程序清单 7.19 指定和删除 CSS 类

```
01 $(function(){
02   $("button:first").click(function(){
03     $("div").toggleClass("c2");
04   });
```

```
05  $("button:last").click(function(){
06    $("div").toggleClass(function(){
07           return "c4";
08      });
09  });
10 });
```

第一次切换时，我们直接指定带有新类名的字符串（第 3 行），第二次我们使用了带有回调函数的变种（第 6 ～ 8 行）。

在单击第一个按钮之后，可以看到所有 div 元素都被指定了 c2 类，但是没有删除任何具有另一个名称的类。相反，该类被当作一个参数，简单地添加到现有的类中。如图 7-12 所示。

如果再次单击按钮，c2 类被删除，返回到原来的情况。当单击第二个按钮然后再次单击，相应地也会发生同样的情况。

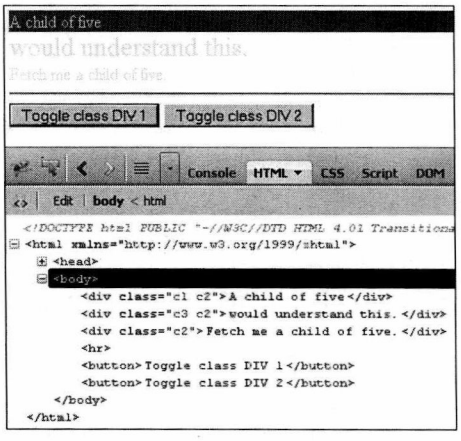

图 7-12 所有 div 元素都被指定了 c2 类

7.2.4 测试一个类：hasClass()

使用布尔方法 hasClass()，可以测试一个元素是否指定了某个类。如果测试应用到多个元素，只要有一个元素满足条件就足够了。

只需将类作为字符串参数。当然，如果在字符串参数中用空格分隔类名，也可以测试多个类。如果条件匹配，返回值为 true，否则为 false。

注意

对于 hasClass()，目前不能指定一个回调函数作为参数。

程序清单 7.20 基本文件（ch7_5.html）

```
...
10    <script type="text/javascript"
11        src="lib/ch7_4_ready.js"></script>
12  </head>
13  <body>
14    <div class="c1">A child of five</div>
15    <div>would understand this.</div>
16    <div>Fetch me a child of five.</div><hr/>
17    <button>Test 1</button>
18    <button>Test 2</button>
19  </body>
20 </html>
```

第一个 div 元素有我们想要测试的 c1 类。下面是具有两个测试函数的 JavaScript 文件

(ch7_5_ready.js)。

程序清单 7.21 测试一个类

```
01 $(function(){
02  $("button:first").click(function(){
03    alert($("div").hasClass("c1"));
04  });
05  $("button:last").click(function(){
06     alert($("div:eq(1)").hasClass("c1"));
07  });
08 });
```

在第一个测试中，我们检查所有 div 元素。因为第一个 div 元素已经指定了 c1 类，测试返回 true。如图 7-13 所示。

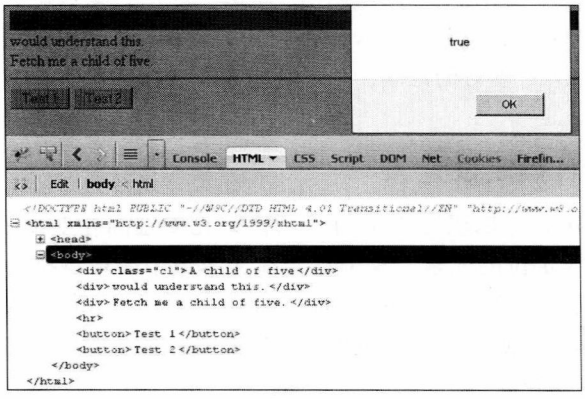

图 7-13 第一个 div 元素通过测试

在第二种情况中，我们只测试第二个 div 元素，它没有指定 c1 类，所以，测试返回 false。如图 7-14 所示。

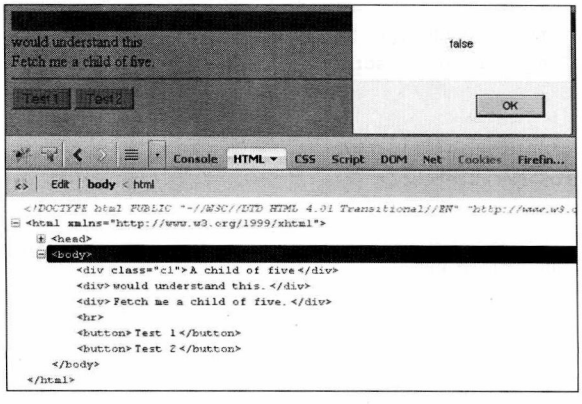

图 7-14 该类还没有在选中的元素集中指定

7.3 定位方法

设计和动态修改网页最大的问题之一是定位元素。通过 CSS，可以用 position 属性结合左上位置，很好地控制网页中元素的定位。[⊖]

注意

右侧和底部位置规格还没有得到各种浏览器的支持，最好避免使用它们。

定位数据基于元素的偏移量（Offset）。坐标系的原点在偏移元素的左上角。jQuery 定位方法用这些属性和坐标作为参考点。

注意

尽管我们在本书中假定读者有基本的 CSS 知识，定位、文本流和偏移的重要性方面的解释将比纯粹应用这一领域大部分 jQuery 方法所需的更加详细。从我在培训课程中的经验来看，尽管 CSS 中还有另外一些好的经验，但是这些方面的知识在许多情况下是相当简单的。

7.3.1 用 position() 确定位置

通过 position() 方法，可以确定元素与其偏移父元素之间相对的 top 和 left 值，这两个值都可以用于通过 CSS 定位的元素，和浏览器在文本流中简单编排的元素。这意味着，以像素表示的定位数据总是被看作和包含元素的偏移父元素左上角有关联性。在许多情况下，偏移父元素就是网页本身（换句话说，DOM 对象 document），但是也可能是 div 元素或者子元素所在的另一个容器对象。

但是（你可能已经知道），这个父元素首先必须明确地转换为一个偏移父元素；[⊖]例如，通过对其绝对定位来实现（如下例所示，ch7_6.html）。

程序清单 7.22 带有 4 幅图像的基本文件

```
...
10      <script type="text/javascript"
11       src="lib/ch7_6_ready.js"></script>
12  </head>
13  <body>
14    <h1>Position of Elements</h1><hr/>
15    <table id="info"></table><hr/>
16    <img src="images/pic1.png" alt="A scanner" />
17    <div id="p1">
18      <img src="images/pic2.png"
19        alt="A portable computer" />
```

⊖ 注意，通过 CSS 绝对定位的元素会被排除在“常规”文本流或网站的元素流之外。例如，如果在 HTML 中有 3 个标题，第 2 个标题通过 CSS 绝对定位，网站的常规文本流中将第 3 个标题紧跟在第 1 个标题之后。但大家应该已经熟悉了这一点。

⊖ 否则，网页仍然是参考点。这是纯粹的 CSS。

```
20      <img id="p2" src="images/pic3.png" alt="Mice" />
21    </div>
22    <img id="p3" src="images/pic4.png"
23      alt="Cable mouse" />
24  </body>
25 </html>
```

在这个网页中，可以看到 4 幅图像。其中两幅在一个 div 区域中，该区域的 ID 为 p1。还应该注意到，第 3 幅和第 4 幅图像具有 ID。

下面是 JavaScript 文件 ch7_6_ready.js，我们从中可以找出 4 幅图像的定位数据，在网页的信息区域（一个表格）中输出。

程序清单 7.23　获取定位数据

```
01 $(function(){
02  var pos = $("img:first").position();
03  $("#info").html(
04    "<tr><td>Image 1</td><td>left:</td><td>" +
05      pos.left +
06      "</td><td>top:</td><td>" +
07      pos.top + "</td></tr>");
08  pos = $("img:eq(1)").position();
09  $("#info").append(
10    "<tr><td>Image 2</td><td>left:</td><td>" +
11      pos.left +
12      "</td><td>top:</td><td>" +
13      pos.top + "</td></tr>");
14  pos = $("img:eq(2)").position();
15  $("#info").append(
16    "<tr><td>Image 3</td><td>left:</td><td>" +
17      pos.left +
18      "</td><td>top:</td><td>" +
19      pos.top + "</td></tr>");
20  pos = $("img:eq(3)").position();
21  $("#info").append(
22    "<tr><td>Image 4</td><td>left:</td><td>" +
23      pos.left +
24      "</td><td>top:</td><td>" +
25      pos.top + "</td></tr>");
26 });
```

在这个例子中，JavaScript 功能不是特别复杂，我们简单地在加载网站时获取 4 幅图像的定位数据，将其保存在一个变量中，并输出每个图像的 left 和 top 属性值。

但是在我们的情况下，有趣的是在我们定位带有图像的 div 区域时值的改变，以及我们没有这么做的时候发生的情况。在我们的例子中，这些改变通过 CSS 进行，让我们来看看这个 CSS 文件 ch7_6_1.css。

程序清单 7.24 CSS 文件

```
01 img {
02  height: 150px;
03 }
04 #info{
05  width: 440px;
06  background: yellow;
07 }
08 #p1 {
09  border: solid 1pt;
10 }
11 #p2 {
12  position:absolute;
13  top:200px;
14 }
15 #p3 {
16  position:absolute;
17  top:200px;
18 }
```

图像的高度都被设置为 150 个像素，信息区域的宽度被设置为 440 个像素。ID p1 是包含第二幅和第三幅图像的 div 区域的格式。这在该区域周围增加了一个框，可以看到它占据的网页空间。这些设置的效果都不是非常壮观。如图 7-15 所示。

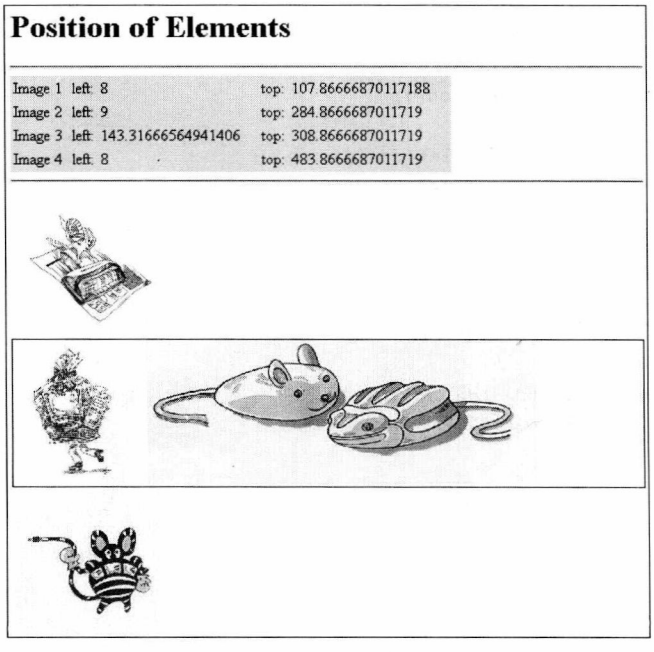

图 7-15 这是没有定位元素时网页的外观

前两幅图像没有受到单独的影响。但是第三幅图像和第二幅图像一起处于 div 区域内，被 ID p2 的规则绝对定位。div 区域之外的第四幅图像的情况也一样。两幅图像的 top 规格完全相同！结果是，第三幅图像和第四幅图像部分重叠，因为它们都被设置相同的 top 位置。如图 7-16 所示。

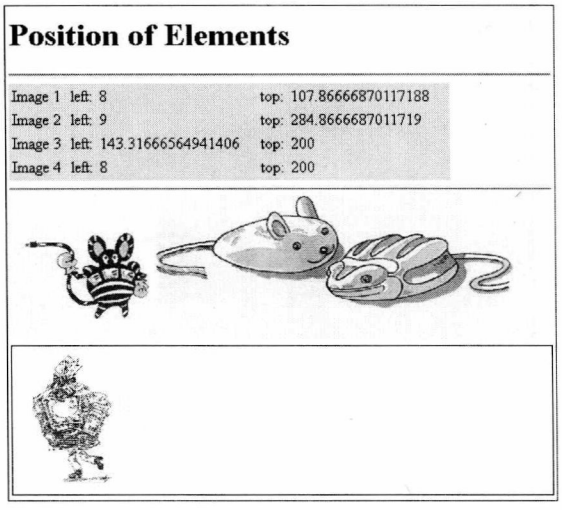

图 7-16 第三幅和第四幅图像的顶部位置重叠

注意，第三幅图像的偏移父元素明显是网页，第四幅图像和网页中的所有其他元素也一样——即使没有被定位也是如此。现在，我们对 p1 规则略作扩展（ch7_2.css）。

程序清单 7.25 修改后的 CSS 规则

```
...
08 #p1 {
09  border: solid 1pt;
10  position:absolute;
11 }
...
```

我们现在对 div 区域应用绝对定位。注意，我们既不指定 top，也不指定 left。指定为 absolute 足够将 div 元素变成内容（例子中是两幅图像）的偏移父元素。

这对第三幅图像的定位产生了很大的影响，因为 top 规格现在从 div 元素的左上角开始计算。所以，第三幅图像[⊖]和第四幅图像[⊖]的 top 位置不再重叠，尽管在 CSS 文件中的数值相同。

而且，影响甚至可能更进一步。内容为两幅图像的 div 元素的宽度基于其内容，而内容仍然直接在这个块元素的内部文本流中。而内部文本流中现在只有第二幅图像，因为第三幅

⊖ 从偏移 div 的左上角开始计算。

⊖ 从网页的偏移开始计算。

图像被绝对定位，从而从其父元素的文本流中清除。

因为绝对定位，div 元素本身已经从整个文档的常规文本流中清除，在这个文本流中，第一幅图像后面紧跟着第四幅图像。第四幅图像已经是绝对定位，但是因为 left 规格还没有设置，所以它按照常规文本流的方向，被定位在第一幅图像旁边。如图 7-17 所示。

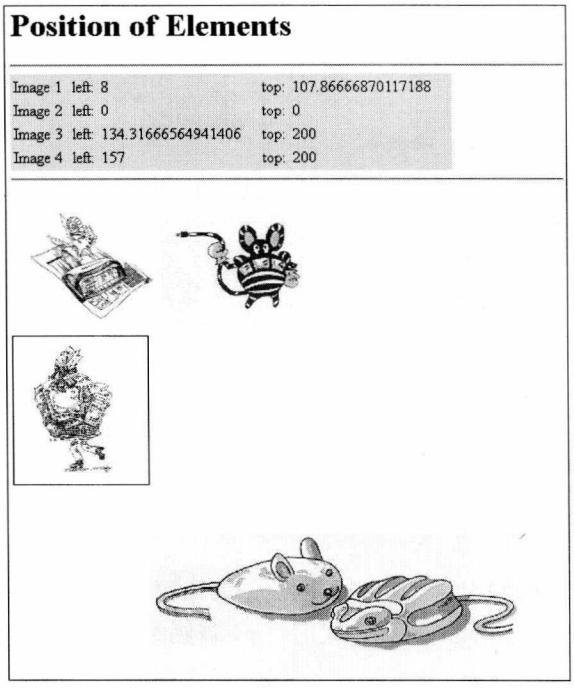

图 7-17 div 区域绝对定位的效果

7.3.2 文档相对定位：offset()

使用 offset()，可以进行文档的相对定位。可以用它定位元素，以及获取元素的位置。

获取位置

offset() 方法提供一个返回值，以像素为单位返回第一个匹配元素的当前偏移（top 和 left）。但是和 position() 不同，返回值总是和文档相关的值，即使有一个偏移父元素也是如此。

新示例基于 ch7_7.html 网页，该网页与 ch7_6.html 完全相同。但是在 CSS 文件 ch7_7.css 中，我们做了几处改动。

程序清单 7.26 新的 CSS 文件

```
01 img {
02  height: 150px;
03 }
```

```
#info {
 width: 640px;
 background: yellow;
}
#p1 {
 border: solid 1pt;
 position: absolute;
 left: 100px;
}
#p2 {
 position: absolute;
 top: 160px;
 left: 10px;
}
#p3 {
 position: absolute;
 top: 290px;
 left: 280px;
}
```

我们完全定位第三幅和第四幅图像。特别是，div 区域作为第二幅图像和第三幅图像的偏移父元素，方向被设置为 left（左）。大家将会看到，这一规格被累加到 offset() 提交的值上。

JavaScript 文件 ch7_7_ready.js 和前一个例子相比只是稍作修改。

程序清单 7.27　文档偏移的实际效果

```
$(function(){
 var offset = $("img:first").offset();
 $("#info").html(
   "<tr><td>Image 1</td><td>Offset left:</td><td>" +
     offset.left +
     "</td><td>Offset top:</td><td>" +
     offset.top + "</td></tr>");
 offset = $("img:eq(1)").offset();
 $("#info").append(
   "<tr><td>Image 2</td><td>Offset left:</td><td>" +
     offset.left +
     "</td><td>Offset top:</td><td>" +
     offset.top + "</td></tr>");
 offset = $("img:eq(2)").offset();
 $("#info").append(
   "<tr><td>Image 3</td><td>Offset left:</td><td>" +
     offset.left +
     "</td><td>Offset top:</td><td>" +
     offset.top + "</td></tr>");
 offset = $("img:eq(3)").offset();
 $("#info").append(
   "<tr><td>Image 4</td><td>Offset left:</td><td>" +
```

```
    offset.left +
    "</td><td>Offset top:</td><td>" +
    offset.top + "</td></tr>");
offset = $("#p1").offset();
$("#info").append(
  "<tr><td>DIV #p1</td><td>Offset left:</td><td>" +
    offset.left +
    "</td><td>Offset top:</td><td>" +
    offset.top + "</td></tr>");
});
```

除了处理与文档相关的偏移之外，还可以看到，我们获得绝对定位的 div 区域的坐标。注意包含在其中的对象的定位，它当然考虑了父元素的定位，如前所述，这对 offset() 的数据有影响。如图 7-18 所示。

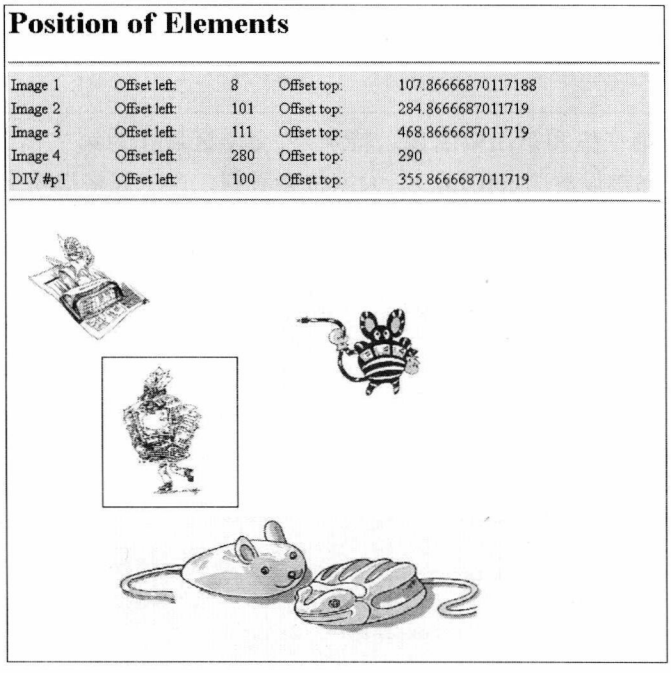

图 7-18 与文档相关的各种偏移规格

设置位置

在 jQuery 1.4 中，还可以在一个选项表达式中指定两个坐标，通过 offset() 设置位置（总是与文档相关）。作为替代，可以用一个回调返回你想要设置的坐标。回调函数有两个可选的默认参数，和前面的类似函数一样。第一个可选参数代表目前处理的元素在前面指定的一组元素中的索引位置。第二个可选参数以坐标对象的形式，表示当前处理元素的 left 和 top 值。

程序清单 7.28 基本文件（ch7_8.html）

```
...
10      <script type="text/javascript"
11          src="lib/ch7_8_ready.js"></script>
12  </head>
13  <body>
14        <h1>Position of Elements</h1><hr/>
15      <button>Image 1</button><button>Image 2</button>
16      <button>Image 2</button>
17    <button>Image 3 and 4</button><hr/>
18      <img src="images/img1.jpg" />
19      <img src="images/img2.jpg" />
20      <img src="images/img3.jpg" />
21      <img src="images/img4.jpg" />
22  </body>
23 </html>
```

在这个网页中，可以看到四幅图像。我们希望通过 offset()，以三种不同的方式定位它们。程序清单 7.29 展示了 JavaScript 文件 ch7_8_ready.js，我们在其中完成上述任务。[⊖]

程序清单 7.29 设置定位数据

```
01 $(function(){
02  $("button:first").click(function(){
03    $("img:first").offset({
04      top: 200,
05      left: 30
06    });
07  });
08  $("button:eq(1)").click(function(){
09    $("img:eq(1)").offset(function(index, coords){
10      return {
11        top: 100,
12        left: 300
13      };
14    });
15  });
16  $("button:eq(2)").click(function(){
17    $("img:gt(1)").offset(function(index, coords){
18      if (index == 0)
19        return {
20          top: coords.top + 10,
21          left: coords.left + 30
22        };
23      else
24        return {
25          top: coords.top - 10,
26          left: coords.left - 30
```

⊖ 本例不关注 CSS 文件。

```
27          };
28      });
29  });
30 });
```

在加载该站点之后，图像仍然处于正常的文本流之内。如图 7-19 所示。

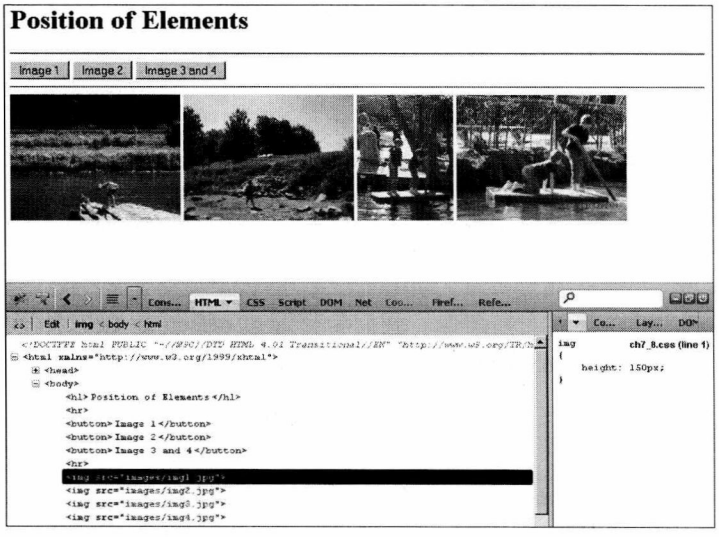

图 7-19　图像仍然像平常一样处于文本流之中

如果用户单击第一个按钮，定位通过第 3 ～ 6 行的语句调用。我们通过一个选项对象设置 left 和 top 属性。这涉及相对定位，在 Firebug 中可以看到。如图 7-20 所示。

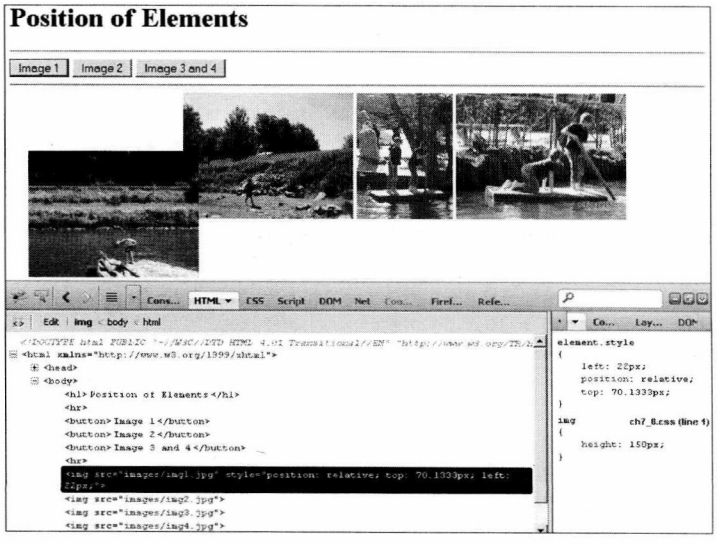

图 7-20　第一幅图像相对定位

注意，实际的定位规格和源文本中的值不相同，但是在后台计算，使其与文档偏移保持一致。这使 jQuery 框架可以弥补可能发生的任何浏览器特殊情况。

第 9 ～ 14 行的语句在用户单击第二个按钮时执行，我们用它们以一个返回匹配选项对象的回调函数，定位第二幅图像。如图 7-21 所示。

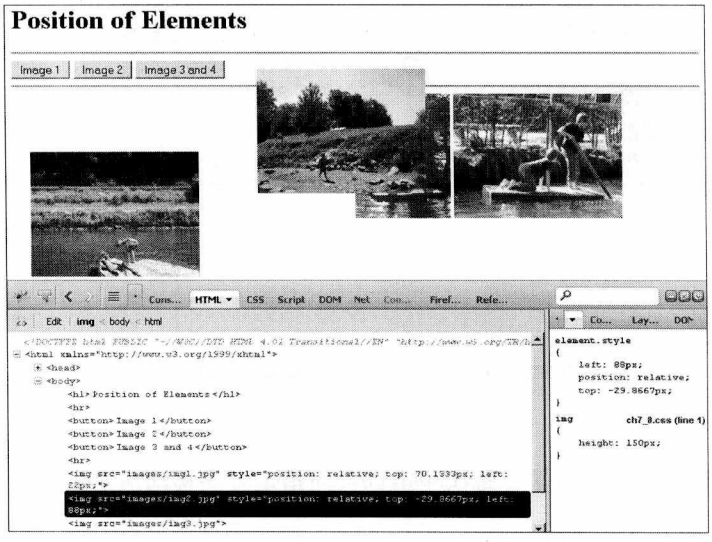

图 7-21 现在，第二幅图像也相对定位

单击按钮 3 时，可以看到如何使用回调函数可选的默认参数创建定位逻辑。如果打算实现一个逻辑，回调函数通常是这种情况下的唯一合适选择。我们再次查看相关的源代码片段。

程序清单 7.30 第 3 个按钮的回调函数

```
17    $("img:gt(1)").offset(function(index, coords){
18      if (index == 0)
19        return {
20          top: coords.top + 10,
21          left: coords.left + 30
22        };
23      else
24        return {
25          top: coords.top - 10,
26          left: coords.left - 30
27        };
28    });
```

我们对图像 3 和图像 4 应用定位。因此，回调函数被调用两次。对于第三幅图像，index 的值为 0。在这种情况下，我们为过去的 top 属性（通过 coords.top 获得）值加上 10 个像素，为过去的 left 属性（通过 coords.left 获得）值加上 30 个像素。

对于所有其他的图像（例中只有第四幅图像），我们相应地减小属性值。如图 7-22 所示。

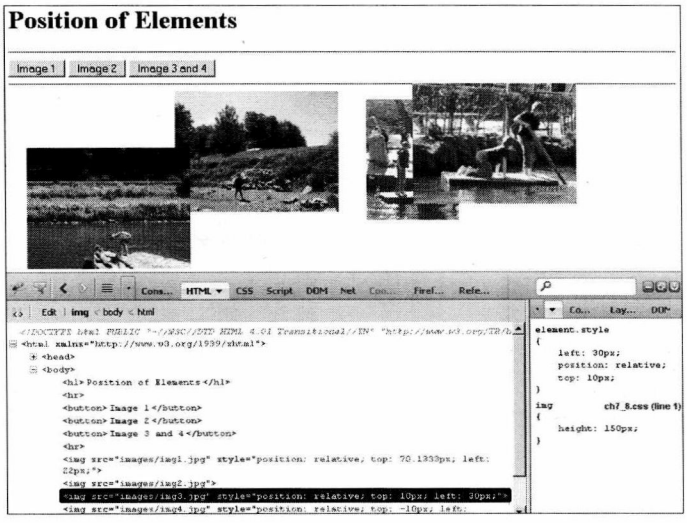

图 7-22 新的编排

因为定位总是指向当前位置，每次单击都会向指定方向移动两个图像。如图 7-23 所示。

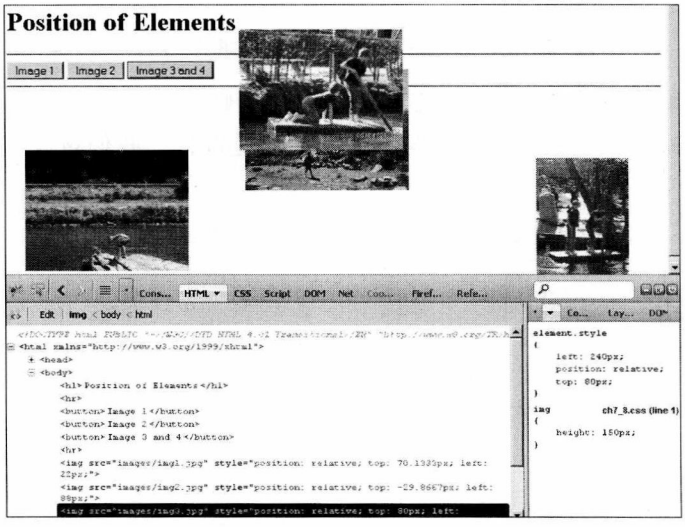

图 7-23 图像 3 和图像 4 在几次单击后被移到更远处

7.4 滚动方法

使用 scrollTop() 和 scrollLeft() 方法，可以获取元素从顶部或者从左侧已经滚动的值。如果向该方法传递一个整数值作为参数，就可以将该元素移动指定的像素数。当然，这个值

被看作与外围的偏移父元素相关。所以，这与用户通过滚动条移动内容的效果相同。

程序清单 7.31 基本文件（ch7_9.html）

```
...
10     <script type="text/javascript"
11      src="lib/ch7_9_ready.js"></script>
12  </head>
13  <body>
14    <h1>Scrolling elements</h1><hr/>
15    <button>Image 1</button>
16    <button>Image 2</button>
17    <div><img src="images/img1.jpg" /></div>
18    <div><img src="images/img2.jpg" /></div>
19    <span id="output1"></span><br/>
20    <span id="output2"></span>
21  </body>
22 </html>
```

在上述网页中，可以看到两幅图像，它们各被插入一个 div 元素。但是 div 区域比图像小，所有超出的内容都被截断。这通过 CSS（ch7_9.css）以常规的方法完成。

程序清单 7.32 图像需要的空间比网页可用空间更大

```
01 img {
02  height: 300px;
03 }
04 div {
05  width: 200px;
06  height: 200px;
07  overflow:auto;
08 }
```

上例显示两幅图像，但是都没有完整地展示，因为 div 太小，而 CSS 特性 overflow 被设置为 auto 和一个固定值。结果是，每个图像都只显示一部分。如图 7-24 所示。

我们希望用户能够单击一个按钮，在相关的区域中滚动图像。下面是实现的方法（ch7_9_ready.js）。

程序清单 7.33 滚动

```
01 $(function(){
02  $("div:lt(2)").scrollTop(0);
03  $("div:lt(2)").scrollLeft(0);
04  $("button:first").click(function(){
05    $("div:eq(0)").scrollTop(190);
06    $("div:eq(0)").scrollLeft(210);
07    $("#output1").text("Image 1 - scrollTop: " +
08    $("div:eq(0)").scrollTop() +
09    ", scrollLeft: " +
10    $("div:eq(0)").scrollLeft());
```

```
11 });
12 $("button:last").click(function(){
13   $("div:eq(1)").scrollLeft(265);
14   $("div:eq(1)").scrollTop(280);
15   $("#output2").text("Image 2 - scrollTop: " +
16   $("div:eq(1)").scrollTop() +
17   ", scrollLeft: " +
18   $("div:eq(1)").scrollLeft());
19 });
20 $("#output1").text("Image 1 - scrollTop: " +
21 $("div:eq(0)").scrollTop() +
22 ", scrollLeft: " +
23 $("div:eq(0)").scrollLeft());
24 $("#output2").text("Image 2 - scrollTop: " +
25 $("div:eq(1)").scrollTop() +
26 ", scrollLeft: " +
27 $("div:eq(1)").scrollLeft());
28 });
```

与此相反，div 容器根据 overflow 属性的值获得滚动条。这也就意味着，div 容器中的图像可以通过这些滚动条（也可以通过 jQuery 或者 JavaScript）滚动。

加载网页之后，我们在第 2 行和第 3 行中首先将图像滚动到位置（0，0）。通过第 21 ～ 27 行中的 scrollTop() 和 scrollLeft()，我们在相关的 div 容器中显示图像的当前移动情况（以像素表示）。

通过两个按钮，用户现在可以在 div 容器中通过左侧或者顶部的固定因子，滚动第一幅或者第二幅图像。[⊖]我们在网页中再次显示新的坐标值。如图 7-25 所示。

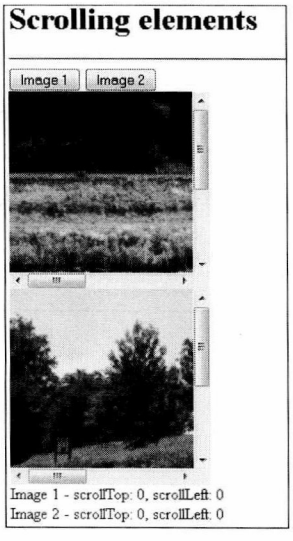

图 7-24 只显示图像的一部分

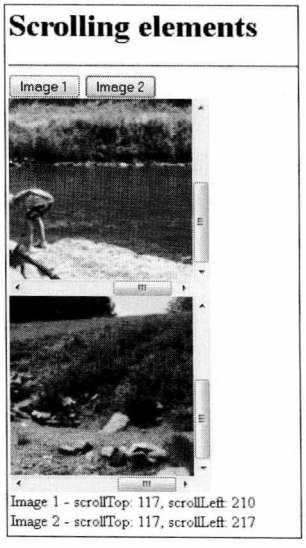

图 7-25 图像已经被滚动

⊖ 当然，你没有必要每次都从两个方向进行重新定位。

7.5 高度和宽度

对于网页中的许多操纵和表示，高度和宽度很重要。jQuery 当然也提供合适的方法，访问元素的高度和宽度。这些方法基于对应的 CSS 属性。

height() 和 width()

height() 和 width() 方法简单地返回第一个匹配元素的高度和宽度（以像素表示）。然而，如果指定一个数字参数代替，就可以设置该元素的高度或者宽度。

提示

和纯 HTML 中一样，常常只需要指定元素的高度或者宽度。对应的属性通常由浏览器自动按比例计算。只减少一个属性的值也绝非少见。相反，指定两个值可能导致失真，当然，在有些情况下你可能是有意为之。不过，如果只操作一个属性，必须确保其他属性确实“清晰”。某些手段（如应用某些 jQuery 方法）可能已经在后台指定了高度和宽度。尺寸也可能通过 CSS 设置。如果只修改两个值中的一个，就可能导致失真，或者使整个布局无法正常工作。从这一点上看，通过 jQuery 同时指定高度和宽度是一个好主意。

程序清单 7.34 基本文件（ch7_10.html）

```
...
10      <script type="text/javascript"
11       src="lib/ch7_10_ready.js"></script>
12  </head>
13  <body>
14         <h1>Width and Height of Elements</h1><hr/>
15      <button>More</button>
16      <button>Less</button>
17      <button>Slightly distorted/button>
18      <hr /><img src="images/img4.jpg" />
19      <div id="output1"></div>
20  </body>
21 </html>
```

在上述网页中，可以看到一幅图片，最初按照原始大小显示。在 CSS 文件中，我们只指定网页背景颜色。特别是，我们没有指定图像的宽度和高度。我们通过 CSS 确定输出区域的位置和大小，所以可以用它作为图形尺寸的比例。

我们希望用户可以单击一个按钮，在相关区域中滚动图像。程序清单 7.35 中是具体的做法（ch7_10_ready.js）。如图 7-26 所示。

程序清单 7.35 访问元素的宽度和高度

```
01 $(function(){
02  $("button:first").click(function(){
03     $("img").width(440);
```

```
04    $("#output1").text("Width: " + $("img").width());
05  });
06  $("button:eq(1)").click(function(){
07    $("img").width(64);
08    $("#output1").text("Width: " + $("img").width());
09  });
10  $("button:last").click(function(){
11    $("img").width(440);
12    $("img").height(64);
13    $("#output1").html("Width: " +
14      $("img").width() + " Height: " +
15      $("img").height());
16    ;
17  });
18 });
```

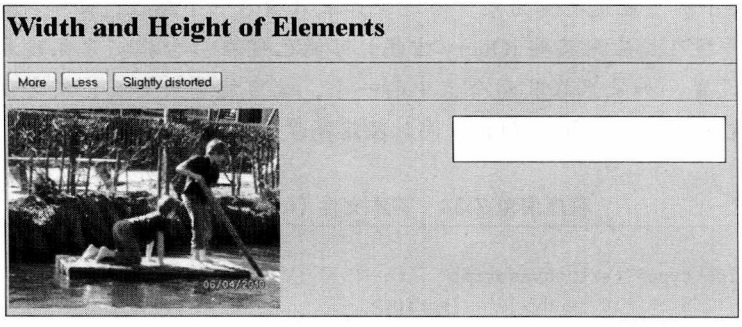

图 7-26 原始图像大小

在本例中，我们访问一幅图像的宽度和高度。单击第一个按钮，图像的宽度在第 3 行中被设置为 440 个像素。高度被自动改变。宽度通过第 4 行的 width() 方法（没有参数）输出到网页中。如图 7-27 所示。

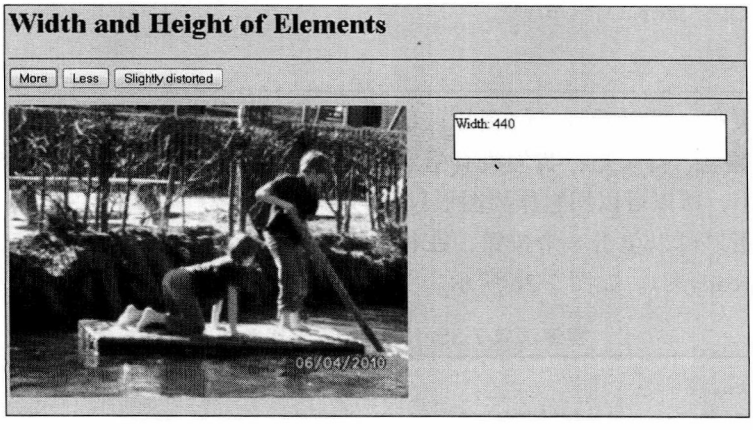

图 7-27 大的图像

通过第二个按钮，宽度被减小到 64 个像素，高度同样自动变化。如图 7-28 所示。

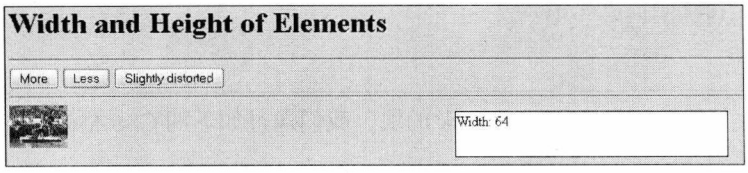

图 7-28 减小的图像

第三个按钮被点击时，我们设置了高度和宽度。我们使图像出现失真。如图 7-29 所示。

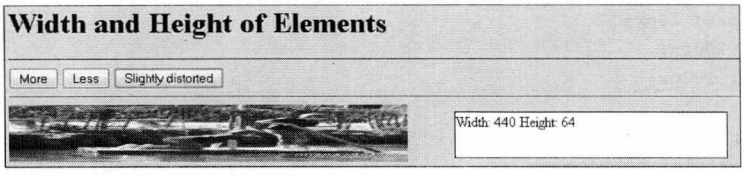

图 7-29 失真的图像

警告

注意，扩大图像的单击操作在设置高度之后不再有效，就像第三个按钮被单击的时候一样。这里，可以明确地看到一种情况，高度不再是清晰的，仅修改其他属性不再有效。

7.6 内部和外部尺寸

网页上有许多视觉元素，某些元素往往会影响超过“自然”尺寸的元素所需的空间。这对边距（margin）、边框（border）和填充（padding）特别有影响。[⊖]所以，还需要考虑元素的内部和外部尺寸。这是经典的 CSS 块模式。

用 innerHeight()，可以获得第一个匹配元素的内部高度（没有边框，但是包含填充），用 innerWidth() 则相应地得到内部宽度。outerHeight() 提供第一个匹配元素的外部高度（包括边框和填充），相应地，outWidth() 提供第一个匹配元素的外部宽度。可以为外部尺寸使用一个可选的布尔参数，指定边距是否考虑在内（如果考虑边距，参数为 true；没有参数意味着 false）。

程序清单 7.36 基本文件（ch7_11.html）

```
...
10      <script type="text/javascript"
11       src="lib/ch7_11_ready.js"></script>
12  </head>
13   <body>
```

⊖ 由外而内。

```
14        <h1>Inner and Outer Dimensions</h1><hr/>
15      <div id="block">www.rjs.de</div>
16      <div id="output"></div>
17  </body>
18 </html>
```

在网页中，可以看到一个 ID 为 block 的块，我们通过如下的 CSS 格式化它（ch7_11.css）。

程序清单 7.37 指定块的尺寸

```
01 #block {
02   background: gray;
03   color: #FFFF00;
04   font-size: 24px;
05   width: 200px;
06   height: 100px;
07   margin: 30px;
08   padding: 50px;
09   border-style: solid;
10   border-width: 5px
11 }
```

我们打算用前面提到的 jQuery 方法读出这些尺寸（ch7_11_ready.js）。

程序清单 7.38 内部和外部尺寸

```
01 $(function(){
02  $("#output").html("width: " + $("div:first").width()
03  + "<br>innerWidth: " +
04  $("div:first").innerWidth() +
05  "<br>outerWidth: " +
06  $("div:first").outerWidth() +
07  "<br>outerWidth(true): " +
08  $("div:first").outerWidth(true) +
09  "<br>height: " +
10  $("div:first").height() +
11  "<br>innerHeight: " +
12  $("div:first").innerHeight() +
13  "<br>outerHeight: " +
14  $("div:first").outerHeight() +
15  "<br>outerHeight(true): " +
16  $("div:first").outerHeight(true));
17  });
```

在例子中，我们获取各种通过 CSS 设置的尺寸数值并输出。例如，可以看到 div 元素的宽度加上边框宽度和填充值得到的外部宽度。[⊖]如果通过布尔参数包含边距，该值也会相应增加。如图 7-30 所示。

⊖ 当然，每种情况下都乘上因数 2。

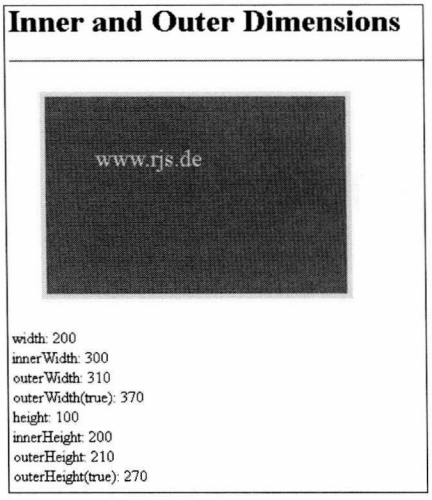

图 7-30 各种尺寸规格

提示

使用 Firebug，可以在 DOM 类别中非常简单地分析元素的内部和外部尺寸。

7.7 小结

本章说明了如何在 jQuery 下动态地使用样式表。通过合适的 jQuery 方法进行元素的动态 CSS 格式化是该框架最明显的一个选项。我相信，这些精选的方法和概念能够打下基础——应用起来非常简单，而且封装了浏览器相关的特殊性和不足。

第 8 章

jQuery 下的事件处理

本章中，我们转向 jQuery 的事件处理。换句话说，我们关注网页中触发某些操作的情况。这无疑是 jQuery 之类框架应用中最为重要的应用之一，因为用 JavaScript 机制处理网页事件的响应并不都很简单——这点可能与你的期望相悖。除了少数例外情况，利用超文本标记语言（HTML）事件处理器的替代响应方法也得到了相当好的支持，在这种情况下，人们从网页上甚至不知道有任何 JavaScript，但是这种方法功能较弱，也不完善。更糟糕的是，结构和功能性混合在一起，我们在较为复杂的 Web 应用中应该加以避免。利用 jQuery，我们就有了一种非常强大的机制，能够成功而安全地应付这一复杂的主题。

8.1 事件、事件处理器、触发器和数据绑定的基本信息

大家可能知道，如果将 JavaScript 语句和函数调用直接写入脚本容器或者外部脚本，它们就会简单地通过在浏览器中加载网页而执行。浏览器加载相关的脚本代码行之后，对应的语句立刻被执行。这是 JavaScript 执行的最简单情况，但是通常只用于初始化。当然，除了刚刚加载的网页之外，在其他一些情况下，你也可能打算调用一个函数或者一条语句。

8.1.1 事件

这里的“事件”这一术语指的是在浏览器中有活动网页时发生的任何事情。这些事件可能包括页面加载到浏览器中，或者从浏览器中删除页面，还有用户执行的操作（例如，单击一个按钮，发送表单数据或者将鼠标指针移到网页的某一部分上）。这些事件很适合于基于语句的调用。

8.1.2 事件处理器的一般信息

因此，我们需要一个机制，通过它响应这些事件，并且在必要的时候触发一条语句、一个函数或者一个对象方法。这个机制称为事件处理器。不同的事件处理器可以用于响应不同

的事件，事件处理器机制通过预定义的名称标识。

事件处理器在纯 HTML 和 JavaScript 中均存在[㊀]。因此，它们存在于两个世界，成为(X)HTML 和 JavaScript 之间最重要的联系。[㊁]另一点也很重要，HTML（版本 4 及更高）和 JavaScript 中触发的事件在功能上是完全相同的。但是 JavaScript 环境中的少数事件处理器在 HTML 中没有直接的等价物。[㊂]

8.1.3 HTML 事件处理器

在 HTML 中，事件处理器简单地由一个标签的特性组成。可以指定大部分标签的事件处理器。[㊃]

注意

W3 联盟在 HTML 语言标准中正式包含事件处理器。这一点特别重要，因为它意味着如果事件处理器写成 HTML 特性的形式，大小写字母的拼写方法就没有作用。但是，在 JavaScript 和 XHTML 中，事件处理器总是必须完全写成小写。

事件处理器的名称通常以音节“on”开始，后面跟上事件的简明描述。我想你一定熟悉这一概念。

程序清单 8.1 HTML 中的不同事件处理器

```
onmouseover
onload
onclick
```

如果对一个 HTML 标签使用这些事件处理器，可以在特性名称之后写上一个等号。和通常的 HTML 特性一样，等号之后跟着一个包含在双引号内的指定值。在这种情况下，指定值是一条 JavaScript 语句或者函数调用。

程序清单 8.2 HTML 事件处理器调用一个函数

```
<h3 onclick="myFunction()">Click me</h3>
```

8.1.4 JavaScript 事件处理器

大家可能已经知道，为了按照建议的做法严格地分离结构和逻辑，应该使用 JavaScript 事件处理器而不是 HTML 事件处理器。如果想要在 JavaScript 环境中响应事件，也可以使用一个事件处理器。JavaScript 中的事件处理器是一个对象的属性。所以，不是写下一个函

㊀ 严格地说是存在于 DOM 概念中，但是我们在此不再赘述。

㊁ 如前所述，这种联系可以从任一端发起。

㊂ onreadystatechange 就是一个例子，它只存在于 JavaScript 环境中，用于实现 AJAX 请求。

㊃ 从本质上说是对所有标签，但是对于某些标签来说这没有任何意义。

数调用作为赋值语句，而是将一个函数引用绑定到网页的一个对象上。我们来看看下面的源代码片段。

程序清单 8.3 在 JavaScript 中用一个函数引用作为事件处理器

```
<form name="myForm">
  <input type="button" name="myButton" value="OK" />
</form>
...
<script language="JavaScript">
  document.myForm.myButton.onclick = myfction;
</script>
```

例子中使用的语法引用 myfction() 函数，如果用户单击表单按钮则调用该函数。这个示例片段展示了用于选择按钮的普通 JavaScript 语法。当然，也可以用 jQuery 引用触发对象。

函数引用不允许使用括号。[⊖]但是，如何向函数传递参数值呢？现在大家已经从本书的许多例子中知道了答案。还可以使用关键字 function 后面跟上一对括号——可能带有参数值——以及一个函数体，可以在其中写下任何 JavaScript 语句（或者函数调用），来代替对函数的直接引用。看看如下的片段。

程序清单 8.4 一个匿名函数

```
document.onclick = function(){
  myFction();
};
```

虽然也可以再次使用命名函数，但是实践中最常使用的是匿名函数。[⊜]一般来说，还可以使用一个通用的回调函数，该函数必须将一个函数引用作为返回值。

8.1.5 事件对象

JavaScript 中的事件处理器的背后有一个特殊的对象：事件对象。用户不断地产生这种事件对象，[⊜]可以用针对性的方式使用这些对象，响应事件。事件对象提供一系列用于该目的的有趣属性。我们以用户鼠标单击网页任何位置的情况作为例子。每次鼠标单击导致浏览器创建一个事件对象，包含如下的信息：

- 使用的鼠标按钮
- 按下的任何附加键（Ctrl、Alt、CapsLock）
- 单击的坐标

进一步的事件创建的其他事件对象当然包含与事件对应的不同信息。例如，如果按下一个键，可以获取关于哪个键被按下的信息。一般来说，事件对象包含几个部分的有用信息，

⊖ 明确地说，这不是一个调用！

⊜ 你从不会需要这种回调函数的标识符。

⊜ 在你将鼠标指针移过浏览器区域时，你创建的是多个事件对象（而不是仅有一个）。

可以用它们创建适应性强的应用程序。

这给我们提出了一个问题——如何使用事件对象？答案既简单又复杂。简单的一部分是，要访问事件对象的属性，可以和平常一样用句点标记法引用对象。复杂的部分是，对于事件对象的名称或者其可用性，必须区分 Microsoft 事件和其他浏览器的概念，关于对象的可见性也有一些差异。Microsoft 对事件对象的属性使用专有的命名约定。jQuery 封装了事件对象来弥补这些差异，即使对 Internet Explorer 也能使用标准化的语法。

警告

Web 应用程序中的大部分事件得到几乎所有现代浏览器的支持。但是，有些事件是浏览器特有的，特别是，即使 HTML 事件处理器在标签支持的事件方面也存在差异。在处理事件对象时事情变得复杂，因为除了不同标识符和访问级别不同引起的所有麻烦之外，如前所述，甚至连事件对象的相同属性在不同浏览器概念中也有不同的命名。[⊖]如果不将浏览器世界分离成 Microsoft 和“其他”，就什么也做不成，即使做了这样的分割，这整件事也还是极其困难。

8.1.6 冒泡

在与事件对象的联系中，我们会遇到“冒泡事件”（Bubble Events）或者“冒泡阶段”（Bubble Phase）这两个术语。如前所述，浏览器不断产生事件对象（不只是鼠标单击等几种事件）。在浏览器区域上移动鼠标的情况发生得很频繁，所以在短期内可能生成数千个事件，也可以考虑一下滚动文本的情况。大部分这类事件都完全没有被应用程序处理，而是简单地忽略。毕竟，除了少数特殊情况，有什么必要对鼠标移动几毫米做出响应呢？

如果树中的一个节点（例如图像或者段落）发生了一个事件，问题是这些对象中哪个负责相关的事件。毕竟，节点在树中嵌套得很深。如果多个对象使用同一个事件处理器，并且都能够响应这些事件，会发生什么？

以上问题通过事件冒泡来解决。这意味着，事件总是在所发生的位置最靠内的元素中处理——如果那里有合适的事件处理器的话。最靠内的元素是离层次结构的根最远的元素。如果这个元素没有合适的事件处理器，事件对象被传递给树的文档对象模型（DOM）中下一个较高层的对象，以此类推。事件对象就像气泡一样上升，直到树根，冒泡的过程经历了整棵树。如果事件对象到达根部时仍未被处理，则该对象被删除。[⊖]事件对象这种未经处理的删除是规则，而不是异常（这是因为大部分事件（例如移动鼠标指针几毫米）没有强制应用程序做出特定的响应）。

⊖ 这是 20 世纪 90 年代发生的不幸的浏览器战争的结果。“如果你打算将一个属性命名为 which，我就绝对不这么做。我要用什么名称来最大程度地产生差异呢？恩，keyCode 怎么样？那会让你的开发人员陷入苦苦的思考之中。”

⊖ 但是有些模型中事件对象会被返回给触发元素，只在回溯的路上没有被处理时才被删除。

8.1.7 数据绑定

为了响应事件，还可以使用数据绑定机制。利用这一机制，可以创建两个变量或者表达式之间的直接关系。[⊖]可以将目标的值和绑定表达式的值关联。这种绑定表达式可以是任何类型的简单值、对象、函数的结果或者一个表达式。数据绑定的处理类似于表格或者电子数据表（如 OpenOffice Calc 或者 Microsoft Excel）中单元格之间的引用关系（特别是那些在公式中使用的单元格引用以及和其他单元格相关的公式）。如图 8-1 所示。

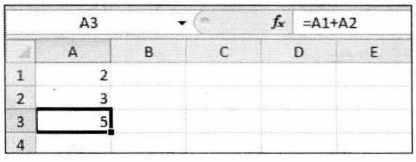

图 8-1 电子数据表软件中的经典数据绑定

如果被应用单元格的值有任何变化，绑定到该单元格的其他表达式按照绑定规则（公式）的描述变化。这种变化的发生没有延迟，更新也没有必要手工触发。

8.1.8 触发器

为了对某些事件做出响应，还有一种被称作触发器（Trigger）的机制。触发器由数据修改的结果触发。与数据绑定相比，还可以在触发器主体内包含复杂的语句。这种代码块每当特定事件发生时执行。如果打算为变量值的变化执行灵活或者复杂的代码，触发器就很有用。

8.2 jQuery 中的事件对象

本章中提出的警告和意见已经很清晰，如果打算支持超过一种浏览器，原生的事件对象处理需要更多的运气，而非明智的编程。加之，处理事件有一整套技术。幸运的是，jQuery 封装了这些问题，对语法进行了标准化。jQuery 事件系统通过 jQuery.Event，按照 W3C 标准对事件对象进行了规范化（包括可用属性和方法的名称），确保在所有支持的浏览器中采用标准化的处理方法。特别是，它能保证事件对象可靠地传递给事件处理器。

8.2.1 jQuery.Event 构造程序

jQuery 中事件对象的构造函数是开放的，可以由触发器以任何方式使用。可以指定一个 jQuery 事件处理器作为参数。[⊖]显然，也可以用构造函数（不使用 new 操作符）创建一个对象。或者，如果喜欢，也可以使用 new，结果都一样——我们将获得一个类型为 jQuery.Event 的对象。

程序清单 8.5 不使用 new 创建一个类型为 jQuery.Event 的新对象

```
var e = jQuery.Event("click");
```

⊖ jQuery 中数据绑定的关键是 bind()，你将在后面看到。

⊖ 不是以 on 开头的经典事件处理器。在 jQuery 中，这些事件处理器也被称作事件助手。详情见后。

下面的语法作用完全相同。

程序清单 8.6 这里使用 new

```
var e = new jQuery.Event("click");
```

8.2.2 jQuery.Event 事件对象的属性

jQuery 中的事件对象有一些属性，它们当然是基于经典的事件对象的属性。

event.type

这个属性描述事件的类型（例如，是否发生鼠标单击或者键盘输入，或者类似的事件）。

event.target

通过这个属性，可以获得触发事件的 DOM 元素。这个元素可能是注册该事件的元素或者它的一个子元素。

event.data

这个属性极为重要，因为如果将 event.data 传递给 bind() 且当前执行处理器被绑定，它能提供可选的数据。

event.relatedTarget

在鼠标移动的情况下，如果鼠标指针移到另一个元素或者离开另一个元素，该属性包含了鼠标之前所在的 DOM 元素。这是一个有关联的节点。

event.currentTarget

通过这个属性，可以获取冒泡阶段中当前的 DOM 元素。这个特性通常等价于函数中的 this 值。

event.pageX/event.pageY

鼠标事件发生的 X 和 Y 坐标。这是相对于文档的坐标。该属性还解决了 Internet Explorer 在坐标标准化记法上的偏差。

event.screenX/event.sceenY

屏幕坐标。

event.result

事件处理器返回的最后值。

event.timeStamp

事件创建的时间戳，以毫秒表示。

一个实用的示例有助于说明 jQuery.Event 对象的这些属性（ch8_1.html）。

程序清单 8.7 解释事件对象特性

```
...
06      <link rel="stylesheet" type="text/css"
```

```
07        href="lib/ch8_1.css" />
08    <script type="text/javascript"
09        src="lib/jquery-1.8.2.min.js"></script>
10    <script type="text/javascript"
11        src="lib/ch8_1_ready.js"></script>
12  </head>
13  <body>
14    <h1>Different Properties of the event Object</h1>
16    <table>
17      <tr><th>Triggering object</th>
18        <th>Information</th>
19      </tr>
20      <tr><td><img src="images/pic1.jpg" id="pic1"/></td>
21        <td id="output1"></td></tr>
22      <tr><td><img src="images/pic2.jpg" id="pic2"/></td>
23        <td id="output2"></td></tr>
24    </table>
25  </body>
26 </html>
```

上述网页在一个表格的第一列包含两幅图像。第二列用于显示事件信息。在引用的层叠样式表（CSS）文件中，我们只将表格的列宽指定为 400 像素。我们不需要进一步关注 CSS 文件，当然，我们应该关注 JavaScript 文件 ch8_1_ready.js。

程序清单 8.8　事件对象属性的具体计算

```
01 $(function(){
02  $("#pic1").click(function(event){
03    $("#output1").html("timeStamp: " +
04       event.timeStamp +
05    "<br />screenY: " +
06    event.screenY +
07    ", screenX: " +
08    event.screenX);
09  });
10  $("#pic2").mousemove(function(event){
11    var pageCoordinates = "( " + event.pageX + ", " +
12       event.pageY + " )";
13    $("#output2").html("e.pageX, e.pageY: " +
14    pageCoordinates +
15    "<br />Triggering node: " +
16    event.target.nodeName);
17  });
18  $("#pic2").mouseout(function(event){
19    $("#output2").html("Related node: " +
20    event.relatedTarget.nodeName);
21  });
22 });
```

例中，可以看到事件触发和处理的三种情况：在第一幅图像上单击（click()），将鼠标指针移到第二幅图像（mousemove()）以及鼠标指针退出第二幅图像区域（mouseout()）。回调函数的参数是 jQuery 事件对象。

注意

在这个例子中，我们使用前面提到过的 jQuery 事件助手，我们将在本章后面更详细地讨论它。但是，如果考虑 JavaScript/DOM 或者 HTML 的“常规事件处理器”，它们应该相当清楚了。而且，我们已经多次使用了 click()。

在第 4 行中，可以看到我们是如何用 event.timeStamp 获得事件的时间戳，而在第 6 行和第 8 行中，我们用 event.screenX 和 event.screenY 获得屏幕坐标。在第 11 行和第 12 行，我们使用 event.pageX 和 event.PageY 确定页面坐标。事件的触发节点通过 evnet.target 确定，因为我们打算显示其名称，所以使用它的 nodeName 属性。如图 8-2 所示。

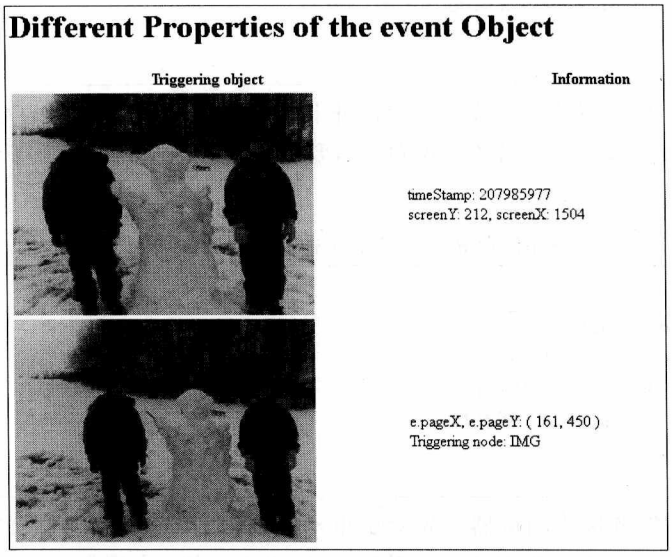

图 8-2 用户单击图像 1 以及鼠标指针在图像 2 区域上

在第 20 行中，我们通过 event.relatedTarget.nodeName 获取退出第二幅图像所涉及的节点。如图 8-3 所示。

图 8-3 涉及节点的名称

8.2.3 jQuery.Event 类型对象的方法

和事件对象相关的还有一些方法。这些方法主要用于阻止浏览器的某些默认响应。

event.preventDefault() 和 event.isDefaultPrevented()

通过 event.preventDefault() 方法，可以阻止浏览器执行这时通常必须执行的默认操作。

乍一看，这一功能有些奇怪，但是它很有用且很明智。例如，可以阻止触发用户单击的超链接。如果打算将单击用于另一项操作（例如启动动画或者通过 Ajax 加载数据），这样做就很合适了。也可以阻止通过提交按钮发送表单数据。在特殊情况下，还可能单独组织某些元素的指定全局默认操作。

利用布尔方法 event.isDefaultPrevented()，可以检查对于事件对象是否已经调用了 event.preventDefault()（ch8_2.html）。

程序清单 8.9 求取事件对象的特性值

```
...
10    <script type="text/javascript"
11        src="lib/ch8_2_ready.js"></script>
12  </head>
13  <body>
14    <h1>Prevent Redirection</h1>
15    <a href="http://rjs.de">Come to me</a><br />
16    <a href="http://ajax-net.de">Or to me</a>
17  </body>
18 </html>
```

上述网页包含两个超链接。我们打算阻止相关的重定向。在引用的 CSS 文件中，我们只指定了超链接的文本装饰。同样，我们不需要对此做进一步的解释。下面是 JavaScript 文件 ch8_2_ready.js。

程序清单 8.10 阻止单击超链接时的重定向

```
01 $(function(){
02   $("a").click(function(event){
03     event.preventDefault();
04     alert("I am not letting you. " +
05     event.isDefaultPrevented());
06   });
07 });
```

如果用户单击超链接，浏览器首先尝试重定向到该 URL。第 3 行中的语句阻止这种操作。它在单击每个超链接时调用的事件助手方法 click() 中。在通知中，我们说明了该方法是否被调用（在这种情况下这当然很简单）。如图 8-4 所示。

event.stopPropagation()、event. isPropagationStopped()、event. stopImmediatePropagation() 和 event.isImmediatePropagationStopped()

通过 event.stopPropagation() 方法，可以阻止事件对象在树中的冒泡过程。这意味着，如果触发元素没有合适的事件处理，事件对象不会冒泡到父元素。如果事件对象在子元素上没有处理，可用于父元素的事件处理程序不会注册这个对象。

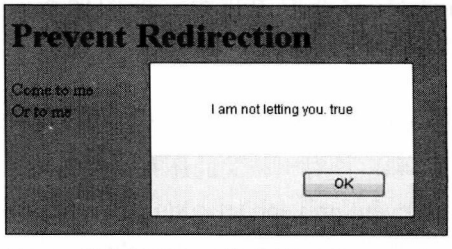

图 8-4 单击被阻止，代之以显示 alert() 窗口

警告

Internet Explorer 使用 event.stopPropagation() 方法时有某些问题。如果使用 Internet Explorer，event.cancelBubble=true；语句将生效。

可以用 event.isPropagationStopped() 检查对事件对象是否调用过 event.stopPropagation() 方法。

注意

调用 event.stopPropagation() 方法不会影响同一个元素上执行的其他处理程序。

event.stopImmediatePropagation() 和 event.isImmediatePropagationStopped() 的工作方式与 event.stopPropagation()、event.isPropagationStopped() 很类似。但是使用 event.stopImmediatePropagation()，浏览器简单地阻止其余处理程序的执行。向父元素冒泡的过程也被阻止，因为 event.stopImmediatePropagation() 间接调用 event.stopPropagation()。类似地，可以使用 event.isImmediatePropagationStopped() 检查事件对象上是否已经调用了 event.stopImmediatePropagation()。

注意

这些方法是特殊的，并不需要经常使用。我们省略了完整的例子。

8.3 坐稳了，出发：$(document).ready()

在本书的所有示例中几乎都使用了 $(document).ready() 方法！这已经清楚地表明了它的重要性。[⊖]但是，调用这个方法也是对一个事件的响应，所以我们应该在本章再次提及这个方法。调用该函数与 DOM 为遍历和操纵做好准备的事件相关。具体地说，一个参数传递给就绪事件处理程序：对一个 jQuery 函数的调用。

如前所述，如果需要，可以在网站中使用任意多的 $(document).ready() 事件。与之绑定的函数按照添加的顺序执行。

所有样式表应该在脚本之前集成。特别是，这种集成应该发生在调用 $(document).ready() 之前。如果没有这么做，在某些浏览器中会导致问题。

8.4 事件助手

对网页中事件的响应所需的最重要技术可能基于事件处理器。这些来自 JavaScript/DOM 或者 HTML 的“常规”事件处理器在 jQuery 中通过所谓的“事件助手”封装。这些事件助手是链接到特定事件的完全正常的方法。至于它们的使用，只需要应用 JavaScript/

⊖ 所以，我们在此不需要再使用明确的示例了。

DOM 事件处理器的知识，简单地在相关事件方法名称中省略前导的 on 即可。

注意

事件助手都是通过 bind() 和特定参数的数据绑定简写标记法。所有方法的返回值都是一个 jQuery 类型的对象。

表 8-1 描述了 jQuery 提供的事件助手。

表 8-1 jQuery 事件助手

名 称	描 述
blur()	不使用参数，调用该方法意味着触发匹配元素的 blur 事件（退出元素）。使用一个参数，这个事件助手为每个匹配元素绑定一个函数
change()	如果没有指定参数，调用该方法表示为每个匹配元素触发 change 事件（修改元素，例如表单输入字段）。如果有一个参数，该事件助手将一个函数绑定到每个匹配元素的 change 事件
click()	支持单击元素的无疑是最重要的事件助手之一。不带参数意味着触发每个匹配元素的 click 事件。如果有一个参数，该事件助手将一个函数绑定到每个匹配元素的 click 事件
dblclick()	尽管双击在经典 Web 应用中没有任何传统，但是在未来的富互联网应用（RIA）中可能起更大的作用。对于 RIA 来说，像桌面应用程序一样的表现是令人渴望的，而这些操作传统上都使用双击。这个 jQuery 事件助手可以触发每个匹配元素的 dblclick 事件。如果有一个参数，该事件助手将一个函数绑定到每个匹配元素的 dblclick 事件
error()	不使用参数，调用该方法代表着每个触发匹配元素的 error 事件，也就代表着发生的错误（例如，请求数据或者资源时）。如果有一个参数，该事件助手将一个函数绑定到每个匹配元素的 error 事件
focus()	如果没有指定参数，调用该方法等价于为每个匹配元素触发 focus 事件（获得焦点，例如一个按钮或者表单中的一个输入字段）。如果有一个参数，该事件助手将一个函数绑定到每个匹配元素的 focus 事件
focusin()	该事件在 jQuery 1.4 中加入，在版本 1.4.3 中进行了更新，在一个元素或者该元素包含的任何元素获得焦点时触发。这个事件与 focus 事件不完全相同，因为它还支持父元素中 focus 事件的发现（换句话说，支持事件冒泡）。到版本 1.4.3 时，可以指定一个 map 类型的对象为第一个参数，通过该参数可以将数据传递到事件处理程序
focusout()	focusin() 的对应事件。焦点从一个元素或者父元素上移开。这种情况也不同于相关的 blur()。这个事件助手也是从 jQuery 1.4 或者 1.4.3 起才出现的
keydown()	键盘事件的支持在传统的 Web 编程遭到相当的忽视。经典的 Web 应用曾经几乎完全基于鼠标操作。但是在现代的 AJAX 应用中，我们特别严重地依赖键盘事件的支持。利用这个事件助手，jQuery 提供了为每个匹配元素触发 keydown 事件（按下一个键）的支持。如果使用一个参数，该事件助手将一个函数绑定到每个匹配元素的 keydown 事件。在键盘上按下一个键的不同阶段可以巧妙地区分。通过 keydown 指定按下一个键，对应的 keyup 表示释放按键。这些事件有很大的不同，因为在按下键的时候，对应的字符还没有在键盘缓冲内，因而在此时无法处理，但是在释放按键的时候，通过键盘码可以得到该字符
keypress()	keypress 事件表示按下任何键。如果没有指定参数，为每个匹配元素触发该事件。如果使用一个参数，该事件助手将一个函数绑定到每个匹配元素的 keypress 事件
keyup()	不使用参数，这个事件助手的作用是为每个匹配元素触发 keyup 事件。如果使用一个参数，该事件助手将一个函数绑定到每个匹配元素的 keyup 事件
load()	加载一个元素。这个事件助手只有带一个参数的形式，将一个函数绑定到每个匹配元素的 load 事件

（续）

名　称	描　述
mousedown()	按下鼠标按钮触发这个事件助手。该方法只有带一个参数的形式，将一个函数绑定到每个匹配元素的 mousedown 事件
mouseenter()	这个事件在鼠标指针移到网页某个可见元素所在区域上的时候发生。该方法只有带一个参数的形式，将一个函数绑定到每个匹配元素的 mouseenter 事件
mouseleave()	如果鼠标移出元素区域，触发该事件。这个事件助手只有带一个参数的形式，将一个函数绑定到每个匹配元素的 mouseleave 事件
mousemove()	这个事件在鼠标指针移入网页某个可见元素所在区域的时候发生。该方法只有带一个参数的形式，将一个函数绑定到每个匹配元素的 mousemove 事件
mouseout()	如果鼠标离开元素区域，触发该事件。这个事件助手只有带一个参数的形式，将一个函数绑定到每个匹配元素的 mouseout 事件
mouseover()	如果鼠标指针移入元素所在区域，触发该事件。这个事件助手只有带一个参数的形式，将一个函数绑定到每个匹配元素的 mouseover 事件
mouseup()	这个事件助手只有带一个参数的形式，将一个函数绑定到每个匹配元素的 mouseup 事件。这个事件在按下的鼠标按钮释放时发生
resize()	这个事件在改变一个元素大小的时候发生。这个事件助手只有带一个参数的形式，将一个函数绑定到每个匹配元素的 resize 事件
scroll()	这个事件助手只有带一个参数的形式，如果用户在某个元素区域内滚动，则将一个函数绑定到每个匹配元素的 scroll 事件
select()	如果一个元素被选中，该方法启动每个匹配元素的 select 事件。如果使用一个参数，该事件助手将一个函数绑定到每个匹配元素的 select 事件
submit()	如果不使用参数，触发每个匹配元素的 submit 事件。当然，这只与表单有关。如果使用一个参数，该事件助手将一个函数绑定到每个匹配元素的 submit 事件
unload()	这个事件助手只有带一个参数的形式，将一个函数绑定到每个匹配元素的 unload 事件

我对单独事件助手的以上解释已经足够了[⊖]，但是我们希望包含方法的更多细节，当然还要通过例子阐述其基本应用。对于 jQuery 提供的某些事件助手方法，可以用一个函数作为参数，也可以不使用参数。其他方法则强制要求一个函数作为参数。如果调用这些方法时没有使用参数，会指定一个元素的默认事件。这个事件会被触发，例如，调用一个合适的触发器。

程序清单 8.11　图像的 click()

```
$("img").click();
```

如果传递一个函数引用或者回调函数作为方法的参数，指定的函数在每个匹配元素发生事件时执行。

程序清单 8.12　在单击一个按钮时执行一个匿名函数

```
$("button:last").click(function () {
  $("img").triggerHandler('dblclick');
});
```

⊖ 以上的解释按照重要性从大到小排列。

回调函数可以拥有一个可选的默认参数。这可能是 jQuery 事件助手概念中最重要的事实（因为这个参数代表了所有浏览器中的 jQuery 事件对象，可以按照前面的例子那样处理，使用标准化的属性和方法名称——即使在 Internet Explorer 中也不例外）。参数的名称当然是任意的。

程序清单 8.13 在处理事件对象时执行一个匿名函数

```
$("button:last").click(function (ev) {
  $("#output").html(ev.pageX);
});
```

8.5 扩展的事件处理方法

在 jQuery 中，有些特殊技术使我们能够用合适的、安全的、特殊而可靠的方法响应事件。这些方法基于我们在前面的事件处理一般性讨论中提到的技术。

8.5.1 bind() 和 unbind() 方法

bind() 方法使用两个或者三个参数，并返回一个 jQuery 类型的对象，这和 jQuery 中几乎所有方法一样。利用这个方法，我们拥有了描述响应机制的最灵活选项之一。我们将一个处理器绑定到每个匹配元素的一个或者多个事件。目前，可以指定作为第一个参数的可能事件（多个事件简单地用空格分隔）是 blur、focus、focusin、focusout、load、resize、scroll、unload、click、dblclick、mousedown、mouseup、mousemove、mouseover、mouseout、mouseenter、mouseleave、change、select、submit、keydown、keypress、keyup、error。

但是也可以使用自己定义的事件。

在大部分情况下，将把自己的事件处理器定义为匿名函数，作为方法的最后一个参数传递。

程序清单 8.14 将一个匿名函数绑定到任何段落的点击事件上

```
$("p").bind("click", function(event){
  $("#output").text("... and there was a click!");
});
```

如果这不可能，或者仍然需要将数值传递给回调函数，可以 event.data 的格式指定附加数据为第二个参数，指定处理器函数为第三个参数。

程序清单 8.15 将一个命名函数绑定到点击事件，并指定附加数据作为第二个参数

```
function myHandler(event) {
  $("#output").text(event.data.p1);
}
$("p").bind("click", {p1: "... and there was a click!"},
myHandler)
```

unbind() 是 bind() 的对应方法，完成与之相反的工作。通过这个方法，可以从每个匹配元素中删除绑定的事件。如果没有指定参数，则解除所有事件的绑定。否则，可用将要解除绑定的事件作为参数，多个参数用空格分隔。

程序清单 8.16 展示了一个完整的例子（ch8_3.html）。

程序清单 8.16 绑定的事件处理

```
...
10    <script type="text/javascript"
11        src="lib/ch8_3_ready.js"></script>
12  </head>
13  <body>
14    <h1>Binding events</h1>
15    <img src="images/pic5.jpg" /><hr/>
16    <button>Unbind</button>
17  </body>
18 </html>
```

上述网页包含一幅图像，我们用它来说明绑定的事件处理。通过按钮，我们解除事件绑定。在引用的 CSS 文件中（在本书中没有明确列出），会发现一个类，我们通过该类，在用户单击图像时为其加上一个外框。如果再次单击，该类去掉图像的外框。我们使用 jQuery 的 toggleClass() 方法来实现。程序清单 8.17 中展示了 JavaScript 文件 ch8_3_ready.js。原始网页如图 8-5 所示。

在这个例子中，如果触发 click 事件图像周围的框在 bind() 方法中动态修改。如图 8-6 所示。

程序清单 8.17 应用 bind() 和 unbind()

```
01 $(function(){
02  $("img:first").bind("click", function(){
03    $(this).toggleClass("frame");
04  })
05  $("button:first").click(function(){
06    $("img:first").unbind("click")
07  });
08 });
```

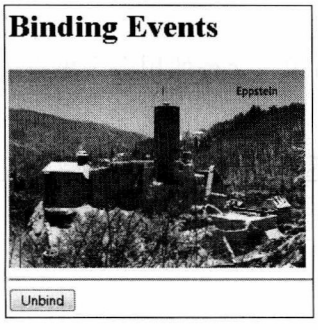

图 8-5 加载之后网页的原始版本

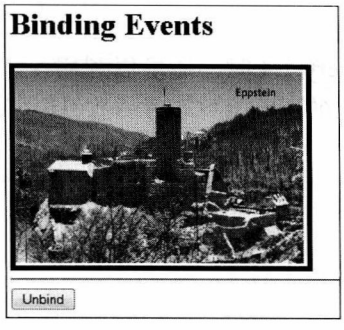

图 8-6 绑定的事件被触发

单击按钮接触单击事件和图像的绑定。后续在图像上的任何点击操作都不再切换类。

bind() 和 bubble：默认操作

处理事件对象时，需要考虑特定元素（例如超链接或者表单的提交）的冒泡阶段和默认操作，以及特定元素的自定义全局默认操作。如果不想执行默认操作，可以用 bind() 阻止默认操作的当前执行以及向父元素冒泡。要达到这种效果，只需在助手函数中返回 false。

程序清单 8.18 阻止表单数据提交

```
$("form").bind("submit", function() {
  return false;
});
```

如果只想阻止默认操作，可以在回调函数中使用 event.preventDefault() 方法。

程序清单 8.19 默认操作被阻止，但是冒泡继续

```
$("form").bind("submit", function(event){
  event.preventDefault();
  ...
});
```

相形之下，纯粹的冒泡可以通过调用 event.stopPropagation() 方法阻止（还因为 Internet Explorer 造成的问题）。

提示

jQuery 甚至支持命名空间。这使你可以组成绑定处理器组，而不需要直接引用它们。

bind() 和多个事件

还可以将多个事件作为参数传递给 bind() 方法。只需在字符串中用空格分隔多个事件助手。

程序清单 8.20 对多个事件的响应

```
$('div:first').bind('mouseenter mouseleave', function() {
  $(this).toggleClass('myClass');
 });
```

到 jQuery 1.4 时，还可以利用一个 Map 类型的对象，交替使用多个事件助手，然后，将一个独立的回调函数或者函数引用绑定到每个事件。

程序清单 8.21 使用 Map 类型的对象

```
$('div:first').bind({
  click: function() {
    // ...
  },
  mouseenter: function() {
    // ...
```

```
  },
  mouseleave: function() {
    // ...
  }
});
```

传递数据

在版本 1.4.3 时，bind() 还有如下的变种。

程序清单 8.22 传递数据

```
bind( eventType, [ eventData ], handler )
```

第二个参数是一个类型为 Map 的数据对象，可以传递给处理器。这些数据成为事件对象 data 特性的一个属性。

程序清单 8.23 将数据传递给函数

```
$("button:first").bind("click", {name: "Felix"},
  function (event) {
    alert(event.data.name);  // Output Felix
});
```

8.5.2 仅有的一个：one()

bind() 有一个直接的替代品：one()。利用这个方法，可以将一个处理器绑定到每个匹配元素的一个或者多个事件。这和我们刚刚看到的 bind() 一样。one() 方法的特殊功能从它的名称就可以看出来：绑定的处理器对每个元素都只执行一次，然后就被解除绑定。除此之外，该方法的表现和 bind() 完全相同。所以，我们没有必要提供例子。

8.5.3 trigger() 方法

对于 trigger() 方法，传递想要“手工”触发的事件[⊖]及所需数据作为参数。在 jQuery 中使用触发器链接到浏览器对在其他地方指定的特定事件或者浏览器自动触发的事件的默认响应。如果调用 trigger() 方法，这个调用会启动浏览器的同名默认响应（如果存在的话）。例如，如果在 trigger() 中传递“ submit”，这将迫使浏览器发送表单数据（如果你的触发器在表单中）[⊖]。

提示

触发事件并不限于基于浏览器的事件，也可以使用改编的事件。但是要记住，jQuery 的不同版本对触发器的实现进行了修改。所以，如果在过去的版本上修改，一定要查看文档，看看是否有变化（这种变化大部分发生在非常特殊的细节上）。

⊖ 例如，由于数据变化。

⊖ 除非你用某个返回值为 false 的绑定函数阻止触发。

元素的事件处理器在触发器启动的时候获得一个规范化的事件对象，没有任何浏览器相关特性（如 keyCode、PageX 或者 PageY）。

对于触发器，可以用三种方式指定事件的类型：

- 可以用字符串方式指定事件名称。
- 可以使用一个 jQuery.Event 类型的对象。如果这样做，可以向这个对象传递数据，这个数据将会送达触发的处理器。
- 可以传递带有数据的字面对象。这个对象被复制到一个类型为 jQuery.Event 的实际对象。在这种情况下，需要指定一个类型特性。

程序清单 8.24 事件处理触发器（ch8_4.html）

```
...
10    <script type="text/javascript"
11        src="lib/ch8_4_ready.js"></script>
12  </head>
13   <body>
14     <h1>Trigger</h1>
15    <h2>Click an image to enlarge.
16       Double-click to shrink the image.</h2>
17    <img src="images/pic3.jpg" />
18     <img src="images/pic4.jpg" /><hr/>
19    <button>Enlarge all images</button>
20    <button>Shrink all images</button>
21  </body>
22 </html>
```

上述网页有两幅图像，我们通过触发器启动一个事件助手——这就是按钮的用途。程序清单 8.25 展示了 JavaScript 文件 ch8_4_ready.js。

程序清单 8.25 单击和双击处理器

```
01 $(function(){
02  $("img").click(function(){
03    $(this).css({
04      width: "400px"
05    });
06  });
07  $("img").dblclick(function(){
08    $(this).css({
09      width: "80px"
10    });
11  });
12  $("button:first").click(function(){
13    $("img").trigger('click');
14  });
15  $("button:last").click(function(){
16    $("img").trigger('dblclick');
17  });
18 });
```

在上述例子中，可以看到两幅图像通过方法 css() 被缩放。在第 2 ～ 6 行中，指定了一个图像单击默认事件。每当用户单击一幅图像，它的宽度通过 css() 方法被扩大到 400 个像素。相应地，在第 7 ～ 11 行中双击图像可以缩小它。如图 8-7 所示。

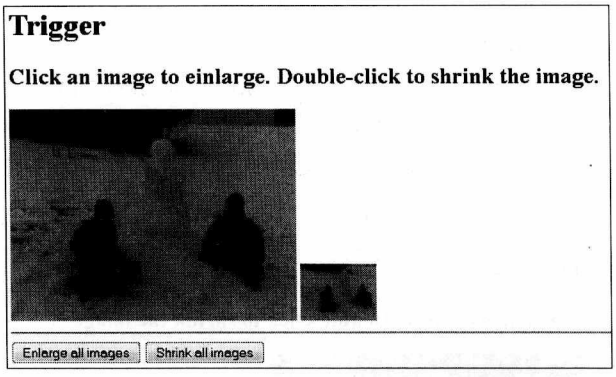

图 8-7 只有右边的图像被缩小

到目前为止仍然没有涉及触发器。但是在单击按钮时，我们使用了触发器。如果用户单击第一个按钮，所有 img 类型的元素的单击触发器都在第 13 行中被触发。

对于所有图像，单击这个按钮都会启动绑定到相应图像单击事件的默认事件。换句话说，如果单击这个按钮，网页中的两个图像都会被扩大。这就是我们在默认事件中所规定的。

相应地，第 16 行中的触发器启动了指定给每个图像双击事件的操作。图像宽度被设置为 80 个像素。这意味着，用户将通过单击第二个按钮缩小所有图像。如图 8-8 所示。

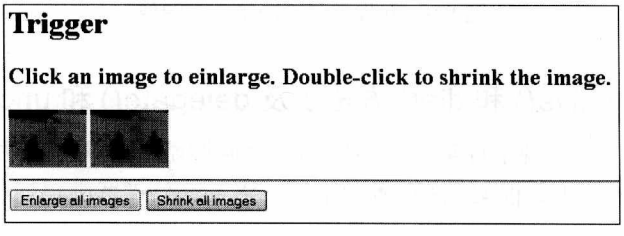

图 8-8 第二个按钮启动 dblclick 的触发器

提示

如前所述，因为 jQuery 支持事件的命名空间，可以创建绑定处理器组，将它们用于触发器。

8.5.4 triggerHandler()

triggerHandler() 方法与 trigger() 方法非常相似，它也执行元素在某个事件类型上连接的所有事件处理器。和 trigger() 方法最重要的区别在于，triggerHandler() 方法既不执行浏览器

的默认操作，也不通过冒泡阶段将事件对象传递给父元素，也不执行任何实时事件（参见下一小节）。

第二个（也是根本性的）不同之处是，只有 jQuery 集合中的第一个元素的事件被触发。这个方法返回一个触发处理器的值而不是一个可链接的 jQuery 对象。所以，如果 jQuery 集合为空，该方法返回 undefined 值。

举个例子，我们来看看 ch8_4.html 的一个变种，在这里我们只用 triggerHandler() 代替 trigger()。可以在本书的配套网站上找到 ch8_5.html 及 CSS/JavaScript 文件的清单。如果用户单击两个按钮中的一个，操作将只影响第一个图像（正如前面所描述的，因为只触发 jQuery 集合中的第一个元素）。所以，只有第一个图像被缩小或者扩大。如图 8-9 所示。

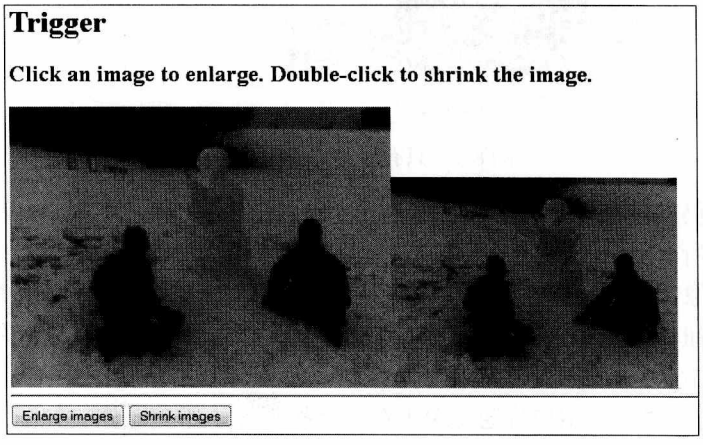

图 8-9 触发器只影响第一个图像

8.5.5 实时事件：live() 和 die() 方法以及 delegate() 和 undelegate()

live() 方法需要一个表现事件类型的参数和一个回调函数。这种方法在 jQuery 中被称作实时事件（live event），于该框架 1.3 版本中推出，将一个处理器绑定到所有当前以及未来匹配元素的某个事件。

在 jQuery 框架的最后几个版本中，可能的事件值改变了数次，所以最好是检查你所使用的版本中特定事件是否可用。例如，在 jQuery 1.3 版本中只支持如下事件：click、dblclick、mousedown、mouseup、mousemove、mouseover、mouseout、keydown、keypress 和 keyup。

在 jQuery 1.4.x 的不同版本中，被遗漏的所有需要经历冒泡阶段的 JavaScript 事件（如 mouseenter 或者 mouseleave）陆续添加进来，然后是 blur 和 foucus，在某个阶段上又添加了 hover。这样，几乎所有事件现在都得到了支持，也可以定义自定义事件。live() 方法的工作方式和 bind() 方法基本上相同，但是也有一些差异：

- 如果通过 live() 方法将某个事件绑定到一个元素，响应不仅在当前 DOM 中的所有元

素中可用，而且在未来集成到 DOM 中的同类元素中也可用。例如，假设有一个具有确定数量项目的列表，并使用 live() 将响应连接到某个项目的单击事件上。这个响应就会自动分配给后续动态加入该列表的所有项目中。[⊖]这与使用 bind() 时的情况不同，那种情况下必须明确地重新链接所有新添加的元素。

- 实时事件不会按照传统的方式冒泡，也无法通过调用 event.stopPropagation() 或者 event.stopImmediatePropagation() 停止。要阻止实时事件的进一步处理，回调函数必须返回 false 值。
- 实时事件只对 jQuery 旧版本的选择器有效。在版本 1.4.1 的时候，也可以指定多个选择器。

我们来看一个例子，例中一个响应函数通过 bind() 绑定到一幅图像，第二个响应函数通过 live() 绑定。然后，我们动态地在网页中创建多幅新图像，看看这些新图像可以使用两个响应函数中的哪一个（ch8_6.html）。

程序清单 8.26 事件处理所用的实时事件

```
...
10    <script type="text/javascript"
11        src="lib/ch8_6_ready.js"></script>
12  </head>
13   <body>
14    <h1>Live Events</h1>
15    <button>New image</button><hr />
16      <img src="images/pic1.jpg" />
17  </body>
18 </html>
```

上述网页包含一幅图像和一个按钮。我们希望图像响应三个事件：

- 用户的单击：这个响应通过 bind() 实现。
- 鼠标移到图像区域之上：这个响应通过 live() 实现。
- 鼠标指针退出图像区域：这个响应也通过 live() 实现。

程序清单 8.27 展示了具体的做法（ch8_6_ready.js）。

程序清单 8.27 使用触发器

```
01 var imageno = 2;
02 $(function(){
03  $("img").bind("click", function(){
04    $(this).toggleClass("frame");
05  });
06  $("img").live("mouseover mouseout", function(){
07    $(this).toggleClass("gross");
08  });
```

⊖ 这涉及被称作委托（delegation）的概念。你可能从 Java 等强大的编程技术中了解过它。这种概念在 jQuery 中也存在，jQuery 中有一个具备相应名称的方法，我们很快就会学到。

```
09  $("button:first").click(function(){
10    if (imageno < 6)
11      $("body").append($("<img />").attr("src",
12          "images/b" + imageno++ + ".jpg"));
13  });
14 });
```

我们通过改变 CSS 类展示响应。一个类在图像周围放置一个框；其他类则改变图像的大小。如图 8-10 所示。

图 8-10 加载后的网页

通过 bind() 方法，对用户单击的响应被链接到 DOM 中此时列出的所有类型为 img 的元素上。

通过 live()，一个切换类的匿名函数被链接到图像区域的进入和退出事件上。[⊖]有趣之处在于，这个响应对于后续动态添加的 img 类型元素也可用。如图 8-11 所示。

通过一个按钮，用户可以在必要时添加其他图像。[⊜]jQuery append() 方法用于这一目的。用户可以添加最多 4 幅新图像。如果鼠标指针移到图像上然后移开，你会看到图像的大小改变。但是对用户单击的响应对于后面添加的图像不可用。如图 8-12 所示。

图 8-11 单击和改变大小之后

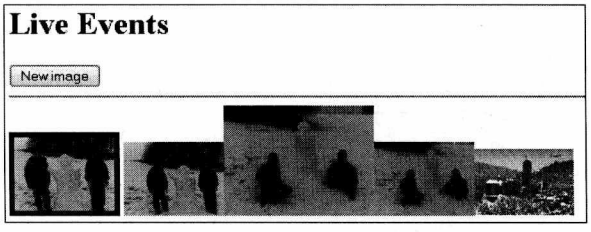

图 8-12 即使输入字段在以后被添加，响应函数仍然可用；第 3 幅图像被增大

delegate()

前面已经提到过，委托（Delegate）技术可以在 Java 等强大的语言概念中找到。在 jQuery 中，可以简单地将 delegate() 方法看作 live() 方法的替代品，用于将一个函数的执行绑定到一个事件上；即使对于以后动态添加的元素也有效。

程序清单 8.28 用 delegate() 代替 live()

```
$("table").delegate("td", "hover", function(){
        $(this).toggleClass("myClass");
 });
```

⊖ 注意，我们在这里明确地指定两个选择器。

⊜ 在我们的特殊例子中，这可以通过一个算法选择图像名称来完成。

die() 和 undelegate()

为了从事件系统中删除一个实时事件，应该使用 die() 方法。这个方法在 jQuery 1.3 中推出，作为 live() 方法的同类方法，它简单地删除已经绑定的实时事件。它有一个可选的参数，可以用它指定类型和回调函数。如果不带参数，所有绑定的实时事件都被删除。如果指定了类型，指定类型的所有实时事件被删除。如果指定一个函数作为第二个参数，只有指定的事件处理器被删除。如果使用 delegate() 代替 live()，可以相应地用 undelegate() 解除绑定。

8.5.6 交互辅助函数

jQuery 框架提供两个有趣的辅助函数，支持常见的交互技术。

hover() 函数

从名称里可以看出，hover() 函数模拟通常的悬停效果，启动鼠标指针到达元素区域以及退出该区域的响应。可以指定两个函数引用作为参数。这样就有了更多选择，而不只是修改元素的 CSS 属性（当然，在这方面也可以做得很好）。

每当鼠标指针移到匹配的元素之上，启动指定的第一个函数。如果鼠标指针离开该区域，启动第二个指定函数（ch8_7.html）。

程序清单 8.29 悬停效果

```
...
10      <script type="text/javascript"
11          src="lib/ch8_7_ready.js"></script>
12    </head>
13    <body>
14      <h1>Hover</h1>
15      <img src="images/pic1.jpg" alt="The snowman"/>
16      <img src="images/pic2.jpg" alt="and"/>
17      <img src="images/pic3.jpg" alt=
18        "the proud builders"/>
19      <div id="output"></div>
20    </body>
21 </html>
```

在上述例子中，可以看到 3 幅图像，指定了 HTML 的 alt 特性。很自然，我们可以通过 jQuery 对象访问这个特性（ch8_7_ready.js）。

程序清单 8.30 应用 hover() 方法

```
01 $(function(){
02   $("img").hover(function(){
03     $("#output").text(this.alt);
04   }, function(){
05     $("#output").text("");
```

```
06  });
07 });
```

通过 hover() 方法，我们得到这个特性的值，并将其写入网页的输入区域。指定为第一个参数的函数完成的就是这个工作。this 表示当前鼠标指针下方的图像。第 4 ～ 6 行中的第二个函数只是将输出区域清空。如图 8-13 所示。

图 8-13　通过 hover() 输出一条消息

toggle()

toggle() 方法的用法和 hover() 几乎完全相同。它响应单击并随着每次点击在两个或者更多函数之间切换，这些函数作为该方法的参数。如果单击了一个匹配元素，第一个指定的函数被启动。第二次单击同一个元素则启动第二个函数。如果还有其他的函数引用，这一过程相应地继续下去，然后回到第一个函数。所有后续的点击都导致在指定的函数调用中循环。该方法和 hover() 很类似，所以我们不提供完整的例子。

提示

除了以函数引用为参数之外，toggle() 方法也可以在没有参数的情况下使用。在这种情况下，每个匹配元素的可见性被切换为开或者关，这涉及 hide() 和 show() 方法的间接调用。也可以用一个布尔值作为该方法的参数。根据这个参数值，所有元素显示（true）或者隐藏（false）。如果你知道在 JavaScript 中数值 0 代表 true，还可以用数字算法设置这个参数。

toggle() 方法的另一个变种是将第一个参数作为速度，第二个可选参数是一个函数引用或者匿名函数。我们来看看这一变种的一个完整示例（ch8_8.html）。

程序清单 8.31　悬停效果

```
...
10    <script type="text/javascript"
11        src="lib/ch8_8_ready.js"></script>
12  </head>
13   <body>
14     <h1>Toggle</h1>
15     <button>Toggle DIVs</button><hr />
16     <div>1</div><div>2</div><div>3</div>
```

```
    <div>4</div><div>5</div><div>6</div>
  </body>
</html>
```

在上述例子中，可以看到6个div区域，我们用CSS进行格式化（ch8_8.css）。如图8-14所示。

程序清单 8.32　div 元素格式

```
div {
  width: 80px;
  height: 40px;
  margin: 5px;
  padding: 5px;
  float: left;
  background: red;
  border: 10px outset;
  cursor: pointer;
  text-align: center;
  font-size: 22px;
}
```

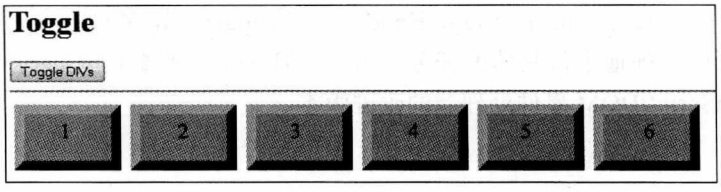

图 8-14　6个div元素都有相同的格式

可以自然地通过一个过滤器访问这些div区域。但是我们进入每个div容器，使这些区域以动画的方式消失（ch8_8_ready.js）。如图8-15所示。

程序清单 8.33　应用 hover() 方法

```
$(function(){
  $("button:first").click(function(){
    $("div:odd").toggle("slow");
  });
});
```

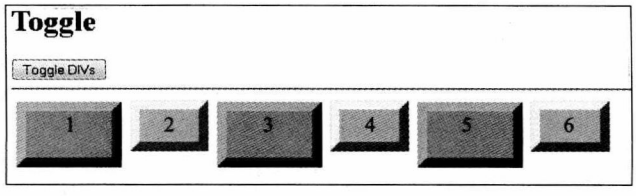

图 8-15　每组中的第二个div区域以动画形式消失

对于速度值，可以使用 jQuery 的默认标志，如“slow”或者指定以毫秒表示的时间间隔。参见第 3 行。按钮上的单击操作按照指定的事件间隔，以动画形式切换可见性。如图 8-16 所示。

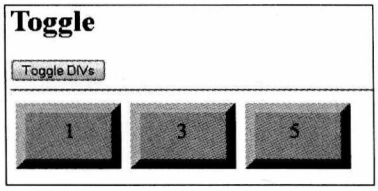

图 8-16 现在只有索引为偶数的 div 可见

8.6 小结

在本章中，我们学习了事件处理的基本知识，并了解了 jQuery 对事件处理的简化以及在不同浏览器上的标准化工作。现在，我们熟悉了事件处理器、冒泡阶段、数据绑定和触发器的一般事实。jQuery 中的事件对象为全部事件处理的使用提供了基础，解决了所有的问题。bind()、live()、die()、one()、triggerHandler() 和 trigger() 等多种 jQuery 方法提供了一个解决方案，可以精确地定位具体的任务。最后（但并非不重要），jQuery 中的特定事件处理器提供了 JavaScript/DOM 事件处理器的改进版本。

第 9 章

特效与动画

本章讨论 jQuery 为 Web 应用程序提供的特效和动画。本质上，动画基于一段规定时间内，网页的一个部分发生的均匀变化或者遵循某个算法产生的变化。这意味着，所有创建动画的 jQuery 技术都是与时间相关的。然而，仅仅为了创建特效，也可以使用某些与时间无关的方法。特效（例如，显示一个元素）在调用该方法时会立即显示。在本书的学习过程中，我们已经遇到了各种特效和动画。

9.1 基本用法

jQuery 动画和特效给人们留下了深刻的印象（特别是对于门外汉来说），但是，在理解了 jQuery 的基本功能之后，它们就只能属于简单技术的类别了。所以，我们只需要简单地研究各种方法，没有必要提供过多的例子。为此，本章是本书中最短的章节之一，这和你的期望可能相悖，至少从我的角度看，这也是本书中最简单的章节。

9.1.1 你所需要的就是速度

本质上，jQuery 动画技术基于网页组件的样式表事件按照时间的控制变化。对于事件无关的特效，这些变化立即发生，但是层叠样式表（CSS）属性仍然以针对性的方式改变。从这一点出发，可以轻松地通过反复操纵 style 对象，手工编写出 jQuery 框架中的所有动画技术。但是，jQuery 方法显著地简化了所有任务，封装了底层处理。和往常一样，jQuery 技术与所有浏览器兼容。

如果想使用动画特效，通常必须指定这些动画的时长。在 jQuery 中，有 3 个默认标志（字符串）可用于这一目的，这些预定义的时间值可在必要时用于任何这类时间规格上（通常作为方法的参数）：

- “slow”
- “normal”

- “fast”

在每种情况下，指定这些动画速度的效果都很明显。然而，因为这些标志所代表的毫秒数在jQuery中预先定义，取决于底层平台，在使用默认标志时对这个时长没有直接的控制。只能接受预定义的行为。

作为替代，可以直接为动画指定毫秒数，从而自己设置动画的时长！但是这一设置并不完全精确。

提示

动画方法的默认时长通常是normal。

动画速度：jQuery.fx.interval

到jQuery 1.4.3时，可以影响动画事件启动的速度（以毫秒表示）。这意味着，可以指定每秒帧数。默认值是13毫秒。如果想要使用其他值，也可以这么做；不一定要使用默认值。较低的值使动画在强大的平台和适合的浏览器上运行得更顺畅，因为每个时间单位中的单独步骤数量更高，使得发生的增量变化更少。但是较高的值占用更多资源，可能对低性能平台和浏览器造成负面的影响。

警告

这个速度不等于动画应该运行的时长。

9.1.2 指定一个回调

jQuery中的几乎所有动画和特效都允许指定一个回调函数或者匿名函数。这个函数在相关的动画结束时执行，如程序清单9.1所示。

程序清单9.1 指定一个在动画结束之后执行的匿名函数

```
$('img:first').fadeIn('slow', function() {
   alert("Done");
 });
```

9.1.3 链接

对于动画，可以通过句点标记法链接它们，当然，这是因为动画和特效方法返回一个jQuery对象，如程序清单9.2所示。

程序清单9.2 通过句点标记法链接

```
$("img").fadeOut('slow').fadeIn('fast').toggleClass('cl');
```

很明显，如果动画和特效没有通过句点标记法连接，而是简单地应用到相同的对象，在其中一个动画和特效运行时并行调用，它们也会被链接起来，如程序清单9.3所示。

程序清单 9.3 不使用句点标记法的链接

```
$("button:first") {
  $("img").fadeOut('slow');
 }
 $("button:eq(1)") {
  $("img").fadeIn('fast');
 }
 $("button:eq(2)") {
  $("img").toggleClass('c1');
 }
```

现在，如果用户在某个动画仍在运行中时单击不同的按钮，当前动画不会被停止；相反，新的动画进入队列中（就像通过句点标记法进行链接一样），在以后处理。

9.1.4 队列

正如前面所讨论的[⊖]，jQuery 中被延迟的函数调用一般在队列中进行管理。通常不需要担心它们的处理和管理。但是，前面介绍的 queue()、clearQueue()、dequeue() 和 delay() 方法使你能够在必要时进行人工干预。可以用 queue() 方法获得对队列中第一个元素（第一个函数）的引用，而用 dequeue() 则可以从队列的开头删除一个函数并执行之。在合适的时候，可以用 delay() 延迟调用。还可以用 clearQueue() 清除队列。但是由于 jQuery 非常强大且容易配置的标准方法，需要使用这些方法的时候很少。特别是，animate() 方法已经使我们能够通过不同选项控制队列。

9.1.5 通过 stop() 和 jQuery.fx.off 停止

如果希望动画停止，可以调用 stop() 方法，如程序清单 9.4 所示。

程序清单 9.4 停止动画

```
$("button:first") {
  $("img").fadeOut('slow');
}
$("button:eq(1)") {
  $("img").stop();
}
```

利用两个可选的布尔参数，可以影响所有链接的后续动画是否应该停止（第一个参数，默认值为 false）以及当前动画是否应该立刻停止[⊖]（第二个参数，默认值为 false）。

作为替代，可以使用 jQuery.fx.off 的值禁用所有动画。将 jQuery.fx.off 的值设置为 true 是一个全局设置，可以禁用所有的动画。这导致网页上的所有元素立即被设置为其最终状态。例如，如果平台太弱或者动画元素出现访问问题时，可以这样做。如果将 jQuery.fx.off

⊖ 所以我们在这里只给出了一个简单的小结，作为提醒。

⊖ 这样，应用动画或者特效的目标值，或者将所有 CSS 值设置回原始状态（根据情况）。

的值设置为 false，动画再次被启用。

9.1.6 永不停止的动画

用 jQuery 创建不停运行[⊖]的动画极其简单，这归功于调用可以链接这一事实。只需要在回调中使用递归调用。切换效果和动画特别适合于这一用途，但是如果用全局变量或者参数控制必要的逻辑，就可以轻松地使用其他特效，如程序清单 9.5 所示。

程序清单 9.5 不停运行的递归动画

```
function ani(){
  $("div.k1").slideToggle(5000,ani);
}
```

9.1.7 动画的类型

利用不同的 jQuery 动画方法，还可以在框架的较新版本中指定一个缓动函数。[⊜]缓动函数定义随时间推移而应用的特效数量。例如，可以指定一个线性特效，或者在结束时降低速度的特效，或者在开始 / 结束时加速的特效，或者摇摆特效。下面是一些当前有效的特效值：

- easeInBack
- easeInBounce
- easeInCirc
- easeInOutBounce
- easeInOutCirc
- easeInOutQuint
- easeInOutSine
- easeInSine
- easeOutBounce
- easeOutCirc
- easeOutQuint
- easeOutSine
- linear
- swing

注意

如果打算应用特定的特效，查看 jQuery 文档，找到这些特效工作方式的附加信息。（文档描述了用于动画技术的常见标准算法）

⊖ 当然，你也可以使用一个计数器限制重复次数。

⊜ 在 jQuery 1.3 中是一个扩展 / 插件，在 1.4.3 中成为标准。

9.2 显示和隐藏：show() 和 hide() 方法

网页元素的时控显隐是基本的动画技术，可以用 jQuery 轻松地做到。

不带参数的 show() 方法直接显示所有匹配元素（如果它们被隐藏）。如果它们已经可见，调用这个方法没有改变任何东西。作为替代，也可以使用带有参数的方法，几乎所有这类方法都是如此。在这种情况下，第一个参数指定效果发生的速度；最后一个可选参数是一个回调函数。到 jQuery 1.4.3，可以使用第二个可选参数——缓动——来影响动画的变化速度。相应地，可以使用 hide() 方法隐藏可见元素。如果元素已经不可见，调用这个方法没有任何改变。

9.3 滑动特效：slideDown()、slideUp() 和 slideToggle()

利用 slideDown()、slideUp() 和 slideToggle() 这三个方法，可以通过元素高度的时控变化来实现显隐元素的滑动效果。可以和 show() 和 hide() 相同的方式使用参数。

这种特效是元素变大（slideDown()）或者变小（slideUp()）。相应地，slideDown() 表示元素的动画显示，slideUp() 则代表隐藏。

这类开发自顶向下或者自底向上进行取决于显示或者隐藏相关的元素类型或者格式。例如，对于 div 元素，动画中通常只有高度（而非宽度）改变。但是，这些方法从 jQuery 以后已经进行了扩展，垂直的填充值和垂直的边距值也可以改变，以创建更为流畅的动画特效。对于图形，宽度在默认设置中也可以动态变化，除非图像的宽度已经被规定（例如，通过样式表属性 width）。注意，对于 div 元素，是否指定这个属性没有关系。正如你所看到的，这种行为有一些困难，但是从本质上说，动画中变化的是一个元素的高度。

顾名思义，第三个方法 slideToggle() 负责根据状态显示或者隐藏元素。对于这三个方法，都可以指定可选的参数——时长、缓动行为和回调函数。

警告

上述方法操纵网页的文档对象模型（DOM）树，在某些情况下，树的原始结构可能被移动。在特定的情况下，会插入新的节点，用于换行。

程序清单 9.6 应用滑动方法（ch9_1.html）

```
...
06    <link rel="stylesheet" type="text/css"
07       href="lib/ch9_1.css" />
08    <script type="text/javascript"
09        src="lib/jquery-1.8.2.min.js"></script>
10    <script type="text/javascript"
11        src="lib/ch9_1_ready.js"></script>
12  </head>
13  <body>
14    <button>Show rectangles</button>
15    <button>Hide rectanqles</button>
```

```
    <button>Show images</button>
    <button>Hide images</button>
    <button>Toggle images and rectangles</button><hr/>
    <div class="c1"></div>
    <div class="c1"></div><hr/>
    <img src="images/pic1.jpg" class="c2" />
    <img src="images/pic2.jpg" />
  </body>
</html>
```

在上面的清单中，可以看到两个矩形和两幅图像。可以用 5 个按钮分别显示和隐藏矩形和图像，也可以同时显示或者隐藏所有对象。

可以看到，我们再次使用一个 CSS 文件（ch9_1.css）。

程序清单 9.7　CSS 文件

```
.c1{
  width: 80px;
  height: 40px;
  margin: 5px;
  background: green;
  border: 5px outset;
  display:inline-block;
 }
 .c2{
      width:300px
}
```

第一个类用于格式化 div 元素，我们用第二个类明确地指定一幅图像的宽度。如图 9-1 所示。

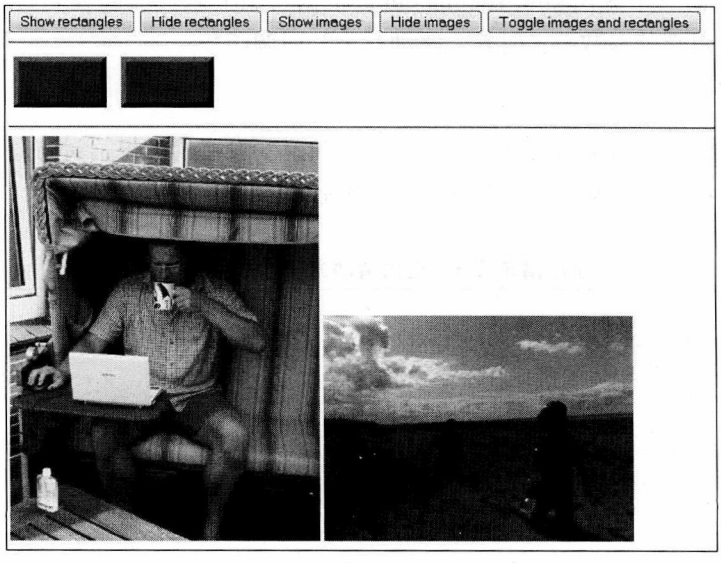

图 9-1　不同的动画对象

程序清单 9.8 展示了我们对方法的应用（ch9_1_ready.js）。

程序清单 9.8 滑动方法

```
01 $(function(){
02  $("button:first").click(function(){
03    $("div.c1").slideDown("slow");
04  });
05  $("button:eq(1)").click(function(){
06    $("div.c1").slideUp("fast");
07  });
08  $("button:eq(2)").click(function(){
09    $("img").slideDown("slow");
10  });
11  $("button:eq(3)").click(function(){
12    $("img").slideUp("slow");
13  });
14  $("button:eq(4)").click(function(){
15    $("div.c1").slideToggle(5000);
16    $("img").slideToggle(10000);
17  });
18 });
```

如果隐藏矩形，可以看到矩形的宽度没有改变。可以尝试，并删除矩形的宽度规格。即使如此，在用动画方法隐藏和显示它们的时候，宽度也将保持不变。如图 9-2 所示。

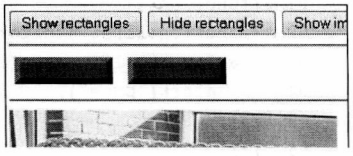

图 9-2 矩形的宽度在动画中变化

但是对于图像，如果宽度已经指定，它起到重要的作用。对于第一幅图像，我们已经规定了宽度，在动画中相应地只有高度变化。对于第二幅图像，我们没有指定宽度，所以在显隐的时候，高度和宽度都动态变化。如图 9-3 所示。

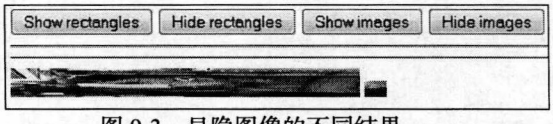

图 9-3 显隐图像的不同结果

9.4 透明度特效：fadeIn()、fadeOut() 和 fadeTo()（加上 toggle()）

利用 fadeIn()、fadeOut() 和 fadeTo() 方法，jQuery 提供了灵活改变元素透明度的选项。这些变化不会将元素从 DOM 中删除。然而，如果不透明度设置为 0，元素不再占据原来指定的网页空间（CSS 属性 display 被设置为 none），这可能导致网站结构的变动。

注意

jQuery 方法对 Internet Explorer 有关 CSS 属性 opacity 的标准偏移做了补偿，在后台用不遵循标准的 filter() 对该浏览器进行操纵。

可以用 fadeIn() 使元素可见，用 fadeOut() 使元素不可见。同样，可以使用与 show() 和 hide() 相同的可选参数。此外，可以为 fadeTo() 指定第二个参数——我们将为元素设置的不透明度目标值（0 ～ 1）。相应地，可选的回调函数将是第三个或者第四个参数（取决于是否指定一个缓动参数）。

程序清单 9.9 应用淡入淡出方法（ch9_2.html）

```
...
10    <script type="text/javascript"
11        src="lib/ch9_2_ready.js"></script>
12  </head>
13  <body>
14     <button>Image 2 at opacity 0.2</button><br />
15    <img src="images/pic3.jpg" />
16    <img src="images/pic4.jpg" />
17  </body>
18 </html>
```

程序清单 9.10 展示了 JavaScript 文件（ch9_2_ready.js）。

程序清单 9.10 动态改变不透明度

```
01 $(function(){
02  $("button:first").click(function(){
03    $("img:eq(1)").fadeTo("slow", 0.2, function(){
04      alert("Done");
05    });
06  });
07 });
```

在上述清单中，我们用前面描述的方法之一改变第二幅图像的不透明度。如图 9-4 所示。

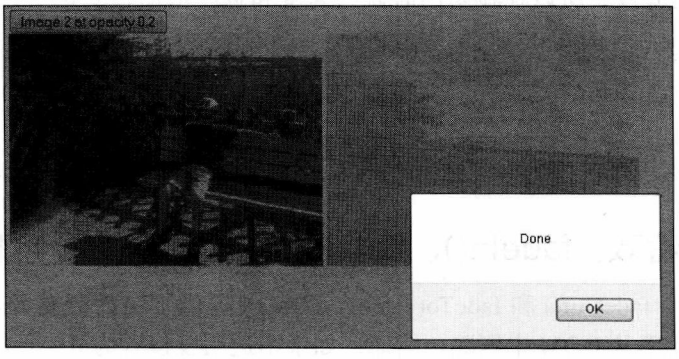

图 9-4 第二幅图像的不透明度目标值已经达到，触发回调函数

toggle() 方法和前三个方法处于同一类别。可以用它简单地切换元素的可见性（不使用参数）。这个方法有两个特殊的变种。使用一个布尔参数，可以指定可见性是应该启用还是禁用。这种做法等价于 show() 和 hide()。jQuery 1.4.3 加入了一种有三个可选参数的变种，

更加有趣。除了常见的时间数据参数和回调函数（第二个可选参数）之外，还指定了不透明度减小和增加的类型。

9.5 用 animate() 实现单独动画

jQuery 中启动特效或者时控动画的默认方法已经很强大且令人印象深刻了。但是，很容易想到，这些方法并不能覆盖你想要实现动画的不同情况。例如，想像一下移动元素位置的情况。

在直接编程中，你可能使用 window.setTimeout() 控制动画时长，并用它动态地改变某些 CSS 属性。你还非常可能通过递归调用优化动画。这类工作在 jQuery 中没有必要，因为它提供了 animate() 方法。这个方法有多个变种，使用不同（可选）的参数。一般的版本如程序清单 9.11 所示。

程序清单 9.11 应用 animate() 的模式

```
animate(style object with properties, [time factor], [easing], [callback])
```

替代的版本之一带有两个参数，如程序清单 9.12 所示。

程序清单 9.12 有两个参数的模式

```
animate(style object with properties, options)
```

对于选项，可以设置表 9-1 所描述的规格。

表 9-1 更加精确规定的 animate() 调用所用的选项

选 项	描 述
duration	通过常用标志“slow”、“normal”、“fast”或者毫秒数指定时长
easing	指定动画结束时的具体特效（字符串形式）。默认为“swing”
complete	指定动画结束时执行的一个函数。该函数对于每个受影响的元素都执行
step	指定一个回调函数，在动画的每步中调用
queue	一个布尔值，默认为 true。这个规格在动画被添加到队列中时起作用。如果该值被设置为 false，动画跳过队列直接开始。这和同时启动多个 window.setTimeout() 调用相同
specialEasing	一个 Map 类型对象，保存 CSS 属性和为相关属性指定的缓动函数

和其他动画方法一样，animate() 方法基于样式表属性可以时控方式改变这一事实。特别地，需要为特定的样式表属性指定一个目标值，对于其他方法（如 fadeTo()），也需要这么做。但是，通过调用 animate() 方法，整个过程变得更加灵活，这是因为实际上可以改变任何样式表属性。以对象的形式指定样式表属性，作为第一个参数，对象的每个键值代表想要在动画中改变的一个样式属性（例如 height、left 或者 opacity）。

指定的属性值表示与某个初始状态相对的目标值。如果属性值是数字，动画使用缓动函数调整数值。如果属性目标值通过字符串（如“hide”、“show”或者“toggle”）指定，

jQuery 自动构造一个默认动画，达到属性的目标状态。但是要记住，所有支持的属性必须用某种形式的数值表示。backgroundColor（或者前景颜色）之类的属性不能以这样的数字形式表示，从而在当前版本的 jQuery 中不能得到支持。[⊖]

但是，在 jQuery 1.2 时，可以使用某些属性的单位（例如，字体大小的 em 或者 %）。从该版本之后还可以使用相对动画。必须在属性值之前加上 JavaScript 标志“ +=”或者“ -=”，从正 / 负方向上相对于当前值做出改变。

jQuery 1.3 也有新功能。如果指定动画时间周期，动画将同步地将所涉及的元素设置为最终状态。

最后一点提示是，当然也可以用这个方法指定动画的时间周期。可以用第二个可选参数按照通常的方法进行（使用毫秒数，或者“ slow”、“ normal”、“ fast”标志）。我们来看看该方法几个变种的实际使用。

程序清单 9.13　使用 animate() 的不同方式（ch9_3.html）

```
...
10    <script type="text/javascript"
11        src="lib/ch9_3_ready.js"></script>
12  </head>
13  <body>
14     <button>Animate DIV 1</button>
15    <button>Animate DIV 2</button>
16    <button>Animate image 1</button>
17    <button>Animate image 2</button>
18    <button>Animate images and DIVs
19    </button><hr/>
20    <div class="d1" id="i1"></div>
21    <div class="d1" id="i2"></div>
22    <img src="images/pic5.jpg" id="i3" class="img1" />
23      <img src="images/pic6.jpg" class="img1" id="i4"/>
24  </body>
25 </html>
```

在上例中，可以看到两个矩形和两幅图像。在引用的 CSS 文件中，我们首先为图像和 div 指定属性。其中，我们指定了高度和位置，加上画框及颜色的数据。这不是非常复杂，但是这些 CSS 属性中的一些在动画中变化（ch9_3.css）。加载后的页面如图 9-5 所示。

程序清单 9.14　初始的 CSS 规格

```
01 .d1 {
02  width: "80px";  height: 40px;
03  margin: 5px;
04  background: green;  border: 5px outset;
05  position: absolute
06 }
```

⊖ 但是，来自 jQuery UI 的扩展方法 animate() 允许的动画甚至包括颜色！

```
.img1 {
 width: 200px;
 position: absolute
}
#i1 {
 width: 200px;
 left: 100px;  top: 50px
}
#i2 {
 width: 200px;
 left: 400px;  top: 50px
}
#i3 {
 width: 200px;
 left: 100px;  top: 150px
}
#i4 {
 width: 200px;
 left: 400px;  top: 150px
}
```

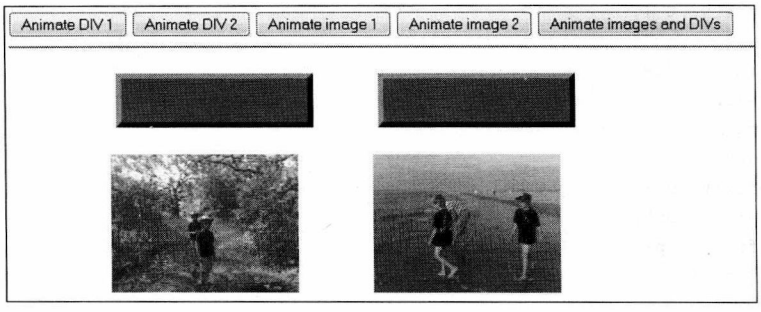

图 9-5 加载后的网页

程序清单 9.15 展示了 JavaScript 文件（ch9_3_ready.js）。

程序清单 9.15 定义自定义动画

```
$(function(){
 $("button:first").click(function(){
   $("div#i1").animate({
     width: "50%",
     opacity: 0.4,
     borderWidth: "1px"
   }, 1500);
 });
 $("button:eq(1)").click(function(){
   $("div#i2").animate({
     width: "-=20",
     opacity: 0.2,
```

```
    borderWidth: "15px"
  }, 1500);
});
$("button:eq(2)").click(function(){
  $("img#i3").animate({
    width: "30%",
    opacity: 0.3,
    top: "200px"
  }, 5000,"swing");
});
$("button:eq(3)").click(function(){
  $("img#i4").animate({
    width: "40%",
    opacity: 0.3,
    top: "+40px",
    left: "-50px"
  }, 1500);
});
$("button:eq(4)").click(function(){
  $("div").animate({
    width: "5%",
    opacity: 0.1,
    top: "100px",
    left: "100px"
  }, "slow");
  $("img").animate({
    width: "5%",
    opacity: 0.3,
    top: "100px",
    left: "100px"
  }, "slow");
});
});
```

利用网页中的5个按钮，可以触发不同的动画。在第一个按钮上单击实现第一个矩形的动画，第二个按钮改变第二幅图像，第三个按钮改变第一幅图像，第四个按钮实现第二幅图像的动画，最后一个按钮实现所有元素的动画。

第一个矩形的宽度和不透明度出现变化。此外，外框的宽度也在动画中变化。第二个矩形的宽度也出现变化。然而，宽度用“-=”标志每次减少20个像素。透明度和外框宽度也再次在动画中变动。

第一幅图像被扩大，透明度减小到30%，位置移到距离顶部200个像素的地方。第二幅图像也移动其位置（水平和垂直方向），并在动画中改变大小和透明度。这里，我们还制定一个缓动因子作为第三参数。如图9-6所示。

单击第4个按钮同时启动两个动画。注意此时指定元素的选择器。特别地，所有元素被放到同一个位置。

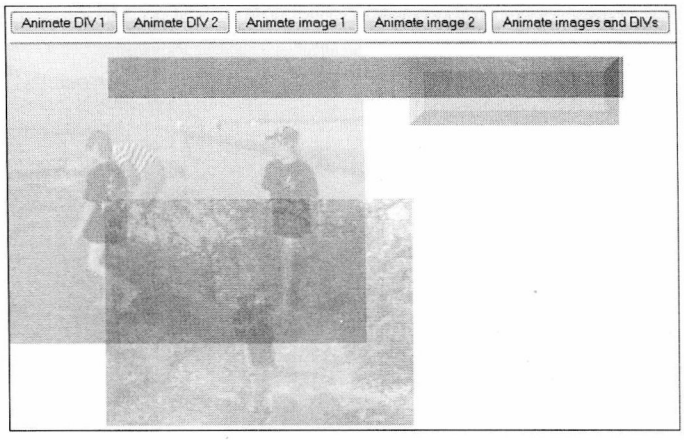

图 9-6 所有元素依次动画

我们再来看看 animate() 第二个变种的一个示例（ch9_4.html）。

程序清单 9.16 animate() 的另一种用法

```
...
10    <script type="text/javascript"
11        src="lib/ch9_4_ready.js"></script>
12  </head>
13  <body>
14     <button>Start animation</button><hr />
15    <img src="images/pic4.jpg" id="i1"/>
16  </body>
17 </html>
```

程序清单 9.17 展示了引用的 CSS 文件（ch9_4.css）。

程序清单 9.17 初始 CSS 规格

```
01 #i1 {
02  width: 200px;
03  position: absolute;
04  left: 100px;
05  top: 150px
06 }
```

JavaScript 文件（ch9_4_ready.js）更加有趣。

程序清单 9.18 链接和控制多个链接动画同时启动

```
01 $(function(){
02  $("button:first").click(function(){
03    $("img#i1").animate({
04      width: "20%",
05      opacity: 0.4
06    }, {
```

```
    duration: 1500,
    queue: false
  }).animate({
    top: "+40px",
    left: "-50px"
  }, {
    duration: 1500,
    queue: false
  }).animate({
    height: "100%"
  }, {
    duration: 5000,
    queue: true
  });
});
});
```

在源文本中，可以看到多个动画被链接起来。所以，它们实际上应该一个接一个运行。但是因为我们指定了 queue:false，3 个链接的 animate() 调用同时启动。如你所见，最后一个方法的 queue 值被设置为 true。后续的 animate() 方法调用将放回到队列中。

9.6 小结

本章介绍了 jQuery 中核心的动画技术和特效。动画本质上基于网页组件的按时变化。在这一领域的 jQuery 方法通常很简单，能够支持这种时间上的操纵。动画和特效的主要方法包括 show()、hide()、slideDown()、slideUp()、slideToggle()、fadeIn()、fadeOut() 和 fadeTo()，以及创建自定义动画的 animate()。

第 10 章
AJAX

2005 年左右，AJAX 给 Web 带来了革新。从此以后，Web 上没有出现过能与之相比的技术。AJAX 就像一颗炸弹，但是事实上它只不过是一个流行语。许多 Web 应用程序开发框架有时会明确地称自己为 AJAX 框架，但是纯粹的 AJAX 功能通常只占其性能范围的一小部分。不过，这一小部分极其重要，因为它组成了现代富互联网应用程序（Rich Internet Application，RIA）的基础。只要想想许多特殊的 Google 功能几乎都基于 AJAX 就不难理解这一点。当然，jQuery 等高端框架也提供简化 AJAX 应用程序创建的所有必要功能。jQuery 中的这些 AJAX 功能是本章的重点。

如果将手工编写的 AJAX 应用程序源代码与使用 jQuery 创建相同功能和质量所需的源代码相比较，你就会看到，代码更为简单，因此质量更高。首先，需要的代码大大减少了。没有必要不断重新开发 AJAX 通信中重复的必要步骤，而可以使用高级的 jQuery 技术。用 jQuery 开发 AJAX 应用程序更为清晰，也更为紧凑。至少，可以避免手工创建 AJAX 应用程序时必须应付的许多复杂性。

10.1 AJAX 和 XMLHttpRequest（XHR）基础

我们首先简单地介绍 AJAX 的基础知识。如前所述，使用 AJAX 来描述确保 Web 应用程序在 Web 服务器请求新数据时都能实时响应的一种方法。这一切的发生不需要在浏览器中重新加载整个网页——只有新请求的数据被加载并插入网页。原来加载的网页保持不变。

具体地说，AJAX 指的只是 Web 上确立已久的技术之间的一种相互作用；这些技术始于 HTML（超文本标记语言）或者 XHTML（可扩展 HTML）和 HTTP（超文本传输协议），通过 JavaScript 和 CSS（层叠样式表）直到 XML（可扩展标记语言）或者 JSON（JavaScript 对象标记法）。即使是网页中异步请求数据的集成也不是新技术，它从 1998 年左右就以稍有不同的方式出现了。只有 AJAX 这个术语是相对新颖的；它于 2005 年左右确立，同时出现的还有时髦的术语“Web 2.0”。为了支持客户端和 Web 服务器之间的这种异步通信，所有现代浏览器都提供了 XMLHttpRequest（作为 JavaScript 对象模型的扩展集成），这是一个控制

客户端编程语言（主要是 JavaScript）中 HTTP 事务的集成接口。这些事务独立地[⊖]取代 Web 浏览器的“常规”数据请求。XHR 或者 XMLHttpRequest 对象直接基于 HTTP 的内部结构，形成了所有 AJAX 请求的骨架。它们以某种形式存在于声称支持 AJAX 的所有框架和工具之中。

为了实现浏览器和 Web 服务器之间的异步通信，XHR 对象允许通过函数引用注册回调函数。这些回调函数评估事务状态的每个变化。此外，还可以通过 XHR 对象访问 AJAX 请求或者响应的所有 HTTP 头标字段。

10.1.1 手工创建一个 XMLHttpRequest 对象

要使用 XHR 对象，必须按照所使用的编程语言的原则创建它。在 JavaScript 中，可以通过合适的构造方法进行。利用 XMLHttpRequest 类型的对象，可以访问 AJAX 异步通信需要的所有相关方法和属性。根据常见的传输方法，[⊜]可以发送数据或者请求信息，而且请求的头标也可以设置和求取。

注意

由于安全原因，通常只使用 AJAX 从请求网页的同一个域请求数据。这常被称作“沙箱原则”。否则，所进行的被称作跨域访问，这可能导致潜在的误用和操纵风险。但是，这种沙箱原则造成了严重的局限性，Web 设计人员和浏览器制造商都致力于寻找解决方案，在不危及安全的情况下消除这些局限性。例如，在 Internet Explorer 8 中，Microsoft 提供了专利的 XDomainRequest 对象，而 Mozilla 和相关的 W3C 兼容浏览器试图对 XHR 对象进行扩展。在 jQuery 中，getScript() 方法提供了一个解决方案，允许从另一个域加载脚本。这个方法的基础使用名为 JSONP 的 JSON 扩展（稍后将作介绍），利用这个扩展，JSON 数据可以从任何地址加载。

各种浏览器的不兼容性迫使我们用不同的方法创建 XHR 对象，以确保这一规程在所有相关浏览器中都能正常工作。基本方法如程序清单 10.1 所示。

程序清单 10.1 用构造程序创建一个 XMLHttpRequest 对象

```
resObject = new XMLHttpRequest();
```

遗憾的是，Internet Explorer 的旧版本无法处理上述语句。它们要求使用一个 ActiveX 控件，甚至使用不同的参数，这取决于浏览器的版本。所以，需要编程自动区分创建通信对象的不同方式。[⊝]利用 jQuery，后台中相对复杂的步骤以在所有浏览器中都兼容的方式封

⊖ 大部分是异步的，如前所述，这是 AJAX 缩略语的一部分。

⊜ 在实践中，通常只使用 POST 或者 GET。其他方法如 PUT 或者 DELETE 从理论上可以使用，但是没有得到所有浏览器的支持，而且很少有需要它们的情况。

⊝ 一般的规则是采用异常处理的概念。你简单地尝试创建 XHR 对象，如果出现错误，则采用替代的方法。通过将尝试集成在一个异常处理中，脚本能够在对象无法生成的情况下稳定运行（jQuery 在后台采用这种方法）。你也可以使用浏览器嗅探，但是该方法不可靠。

装，使这一任务变得非常简单。

10.1.2 XHR 对象方法

XMHttpRequest 类型的对象提供多个方法，我们在此简单地加以讨论。它们形成了理解 jQuery 的 AJAX 方法的基础：

- abort() 方法停止当前浏览器请求。更准确地说，客户端在调用该方法之后不再对来自 Web 服务器的最新响应做出反应。
- 通过 getAllResponseHeaders()，可以字符形式获取所有服务器发送的头标字段。
- getResponseHeader("headerLabel") 返回指定为字符串参数的头标字段值。
- open("method","URL"[,asyncFlag[,"username"[,"password"]]]) 是 AJAX 通信中最重要的方法之一。可以用它指定与 Web 服务器连接的数据。前两个参数是必需的——数据传输方法（GET 或者 POST）和 URL。第三个参数指定请求是同步（false）还是异步（true）处理。如果指定了同步处理，后续通过 send() 方法发送的数据阻塞脚本的执行，直到完全收到服务器的响应。[⊖]对于异步数据请求，则避免了浏览器等待响应的阻塞行为，脚本在请求发送之后继续运行。其他的可选参数是可能需要的用户名和相关密码。
- send(content）方法用于发送请求。它在 open() 方法之后调用。参数 content 可以为 null（对于 GET 方法）或者一个查询字符串（对于 POST 方法）。对于 POST 和 content 参数值，重点在于将客户端用户输入等数据发送到服务器。因为 GET 时参数值为 null，请求数据如何发送到 Web 服务器就成了问题。答案很简单：和往常一样，用 GET! 对于 GET 方法，数据在发送之前简单地附加到数据传输目标（URL）之后，数据和原始 URL 之间用一个问号（？）分隔。[⊜]将数据以“&”标志分隔的名称 - 值配对方式发送给评估脚本或者程序。这组名称 - 值配对作为 URL 字符串的一部分传送，必须由一个 Web 脚本进行拆解。[⊜]
- 可以用 setRequestHeader("label", "value") 设置单独的头标字段。
- 通过 setMimeType("mimetype")，可以设置请求数据的 MIME 类型。但是这个方法在某些浏览器中没有得到支持。

10.1.3 XHR 对象属性

除了方法之外，XHR 对象有一些属性，其功能在 jQuery 中当然也提供：

- 使用 onreadystatechange，可以拥有一个在 XHR 对象的连接状态（readyState）变化时

⊖ 例如，如果多个 AJAX 请求需要顺序运行，后续的 AJAX 请求必须在前一个请求的结果上进行操作时，这就很有必要。

⊜ 用户将在表单发送之后，在浏览器地址窗口中看到使用经典数据请求（而不是 AJAX 请求）的伪 URL。

⊜ 这样的字符串结构大约如下：name1=value1&name2=value2&name3=value3。这个字符串必须由处理脚本或者程序从 & 符号和等号（=）处分成各个部分。

调用的事件处理器。对于这个事件处理器，一般注册一个对回调函数的引用。

- readyState 属性包含当前事务的连接状态。可能值为 0（UNINITIALIZED）、1（LOADING）、2（LOADED）、3（INTERACTIVE）和 4（COMPLETED）。[⊖]
- responseText 属性包含服务器发送的数据（文本形式），responseXML 是服务器发送的 XML 数据（换句话说，可以浏览的一棵树）。如果数据不以 XML 方式发送，responseXML 包含 null 值。
- status 属性包含一个表示连接的 HTTP 状态的数字值，statusText 属性包含相关的文本消息（如果发送的话）。

10.1.4 不使用特殊 jQuery 方法的数据请求实用示例

我们来看看如何在不使用 jQuery 框架的特殊方法[⊖]的情况下实现 AJAX 数据请求（ch10_1.html）。

程序清单 10.2 通过 AJAX 加载数据的网页

```
...
06    <link rel="stylesheet" type="text/css"
07        href="lib/ch10_1.css" />
08    <script type="text/javascript"
09        src="lib/jquery-1.8.2.min.js"></script>
10    <script type="text/javascript"
11        src="lib/ch10_1_ready.js"></script>
12  </head>
13  <body>
14    <button>Request text via AJAX</button>
15    <div></div>
16  </body>
17 </html>
```

注意

在本章的其他示例中，我们通常采用与目前这个例子相同的网页结构作为出发点。所以，为了节约空间，我们将不再将其列出，可以在本书的配套网站上找到详细的程序清单。

单击按钮加载一个文本文件并在 div 区域中显示。这通过如下的简单脚本发生，这个脚本使用了前面提到的 XHR 对象属性和方法（ch10_1_ready.js）。

程序清单 10.3 AJAX 请求

```
01 $(function(){
02  var xhr = new XMLHttpRequest();
```

⊖ 有些来源规定 0 为未初始化，1 为连接建立，2 为请求已接收，3 为应答处理中，4 为结束。两者等价。

⊖ 但是，我们已经使用 jQuery 访问网页的元素。然而，我们也可以使用传统的 DOM 方法，而不使用任何 jQuery。

```
03  $("button:first").click(function(event){
04    xhr.open("get", "ajax.txt", true);
05    xhr.send(null);
06    xhr.onreadystatechange = function(){
07      if (xhr.readyState == 4)
08        $("div:first").html(xhr.responseText);
09    }
10  });
11 });
```

在第 2 行中创建了一个 XHR 对象。（注意，这在 Internet Explorer 旧版本中无法正常工作[⊖]）在第 4 行中，可以看到连接是如何初始化的，第 5 行发送请求，第 6 ～ 8 行接收响应。

事实上，这个简短的脚本已经实用地展示了组成 AJAX 核心的各个部分。然而，jQuery 提供了必要的工具，在简化 AJAX 请求和避免浏览器兼容性问题方面有多种好处。

10.1.5 AJAX 通信的数据格式

如果客户端通过 AJAX 从服务器请求数据，Web 服务器总是发送普通文本作为响应（正如大家在第一个例子中所看到的）。这种普通文本可以由服务器端脚本或者程序（例如 Java、PHP 或者 ASP.NET）生成，也可能是服务器上存在的文本文件。[⊖]这些普通文本可能是完全无结构的，或者具有各种不同的内部结构。对于结构化的普通文本，通常在 HTTP 头标中指定特定的 MIME 类型，但是它仍然是普通文本。即使对于结构化的文本，这些文本是由服务器脚本或者程序生成还是已经是一个完成的文本也没有关系。

在实践中，已经确立了 4 种不同的传输格式，它们也在 jQuery 中的不同 AJAX 方法中得到反映：

- 没有任何结构的纯文本。
- 含有 HTML 标签的纯文本；换句话说，就是 HTML 片段。这些文本甚至包含脚本，但是有些浏览器不会执行这种方法加载的脚本。
- XML。
- JSON。

如果服务器发送纯文本或者含有 HTML 标签的纯文本，通常在服务器上通过动态 HTML（DHTML）（或者 jQuery）集成到网页中，大部分不需要在客户端进行响应的进一步处理。Web 服务器的响应通常不作更改地显示，必须在服务器上事先成为最终形式——换句话说，我们希望在客户端上使用的形式。在许多 AJAX 应用程序中，这就已经足够了。

警告

如果通过 AJAX 发送 HTML，这将只是一个片段。不应该通过 AJAX 重新加载完整的网页。

⊖ 如前所述，我们在这里应该使用异常处理，并使用可以正常工作的其他对象创建方法。但是为了简洁起见，我在这里不打算这么做。

⊖ 没有图像等多媒体数据或者其他二进制数据；那些数据与 AJAX 无关。

如果用可在客户端中以针对性方式使用的结构化格式扩展服务器响应，就能创建更高要求的应用程序。构造传输信息提供了分布式业务处理的更多选择。如前所述，AJAX 的结构化格式通常只有 JSON 或者 XML。所以，如果 Web 服务器发送 XML 或者 JSON，可以在客户端操作响应，将业务逻辑转移到客户端。

警告

在 Web 应用程序中，将逻辑转移到客户端有时候很难。应该非常小心地考虑这种方式是否合适。

不管 Web 服务器和客户端之间的这种业务逻辑分布是否明智，客户端通过纯粹的 JavaScript 处理 XML 都是有疑问的。XML 只是元素和结构语法的一种描述。原则上，XML 的严格规则使得这种文档容易自动化验证和处理。乍一看，XML 注定是用于数据交换的。但遗憾的是，这些规则在不同平台上的实现并不充分。尽管很难理解成因——除非是市场目标的驱动——XML 在不同浏览器中的处理方式有严重的差异。jQuery 方法减少了这类问题，至少从理论上使业务逻辑的分布成为可能。但是 JSON 仍然是实践中 Web 服务器结构化响应的更好选择。因为如果纯文本提供的逻辑不充分而 XML 太难处理，JSON 就是两个极端之间的一个替代品。利用 JSON，有了一个传输格式的结构，这个格式比 XML 简单，更重要的是，在不同 Web 浏览器中的处理更为一致。此外，JSON 格式对应于我们在 jQuery 中使用的选项格式对应——对象字面语法。程序清单 10.4 展示了简单 JSON 文档的一个例子，我们将在稍后再次使用它（ch10.json）。

程序清单 10.4 JSON 结构

```
01 {
02   "name" : "Ralph Steyer",
03   "job" : "Masters in Maths",
04   "place : "Eppstein",
05   "websites" : {
06     "url1" : "www.rjs.de",
07     "url2" : "blog.rjs.de",
08     "url3" : "www.ajax-net.de"
09   }
10 }
```

10.1.6 AJAX 请求处理

下面大家将看到每个 AJAX 应用程序在系统中工作的原则的一个总结。AJAX 请求可能在细节上各不相同，可能被框架或者工具包所隐藏，但是其原则总是遵循如下的模式：

1. 创建一个 XHR 对象，通过它进行通信。

2. 注册一个回调函数作为 XHR 对象的函数引用。这个函数在每次事务状态变化时被调用。指定的函数在 XHR 对象中每次状态变化时调用。readyState 表示回调函数调用时事务的

当前状态。这就能区分数据传输的不同状态。最重要的情况是状态码为 4 的 COMPLETED（完成）状态。只有这个状态在各种浏览器中得到了一致的处理。因此，几乎所有 AJAX 应用程序都检查该状态。jQuery 也这么做，但是具体的步骤通过框架的 AJAX 方法隐藏起来。

3. 调用 XHR 对象的 open() 方法打开连接。但是这还不是具体的请求。

4. AJAX 请求通过 send() 方法发送到 Web 服务器。

5. Web 服务器响应——有结构或者无结构的纯文本——得到处理。XHR 对象的状态变化可以明确地用于这个目的。

10.2 jQuery 中的特殊 AJAX 支持

jQuery 中的 Ajax 支持当然直接基于 XHR 对象及其方法和属性，以及 AJAX 请求的通用过程。如果掌握了 AJAX 的工作原理，许多这类 jQuery 方法也就变得很直观了。AJAX 通信的单独步骤通常在 jQuery 的不同 AJAX 方法中一起执行。而且，jQuery 有一些支持 AJAX 的特殊功能，它们已经进行了很大的扩展，特别是在 jQuery1.4.3 和 1.5.x 中。一般来说，jQuery 1.5.x 在表面上没有做大的改变，更多的是在框架的内部工作上。版本 1.5.x 绝不是 jQuery 的一个里程碑，1.3 或者 1.4.3 也是如此。唯一的例外是 AJAX，在这些版本中添加了许多新功能，但是在某些情况下，这些新功能相当特殊，仅在某些特定情况下，高级用户才对此感兴趣。

10.2.1 JSONP 和远程请求

至于 JSON，jQuery 提供了名为 JSONP[⊖] 的扩展，允许跨服务器脚本和 JSON 数据的灵活加载。[⊖]如前所述，AJAX 请求只能发送到网页加载的相同域。但是从任何服务器完全独立于 AJAX 将 JavaScript 服务器集成到网页总是有可能的。[⊜]这正是 JSONP 的用武之地。一方面，使用 jQuery 和创建一个脚本。另一方面，脚本通过 AJAX，使用 <script> 标签的 DOM 表现形式加载并执行。所有从不同域加载的脚本被称作“远程请求”。大家将在后面看到一个更大的示例。

10.2.2 jqXHR 对象

在 jQuery 1.5 中，该框架提供了原生的 XMLHttpRequest 对象的一个特殊扩展。它被称作 jQuery XMLHttpRequest，或者简称为 jqXHR。例如，该对象被用于控制 AJAX 活动的最通用 jQuery 方法（$.ajax()）当作返回值。该对象提供所有传统的 XHR 方法和属性——有一个重要的例外，因为它没有直接提供 onreadystatechange 事件处理器。这不是一个问题，因为新的 jqXHR 对象提供了新的接口，允许用于响应 AJAX 请求成功或者失败的特定单独机制。

⊖ P 是填充（Padding）的缩写。

⊖ ajax() 方法允许这种操作。

⊜ 至少可以使用 JavaScript。

jqXHR 对象的特殊能力首先表现在通过 AJAX 加载特殊格式时。具体地说，它考虑到了 JSONP 请求所用的 <script> 标签。在那种情况下，jqXHR 对象尽可能地模拟了原生 XHR 功能。

10.2.3 jQuery 中的 AJAX 请求方法

如前所述，AJAX 请求总是遵循相同的模式。本质上，我们需要服务器文件或者服务器端脚本的 URL，数据传输方法，发送到服务器的任何数据如用户输入，信息发生采用的是异步还是同步方式，以及用于响应的回调函数。有了这一背景知识，jQuery.get(url,data,callback,type) 以及 jQuery.post(url,data,callback,type) 方法[⊖]及其参数的意义、使用方法就都很清楚了。

上述两个方法提供了一个 XHR 对象，以及所有请求方法。data（数据）参数是可选的值对，包含发送到服务器的数据。可选的 type（类型）参数表示通信应该异步发生（true，默认值）还是同步发生（false）。回调函数在请求成功完成的时候执行。所以，我们仍然需要提及的，就是如何在回调函数中使用返回的数据。但是，jQuery.getJSON(url, data, callback) 方法对我们来说也应该是不言自明的。如果查看一个使用的示例，会更有意义。当然，我们将在接下来介绍几个相关的例子。

10.2.4 指定数据类型

基本上，AJAX 方法智能地响应来自服务器的数据类型。jQuery 框架试图明智地处理服务器通过头标提供的数据类型信息。但是文件扩展名也被考虑在内。如果信息表示传递的是 XML，可以通过 XML 方法或者 jQuery 选择器，以正常的方法访问 AJAX 方法的返回值。如果标识的是另一个类型（例如 HTML），数据被当作文本处理。

有时候，服务器不能正确地指明普通文本的类型，特别是在服务器数据的文件扩展名没有链接到特定格式或者发送了不合适的 MIME 类型时。[⊖]往往只需要调整服务器上的某个设置。除非你恰巧自己运行服务器或者可以使用脚本设置该头标，否则不太可能做到这一点。你也很有可能想要以不同的方式处理发送的数据。

在这两种情况下，都可以明确地指定大部分 jQuery 方法的数据类型，确定服务器解释数据的方法。在 $.GET() 或者 $.POST() 中，可以根据需要指定最后一个参数为 xml、text、json、jsonp、script 或者 html。在 $.ajax() 中，可以指定相应的细节。

提示

如果你认为 jQuery 为 AJAX 通信提供的默认数据类型不够，在 jQuery1.5 版本下也可以创建自定义数据类型。但是这一主题超出了本次讨论的范围，而且这种自定义类型只在少数情况下需要。为此，可以在 $.ajaxSetup() 中使用 converters 选项。在本章结束时可以找到一个简单的解释，当然，可以在 jQuery 文档中看到更多的解释。

⊖ 或者简写方式 $.get() 和 $.post()。

⊖ 例如，XML 应该由服务器以 text/xml 或者 application/xml 类型发送，以便浏览器可以处理它。但是在 application/xml 的情况下，某些浏览器存在一些已知的问题，因此如果有必要，你可能必须加以干预。

对于 text 和 xml 类型，jQuery 方法完全不处理数据，而是简单地传递给处理器 success()。在后台，处理涉及 responseText(text) 或者 responseXML(xml) 方法。

如果数据被标记为 html，接收到的数据中每段嵌入的 JavaScript 应该在实际的 HTML 作为字符串返回之前执行。如果数据被标记为 script，也适用相同的规则，除非在其执行之后没有任何数据返回。

如果数据被标记为 json，它被解析为 JavaScript 对象。如果浏览器支持，jQuery.parseJSON() 方法在后台运行。否则，使用 JavaScript 类 Function 的构造方法，而不使用可能不安全的原生函数 eval()。如果数据标记为 jsonp，jQuery 框架在 URL 后附加查询串参数 callback=？。服务器应该在 callback 标识符之后附加 JSON 数据，以发送一个有效的 JSONP 响应。

提示

如果必要，可以在 $.ajax() 方法中通过 jsonp 选项为 callback 指定一个代用的参数标识符。

如果 jQuery 框架的 AJAX 方法从服务器接收数据，error 回调和全局事件就绝不会启动。

在某些情况下，为以后要加载的文本文件添加 .html 扩展名可能是合适的（即使它不是一个完整的网页）——特别是，如果文本包含想要在浏览器中加载之后解释的 HTML 标记的话。然而，在大部分实现中，可以不使用这个文件扩展名，简单地使用常规的 .txt。

10.2.5 避免缓冲

对于 AJAX 请求，GET 请求中的数据缓冲存在根本性的问题。遗憾的是，即使有必要，服务器也很少在 AJAX 请求中重新请求数据。浏览器简单地从本地缓存中读取数据。[⊖]这个问题首先影响到 Internet Explorer。在手工编程中，在查询字符串后附加由 JavaScript 生成的随机数或者时间戳（不管在服务器还是在客户端，这个数值基本上都没有进一步的用途）是一个好的变通方法。这种方法只是为了告诉浏览器必须重新加载数据。事实上，这种技巧是目前避免缓冲行为（在传统的 Web 数据请求中确实很有用的一种行为）的唯一简单而可靠的方法。明确提供这类机制的框架（例如 YUI）在后台所做的也就是这些。在 jQuery 中，目前必须为大部分方法手工创建一个额外的参数，并将这个参数附加到 URL 中。只有 ajax() 方法通过特殊参数提供了对应的避免缓冲选项。

10.3 $.get() 和 $.post()

我们首先来看看用于通过请求加载数据的两个最重要的方法[⊜]——$.get() 和 $.post()。本质上，这两个方法的用法如程序清单 10.5 所示。

⊖ 你必须考虑到一点，浏览器在历史上从不是为异步数据请求设计的。

⊜ 如果你不想指定 AJAX 请求的特殊配置的话。

程序清单 10.5 方法使用示意

```
jQuery.get( url, [ data], [ success(data, textStatus, jqXHR) ], [ dataType ] )
jQuery.post( url, [ data ], [ success(data, textStatus, jqXHR) ], [ dataType ] )
```

唯一强制的参数是请求发送的 URL。可选的数据是一个 Map 类型的对象或者一个字符串，包含想要发送给服务器的数据。

第三个参数是成功时调用的回调函数。它的参数代表来自服务器的数据、包含状态的文本和 jqXHR 对象。jqxHR 对象在 jQuery 1.5 之后的版本中只用做参数——在此之前，它就是传统的 XMLHttpRequest 对象。

10.3.1 只从 Web 服务器请求普通文本

我们首先看看最简单的情况：只打算从 Web 服务器请求普通文本。在下面的例子中，这一功能通过 AJAX 和两个按钮完成。我们将同时使用 $.get() 和 $.post()，以说明这两个方法。注意，源代码中和前面没有差异。可以在配套网站中找到 ch10_2.html 网页。程序清单 10.6 展示了 JavaScript 文件（ch_10_2_ready.js）。

程序清单 10.6 通过 GET 和 POST 发起 AJAX 请求

```
01 $(function(){
02  $("button:first").click(function(){
03    $.get("ajax.txt", function(data){
04      $("div:first").text(data);
05    });
06  });
07  $("button:eq(1)").click(function(){
08    $.post("ajax.txt", function(data){
09      $("div:first").text(data);
10    });
11  });
12 });
```

在例子中，我们通过 AJAX 从 Web 服务器请求一个没有任何结构的简单文本文件。在单击第一个按钮时，文件通过 GET 请求文件，单击第二个按钮时则通过 POST。从功能上说，这个简单例子中的两个数据请求方法之间的差异并不是最轻微的。在内部，HTTP 头标看上去略有不同，当然，服务器根据相关的方法处理数据。[⊖]在第一个例子中，来自客户端的数据不会被发送到 Web 服务器，所以 AJAX 数据请求方法只需要两个参数。第一个参数当然是 URL；第二个参数是回调。这里，我们在两种情况下使用了结构完全一样的匿名函数。该函数的参数包含了 Web 服务器的响应，它已经不能再简单了。在第一个例子中，Web 服务器的响应直接集成到网页中，没有任何进一步的操作。

如果现在请求包含 HTML 片段的数据，只要使用方法 text()，[⊖]这些片段就不会被解释。

⊖ 这是 jQuery 的一个显著优势，不仅 AJAX 请求本身得到简化，而且不同方法的处理也变得统一。

⊖ 这个方法与文件扩展名 .txt 无关。

但是，当然可以像第一个例子一样，用 html() 方法代替 text()，这样浏览器的响应被相应地解释。在实践中，发送这样的 HTML 片段并简单地将其集成到网页是很常见的。注意，如前所述，除了少数例外情况，我们不能发送完整的 HTML 页面。这样的 HTML 页面已经在浏览器中加载，我们只希望替代页面的一小部分。

注意

AJAX 请求几乎总是要求使用 Web 服务器。毕竟，我们希望从 Web 服务器上请求数据，将其集成到现有的网站上。为此，通过 HTTP 重新加载数据。对于测试，也必须通过一个 Web 服务器[⊖]加载网页和请求的数据。简单地从文件系统打开一个网页通常是不可能的。所以在启动本地 Web 服务器时，它通常可以用 URL localhost 访问。在其他情况下，在地址栏中输入 Web 服务器所在的 IP 地址或者计算机名。如果用 Aptana 等 IDE 将页面当作 JavaScript Web 应用程序运行，该页面总是通过 Web 服务器请求。

10.3.2 通过 $.get() 和 $.post() 向 Web 服务器发送数据

我们现在来看看，如何在一个 AJAX 请求中向 Web 服务器发送数据。这些数据主要是用户输入，可能是 Web 表单中用户输入的数据或者基于用户鼠标操作的数据。[⊖]你将会看到 jQuery 通过 Map 类型对象显著地简化和标准化了这种数据发送。

提示

在默认设置中，AJAX 请求通过 GET 发送。如果使用 POST，发送的数据用 UTF-8 编码。如果打算发送一个查询字符串，数据被该方法转换为合适的形式。如果不希望这些现象发生，可将 processData 参数设置为 false。这在打算向服务器发送 XML 对象时有意义。在那种情况下，应该将 contentType 选项从 application/x-www-form-urlencoded 更改为更合适的 MIME 类型。

注意

从客户端发送到 Web 服务器的数据也必须在服务器上处理。对我们来说，指定服务器上的纯文本文件作为 URL 不再有意义。必须调用服务器上的一个脚本或者程序来接收和处理数据。然后，这个脚本或者程序当然会再次在客户端生成纯文本，或者具有任何结构的响应——这无关紧要。我们在此不打算详细地讨论服务器端的情况（这明显不是我们的主题），而是观察一个脚本，它为我们提供了一个发送数据的“黑盒子”（在配套网站上，能够找到本例中使用的 PHP 文件：ch10_3_get.php 和 ch10_3_post.php）。

⊖ 严格地说，甚至包括网页所在的同一个 Web 服务器（关键词——沙箱原则）。

⊖ 鼠标操作组成了最广为人知的 AJAX 应用之一——Google 地图的基础。在这个应用中，拖动地图或者滚动鼠标滚轮的动作被从后台发送给 Google Web 服务器，根据这一数据，用户接收到显示不同地图区域的新图像。

作为特殊例子的访问验证

作为一个特殊的例子，我们打算编程如下的情形：我们打算创建一个 AJAX 应用，为 Web 应用的一个封闭区域进行用户登录验证（访问验证或者访问控制）。为此，我们使用一个 Web 表单，包含一个一行的输入字段和一个密码字段。此外，网页需要一个按钮，以便用户登录。单击该按钮触发 AJAX 请求。

但是，我们不希望用常规的方式将 Web 表单的数据发送给服务器，而是使用 AJAX 请求。根据访问数据正确与否，用户可以在网页的一个区域看到不同的响应。我们希望服务器响应在不重新加载网页的情况下显示。毕竟，这是 AJAX 的必要条件，也是将我们的应用程序和传统的应用程序（在全新的网页中显示 Web 服务器响应）区分开来的特征。

因此，对于 Web 表单中的按钮，无法使用提交按钮，而不得不使用一个常规的按钮或者一个链接，无法通过提交发送 Web 表单的数据。必须避免这种表单数据的“常规”发送。否则，会请求一个新的网页。而且，action 和 method 的常规数据不在表单中，而是在 AJAX 方法中写入。

提示

当然，可以在服务器端通过 AJAX 调用的脚本或者程序中存储硬编码的访问数据。但是使用一个数据库表来存储访问数据更为明智。如果使用 XAMPP，可以使用 MySQL 和管理工具 phpMyAdmin。这样，就能在 MySQL 数据库中用 phpMyAdmin 创建这样的表。这个表至少必须包含两个字段，一个字段用于用户 ID，另一个用于相关的密码。必须检查这两部分数据。注意，两个值的组合始终必须匹配。可以在配套网站上找到创建一个表格的 SQL 脚本（ch10.sql）。

现在，我们来看看具有 AJAX 功能的网页是什么样子。在这个例子中，我们需要和本章的大部分其他例子略有不同的结构（ch10_3.html）。

程序清单 10.7　通过 AJAX 验证用户输入的网页

```
...
11        src="lib/ch10_3_ready.js"></script>
12  </head>
13  <body>
14     <h1>Please enter your user name and password</h1>
15    <form>
16      <table>
17        <tr><td>Name</td>
18          <td><input type="text" size="30"></td></tr>
19        <tr><td>Password</td>
20          <td><input type="password" size="30"></td></tr>
21      </table>
22    </form>
23    <button>AJAX login via $.get()</button>
24    <button>AJAX login via $.post()</button>
```

```
25    <div id=output"></div>
26  </body>
27 </html>
```

在例子中，可以看到一个简单的表格，包含两个输入字段，用 CSS 设置了一些格式。用户必须在一个字段中输入用户 ID；在另一个字段中则输入密码。因为通过 GET 和 POST 发送数据没有什么差别，可以看到两个按钮，用户可以通过它们发送数据。这样我们就没有必要提供两个本质上完全相同的 GET 和 POST 示例。如图 10-1 所示。

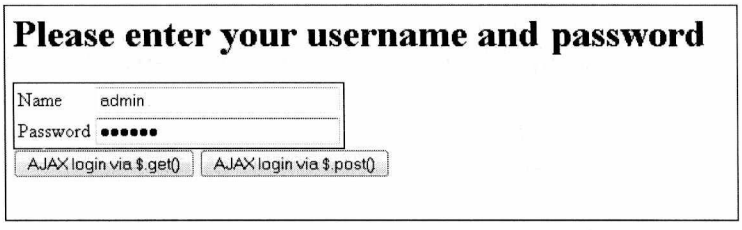

图 10-1 简单的输入表单

AJAX 函数大体上与前一个例子相同。唯一的区别是包含输入用户数据的附加参数（ch10_3_ready.js）。

程序清单 10.8 发送数据

```
01 $(function(){
02  $("button:first").click(function(){
03    $.get("ch10_3_get.php", {
04      username: $("input:first").val(),
05      password: $("input:last").val()
06    }, function(data){
07      $("#output").html(data);
08    });
09  });
10  $("button:last").click(function(){
11    $.post("ch10_3_post.php", {
12      username: $("input:first").val(),
13      password: $("input:last").val()
14    }, function(data){
15      $("#output").html(data);
16    });
17  });
18 });
```

在第 3 ～ 6 行和第 11 ～ 14 行中，分别可以看到发送给服务器的值对。这些值对写入波形括号中，这在 jQuery 中是常规的做法。和平常一样，我们使用一个选择器访问输入字段。val() 方法提供相关输入字段的内容。因为我们还需要为值对取一个名称，以获得和分配服务器端的传递值，我们在前面添加了标志“username”和“password”。

注意

实际的 form 标记中包含的参数明显少于传统的 HTML 表单。特别是，输入字段没有 name 参数。我们将使用 jQuery 选择器访问这些字段。而且，form 标记的参数完全删除。这些参数也由 jQuery 的 AJAX 函数参数代替。

如果用户在输入字段中输入正确的访问数据，Web 服务器的响应出现在表单下方，不需要重新加载网页。成功登录和不成功登录分别如图 10-2 和图 10-3 所示。

这适用于成功的登录和错误的访问数据组合。

图 10-2 成功的登录

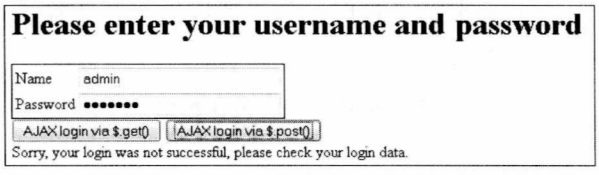

图 10-3 输入了错误的访问数据

10.3.3 获取和解析 XML 数据

如果打算通过 AJAX 请求普通文本，问题就不大。只需要输入一个 XML 文件的对应 URL 或者使用一个脚本 / 程序在服务器上生成 XML 代码。[⊖]这只有在客户端上才会令人激动，因为如果用 responseXML 接收 XML 数据，就能够访问一个完整的文档对象模型（DOM）！在 jQuery 方法中，这个 DOM 可以通过回调函数的参数访问。

如果从配套网站下载示例 ch10_4.html，就会看到网页上有一个按钮和一个输出区域。同样，JavaScript 很有趣（ch10_4_ready.js）。

程序清单 10.9 请求一个 XML 文件

```
01 $(function(){
02  $("button:first").click(function(){
03    $.get("ch10.xml", function(data){
04      $("#output").text("");
05      for (i in data) {
06        $("#output").append(i + ", ");
07      }
```

⊖ 这里，你可能必须设置正确的 MIME 类型，但是这些服务器端细节不是我们关心的。

```
08    });
09  });
10 });
```

我们简单地请求任何 XML 文件，并在第 5 ～ 7 行的 for 循环中读取 Web 服务器响应（data）中的所有元素。这和通过 GET 还是 POST 请求数据无关。如果观察输出，就会看到响应时一个完整的 DOM 对象。[⊖]这意味着，在这里可以看到可以用在网页 DOM 的所有常规属性和方法，以及 jQuery 的特殊对象。如图 10-4 所示。

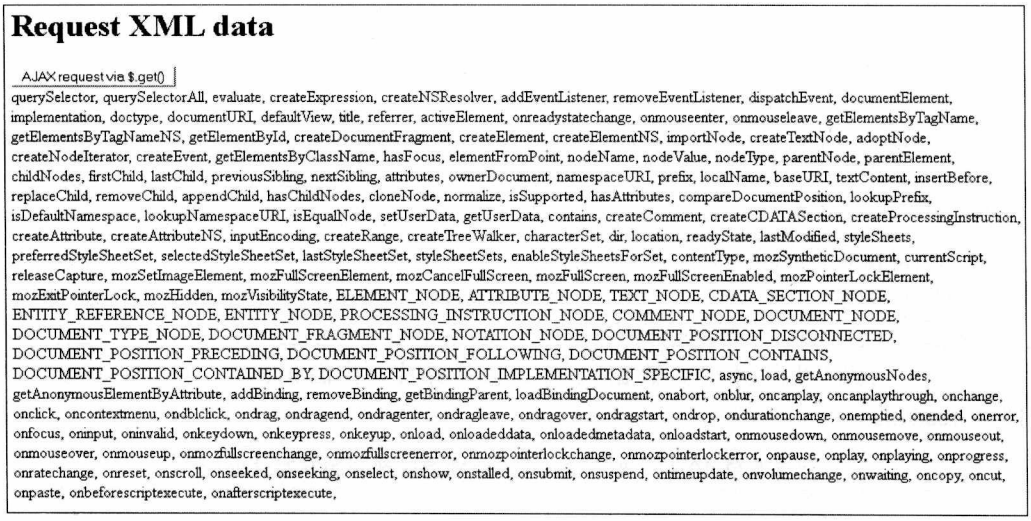

图 10-4　Web 服务器的响应是一棵完整的 DOM 树

原则上，现在可以在客户端中进一步使用 DOM，但是这超出了本书的讨论范围，在许多情况下也并不明智，因为 JSON 有一个更简单的替代方案。但是，我们仍然要简单地总结一下可以进行的工作。

为了浏览来自 Web 服务器的响应，实际上可以使用通常的方法：getElementById()、getElementByName() 或者 getElementByTagName()；当然，也可以使用所有用于选择、遍历和浏览树的 jQuery 技术。至于通过 AJAX 的数据请求，也可以非常针对性的方法浏览响应的任何部分。

不过，确实要小心，因为正如前面所提到的，DOM 的构造在不同浏览器中各不相同。关键点总是在于通过合适的方法，选择和目标节点尽可能靠近的元素。如果从响应的根开始遍历树的各个层次，可能在不同的浏览器中得到不同的结果。

我们以如下的 XML 文件 ch10.xml 作为基础。

程序清单 10.10　基本 XML 文件

```
01 <?xml version="1.0" encoding="UTF-8"?>
```

⊖ 这个循环输出一个 DOM 对象的属性和方法。

```
02 <rjs>
03  <data>
04    <name job="Masters in Maths">
05      Ralph Steyer
06    </name>
07    <place>
08      Eppstein
09    </place>
10  </data>
11  <websites>
12    <url>
13      www.rjs.de
14    </url>
15    <url>
16      blog.rjs.de
17    </url>
18    <url>
19      www.ajax-net.de
20    </url>
21  </websites>
22 </rjs>
```

原则上，我们可以对这些数据行进行某些操作（我们仍然只关注 JavaScript 文件 ch10_5_ready.js）。

程序清单 10.11　对 XML 响应单独组成部分的针对性处理

```
01 $(function(){
02  $("button:first").click(function(){
03    $.get("ch10.xml", function(data){
04      $("#output").text("");
05      y = data.getElementsByTagName("url");
06      for (i = 0; i < y.length; i++) {
07        $("#output").append(y[i].childNodes[0]
08          .nodeValue + "<br />");
09      }
10    });
11  });
12 });
```

在例子中，我们使用 getElementByTagName() 方法，从服务器响应（data）中选择特定元素。该方法返回一个数组，所以我们可以用 length 获得包含元素的数量（第 6 行）。

元素的文本内容本身也是一个节点，所以我们需要取第一个子元素，才能得到这一内容。可以通过 nodeValue 获得文本节点的内容。在第 6 ～ 9 行中，我们在元素数组上循环，输出类型为 url 的元素的内容。如图 10-5 所示。

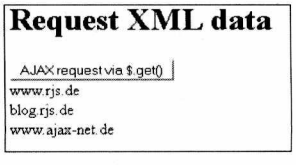

图 10-5　类型为 url 的元素内容

现在，如果还记得自己能够通过 nodeType 区分不同元

素类型，以及可以使用 DOM 概念和 jQuery 中的所有导航选项和过滤器，就应该很清楚 XML 响应的针对性使用如何进行了。

如果将 XML DOM 读入 jQuery 命名空间，处理起来就更容易了，只要通过 $(data) 访问即可。然后，可以用通常的选择器来处理这个 jQuery 对象。但是，如前所述，这仍然是一件费力的工作。

10.4 获取和解析 JSON 数据：getJSON() 和 parseJSON()

如前所述，JSON 提供了和 XML 几乎一样的功能和灵活性，且没有后者的复杂性和在不同浏览器中有疑问的求值问题。因此，如果在客户端确实需要处理响应，JSON 已经成为 AJAX 应用程序的标准格式。原则上，可以使用 $.get() 和 $.post() 从服务器请求 JSON 数据，但是以后必须进一步处理数据才能使用它。例如，可以使用原生的 JavaScript 函数 eval() 将 JSON 转换为一个 JavaScript 对象。不过，这个方法非常缓慢，迫使浏览器关闭其即时编译器，并且有着极不稳定的坏名声。作为替代，可以使用原生 JavaScript 类 Function 或者使用 jQuery 1.4.1 中推出的方法，将一个 JSON 字符串转换为一个 JavaScript 对象（只要浏览器支持）。

但是，在 AJAX 请求中甚至不需要这样的努力。jQuery 框架中的 AJAX 方法 getJSON() 专门用于这种格式，使得 JSON 的处理像小孩的游戏一样简单。

10.4.1 简单的 JSON 应用

我们以程序清单 10.4 中看到的 JSON 结构作为基础。使 JSON 在网页上可用的细节与我们的目的无关。我们只关注它们在客户端中的处理方法（ch10_6_ready.js）。

程序清单 10.12 请求和处理 JSON

```
01 $(function(){
02   $("button:first").click(function(){
03     $.getJSON("ch10.json", function(data){
04       $("#output").append(data.name + ", " +
05       data.websites.url1);
06     });
07   });
08 });
```

我们以 JSON 的形式请求数据，来自 Web 服务器的响应仍然通过 data（回调函数的第一个参数）访问。令人宽慰的是，getJSON() 方法为我们提供的数据使我们可以使用简单的句点标记法访问 JSON 结构中的单独元素（当然，这是嵌套的结构，我们在结构示例中已经使用过）。这种方法简单而可靠。如图 10-6 所示。

图 10-6 JSON 响应的一部分已经得到处理

10.4.2 通过 JSONP 请求 Twitter Tweet

现在，我们来看看如何使用 JSON。这里借助 Twitter 来进行观察。这意味着，使用 AJAX 访问另一个服务器，这种做法原先会遭到沙箱的阻止，但是 JSONP 解决了这个问题。

Twitter 是目前广泛使用的主流社交网络，人们通常用它在互联网上发布可供公众访问的日志。[⊖]注册用户可以输入并发布最大长度为 140 个字符的消息（被称作 Tweet）。这个社会化网络基于一个事实：你可以订阅其他用户的 Tweet。读者被称作粉丝。

对于我们的目的来说，访问 Tweet 的文件格式很有趣。Twitter 文稿的格式可以使用 XML 和 JSON，也可以使用其他格式访问。在程序中也可以使用这种按照日期降序排列的项目列表，这正是我们所要做的。我们使用 JSONP 是因为 Tweet 不是来自我们自己的服务器，而是来自 Twitter（http://twitter.com）。所以这一概念很理想。

注意

在下面的例子中要记住，我们必须接受 Tweet 的数据结构。可以在 http://apiwiki.twitter.com/ 找到关于 Tweet 准确含义和结构的进一步信息。

程序清单 10.13 通过 JSONP 加载 Twitter Tweet（ch10_7_ready.js）

```
01 $(function(){
02   $("button:first").click(function(){
03   $.getJSON("http://twitter.com/status/user_timeline/
       rjsedv.json?count=10&callback=?",
04   function(data){
05      var name = "";
06      var screen_name = "";
07      var profile_image_url = "";
08      var id = "";
09      //Determine user data
10      for (i = 0; i < data.length; i++) {
11         jQuery.each(data[i].user,
12           function(index, value){
13          if ((index == "name") && (name == "")) {
14            name = value;
15          }
16          if ((index == "screen_name") &&
17            (screen_name == "")) {
18            screen_name = value;
19          }
20          if ((index == "profile_image_url") &&
21            (profile_image_url == "")) {
22            profile_image_url = value;
23          }
```

⊖ 也被称作微博。

```
24          if ((index == "id") && (id == "")) {
25            id = value;
26          }
27        });
28      } // End for – all user data determined
29      var title = "<h1>The Twitter Tweets of " +
30      screen_name +
31      "</h1>";
32      $("#output").html(title + "<br />" +
33      "Name: " + name + ", ID: " + id + "<hr />");
34      if (profile_image_url != "")
35        $("#output").append("<img src='" +
36           profile_image_url + "' /><hr />");
37      // The specific tweets
38      for (i = 0; i < data.length; i++) {
39        jQuery.each(data[i], function(index, value){
40          if (index == "text") {
41            $("#output").append(value + "<br />");
42          }
43          if (index == "created_at") {
44            $("#output").append("Created: " +
45            value +
46            "<hr />");
47          }
48        });
49      } // End for - Tweets
50    });
51  });
52 });
```

在第 3 行中，我们使用 getJSON() 请求别名为 rjsedv 的作者的 JSON 数据。

提示

可以用任何 Twetter 别名来代替 rjsedv，只要该作者的 Tweet 可以公开访问。

这个例子有些复杂，因为我们操作于 Twitter Tweet 的结构之上。我们无法、也不打算详细地讨论这个结构。但是，我当然会解释例子中使用的特定结构，这也是 JSON 格式中最关键的细节。

在 Twitter 的 JSON 格式中，有一个名称为 text 的关键字。你可能已经猜到，它包含了相关 Tweet 的内容。还有一个关键字名为 created_at，这代表相关 Tweet 的创建日期。在第 38 ～ 49 行中，我们使用 each() 循环读取通过 AJAX 从服务器接收到的所有 Tweet。通过 index 参数，可以得到结构中的关键字名，通过 value 获取特定的键值。利用 if 结构，我们只处理这两个关键字，将相关的键值添加到输出区域。

除了 Tweet 文本本身和日期之外，我们还希望在网页中显示一些关于文本作者的信息。每个 Tweet 以名为 user 的数组的形式包含这一信息。所以，如果这些信息不变，那么它们

就是多余的。但是，用户也可能更新这一档案。例如，他可能上传或者替换照片。Tweet 的顺序结构确保最新的消息出现在 JSON 结构的开始处。如果用循环读取 Tweet，第一条 Tweet 包含的作者信息是最新的。我们在程序中利用了这一点。

我们将对感兴趣的作者数据使用不同的变量。这包括作者的姓名、屏幕名称（别名）、ID 和照片。在第 10 ～ 28 行中，我们寻找每个 Tweet 的作者数据。我已经指出，我们只需要搜索第一条 Tweet（data[0]），因为它包含了作者的最新数据。但是我们的通用方法在需要处理作者数据的任何变化（在例子中没有这么做）时，也可以搜索旧的 Tweet。但是按照时间排序的结构也造成了一个问题：我们必须小心地避免在循环读取所有 Tweet 时用旧的信息覆盖更新的信息。所以，我们检查感兴趣的变量值是否已经填写，只在它尚未填写时用对象 user 中的属性值为其赋值。如图 10-7 所示。

程序清单 10.14　格式化数据输出区域

```
01 #output {
02  width: 900px;
03  max-height:600px;
04  overflow:scroll;
05  background: yellow;
06  color: blue;
07 }
```

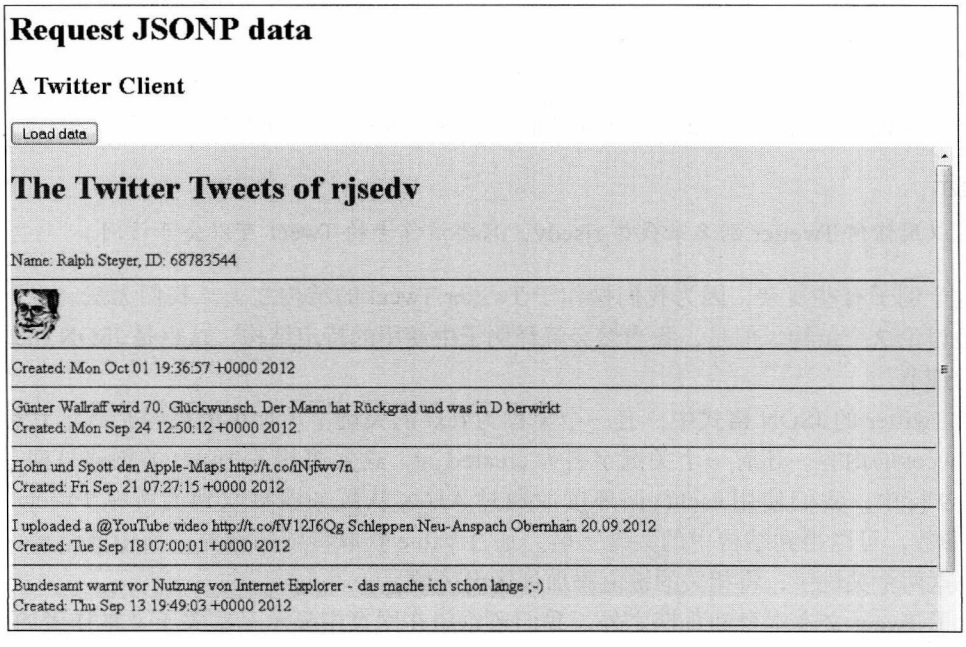

图 10-7　在网页中显示的 Twitter Tweet

10.5 通过 AJAX 在以后加载脚本：jQuery.getScript()

如果在某个时点需要程序库中的功能，通常在加载网页之后立刻加载一个 JavaScript 文件。但是，我们并不总是需要 JavaScript 程序库中的所有功能。能够将功能分隔到多个程序库中，以便在以后需要的时候加载特定功能，是很有意义的。可以通过 AJAX，轻松地在任何浏览器中加载脚本指令，只要简单地在一个 HTML 片段中编写一条脚本语句即可，如程序清单 10.15 所示。

程序清单 10.15 包含一个脚本的文本片段

```
This is <i>AJAX</i><script type="text/jscript">alert("Loaded later");</script>
```

如果通过 AJAX 加载文本文件，[⊖]大部分浏览器将执行该脚本（但不是所有浏览器都是如此）。特别是 Internet Explorer 的反应引人注目，[⊜]不过可以在 jQuery 方法 $.get() 或者 $.post() 中指定文件类型，这往往能够解决问题。但是，为什么不直接使用为此目的设计的方法——jQuery.getScript() 呢？

用 jQuery 方法 $.getScript() 能够做什么呢？该方法加载一个 JavaScript 程序库并同时执行。所以在默认的设置下，该方法基于明确发送 JavaScript 的服务器。

在函数集合的情况下，这意味着函数不仅加载到网页的全局 JavaScript 命名空间，可用于调用（和传统的情况一项），而且还可明确地用于所执行的可选回调函数中。Internet Explorer 也可以完成这一操作，通过 HTTP GET 请求（不可能采用 POST）加载文件，就像通过 GET 进行的常规 AJAX 请求一样。

注意

在 jQuery 1.2 版本中，还可以通过 getScript() 从外部域加载脚本。在 1.2 版本之前，只能加载和原始网页同域的脚本。前面已经学习了沙箱原则和 jQuery 的 JSONP 扩展。加载[⊜]和同时自动执行[⊛]外部域的选项显然大大扩展了功能性。但是大家应该知道访问外部域的选项去除了浏览器的重要安全准则。沙箱原则在浏览器中的实现有非常正当的理由，去除它的做法会降低 Web 应用程序的安全性。我的看法是，这种扩展确实有可能——尽管它只可能在有限的程度上遭到误用——毁了人们对 JavaScript 本质上是一种非常安全的技术这一认识。所以，我认为这种扩展是把双刃剑。

⊖ 不管使用人工 AJAX 还是 $.get() 或者 $.post()。

⊜ 例如加载它并不知晓的标记。这些标记在 Internet Explorer 中被作为空元素显示，你无法通过 CSS 对其进行格式化。此外，Internet Explorer 取出内容，将其作为附加的文本节点插入网页中，这将改变节点的数量。

⊜ 加载并不重要；这总是有可能完成的。

⊛ 这是特殊的功能。

程序清单 10.16 加载 JavaScript 文件（ch10_8_ready.js）

```
01 $(function(){
02  $("button:first").click(function(){
03    $.getScript("lib/random.js", function(){
04      $("#output").text(randomNumber());
05    });
06  });
07  $("button:last").click(function(){
08    $.getScript(
09      "http://rjs.de/jquery/zufall.js", function(){
10        $("#output").text(zufallsZahl());
11      });
12  });
13 });
```

如你所见，该方法的使用非常简单。在第一个参数中，指定了 JavaScript 程序库的 URL，第二个参数是一个回调函数（如果需要的话），在该函数中使用了来自程序库的一个特定功能（通常是一个函数）。在我们的例子中，从网页的同一个域加载（通过第一个按钮）程序库，我们也可以从另一个域中加载。⊖

程序清单 10.17 展示了被调用程序库 random.js 的结构，但是这与我们的论述无关。

程序清单 10.17 random.js 程序库

```
01 function randomNumber(){
02      return Math.random();
03 }
```

对于我们的目的，在程序库中定义一个函数就足够了。可以看到，程序库中的 randomNumber() 函数在回调函数中被调用。如果单击相关按钮，该程序库被加载，该函数可供调用。如图 10-8 所示。

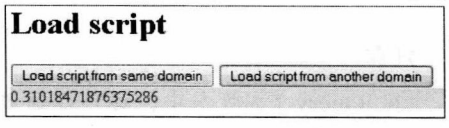

图 10-8 加载并执行外部 JavaScript 程序库

警告

在某些浏览器中，不同的版本对这个方法造成了某些复杂性。例如，在 Safari 2 和更早的版本中，不可能在全局上下文中同步执行脚本。因此，调用通过 getScript() 加载的程序库中的一个函数不得不略有延迟（例如，通过 DOM 方法 window.setTimeout()）。如果打算使用这个方法，就要检查文档，确定是否会在目标浏览器中遇到潜在的问题。

10.6 加载数据的通用变种：load()

用 load() 方法，可以从服务器加载数据，并插入网页的 DOM 中。这个方法的特性是可

⊖ 为了不失去通用型，我们的例子在两种情况下使用同一个程序库（完全没有差别）。

以非常通用的方式工作。但是在实践中，一般通过该方法加载 HTML 片段。然而，也可以请求 XML。该方法在大部分浏览器中工作得很好，只有 Internet Explorer 再次出现问题，这是因为 Internet Explorer 处于常规网页的解释模式中时对未知标记的处理。原则上，可以通过 load() 方法加载纯文本和 JSON，但是这个方法的主要好处（过滤，接下来将会讨论）会因此失效。这就是通常用它加载 HTML 片段的原因。该方法签名的模式如程序清单 10.18 所示。

程序清单 10.18 load() 模式

```
load( url, [ data ], [ complete(responseText, textStatus, XMLHttpRequest) ] )
```

在基本设置中，使用该方法将一个 GET 请求发送给服务器。

程序清单 10.19 默认设置：通过 GET 加载 HTML

```
$("#output").load("rjs.html");
```

但是，如果使用对象 / 映射（关键字 – 值配对）形式的附加参数，请求通过 POST 自动发生。

程序清单 10.20 通过 POST 发送数据（一个值对）

```
$("# output ").load("processvalues.php", { names[]': ["Felix", "Florian"] } );
```

附加的参数是一个字符串，但是被作为 GET 请求执行。

从方法签名的示意中可以看到，可以指定一个回调函数作为可选的第三参数。这个函数有三个标准参数——resposeText、TextStatus 和 XMLHttpRequest。

指定过滤器

下面我们转向该方法可能最有用的一种用法。在 jQuery 1.2 中，可以为 load() 的 URL 中指定一个 jQuery 选择器。如果这么做，加载的 HTML 片段[⊖]被看作 DOM 并加以过滤。这意味着，只有响应 DOM 中匹配选择器的那些元素才会被集成到网页的 DOM 中。选择器用空格与 URL 分隔。

程序清单 10.21 通过过滤器加载

```
$('#info').load('ajax/rjs.html #myID');
```

提示

为 load() 指定过滤器的选项是使用 XML 或者 JSON 的简单替代方案。可以使用纯 HTML 作为传输格式中的结构信息，在客户端中以合乎逻辑的方式选择它。在这种情况下，如果过滤框架结构，甚至可以加载一个完整的 HTML 文件。[⊜]

⊖ 也可以是一个 XML 文件，除非使用的浏览器是 Internet Explorer。

⊜ 只要你想这么做（不管什么理由）。

程序清单 10.22 使用 load() 的不同方法（ch10_9.html）

```
...
10    <script type="text/javascript"
11        src="lib/ch10_9_ready.js"></script>
12  </head>
13  <body>
14    <h1>Filtering with load()</h1>
15    <button>Load whole table/button>
16    <button>Load data cells</button>
17    <button>Load header cells</button>
18    <button>Load XML</button>
19    <button>Load structure with custom tags</button>
20    <div id="output"></div>
21  </body>
22 </html>
```

下面是 JavaScript 文件（ch10_9_ready.js）。

程序清单 10.23 使用过滤器的 load()

```
01 $(function(){
02  $("button:first").click(function(){
03    $("#output").load("load.html").css({
04      background: "red",
05      color: "yellow",
06      width: "300px"
07    });
08  });
09  $("button:eq(1)").click(function(){
10    $("#output").load("load.html td");
11  });
12  $("button:eq(2)").click(function(){
13    $("#output").load("load.html th");
14  });
15  $("button:eq(3)").click(function(){
16    $("#output").load("ch10.xml url").css({
17      background: "red",
18      color: "yellow",
19      width: "300px"
20    });
21  });
22  $("button:eq(4)").click(function(){
23    $("#output").load("load2.html").css({
24      background: "red",
25      color: "yellow",
26      width: "300px"
27    });
28  });
29 });
```

在第 13 行和第 16 行中，可以看到过滤器当然是基于后面加载的 HTML 文件的结构，这就是已经加载的 HTML 文件 load.html。

程序清单 10.24 后来加载的 HTML 片段

```
01 <table>
02   <tr><th>Name</th><th>Age</th></tr>
03   <tr><td>Felix</td><td>11</td></tr>
04   <tr><td>Florian</td><td>11</td></tr>
05 </table>
```

加载的 HTML 片段是一个具有表头和数据单元格的小型表格。单击第一个按钮加载完整的 HTML 片段并集成到网页。注意，load() 的返回值是 jQuery 对象，可以像平常一样格式化。如图 10-9 所示。

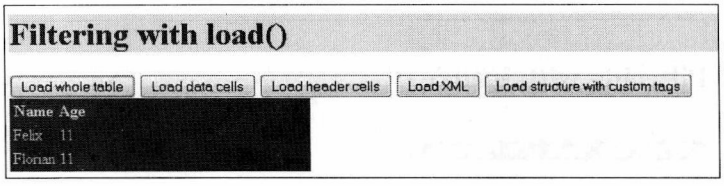

图 10-9 完整的表格加载并格式化

单击第二个按钮只加载表格所包含的数据单元格，第三个按钮只加载表头单元格。我们明确地应用过滤器 th 和 td。至于格式，是否已经首先触发一个按钮，对输出区域进行格式化就很重要了；在必要的时候，格式保持不变。如图 10-10 所示。

当然，在实践中，提取链接结构（如表格）的一部分只能在有限的程度上进行。但是，这个例子至少很好地说明了过滤器的工作方式。

现在，可以看到单击第 4 个按钮请求一个 XML 文件，并通过 css() 对其进行格式化，这在所有相关浏览器中都工作得很好。如图 10-11 所示。

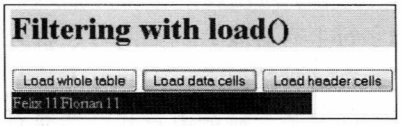

图 10-10 只有数据单元格从 HTML 片段中过滤出来插入网页的 DOM

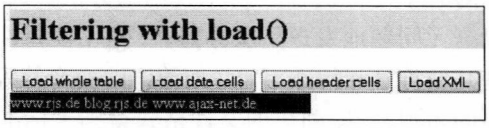

图 10-11 经过过滤的 XML 文件元素

但是，这个例子在 Internet Explorer 中不能正常工作，这意味着如果打算支持该浏览器，这种方法在实践中就不可行。

有趣的是，一种替代方案工作得相当好，在这种方案中，我们简单地打破有效 HTML 的原则，使用在容错原则下被忽略的元素。请看下面的文件 load2.html。如图 10-12 所示。

程序清单 10.25 使用 HTML 中不存在的标记的一种结构

```
<tab>
 <row>
   <headercell>Name</headercell>
   <headercell>Age</headercell>
 </row>
 <row>
   <cell>Felix</cell>
   <cell>11</cell>
 </row>
 <row>
   <cell>Florian</cell>
   <cell>11</cell>
 </row>
</tab>
```

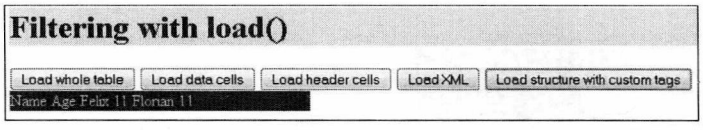

图 10-12 现在即使在 IE 中也能正常工作

所以，如果能让 Internet Explorer 将以后加载的数据看成 HTML，[⊖]就可以通过 load() 将这些数据集成到网页，并将其格式化。引人注目的是，如果通过 AJAX 人工加载，这种技巧在 Internet Explorer 中无效。正如前面在 jQuery 数据类型中所描述的，jQuery 框架明显在后台进行了干预。这一技巧在所有其他相关浏览器中也有效。[⊜]

10.7 序列化数据

jQuery 还提供了序列化数据的方法。通常，jQuery 使用序列化以准备用户输入，使其可以毫无问题地发送给服务器。[⊜]这主要是在某些特殊字符上有必要，例如问号、百分号和一些其他符号和特殊字符（如德语的变音符等外语字符；在这种情况下，jQuery 方法甚至有不同的表现）。序列化的文件格式得到了标准化，并与大部分服务器端语言和框架兼容——这被称作 URL 编码。

⊖ 这也可以通过服务器上的 MIME 类型，用头标信息来完成；文件扩展名 .html 只是一种简单的方法。在其他 jQuery 方法中，你甚至可以直接指定文件类型。

⊜ 人工加载通常也有效。

⊜ 特别是通过 AJAX 人工发送数据的情况下，你通常必须自己完成这项工作，而 jQuery 的大部分 AJAX 方法自动在后台负责序列化。

10.7.1 serialize() 方法

jQuery 框架提供 serialize() 方法，用于序列化数据串（换言之，一个 Web 表单）中的一组输入元素。每个需要序列化的表单元素必须有 name 特性，该方法才能正常工作。

该方法在代表表单的 jQuery 对象上操作，生成标准 URL 编码标记法的一个文本字符串。原则上，也可以直接对单独的表单元素应用该方法，但是通常使用整个表单将所有输入合并到一个字符串。该方法可以编码特殊字符和变音符号。我们来看一个例子（ch10_10.html）。

程序清单 10.26 需要序列化数据的表单

```
...
10    <script type="text/javascript"
11       src="lib/ch10_10_ready.js"></script>
12  </head>
13  <body>
14     <h1>Serializing Data</h1>
15    <form>
16      <p>
17        <input type="text" name="name" />
18           <input type="text" name="firstname" /><br />
19           <input type="button" value="Send data" />
20      </p>
21    </form>
22   </body>
23 </html>
```

用户输入到表单的所有数据必须序列化。这很容易完成，如程序清单 10.27 所示（ch10_10_ready.js）。

程序清单 10.27 序列化表单

```
01 $(function(){
02  $("input:button:first").click(function(){
03   alert($("form:first").serialize());
04  });
05 });
```

在第 3 行中可以看到，我们对页面中的第一个表单应用该方法。在这个简单的例子中，我们只在回调函数中输出经过序列化的输入。如图 10-13 所示。

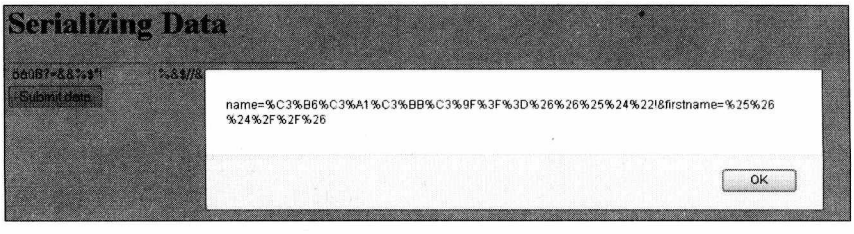

图 10-13 可以清晰地看到，特殊的字符被编码

现在，可以在 AJAX 或者传统的 Web 表单中取得序列化的表单数据，传递给服务器。

10.7.2 serializeArray() 方法

serializeArray() 方法也序列化所有的表单和表单元素（和 serialize() 方法一样），但是它返回一个嵌套的 JSON 数据结构，可以对其进行操作。利用参数 name，可以获得表单字段的名称，而用 value 则可以得到对应的值（任何形式）。与 serialize() 方法相反，特殊字符被编码，而变音符号则不能。我们来看看下面的脚本 ch10_11_ready.js。

程序清单 10.28 使用 serializeArray()

```
01 $(function(){
02  $("input:button:first").click(function(){
03    var data = $("form:first").serializeArray();
04    $("div:first").html();
05    $(data).each(function(index, v){
06      $("div:first").append(index + ": " + v.name
07         + ": " + v.value + "<br />");
08    });
09  });
10 });
```

作为返回值，我们得到一个 JSON 对象，可以循环读取它。通过这个对象，我们循环读取单个表单字段。如前所述，我们通过 name 获得相关字段名，通过 value 得到序列化之后的值。如图 10-14 所示。

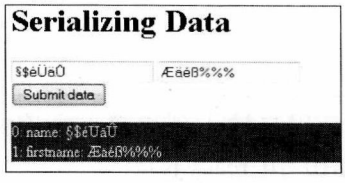

图 10-14 德语变音符号没有得到编码

10.7.3 通用版本：param()

利用这个方法，可以序列化一个通用的数组或者对象，前面提到的两个方法在内部也使用该方法。第一个参数是对象或者数组。在版本 1.4 时，可以通过第二个布尔参数，指定数据的序列化方式——递归式（默认值，false）或者传统式（true）。[⊖]

10.8 AJAX 的默认值

如果不喜欢 jQuery 处理 AJAX 请求的默认设置，可以对这些值进行全局调整。可以通过 jQuery.ajaxSetup() 方法做到这一点，该方法有一个参数是个选项对象，可以在这里进行全局调整。这些设置只要不被单独的方法设置所覆盖，就一直有效。可能的设置组合对应于常用的选项。

⊖ 你也可以通过 jQuery.ajaxSettings.traditional=true; 进行全局设置。

程序清单 10.29 将默认设置方法设置为 POST

```
$.ajaxSetup({
   type: "POST"
 });
```

程序清单 10.30 集中初始化 AJAX 请求的另一个示例

```
$.ajaxSetup({
  url: "test.php", global: false, type: "GET"
});
```

所有设置都是可选的。

警告

不应该用这个方法设置 complete、error 或者 success 的处理器，但是可以使用全局的 AJAX 事件。

10.9 AJAX 事件和 AJAX 事件处理器

发送 AJAX 请求的不同方法通常都是完备的。换言之，在许多情况下只需调用对应的 AJAX 方法，请求就完成了。但是，有些时候希望用针对性的方式响应 AJAX 请求中发生的特定事件。AJAX 请求生成不同的事件，可以做出特殊的反应。全局和局部事件之间存在着差别。

10.9.1 局部事件

局部事件这一术语指的是可以在 XHR 对象中指定响应的一些回调。例如，这包括错误或者成功时被调用的函数。在本章的后面，大家将看到 $.ajax() 方法的描述，该方法中使用了局部事件（但是这有些超前了）。

表 10-1 描述了局部事件。

表 10-1 jQuery 中的 AJAX 局部事件

事 件	描 述
beforeSend	局部事件，在 AJAX 请求前触发。例如，可以在指定的回调函数中修改 XHR 对象
complete	这个局部事件描述请求完成的状态，不管请求是否成功。在两种情况下，都能得到一个完整的回调，甚至对同步请求也是如此
error	这个局部事件只在错误发生时触发。相应地，对于同一个请求不可能同时有 error 和 success 回调
success	这个局部事件发生（也仅发生）在成功时。服务器发送的任何错误信息或者数据结构中的任何错误都会阻止该事件发生

jQuery 1.5 版本引入的 jqxHR 对象提供了一个对服务器响应做出反应的接口，由此可以

使用 error()、success() 和 complete()。这些方法使用一个函数引用作为参数。这个回调函数在 $.ajax() 触发的请求结束时调用。回调函数使用和相应名称的 $.ajax() 回调相同的参数。

链接局部回调事件

在 jQuery1.5 时，甚至可以将多个回调连接到单个 AJAX 请求（例如，如程序清单 10.31 所示——如果使用句点标记法的话）。

程序清单 10.31 链接回调

```
var jqxhr = $.ajax({ url: "ajax.txt" })
  .success(function() { $("#output".html("All okay")); })
  .error(function() { $("#output".html("Gone wrong")); })
  .complete(function() { $("#output".html(
    "All done")); });
```

回调队列

在 jQuery1.5 版本中，error、success 和 complete 回调被当成一个先入先出（FIFO）队列进行管理。如果为一个请求注册多个回调，它们以相应的顺序处理（如果它们被指定的话）：

1.beforeSend

2.error

3.dataFilter

4.success

5.complete

10.9.2 全局事件

jQuery 框架以对象的形式提供事件，可以用多种不同的方式使用它。这些事件被传递给 DOM 中的所有元素。这使得可以在监听这些事件的时候，于任何位置对其作出响应。对全局事件的这种监听多少有些类似程序清单 10.32。

程序清单 10.32 绑定到全局事件

```
$("#output").bind("ajaxSend",
  function(){
   $(this).show();
  }).bind("ajaxComplete", function(){
    $(this).hide();
});
```

可以在这个例子中看到传统的函数与特定事件绑定通过 AJAX 完成。

提示

正如你所看到的，可以禁用全局事件。为此，需要将 global 选项设置为 false。ajaxSend() 或者 ajaxError() 之类的方法此后不会像平常那样被请求触发。

也可以在一个节点中将事件当作方法写入，并指定一个回调函数或者匿名函数作为参数（例如下面的程序）。

程序清单 10.33 响应一个错误

```
$("#output").ajaxError(function(event, request, settings){
   $(this).text("Error: " + settings.url + "!");
});
```

程序清单 10.34 响应 AJAX 请求结束

```
$("#loadindicator").ajaxStop(function(){
   $(this).hide();
 });
```

表 10-2 是一个可能在 AJAX 通信中发生的事件列表。除了 ajaxStart 和 ajaxStop 之外，所有事件都为所有 AJAX 请求所触发（除非设置了 global 选项）。ajaxStart 和 ajaxStop 事件所指的是所有 AJAX 请求。

表 10-2 jQuery 中的全局 AJAX 事件

事 件	描 述
ajaxComplete	这个全局事件的表现和 complete 事件完全相同，在 AJAX 请求完成时触发
ajaxError	这个全局事件的表现和局部事件 error 类似。在错误发生的时候触发
ajaxSend	这个全局事件在请求发送之前触发
ajaxStart	启动 AJAX 请求时的核心事件。该事件在 AJAX 请求开始且当前没有其他 AJAX 请求运行的时候触发
ajaxStop	这个全局事件在没有其他 AJAX 请求执行时触发
ajaxSuccess	这个全局事件只在调用成功时发生

可以在 jquery.com 的 jQuery 文档中看到 $.ajax() 方法的完整例子。

10.10 完全控制

前面讨论过的用于 AJAX 请求的 jQuery 方法通常非常实用，也很容易使用；它们在后台处理许多我们在 AJAX 中需要留心的工作。它们在 jQuery 文档中被视为更高的抽象层次，使编程人员不需要了解 AJAX 通信的细节。但是，有时候你可能对 AJAX 通信的这些细节感兴趣，jQuery 也为此提供了合适的方法——jQuery.ajax()。

10.10.1 jQuery.ajax()

jQuery.ajax() 方法使你在必要的时候可以完全控制 AJAX 通信过程，可以任何方式规定数据交换的细节。利用 jQuery.ajax()，可以直接返回类型为 jqXHR 的对象，该对象具有目

前为止讨论过的所有属性和方法。[⊖]这代表着 jQuery 对 AJAX 的低级实现。此外，还可以得到合格的错误回调，而且可以访问字符集和 MIME 类型。目前讨论的 AJAX 方法只不过是这个方法特殊配置变种的简写标记法。

注意

记住，$.ajax() 方法并不完全是容易使用的，首先要求对 AJAX 有基本的知识。而且，有些设置要求你观察不同的服务器端规则，才能实现合适的结果。特别是在这个方法的上下文中，jQuery 1.5 已经添加了一些重要的扩展，它们的应用相当特殊，其细节超出了本书的范围。但是，我们仍然会介绍一些部分。

作为参数，可以向该方法传递一个类型为 Map 的对象。这个对象代表着用于初始化 XHR 对象和处理请求的关键字 - 值配对。当然，这些选项基于 XHR 对象的属性和方法，但是特别到了 jQuery 1.5 的时候，它们的潜力远超过原生的 XHR 对象。

到 jQuery1.5 时，该方法还有一个变种，可以将第一个参数指定为想要发送请求的 URL。可选的第二个参数是包含设置的 Map 对象。

表 10-3 是可以为这个方法指定的选项列表。只在选项值偏离默认值或者想要明确地规定这些选项时才需要为 ajax() 方法指定选项。

表 10-3 可以用来作为 ajax() 参数的选项

名 称	描 述
accepts	类型为 Map 的对象，可以通过它告诉服务器接受哪一种类型的应答
async	布尔值，用于确定数据请求是同步还是异步。默认值为 true（异步）。例如，以 JSONP 形式访问其他域，可能无法同步进行。一般来说，同步 AJAX 请求并不完全是没有疑问的，因为它们可能在请求处于活跃状态时暂时锁住浏览器。到 jQuery1.8 中，async:false 被废弃，代之以 complets、success 和 error 回调
beforeSend	局部事件，在 AJAX 请求之前触发，与一个对应的回调函数相关。作为参数，可以使用一个 jqXHR 对象和一个用于设置的 Map 类型的对象
cache	通过这个布尔参数[⊜]，可以控制缓存问题。如果参数设置为 false，通过 AJAX 请求的网站不会在浏览器中被缓存。默认值一般是 true，但是 false 可用于 dataType 选项值为 script 和 jsonp 的特殊情况
complete	这里指定的函数在请求完成的时候调用，只适用于成功和错误的回调完成之后的情况。作为参数，传递 XHR 对象和一个描述性的字符串（"success"、"notmodified"、"error"、"timeout"、"abort" 或者 "parsererror"）。到 jQuery1.5 时，还可以指定一个包含设置函数的数组。每个指定的函数在该回调被调用时执行
contents	在 jQuery 1.5 中，可以使用这个选项指定一个 Map 类型的对象，包含字符串和一个正则表达式的配对，通过它来确定响应的解析方式

⊖ 在 jQuery 的早期版本中，返回的是 XMLHttpRequest 类型的对象。

⊜ 指在 jQuery1.2 中。

（续）

名　称	描　述
contentType	当发送用户输入等数据到服务器时，某些字符不会被原封不动地传输（例如，德语变音符号或者特殊字符，如前所述[⊖]）。发送数据之前，通常会发生所谓的 URL 编码处理。这一过程包括用加号代替所有空格，以及德语变音（某些时候）等关键字符，或者首先将特殊字符转换为等价的十六进制值。转换后的符号有百分号前缀，后面跟着十六进制编码。 在传统的表单数据发送中，这种转换由浏览器自动完成。但是当通过 AJAX 人工发送数据时，URL 编码不会由浏览器或者 AJAX 通信对象自动进行，必须人工进行。幸好，jQuery[⊖] AJAX 方法减轻了我们的负担，它们会自动执行 URL 编码。在 ajax() 方法中，可以在必要时对具体的编码加以干预，但是很少有必要这么做。编码的默认值是 application/x-www-form-urlencoded。这在大部分情况下也是最合适的设置
context	我们已经在 jQuery 基础知识中讲述了上下文的重要性。这个对象指定所有与 AJAX 相关的回调的上下文。例如，可以这样指定一个 DOM 元素： $.ajax({ … context:document.body, … });
converters	到 jQuery1.5 的时候，jQuery 中的数据类型有一些转换功能。通过 Map 类型的对象，本质上可以指定现有数据类型的处理方式，默认为： {"* text":window.String, "text html":true, "text json": jQuery.parseJSON, "text xml": jQuery.parseXML} 转换器中的每个值是一个函数，返回 AJAX 请求响应的转换值
crossDomain	到 jQuery 1.5，可以指定是否允许跨域访问。默认值为 false，只允许访问同域
data	发送到服务器的数据以查询串的形式出现。这个查询串是一个具有特殊结构的字符串，我们已经在前面多次看到。数据以名称 – 值配对的形式传输给服务器上的一个处理脚本或者程序。名称代表一个变量，值则是该变量应该传递给服务器的值。例如，变量可以是一个用户输入。这样的查询串结构大致如下： 如果传递一个数组，jQuery 用同样的键值序列化多个值，例如： {names:["Steven", "Phil"]} 被转换为如下的查询串： info%5D%5B%5D=Steven&info%5Bnames%5D%5B%5D=Phil
dataFilter	用这个选项，可以指定一个直接处理 XHR 对象应答数据的函数。该函数接受来自服务器的数据作为参数，另有一个 dataType 类型的参数
dataType	如果在一个 AJAX 请求中从服务器接收到响应，数据类型将会决定客户端如何处理应答。前面已经提到，响应被发送两次。更准确地说，它存在于 XHR 对象的两个属性中。responseText 属性包含服务器发送数据（文本形式），responseXML 则包含 XML 数据（换句话说，使你可以浏览的一棵树）。dataType 参数使你能控制客户端的这一决定。前面已经提到，jQuery 中的默认设置为“ intelligent”。这时，jQuery 将根据发现的响应 MIME 类型决定使用 responseText 还是 responseXML 的值。现在大家已经熟悉了可用的类型（xml、html、script、json、jsonp 和 text）以及用法
error	在此可以指定一个特殊的回调函数，在出现错误时调用（换言之和，在 AJAX 请求失败时）。可以指定三个参数，XHR 或者现在的 jqXHR 对象，发生的错误的描述性字符串以及一个可选的异常对象。第二个参数的值可能为 null，也可能是 timeout、error、notmodified 和 parsererror。到 jQuery1.5，还可以指定一个包含设置函数的数组。每个指定的函数在回调被调用时执行

⊖ 在序列化的时候。

⊖ 以及其他 AJAX 构架。

（续）

名　称	描　述
global	布尔值，默认为 true，指定全局 AJAX 事件处理器是否在请求时被触发。如果该值被设置为 false，ajaxStart() 或者 ajaxStop() 等全局事件处理器将不被触发
headers	到 jQuery 1.5，我们拥有了指定一个 Map 类型对象的选项，该对象包含支持头标信息。和平常一样，指定一个关键字 – 值对，它和请求一起发送。这些设置应该在调用 beforeSend() 回调函数之前输入
isLocal	利用这个选项，可以将当前环境标记为本地，即使 jQuery 没有这样的默认设置
jsonp	利用这个选项，可以覆盖 JSONP 请求中的回调函数。该值被用于替代 GET 请求查询串中“ callback=？”部分中的“ callback”或者 POST 请求中的数据。例如，{jsonp:'onJsonPLoad'} 导致’ onJsonPLoad=？’通过查询串发送给服务器。这个选项在 jQuery 1.5 中已经更改。如果将该选项设置为 false，将使用“ =？”而不是“？ callback”。在那种情况下，应该显式指定 jsonpCallback。例如：{jsonp:false, jsonpCallback:”myCallback”}
jsonpCallback	利用这个选项，可以指定想要在 JSONP 请求中触发的回调函数。该值用于替代框架随机生成的唯一名称。到 jQuery 1.5 时，还可以通过一个函数的返回值设定该名称
mimeType	在 jQuery 1.5.1 版本中，可以用这个选项覆盖 XHR 对象中的 MIME 类型
password	HTTP 访问中的密码
processData	布尔值（默认为 true），影响数据发送到服务器的方式。在默认设置中，通过 data 属性发送的数据以内容类型为 application/x-www-form-urlencoded 的查询串形式出现。如果打算发送 DOM 文档或者其他数据结构，将该选项设置为 false
scriptCharset	如果使用 jsonp 或者 script 作为 dataType 的值，使用 GET 方法（type 的值），这个选项强制某种字符集的翻译。只有在本地和远程内容的字符集不同时才需要这样做

statusCode	jQuery 1.5 的另一种新特性是数字 HTTP 编码和函数的映射，这些函数在服务器响应包含对应编码时调用。可以使用这个选项作出单独的反应。例如：
	`$.ajax({`
	`  statusCode: {`
	`  200: function() {`
	`  $('#output').html("All okay");`
	`  },`
	`  404: function() {`
	`  $('#output').html("Gone wrong");`
	`  }`
	`});`

success	在此可以指定 AJAX 请求成功时调用的回调函数。可以指定 3 个参数：与 dataType 值对应的由服务器传递的数据，状态的描述字符串，以及 jqXHR 对象（在 jQuery 1.5 之前是 XMLHttpRequest 对象）。到 jQuery 1.5 时，还可以指定一个包含设置函数的数组。每个指定的函数在回调被调用时执行
timeout	通过这个选项，可以在客户端指定一个以毫秒表示的时段，表示等待 Web 服务器响应的时长。这一指标覆盖全局变量 timeout 的值（如果该值已经通过 $.ajaxSetup() 设置）
traditional	可以用这个选项指定序列化类型。true 表示按照传统方式执行，否则采用递归方式

（续）

名　称	描　述
type	请求的类型（方法），默认为 GET
url	包含请求地址的字符串。默认值为当前网页
username	字符串形式的用户名，用于有访问限制的 HTTP 访问
xhr	用于创建 XMLHttpRequest 类型对象的回调函数
xhrFields	在 jQuery1.5.1 中，可以指定一个映射，在其中可以指定扩展原生 XHR 对象的值对

从上面这张大表中很容易看出，$.ajax() 方法就像一把瑞士军刀。诚然，没有必要单独地配置大部分的选项，也完全没有必要访问这些低级函数。但是这些选项的存在是件好事，可以避免 jQuery 的默认技术无法正常工作的情况。我们来仔细地观察几个例子。

10.10.2 JSONP 请求

程序清单 10.35 展示了使用 JSONP 的一个示例（ch10_12_ready.js）。

程序清单 10.35　用 ajax() 发出的 JSONP 请求

```
01 $(function(){
02   $("button:first").click(function(){
03     $.ajax({
04       dataType: 'jsonp',
05       jsonp: 'jsonp_callback',
06       url: 'sendJSONP.php',
07       success: function(data){
08         $("div:first").text(data.website);
09       }
10     });
11   });
12 });
```

可以看到使用选项描述性规格的 $.ajax() 的基本使用方法。在第 4 行中，数据类型被设置为 JSONP；下一行指定 JSONP 回调。在第 7 ～ 9 行中，可以看到成功时调用的匿名回调函数。

现在，还需要考虑某些服务器端的细节，它们超出了本书的范围。但是对于 PHP，我们希望指出最重要的一点，以便使 JSONP 请求的 JSON 代码能够封装到一个函数中，以客户端能够处理的方式发送。程序清单 10.36 展示了所请求的 PHP 文件 sendJSONP.php 的结构。

程序清单 10.36　以合适的形式发送 JSON 数据

```
01 <?php
02   $data = json_encode(Array(
03     "website => "www.rjs.de"));
04   echo $_GET['jsonp_callback'] ."(". $data .");";
05 ?>
```

最重要的部分是 json_encode() 函数，该函数可以在 PHP5.2 之后的版本中找到。如图 10-15 所示。

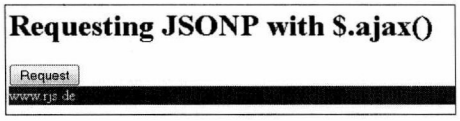

图 10-15 JSON 数据到达

10.10.3 加载和执行 JavaScript 文件

我们已经从 getScript() 中了解了 JavaScript 文件是如何加载和执行的。程序清单 10.37 是使用 ajax() 的低级等价程序。

程序清单 10.37 加载和执行一个 JavaScript 文件（ch10_13_ready.js）

```
01 $(function(){
02  $("button:first").click(function(){
03    $.ajax({
04      dataType: 'script',
05      type: 'GET',
06      url: 'lib/random.js',
07      success: function(data){
08        $("#output").text(randomNumber());
09      }
10    });
11  });
12 });
```

重要的部分是数据类型、JavaScript 程序库规格和数据请求方法。关于这个例子没有需要多加说明的地方。

10.10.4 发送数据并评估成功

我们再来看看如何通过 $.ajax() 发送数据。我们使用 POST 方法，但是这本身没有关系。对于 GET，程序看上去也是一样。我们使用 Web 表单 ch10_3.html 作为基础，它的新版本 ch10_14.html 的不同之处仅仅是引用了 JavaScript 文件 ch10_14_ready.js。

程序清单 10.38 通过 POST 发送数据

```
01 $(function(){
02  $("button:first").click(function(){
03    $.ajax({
04      data: {
05        username: $("input:first").val(),
06        password: $("input:last").val()
07      },
08      type: 'POST',
```

```
09      url: 'ch10_3_post.php',
10      success: function(data){
11        $("#output").html(data);
12      }
13    });
14  });
15 });
```

通过 data 选项，可以简单地指定包含发送数据的查询串，或者像我们的例子一样使用 Map 对象。如图 10-16 所示。

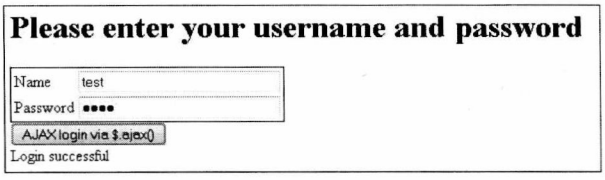

图 10-16 数据已经被服务器接收并处理了，然后将响应发送到客户端

10.10.5 $.ajax() 的扩展技术

在 jQuery 1.5 中，引入了 $.ajax() 的 3 个扩展功能（前面已经简短地提到过，在此将更加详细地讨论）。

前置过滤器

前置过滤器（prefilter）指的是任何 AJAX 请求被发送之前和 $.ajax() 处理任何选项之前执行的回调函数。

前置过滤器通常类似程序清单 10.39。

程序清单 10.39 前置过滤器回调函数

```
$.ajaxPrefilter(
  function( requestOptions, originalOptions, jqXHR ) {
    /* Modify options, control the original options, store jqXHR object etc */
});
```

转换器

转换器指的也是一种新型的回调函数。转换器在服务器以预期之外的数据类型发送响应数据时调用。可以在回调函数中采取相应的措施，转换数据类型或者引入自己的数据类型。转换器保存在 ajaxSettings 中，可以全局添加，如程序清单 10.40 所示。

程序清单 10.40 典型的转换器

```
({
  converters: {
    "text mydatatype": function( textValue ) {
      if ( valid( textValue ) ) {
```

```
      if ( valid( textValue ) ) {
        // some logic
        return mydatatypeValue;
      } else {
        // indicate parse-error
        throw exceptionObject;
      }
    }
  }
});
```

在关于数据类型的小节中已经提到，可以用转换器创建自定义数据类型（数据类型必须采用小写字母）。

如果观察前一个程序清单中的典型转换器，就可以请求类型为 mydatatype 的数据，如程序清单 10.41 所示。

程序清单 10.41　使用自定义数据类型

```
$.ajax( url, {
  dataType: "mydatatype"
});
```

分发器

jQuery 中的分发器（transport）是提供以下两个方法的一个对象：

- send()
- abort()

上述两个方法都由 $.ajax() 在内部使用，可以用于扩展 $.ajax()。但是，jQuery 文档还指出，分发器应该只作为最后手段，在前置过滤器和转换器都不足以影响 AJAX 请求时使用。

因为每个请求需要自己的分发器对象实例，这些对象不能被直接注册。作为替代，应该提供一个函数来返回这样的对象。这种生成对象的函数被称为工厂（Factory）。分发器工厂的注册如程序清单 10.42 所示。

程序清单 10.42　典型的分发器

```
$.ajaxTransport( function( requestOptions, originalOptions,  jqXHR ) {
  if( /* conditions for transport */ ) {
    return {
      send: function( header fields as map, callback ) {
        /* code for sending */
      },
      abort: function() {
        /* code in case of abort*/
      }
    };
  }
});
```

10.11 小结

本章介绍了基于 AJAX 以及代用数据格式的现代 RIA 编程的基础知识和特殊性。jQuery 覆盖了整个主题，通过添加 JSONP 和 getScript 等有趣的扩展，产生了比现有的 AJAX 更实用的技术。备受瞩目的 jQuery 方法如 jQuery.get(url, data, Callback，Type）和 jQuery.post(Url, Data, Callback, Type) 使 AJAX 的处理变得非常容易，而 $.ajax() 使我们在必要的时候可以有完全的控制。jQuery1.5 特别添加了强大的 AJAX 请求控制功能。在本章中大家还学到了如何在客户端评估和处理以结构化格式发送的服务器响应。除了 XML 以外，JSON 也适合于这一目的。

第 11 章
jQuery UI

在本章中，我们讨论 jQuery 框架（更准确地说——基于框架的一个扩展）提供的视觉控制元素。这些元素被统称为 jQuery UI。[⊖]这些功能是一组对 jQuery 进行补充的强大交互插件，以及构建图形界面的窗口组件，这样构建的界面还可以由具备预定义主题的层叠样式表（Cascading Style Sheet，CSS）库加以扩展。在此之上，jQUery UI 提供了 jQuery 默认特效的各种不同扩展。大部分组件都可以广泛地进行配置。

注意

一方面，jQuery UI 的组件强大且引人注目；另一方面，这个程序库只提供目前在所有主流 Web 框架中可用的功能。在这一个领域中，人们熟悉的框架 Dojo、YUI 或者 jQuery 都很相似。

但是，尽管 jQuery UI 的功能很强大，但是使用预先设计的结构实际上很简单，通常不言自明。

在 jQuery 框架中，还有一些基于 Web 的工具，使之更加易于使用。我喜欢这样形容它们：获得越多，所需要付出的代价越多。据此，我们在本章中没有必要详细地讨论每个组件，而只要粗略地了解一下最重要的元素，以及 jQuery UI 的基本用法，然后详细地了解几个精选的组件和方法。

11.1 什么是 jQuery UI

统称为 jQuery UI 的程序库提供了对交互和动画（例如拖放、排序和选择）的低级抽象。我们还可以在这个程序库中找到高级别的扩展特效和适应性很强的窗口组件；这些效果都基于常规的 jQuery JavaScript 程序库。jQuery UI 的功能都可以配置，受到大部分现代浏览器

⊖ UI——用户界面（User Interface）。

的支持。有趣的是，在本书编写的时候，jQuery UI（1.8.24）的浏览器需求低于 jQuery 当前版本自身的要求[⊖]（明确支持的是 Internet Explorer 6.0+[⊖]，Firefox 3+，Safari 3.1+，Opera 9.6+ 和 Google Chrome）。

jQuery UI 在常规的 jQuery JavaScript 程序库中没有提供，这是因为 jQuery UI 是作为整体 jQuery 框架的一个单独项目来管理的。该项目的首页在 http://jqueryui.com。

11.1.1 支持交互的组件

jQuery UI 框架包含一些用于支持复杂交互行为如拖放（draggable() 和 droppable() 方法），改变大小（resizable()）、选择（selectable()）和排序（sortable()）等的组件。这些处理显著地得到了简化。一般来说，只需要一行源代码就可以实现对一个元素或者一组元素的支持，我们很快将讨论这一主题。

11.1.2 窗口小部件

窗口小部件（Widget）一般是较为复杂的元素，可以用它们组成图形用户界面（GUI）。利用超文本标记语言（HTML）或者可扩展 HTML（XHTML），可以轻松地生成按钮、文本输入框、标签、标题、表单和表格等简单元素。但是在桌面应用领域有一些更复杂的 UI 元素，在没有 CSS/JavaScript 或者附加程序库的情况下，它们没有受到浏览器的支持（例如，用于合格日期和时间输入的输入字段，对话框，滑动条和进度条）。这些 UI 组件都必须用纯粹的 HTML 和 CSS 渲染，只能靠 JavaScript 实现逻辑。这绝不简单，特别是考虑到不同浏览器的不同反应。通过窗口小部件，网页开发人员能够很轻松地使用这些用于富互联网应用程序（RIA）的更为复杂的元素。jQuery UI 提供了如下的窗口小部件：

- Accordion
- Autocomplete
- Button
- Datepicker
- Dialog
- Progressbar
- Slider
- Tabs
- Tooltip（jQuery UI 1.9 及更新版本）

此外，相对新的工具脚本 Position 使我们能够将每个窗口小部件根据窗口、文档、特定元素或者鼠标指针的相对位置放置。

⊖ 以前有其他的变通方法。但是这可能在 jQUery UI 新版本发行的时候出现变化。

⊖ 但是，Internet Explorer 没有受到完整的支持（特别是框架的 CSS 主题）。例如，现在的 CSS 主题不支持 Internet Explorer 的圆角，也不支持创建自定义 CSS 主题的工具。

注意

你可能对 jQuery UI 中的窗口小部件数量之少感到惊讶。从其他框架中你可能了解更多的组件。只要想想导航树就知道了。但是，在 jQuery 中有插件的概念，通过这些外部插件扩展了许多窗口小部件。大家将在下一章中了解这一概念。

11.1.3 扩展特效

我们已经知道，jQuery 本身已经提供了许多特效。利用 jQuery UI，这些特效得到了多种其他特效方法的扩展，例如 effect()、show()、hide()、toggle()、animate()、addClass()、removeClass() 和 toggleClass()。大部分特效方法的名称大家应该都很熟悉。实际上，这些已经令人熟知的方法通常只是在 jQuery UI 中扩展了功能，但是有时候这种扩展的程度很可观。例如，在 jQuery UI 中还可以通过 animate() 修改动画中的颜色，这在基本版本中无法做到。不过，本质上使用同名方法的方式与核心框架中相同，所以学习它们很简单。

提示

可以在 http://jqueryui.com/demos/ 中找到 jQuery UI 的文档。jQuery UI 的更多帮助和特殊问题可以访问支持中心：http://jqueryui.com/support。

11.1.4 主题框架和 ThemeRoller

jQuery UI 的另一个组件是基于 CSS 的大规模主题框架。这是一个纯粹的 CSS 库。它还有一个 ThemeRoller 工具，用于选择、改编和使用预建的设计，以及一个包含框架可用组件预建设计的陈列站点。

注意

本书基于 jQuery1.8.24[⊖]稳定版本和 1.9 Beta 版本。在后几个版本中不可能出现不兼容的问题，但是，我们当然不能完全排除这种可能性。未来版本的许多情况已经为人所知，在版本 2.0 中，我们将会看到一个具备完全原型继承的窗口小部件工厂、无需调用 init() 方法的直接实例化、改进的触发器处理等功能。应用程序编程接口（API）也将得到清理。

11.2 入门

如果查看了 http:/jqueryui.com/demos/ 上的官方示例，jQuery UI 提供的潜力就很容易掌握。该网站提供了多个交互式示例，演示了可用的组件，尤其是有一个详尽的描述。

⊖ 本质上，你可以只依靠稳定版本号，可以用尚未完全发行的版本进行大部分试验，但是不要在实践中采用它们。

Autocomplete 的例子如图 11-1 所示。

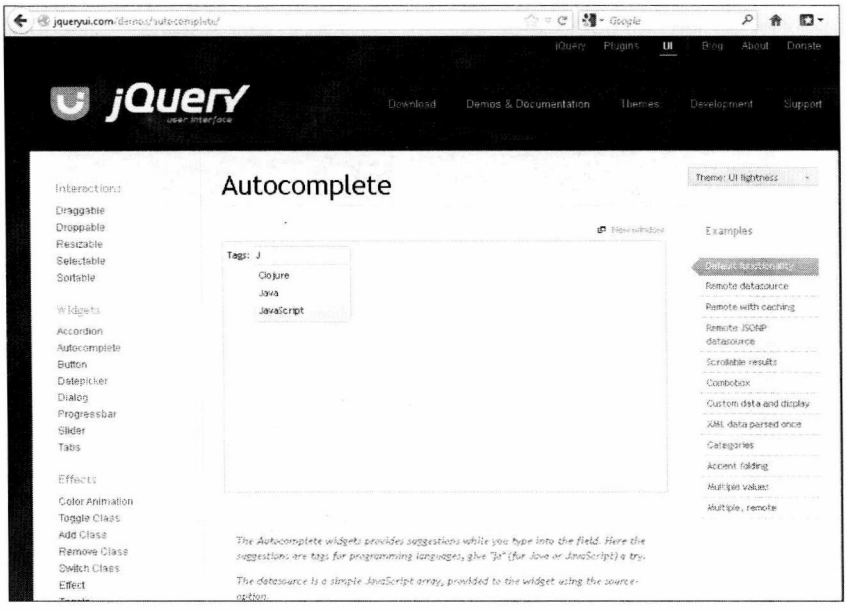

图 11-1 Autocomplete 的示例

每个例子的源代码可以通过页面上的“View Source”链接直接查看。

提示

注意每个例子右侧的“Examples”栏目。在这个标题下，可以选择一个组件或者窗口小部件的不同变种。可以获得这种形式的交互示例和一个完整的列表。可以通过很快将要讨论的选项、事件和方法来实现这些改编。

11.3 如何使用 jQuery UI

在查看示例之后，你可能决定要将这个程序库用于自己的网页的全体或者部分。下面的小节说明使用 jQuery UI 的基本方法。

11.3.1 下载和 ThemeRoller

前面已经提到，jQuery UI 不包含在常规的 jQuery 程序库中。所以，必须单独下载这个扩展程序库，使其可用于自己的网页。

下载构建器

从 jQuery 项目的首页，可以通过对应的“Download”（在顶部显示为一个选项卡）链接

下载该程序库。作为替代，可以在起始页面寻找一个“Bulid”链接来自定义下载。在两种情况下，都会到达同一个 jQuery UI 下载页面：Download Builder（下载构建器）。如图 11-2 所示。

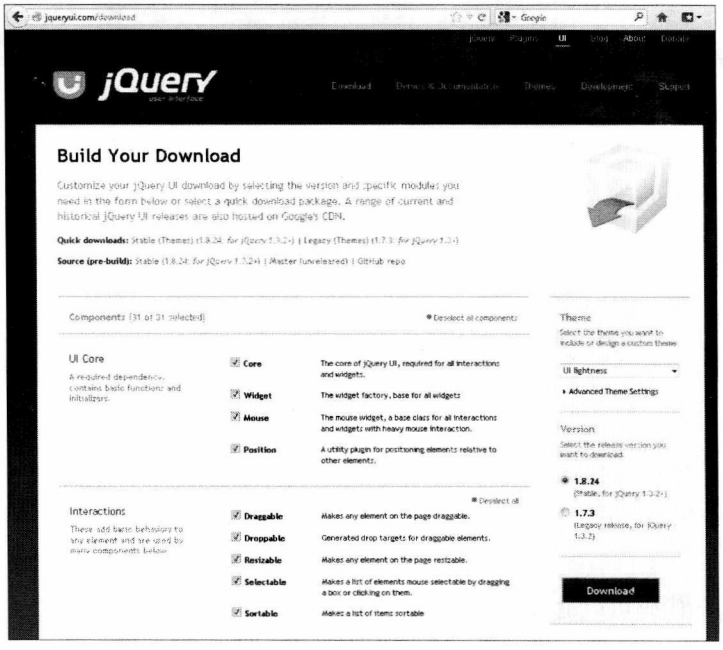

图 11-2 选择所需资源

在下载构建器的主栏目中，可以看到 jQuery UI 中所有 JavaScript 组件的列表，这些组件被分为多个组。具体地说，包括 UI core(UI 核心)、Interactions(交互)、Widgets(窗口小部件)和 Effects（特效）。很容易发现，可以选择需要支持的单独组件。如果确定自己的 Web 应用中不需要某些组件，可以取消选择。这样做使得程序库更小，当然也就有利于访问者的下载时间。也可以单独使用程序库的其他组件中的大部分项目组件（但不是所有的）。[⊖]所以，要考虑特殊功能的潜在相关性，在必要时集成其他文件。可以在文档中指定每个功能的相关性，从工具本身也能够得到支持。如果简单地启用 Download Builder 中组件旁边的复选框，所有必需的相关组件在下载期间自动解析，必要的程序库组件被选中。所选的组件合并到一个 JavaScript 程序库中，以后必须将其集成到自己的网页中。[⊖]

如果以后发现确实需要取消选择的组件，可以简单地重新加载扩展的程序库，用新生成的程序库替换旧的 JavaScript 文件。但是我建议一开始就下载所有组件。

在 Download Builder 的右侧，可以选择想要下载的 jQuery UI 版本。根据项目所处

⊖ 一般的规则是，总是需要核心组件。

⊖ 还将得到一个必需的 CSS 库（称作主题，后面将讨论）。

阶段，你可能发现有多个稳定版本可供选择。如果必要，通过 Google CDN[⊖]（http://code.google.com/apis/libraries/devguide.html#jqueryUI）也能获得其他版本。[⊖]该网页上的许多链接指向各种不同的版本，有些版本更新，但可能还不稳定。

在 Download Builder 的右侧，还可以看到 Theme（主题）分类。从下拉式窗口中，可以选择窗口小部件和组件使用的 CSS 主题。根据主题，窗口小部件和组件的外观（有时候甚至包括行为）将会改变。

警告

没有链接的 jQuery UI CSS 主题，可能无法正确地使用组件和窗口小部件。

jQuery UI 有许多不同的主题，但是也可以定制主题。例如，在下拉菜单之上有一个链接，可以用它设计自定义主题。也可以通过 Themes 选项卡到达指定目标。这将把你带到一个网站，在那里对 CSS 主题进行视觉上的更改。如图 11-3 所示。

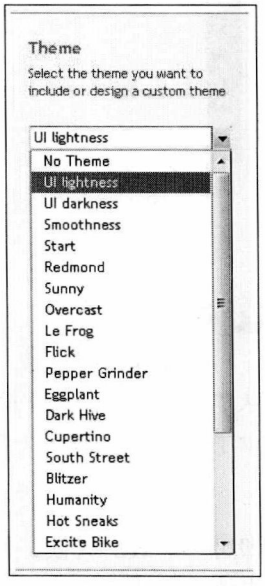

图 11-3 主题下拉式菜单

ThemRoller 和 jQueryUI CSS 框架

如前所述，jQuery UI 框架中的主题只不过是高级的 CSS 库，没有它们，框架中的组件和窗口小部件就无法在网页中显示。这很重要，但是如果你有 CSS 的基础知识，创建或者修改底层文件并不困难。为了简化创建和改写 CSS 规则这种并不困难但却费时的工作，jQuery UI 提供了一个名为 ThemRoller 的特殊 RIA。你现在已经进入这个网站，我们来更仔细地看看它。

如果观察 ThemeRoller 网页，就会看到左侧的用户界面。可以用它设计所有 jQuery UI 窗口小部件使用的元素（Roll Your Own）。单击树状结构，可以展开分支，设置不同类别的 CSS 属性值，在某些情况下，操作甚至是图形化的。如图 11-4 所示。

提示

在该栏目的右侧有各种组件的示例，立刻就可以看到自己的更改对当前设置的影响。

ThemeRoller Gallery 选项卡显示了选择的一些预定义主题。这些主题和你在下载构建器中看到的一样。在这里，可以直接看到选中的主题对窗口小部件的影响（预览可以交互式变化）。如图 11-5 所示。

⊖ CDN 是 Content Distribution Network 或者 Content Delivery Network（内容分发网络）的简称，指的是通过互联网连接的本地分发服务器网络，内容通过该网络以尽可能高效的方式交付。数据在互联网上缓存。

⊖ 一般来说，使用最新版本是个好主意，但是在某些情况下你必须支持旧的浏览器，使用旧版本的 jQuery 或者 jQuery UI 就有意义。

图 11-4 通过 GUI 和预览改写不同的 CSS 属性

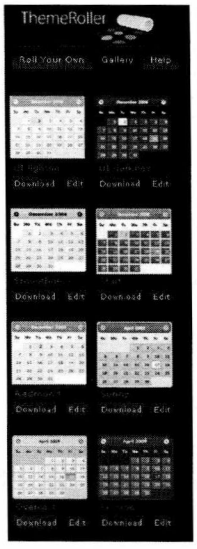

图 11-5 可以用 Gallery 选择预定义主题

提示

如果选择库中的某个特定主题，它就被用作 Roll Your Own 中设置的基础。所以，如果从陈列中选择一个和你心中所想很接近的主题，只需要改变你还不完全满意的那些细节就可以了。当然，也可以不加修改地使用一个主题。

一旦设计完成，只需要下载它，使其可在服务器上访问，并将其集成到自己的网页中即可。下载的时候，你会发现 ThemeRoller 网页的左上角有一个 Download Theme 按钮。单击该按钮或者 Download 选项卡，就会进入 Download Builder，选择或者自定义的主题将在 Theme 下拉式菜单中被预先选中。

提示

可以在任何必要的时候，用编辑器手工改写下载的 CSS 文件。也可以重新在 ThemeRoller 中加载该文件，用那里提供的方便工具编辑。在 CSS 文件中，你将看到程序清单 11.1 中展示的文本。

程序清单 11.1 CSS 文件中注释区域的 URL 开头

```
"To view and modify this theme, visit ..."
```

只需要复制上述文本之后的长 URL，在浏览器中打开。这将把主题加载到 ThemeRoller 供编辑，包含所有以前自定义的设置。可以进行更改，然后再次下载。

11.3.2 在网页上使用 jQuery UI

下载完成后，你会得到一个压缩的 Zip 文件，可以在服务器上解压它，然后将其集成到网页中，就像 jQUery 程序库那样。

警告

至于版本，记住，jQuery UI 版本总是和一个特定的 jQuery 版本搭配，如果版本不匹配，可能会发生兼容性问题。[⊖]例如，jQuery UI 1.8.24 基于 jQuery 1.8 和更高版本，可以结合使用 jQuery 1.6 和较早的版本 jQuery 1.2.6。阅读网页上关于你将使用的 jQUery UI 特定版本的说明。幸运的是，jQuery UI 的下载文件包含了适合的 jQuery 版本。

解压后的程序库包含如下结构：

- /css/（主题；也可以在这里放入其他主题）
- /development-bundle/（演示和文档；这个目录没必要上传到服务器上供访问使用）
- /js/（jQuery 和 jQuery UI 的 JavaScript 文件）
- index.html（演示和文档的索引页面；不需要上传该文件）

在文件 index.html 中可以看到，要使用 jQuery UI，需要在网页中集成 3 个文件。表 11-1 中是一些必要的引用的列表。

表 11-1 必要的引用

描　述	示　例
特定 CSS 主题的引用。这个链接必须在脚本引用之前集成。如果打算修改应用程序布局，只需要引用另一个 CSS 主题。还应该在所有自定义 CSS 库之前集成该引用	`<link type="text/css" href="css/dot-luv/ jquery-ui-1.8.24.custom.css" rel="Stylesheet" />`
对常规 jQuery 程序库的引用。该程序库也可以通过 jQuery UI 访问，但是，当然可以保持自己的路径，只要版本合适。该引用必须先于 jQuery UI 程序库的引用	`<script type="text/javascript" src="js/ jquery-1.8.2.min.js"></script>`
对 jQuery UI 程序库的引用。之后，可以编写对自己的 JavaScript 文件的引用	`<script type="text/javascript" src="js/ jquery-ui-1.8.24.custom.min.js"></script>`

正如前面所提到的，jQuery 和 jQuery UI 也可以通过各种内容分发网络（CDN）访问。通过这些网络，可以用对应的 URL 将 jQuery UI 和 jQuery 集成到网页中。这减轻了你的 Web 服务器的压力，但是以后将依靠 CDN 提供商确保资源的可靠访问。

警告

当然，必须改写用于特定情况的路径和名称，所以必须确保 CSS 主题名称正确选择，你将要使用的 JavaScript 程序库被正确引用。遗憾的是，jQuery 框架（更确切地说是使用它且有问题的浏览器）有一种倾向——在出现某些错误时显示不具体的错误信息。我们最多在

⊖ 但是，通常只有在 jQuery 本身的版本过低时才会发生。

错误面板上看到一个错误，但是屏幕常常简单地显示空白，特别是在引用不正确的情况下。

11.3.3 jQuery UI 样板网页

如果我们使用 jQUeryUI 默认主题并用现有的 jQuery 程序库和引用结构代替 jQuery UI 自带的 jQuery 版本，使用 jQuery UI 的样板模板如下所示。

程序清单 11.2 jQuery UI 模板

```
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="Content-Type"
     content="text/html; charset=utf-8" />
   <title>jQuery UI</title>
   <link type="text/css" href=
     "lib/css/ui-lightness/jquery-ui-[...].custom.css"
     rel="Stylesheet" />
   <link rel="stylesheet" type="text/css"
     href="lib/[...].css" />
   <script type="text/javascript"
     src="lib/jquery-[...].min.js"></script>
   <script type="text/javascript" src=
     "lib/js/jquery-ui-[...].custom.min.js"></script>
   <script type="text/javascript" src=
     "lib/[...].js"></script>
 </head>
 <body>
 </body>
</html>
```

11.4 使用 jQuery UI 中的组件

网页中引用了程序库之后，马上可以在源代码中使用组件和窗口小部件，这和使用常规的 jQuery 功能一样。

提示

jQuery UI 的组件和窗口小部件在使用具有相匹配的 CSS 类的主题时才能正常工作。类明确地对针对这些组件和窗口小部件。但是，这并不能阻止我们以自己喜欢的方式使用 CSS 框架的类。这意味着，可以为网页的任何元素指定组件或者窗口小部件标准使用的类。例如，可以像程序清单 11.3 中所示的那样做。

程序清单 11.3 将 CSS 框架中的一个类应用到类型为 h1 的元素

```
<h1 class="ui-corner-all">A heading</h1>
```

提示

使用来自可用 jQuery UI CSS 库中的一个类，并将其赋予常规的 1 级标题，尽管它并不是 jQuery UI 的组件。这个类提供了圆角。这种尽可能使用来自 jQuery 可用 CSS 库类的方法很受推崇。甚至可以在应用程序的“常规”HTML 元素上实现统一的设计。

11.4.1 默认设置

一般来说，即使使用 jQuery UI 中的复杂组件也只需要一行代码，特别是在使用文档中显示为默认的组件默认版本时。[⊖]下面我们来看看，在网页中使一个对象可以通过拖放来移动有多么简单。

注意

我们在本小节中用作例子的组件纯粹被视为占位符。当然可以使用其他组件。

程序清单 11.4 使一幅图像可以拖动（ch11_1.html）

```
...
18  <body>
19    <h1>Above</h1>
20    <img id="draggable" src="images/pic1.jpg" />
21    <h1>Below</h1>
22  </body>
23 </html>
```

在本书的配套网站上，可以看到我们在这里没有列出的代码行，包含来自样板模板的引用以及对外部 JavaScript 和 CSS 文件的引用，它们的名称和平常一样，是基于 HTML 文件的名称。下面是包含编程逻辑的 JavaScript 文件 ch11_1_ready.js。

程序清单 11.5 图像变成可拖动的

```
01 $(function(){
02  $("#draggable").draggable();
03 });
```

还能有更简短的程序吗？[⊖]如图 11-6 是原始图像。

第 2 行是为了使网页上的一个元素或者一个元素可用鼠标拖动所需做的所有事情。只要为想要拖动的元素使用 draggable() 方法就行了。用 Firebug 分析网页会发现这很有趣。虽然源文本中没有指定 CSS 定位，draggbel() 方法已经生成了一个 style 特性，并在后台指定 position:relative 属性。效果如图 11-7 所示。

⊖ 所有其他变种开始时都被隐藏，你必须在右侧选择它们。

⊖ 拖动元素的大量功能用一条简单的语句提供。

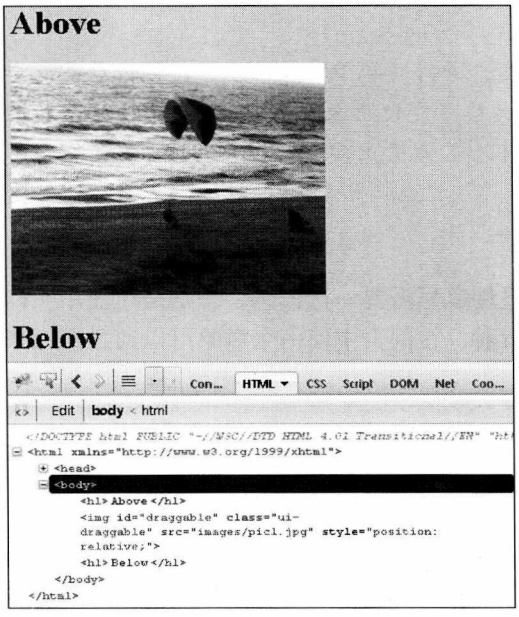

图 11-6 开始，图像位于两个标题之间，注意 Firebug 中的相对位置

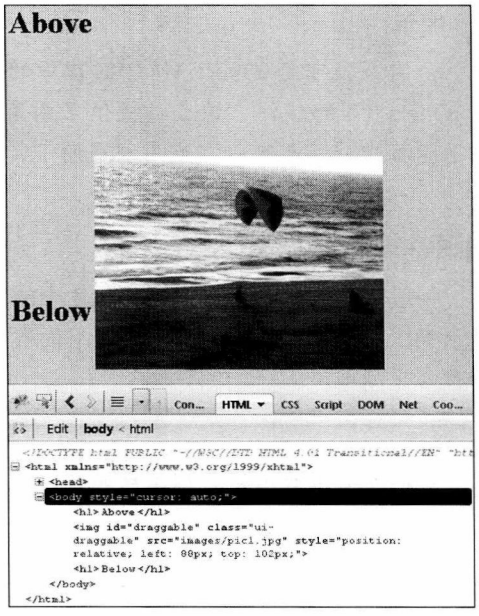

图 11-7 图像已经用鼠标移动

注意

乍一看，让网页中的某个元素可以用鼠标移动似乎是小把戏。如你所知，可以通过事件对象读出鼠标的坐标和按下的按钮。对象的位置可以轻松地通过 CSS 的 position、left 和 top 指定和改变。这样，只要结合这两个方面就完事了。事实并非如此，因为如果尝试这个例子，就会发现图像上下的标题在你拖动图像时保持在原始位置。此外要注意，原文本中没有包含图像的任何 CSS 定位。在许多其他的窗口小部件、组件和特效中，jQuery UI 中的一个方法确保了各种 CSS 格式在后台自动完成。因此，后台发生许多事情，掩盖了这种方法极其简单的用法，这些细节很容易在看似简单的人工编程中被忽略。

11.4.2 组件和窗口小部件的一些基本原则

有些基本原则几乎总是适用于 jQuery UI 组件和窗口小部件的使用。如果从一开始就了解它们，就会发现 jQuery UI 很容易掌握：

- 和 jQuery 方法不同，创建组件和窗口小部件的方法不能用句点标记法链接。单独的对象过于复杂，而且一般没有多大的意义。
- 用默认设置创建组件或者窗口小部件时，使用不带参数的对应方法。
- 对 div 元素应该应用多个 jQUery UI 窗口小部件。这将使来自 HTML 的“溢出效应”[⊖]减到最小。在必要的时候，jQuery UI 方法会在后台改写元素。但是，可以使

⊖ 例如，对于对话框、按钮、进度条或者滑轨来说。

用其他基本元素，尽管不建议这么做。

- 窗口小部件明确使用默认主题 CSS 文件。通常在下载时用 ThemeRoller 工具创建对应的 CSS 文件。但是如果希望或者需要进行单独更改，[⊖]每个窗口小部件都有一个以 jquery.ui 开头，后面跟上相关窗口小部件名称的 CSS 文件（例如，jquery.ui.slider.css）。在这个文件中，将找到可以用编辑器自定义的类，以 ui-[窗口小部件 / 超类名称] 开头（例如 ui-slider，ui-slider-horizontal，ui-widget，ui-widget-content，ui-corner-all）。如果分析窗口小部件的生成代码，也可以在 Firebug 中查看这些类。[⊖]更多细节请参考 Theming 选项卡下每个窗口小部件的说明。但是，当然也可以用自定义的 CSS 规则影响窗口小部件和组件。

11.4.3 组件的属性 / 选项

大家在第一个例子中已经看到，使用 jQUery 的 UI 组件通常很容易。如果喜欢默认设置，常常只需要一行代码。如果想要改写组件，只需要覆盖需要的属性或者选项。这告诉我们，必须了解特定组件提供的选项。我们现在可以列出和描述所有 UI 组件的属性，但是我认为这是多余的，因为如果在 http://jqueryui.com/demos/ 上选择，就可以通过每个组件的文档找到列表和描述。只需要单击“ options”选项卡，然后就能看到一个组件或者窗口小部件可以指定哪些选项。只需指定创建组件时需要的相关组件或者窗口小部件的选项。如图 11-8 所示。

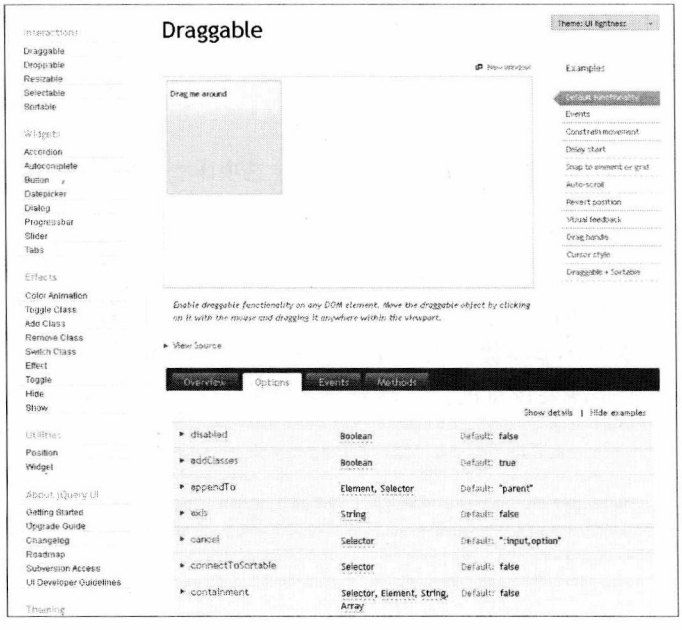

图 11-8 draggable 选项

⊖ 但是很少有必要这么做。

⊖ 在那里你还将找到标识这些类所需的信息，可以将这些类应用到窗口小部件之外的元素，完成更为统一的设计。

只需要知道特定选项的作用，当然，还要知道如何应用它。但是文档中指定的每个选项都很灵敏，如果单击其中一个选项，就会看到一个包含效果说明、使用方法、允许值的下拉菜单和一个程序清单示例。

在这里，还可以看到每个选项的默认值。如果没有指定进一步的选项，默认值就会生效。最后一个问题是如何使用选项。可以设置选项或者获取其取值。如图 11-9 所示。

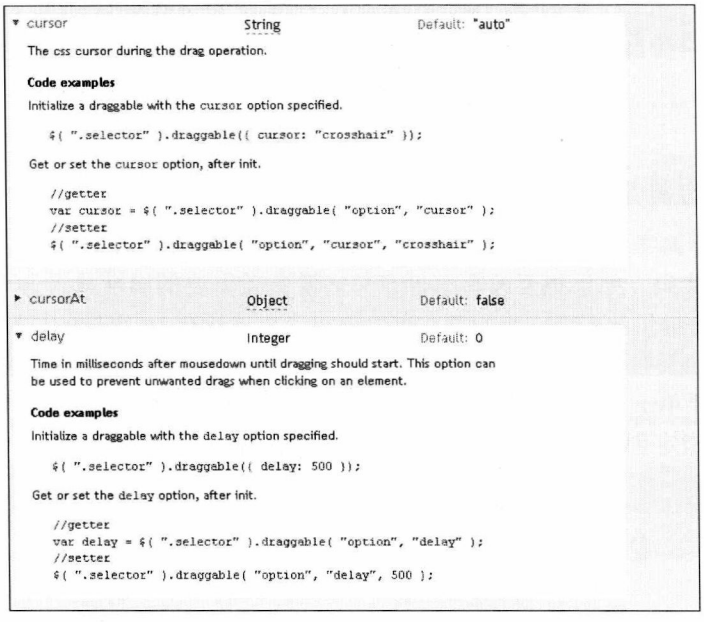

图 11-9 cursor 和 delay 选项详情

设置选项值

如果打算修改默认值，可以在创建组件或者窗口小部件时使用选项。通常，只需在生成组件对象时用一个参数声明选项，例如 jQuery 中常见的 Map(特别在想要指定多个选项时)。

作为替代，可以使用 3 个参数。第一个参数中指定“ option”，第二个参数是字符串形式的选项名称，第三个是需要设定的值。这种标记法在只指定一个选项时很有意义，可以作为一种设值方法。[⊖]

获取选项值

取值方法与设值方法构造方式类似，采用 3 个参数；只要省略第三个参数，返回值就是特定的选项值。

提示

替代的方法是，有些方法将某些组件和窗口小部件的选项值或者属性当前值作为返回

⊖ 在我的实验中，使用 3 个参数的标记法出现了奇怪的不可靠表现，但是我无法确定原因，所以它可能只出现在某些个体中。我建议你总是用 Map 设置选项（即使你只有一个选项)。

值。接下来我们将仔细地研究这一点。

程序清单 11.6 使两幅图像可拖动，并输出选项值（ch11_2.html）

```
...
18  <body>
19    <h1>Two draggable images</h1>
20    <img src="images/pic1.jpg" />
21    <img src="images/pic2.jpg" />
22    <div></div>
23  </body>
24 </html>
```

在上述网页中，可以看到两幅图像变成可拖动，且修改几个参数。我们还将获取选项值（ch11_2_ready.js）。

程序清单 11.7 设值方法和取值方法

```
01 $(function(){
02  $("img:first").draggable({
03    "opacity": "0.35"
04  });
05  $("img:last").draggable({
06    cursor: 'crosshair',
07    delay: 500
08  });
09  $("div:first").html(
10     $("img:last").draggable("option", "cursor"));
11 });
```

第一幅图像的移动用一个选项进行初始化，第二幅图像的移动用两个选项初始化。在第 10 行中，可以看到我们如何读出选项值并进一步使用。[⊖]如图 11-10 所示。

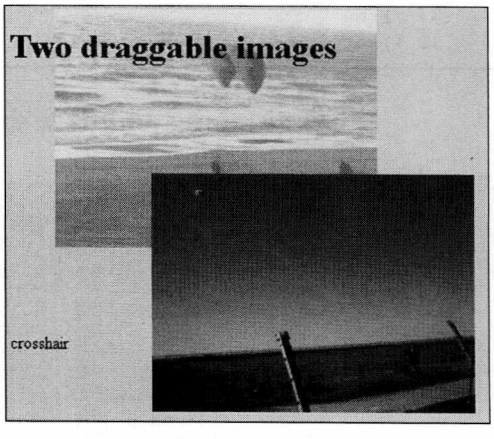

图 11-10 使用取值方法和设值方法

11.4.4 组件的方法

每个组件也有方法——因为组件当然也可以看作对象，而对象通常除了属性之外还提供方法。具体地说，这些方法在 jQuery UI 中用于构建（没有参数或者带有选项）、启用、禁用、配置或者删除对象。“方法”这一术语在 jQuery UI 组件和窗口小部件的上下文中与“常规”对象中的含义略有不同。在 jQuery UI 框架文档中，Methods 选项卡总是只描述组件和窗口小部件的一个方法。但是通过参数的默认值，这个方法可以实现不同的行为，在文档中列出了这些

⊖ 在例子中，只是简单的输出。

参数默认值。[⊖]这些参数值指定了执行和行为，以及方法相关的返回值（ch11_3.html）。

程序清单 11.8 使一个对象可拖动，然后再次禁用

```
...
18  <body>
19    <h1>Making an image draggable and then
20       disabling it again</h1>
21    <img src="images/pic3.jpg" /><br />
22    <button>Enable</button>
23    <button>Disable</button>
24  </body>
25 </html>
```

在网页中，将看到一幅图像在加载时变成可拖动的。通过两个按钮，我们可以动态地启用和禁用这种行为（ch11_3_ready.js）。如图 11-11 所示。

程序清单 11.9 动态启用和禁用拖动

```
01 $(function(){
02  $("img:first").draggable({
03    cursor: 'crosshair',
04    "opacity": "0.35"
05  });
06  $("button:first").click(function(){
07    $("img:first").draggable("enable");
08  });
09  $("button:last").click(function(){
10    $("img:first").draggable("disable");
11  });
12 });
```

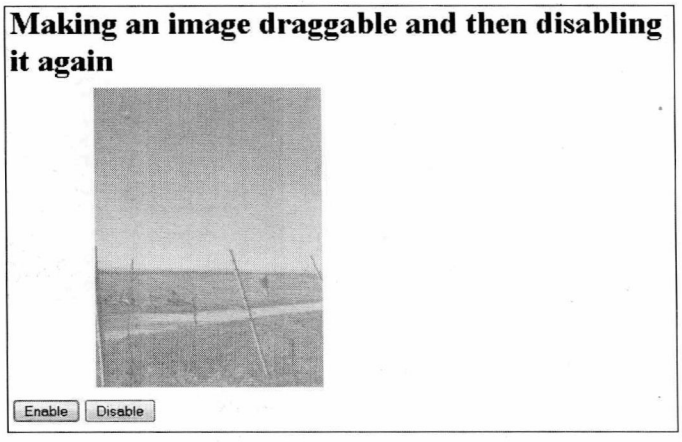

图 11-11 图像被拖动

⊖ 这一部分更恰当的命名是“组件方法参数”（这个名称当然过长了，但是更为合适）。

在第 7 行和第 10 行，可以看到 draggble() 方法是如何用某些参数调用的。单击第一个按钮启用拖动，单击第二个按钮禁用拖动。如图 11-12 所示。

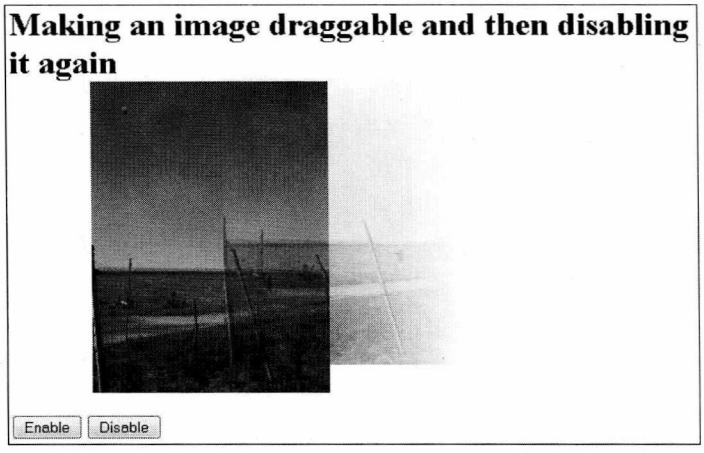

图 11-12 不透明度和光标表示图像不能被拖动

我们来看另一个例子，例中通过一个方法处理窗口小部件的属性值。例如，我们使用一个 Slider 类型的对象（ch11_4.html）。

程序清单 11.10 一个滑动条

```
...
18  <body>
19    <h1>A slider</h1>
20    <div id="sl"></div>
21    <form>
22      <input type="text" /><br/>
23      <input type="button" value="Set value" />
24    </form>
25    <div></div>
26  </body>
27 </html>
```

在网页中，我们希望将一个 div 元素变成滑动条。你将会再次看到，基本设置只需要一行代码。但是我们将其变得更加“复杂”，设置几个选项。我们希望通过按钮改变滑动条的值，并将滑动条设置为在输入字段中输入的值（ch11_4_ready.js）。

程序清单 11.11 处理属性值

```
01 $(function(){
02  $("#sl").slider({
03    orientation: 'horizontal',
04    min: -50, max: 50, value: 0
05  });
06  $("input:button:first").click(function(){
```

```
07    $("#sl").slider("value",
08       $("input:text:first").val());
09    $("div:last").html($("#sl").slider("value"));
10  });
11 });
```

在第 2 ～ 5 行，我们创建了一个滑动条，它具有某种对齐方式以及最小 / 最大值，还有滑动条值的默认设置。在第 9 行中，我们读出 value 属性值，简单地将其显示在页面上。

在第 6 ～ 10 行中，可以看到对按钮单击的响应。首先可以在第 7 行和第 8 行中看到 slider() 方法和两个参数如何将滑动条设置为用户在输入字段中输入的值。如图 11-13 所示。

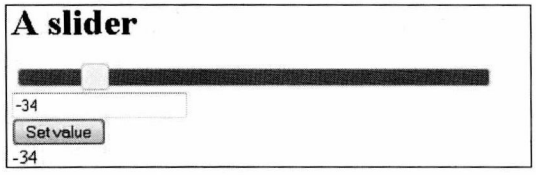

图 11-13　设置和获取滑动条的值

如果这个值大于或者小于最小或者最大值，滑动条简单地回到左右终点，尽管指定的 value 超出限值。如图 11-14 所示。

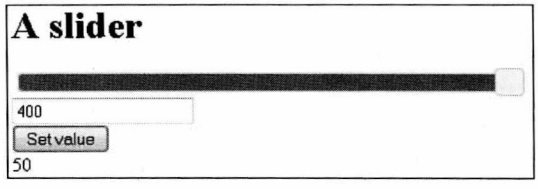

图 11-14　输入的值过大，该值被设置为最大值

现在应该注意到，滑动条通常不会以外部操纵值的方式使用。[⊖]作为替代，我们希望对滑动条的变化做出响应。这当然通过事件来完成，对于组件和窗口小部件来说，事件也是可用的。

11.4.5　组件和窗口小部件中的事件

组件和窗口小部件支持的事件也在文档中列出，在 Events（事件）选项卡中描述。

例如。对于可用鼠标拖动的元素，可以对拖动操作启动、结束和特定拖动阶段做出响应。在这三种情况下都会触发一个事件，可以为其绑定一个回调函数。这是完全常规的 jQuery 事件处理。也可以对组件中的一个选项卡做出反应，如程序清单 11.12 所示。

程序清单 11.12　对 Tabs 窗口小部件显示一个选项卡的反应

```
$('.selector').tabs({
   show: function(event, ui) { ... }
});
```

也可以使用通过 bind() 的数据绑定（例如，如果想对被拖动的滑动条做出反应，可以使用程序清单 11.13 所示的代码）。

程序清单 11.13　用数据绑定对拖动滑动条做出反应

```
$('.selector').bind('slide', function(event, ui) {
  ...
});
```

⊖ 不过，正如你所看到的，这当然是可能实现的。

我们用一个完整的例子来说明这一点，例中我们使网页中的一幅图像可以拖动，并响应拖动操作的不同状态。此外，我们还将对被移动的滑动条做出反应（ch11_5.html）。

程序清单 11.14 响应事件

```
...
18  <body>
19    <h1>A slider</h1>
20    <div id="sl"></div><br />
21    <img src="images/pic3.jpg" id="image" />
22    <div id="info1"></div><div id="info2"></div>
23    <div id="info3"></div>
24  </body>
25 </html>
```

我们使图像可拖动，并将第一个 div 元素变成滑动条。在最后 3 个 div 区域中，我们输出状态信息（ch11_5_ready.js）。

程序清单 11.15 对组件事件做出响应

```
01 $(function(){
02  $("#sl").slider({
03    min: -50,max: 50, value: 0,
04    slide: function(event, ui){
05      $("#info1").html("Value of the slider: " +
06      $("#sl").slider('value'));
07    }
08  });
09  $("#image").draggable({
10    start: function(event, ui){
11      $("#info2").html("Dragging starts: " +
12      event.timeStamp);
13    },
14    stop: function(event, ui){
15      $("#info2").html("Dragging stops.<br />" +
16          "Positions in relation to original position:" +
17          "<br />Top: " + ui['position'].top + ", Left: "
18          + ui['position'].left);
19    }
20  });
21  $('#image').bind('drag', function(event, ui){
22    $("#info3").html("Dragging in progress: " +
23       event.timeStamp + ", X/Y: " +
24      event.pageX + "/" + event.pageY);
25  });
26  $('#image').bind('dragstop', function(event, ui){
27     $("#info3").html("Dragstop triggered");
28  });
29 });
```

在第 2 ～ 8 行中，我们创建一个滑动条，并直接指定几个选项。我们所感兴趣的首先是第 4 行中的 slide 选项。它指定一个事件每当滑动条控件被拖动时触发。我们在第 6 行中通过 slider() 方法并指定“ value”参数，分配了一个匿名函数和处理滑动条值。我们在网页中不断地输出返回值。[⊖]如图 11-15 所示。

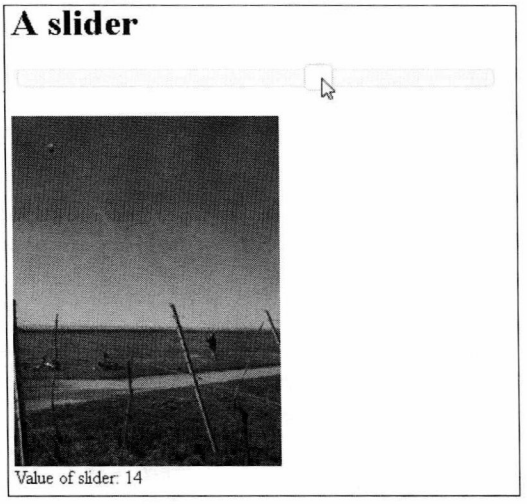

图 11-15 获取滑动条值

在例子中，我们还在创建 Draggable 类型对象时指定选项。此外，可以再次看到操作开始时（第 10 行）和操作结束时（第 14 行）的函数引用。这是经典的事件处理器。在匿名函数中，我们使用 jQuery 事件对象，在第 12 行中于事件发生时求取该时刻。具体地说，这就是对滑动条拖动操作开始的响应。

在第 21 ～ 25 行以及第 26 ～ 28 行中，还可以看到，我们是如何通过 bind() 使用数据绑定的。对于 drag 事件，我们用事件对象求取事件创建时鼠标指针的 X 和 Y 坐标。你将会看到，连续拖动图像会生成 type 类的事件，在网页中显示的值不断更新。如图 11-16 所示。

相应地，我们对 dragstop 做出响应，在该事件发生时输出一条消息。同时，第 14 ～ 19 行指定了对 stop 事件的响应。在此，我们使用匿名回调函数的第二个参数——触发的视觉对象。我们输出与拖动前原始位置相对的新坐标。如图 11-17 所示。

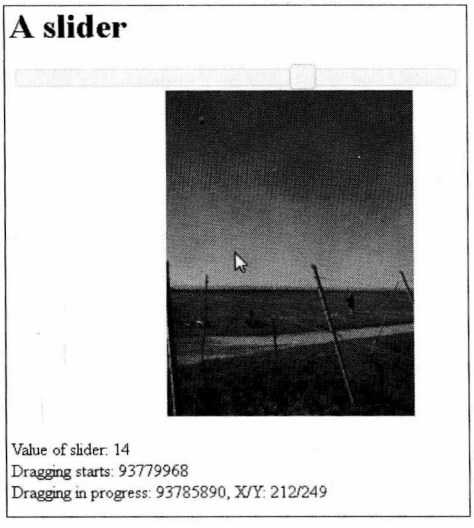

图 11-16 鼠标按钮被按下，启用图像拖动（drag 状态）

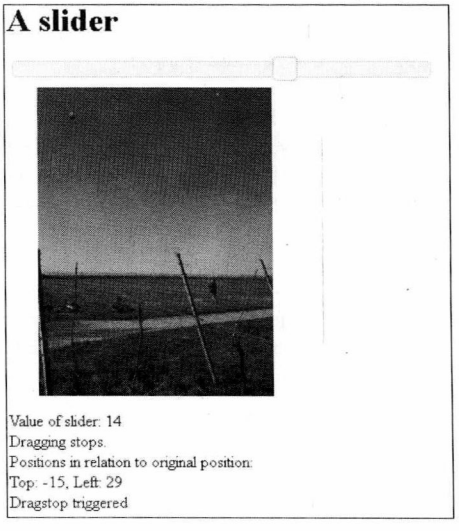

图 11-17 在拖动操作结束之后

⊖ 换言之，在每次滑动条控件被拖动时。

注意

并行使用数据绑定和组件事件响应在实践中很少使用。我们组合这两种方法只是为了教学目的。

11.5 组件和窗口小部件概述

前面已经提到过，本质上只需仔细地观察一个组件和一个窗口小部件，jQuery UI 中其他组件和窗口小部件的使用就应该显而易见了。不过，我们仍然将简单地介绍 jQuery UI 中可用的组件和窗口小部件。

11.5.1 交互组件

我们首先简单地看一下可用的交互组件。

Draggable

利用 Draggable 组件，能够启用网页元素通过鼠标的拖动。我们已经在上面的例子中详细地讨论了它。

Droppable

利用 Droppable 组件，可以指定一个元素为拖动操作的潜在目标。所以，这样的组件只在与 Draggable 类型的元素结合使用时才有意义。如果被拖动的元素被放到标记为 Droppable 的目标区域，可以触发特定的操作。

程序清单 11.16 敏感区域示例

```
$("#targetarea" ).droppable({
  hoverClass: "ui-state-active",
  drop: function( event, ui ) {
    $( this ).html( "Element dropped" );
  }
 });
```

Resizable

有些组件与某些需求的正常运作相关联（例如 Resizable 类型）。利用 resizable() 方法，这些对象的大小可以改变。但是这些组件只在之前通过 CSS 指定宽度 / 高度，且改变大小的区域使用另外几条 CSS 规则[⊖]指定时，才能改变大小。但是，这并不困难，也不复杂。我们来看一个完整的示例（ch11_6.html）。

程序清单 11.17 Resizable 方法示例

```
...
18  <body>
```

⊖ 特别是使用 class="ui-widget-content"。

```
19    <h1>A resizable DIV area</h1>
20    <div id="resizable" class="ui-widget-content">
21       <h3>www.rjs.de</h3></div>
22  </body>
23 </html>
```

我们使 div 元素的大小可以改变，并通过 CSS 进行一些格式化（ch11_6_ready.js）。如图 11-18 所示。

程序清单 11.18　div 容器的大小可以改变

```
01 $(function(){
02  $("#resizable").resizable();
03 });
```

Selectable

如果用 selectable() 方法标记元素，它们就可以被选中。因为该方法不使用参数，没有其他需要说明的地方。

Sortable

用 sortable() 方法标记元素，它们就可以在一个组中被排序。例如，在列表项目中这就有意义。

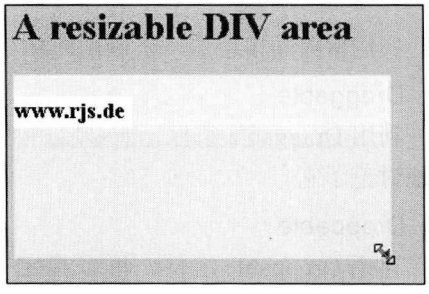

图 11-18　光标显示 div 容器可以通过鼠标扩大

11.5.2　窗口小部件

现在我们来看看 jQuery UI 提供的窗口小部件。

Accordion

Accordin 组件使我们可以垂直编排重叠的内容。这些内容可以交互式显示和隐藏，一次只显示一个内容区域，隐藏其他内容。这个窗口小部件较为复杂，因为它基于 div 元素的嵌套式结构，但是它的结构仍然很清晰。我们首先来看一下 HTML 结构（ch11_7.html）。

程序清单 11.19　基本结构

```
...
18  <body>
19    <h1>An Accordion Component</h1>
20    <div id="accordion">
21      <h3><a href="#">Section 1</a></h3>
22      <div>
23        Two things are infinite:
24        the universe and human stupidity;
25        and I'm not sure about
26        the universe.
27      </div>
28      <h3><a href="#">Section 2</a></h3>
```

```
29      <div>
30        You can lead a horse to water
31        but you can't make it drink.
32      </div>
33      <h3><a href="#">Section 3</a></h3>
34      <div>
35        Be not afraid
36        of going slowly,
37        be afraid only of standing still.
38      </div>
39    </div>
40  </body>
41 </html>
```

外部结构是一个 div 元素（第 20 ～ 39 行）。div 中的单独内容区域（默认版本中）包含一个类型为 h3 的标题，我们在这里写入一个带有标签的链接（通过一个 # 符号使其不活跃）。下一个 div 区域包含该区域的内容。然后，这种内部结构再次重复，指定另一个内容区域，以此类推。我们现在用一行代码将 div 元素变成 Accordion 组件（ch11_7_ready.js）。

程序清单 11.20　div 容器变成 Accordion 组件

```
01 $(function(){
02  $("#accordion").accordion();
03 });
```

现在，我们有了一个 3 部分的 Accordion 组件。第一个部分在加载网页时显示；其他两个部分隐藏。如图 11-19 所示。

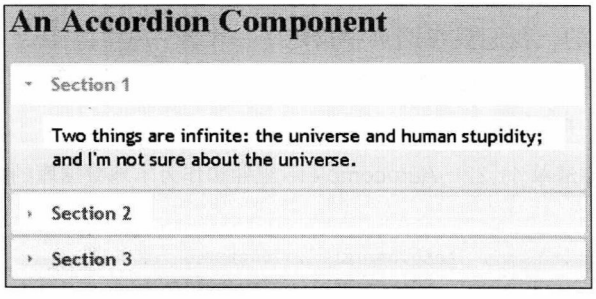

图 11-19　第一个部分可见

如果用户单击隐藏部分的标题，文本以生动的方式显示。如图 11-20 所示。

注意

在 jQuery UI 1.9 中，这个组件的内部进行了重新设计，但是使用方式相同。

Autocomplete

通过 autocomplete() 方法，可以访问一个相当新颖的 AJAX 驱动窗口小部件。它用于在

输入字段下面，于用户输入字符时显示一个适合的建议列表。

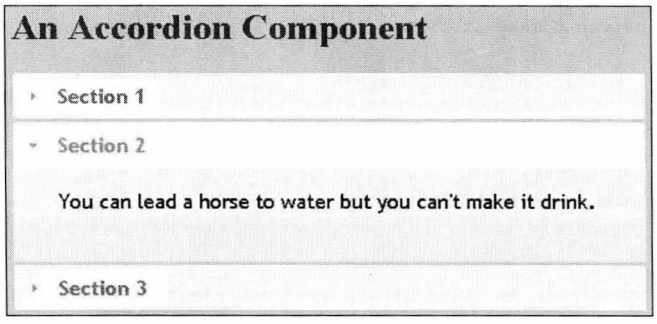

图 11-20 现在，中央部分可见

在默认版本中该方法使用一个 JavaScript 数组提供自动建议的可接受值。然而，这个窗口小部件有很多可配置选项，可以用不同的数据结构进行管理。这些建议可以通过 AJAX 从服务器加载，以 JSON 和 XML 格式提供。我们只打算简单地看看默认配置的一个变种（ch11_8.html）。

程序清单 11.21 基本结构

```
...
18  <body>
19    <h1>An Autocomplete Component</h1>
20    <input id="tags" />
21  </body>
22 </html>
```

正如大家所看到的，我们只在网页中指定了一个输入字段。注意，这里没有 div 区域，没有列表或者任何类似的可以显示建议列表的元素。这些结构完全动态生成。下面是包含数据数组的脚本 ch11_8_ready.js。

程序清单 11.22 Autocomplete 组件和作为本地数据库的数据

```
01 $(function(){
02  var db = ["Accordion", "Autocomplete", "Button",
03    "Datepicker", "Dialog", "Draggable", "Droppable",
04    "Progressbar", "Resizable", "Selectable", "Slider",
05    "Sortable", "Tabs"];
06  $("#tags").autocomplete({
07    source: db
08  });
09 });
```

关键点是本地数组和 Autocomplete 组件之间通过方法选项 source 的链接（第 7 行）。

如果在 Firebug 中观察生成的结构，将看到根据该数组创建了一个隐藏的列表。如图 11-21 所示。

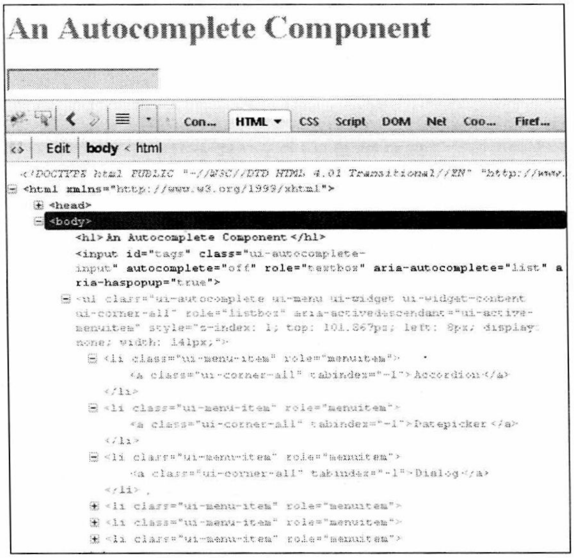

图 11-21 从数组生成了一个隐藏列表

如果用户现在在输入字段中输入字符，将显示一个建议列表，这个列表随着每个字符的输入变得越来越精确，单击建议列表中的某一项，就可以将相关的词语输入到字段中。如图 11-22 所示。

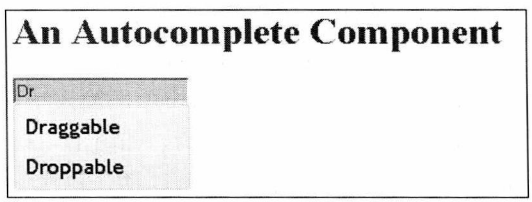

图 11-22 随着用户输入的每个字符，建议变得越来越精确

Button

Button 类型的元素提供按钮，这一点你应该能猜到。从功能上说，jQuery UI 提供的按钮没有什么特别之处。按钮的窗口小部件版本的优势在于，它们可以轻松地进行广泛的配置，可以从视觉上加以改变。本质上，可以通过指定 jQuery UI 中提供的图标（icons 选项）影响外观。需要做的主要是选择正确的类，然后将其赋予网页中的元素（通常是一个 div 元素或者 button 类型的元素），如程序清单 11.23 所示。

程序清单 11.23 带有一个图标的按钮

```
$( "#play" ).button({
  text: false,
  icons: {
    primary: "ui-icon-play"
  }
 })
```

Datepicker

Datepicker 窗口小部件是为用户输入日期提供视觉支持和全部合理性检查的日期组件。

这种视觉支持已经很复杂了，而交互式日期组件（或称日历）的逻辑就更为复杂。[⊖]如果对默认设置感到满意，jQuery UI 中的 Datepicker 类窗口小部件甚至将这种日期组件的使用变成一行代码就能完成任务（ch11_9.html）。

程序清单 11.24 基本结构

```
...
18  <body>
19    <h1>A Date Component/h1>
20    <p>Date: <input id="datepicker" type="text"></p>
21  </body>
22 </html>
```

如你所见，我们只在网页中指定了一个输入字段，没有任何 div 区域或者类似的可以在以后显示日历的区域。这些结构纯粹动态生成。程序清单 11.25 展示了脚本 ch11_9_ready.js。

程序清单 11.25 选择日期用的交互式日历

```
01 $(function(){
02   $("#datepicker").datepicker();
03 });
```

为了将选择日期的校验选项集成到网页中，需要的只是一个文本输入框。简单地调用这个元素的 datepicker() 方法，就能确保在用户单击输入字段时，以生动的方式在输入框下面打开日历。如图 11-23 所示。

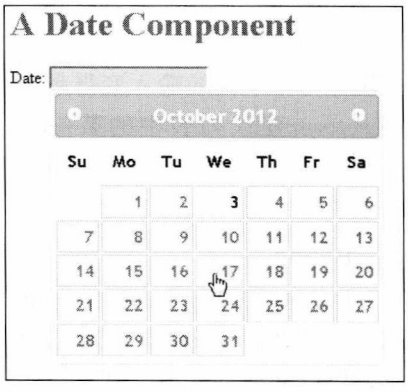

图 11-23 通过日历选择日期

如果用户单击日历中的某个日期，这个日期自动输入字段中。所发生的这一切不需要编写任何程序。

Dialog

Dialog（对话框）元素在 RIA 中越来越重要。它们越来越多地承担过去弹出式控件完成的工作。因为弹出窗口代表着浏览器的新窗口实例，常常被弹出式窗口拦截程序拦截，网页设计人员现在倾向于使用对话框元素代替，这些元素在内部是纯粹动态生成和格式化的 div 结构，实际上并不代表着新的窗口（所以不会被弹出式窗口拦截程序拦截）。

在 jQuery UI 中，还可以用一行代码创建对应于现代桌面环境的外观和行为的对话窗口。[⊜]在这种情况下，只需和平常一样使用默认配置。“模拟”的窗口已经能够满足用户在桌面应用程序中现代对话窗口的所有需求（基本设置下是非模态的）。它可以通过单击“X”图标关闭（通常在右上角，但是可以在主题中进行修改）、用鼠标扩大和缩小以及拖动。

对话框也可以用选项进行广泛的配置——不仅大小，还包括可否拖动、模态、显示的标

⊖ 记得我们自己的日历吗？

⊜ 不只是简单的 alert()。

题行等。我们将用一个例子尝试，使其看上去不那么粗糙。我们还要展示如何使用一个方法。具体地说，我们在下面的例子里通过两个按钮，选择打开两个对话框（ch11_10.html）。

程序清单 11.26 基本结构

```
...
18  <body>
19    <h1>Different Dialogs</h1>
20    <button>Open modal dialog</button>
21    <button>Open dialog</button>
22    <div id="b1" title="My webpage">
23      www.rjs.de</div>
24    <div id="b2" title="My blog">
25      blog.rjs.de</div>
26  </body>
27 </html>
```

在第 22 ～ 23 行中，可以看到一个 div 容器，包含一些随机的文本内容（第 24 ～ 25 行中也一样）。这些内容中可以包含常规的 HTML 标记。注意，这个容器在加载或者修改为对话框时最终是不可见的。（如果它被变为对话框组件，显示由框架动态完成）。这两个按钮在相关对话框被关闭时会重新打开它们。

程序清单 11.27 展示了脚本 ch11_10_ready.js。

程序清单 11.27 对话框元素

```
01 $(function(){
02  $("#b1").dialog({
03    autoOpen: false,
04    modal: true,
05    buttons: {
06      "Ok": function(){
07        $(this).dialog("close");
08      }
09    }
10  });
11  $("#b2").dialog({
12    position: [150, 200],
13    height: 100,
14    width: 400
15  });
16  $("button:eq(0)").click(function(){
17    $("#b1").dialog("open");
18  });
19  $("button:eq(1)").click(function(){
20    $("#b2").dialog("open");
21  });
22 });
```

在第一种情况下，我们使用一个模态对话框（第 4 行），该对话框在网页加载时（第 3

行）不显示[⊖]，为此显示一个具有回调函数的特殊按钮（第 5 ～ 9 行）。单击第一个按钮可以打开对话框（第 17 行）。如图 11-24 所示。

第 11 ～ 15 行定义第二个按钮。在此，我们用 X 和 Y 坐标指定位置、高度和宽度。这个对话框在网页加载时已经自动打开，但是如果用户关闭了它，可通过第二个按钮重新打开。

因为对话框相互独立，它们也可以同时打开（至少在模态对话框第二个打开的情况下）。如图 11-25 所示。

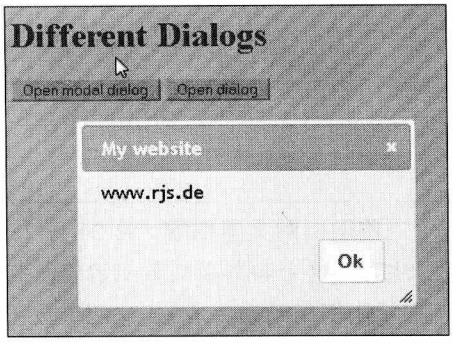

图 11-24 模态对话框阻塞网页

图 11-25 两个对话框元素都可见

Progressbar

名称已经说明了一切：这个窗口小部件是进度条。它本身很简单。进度条的关键点是 value 选项或者该方法的对应参数。这个值应该与实践中的某个过程相联系，进度条才有意义。数据绑定对此很合适。我们稍后将观察一个例子，将其与滑动条结合起来。

Slider

大家已经在实践中看到了滑动条，这里没有更多有趣的话题可讲。然而，下面的例子结合了一个 Slider 类型的对象和一个 Progressbar 类型的对象，确保进度条中显示的值与滑动条的值同步（ch11_11.html）。[⊖]

程序清单 11.28 同步滑动条和进度条

```
...
18  <body>
19    <h1>A Slider and a
20      Progress Bar</h1>
21    <div id="sl"></div>
22    <div id="progressbar"></div>
23  </body>
24 </html>
```

我们将第一个 div 元素转化为滑动条，第二个转化为进度条（ch11_11_ready.js）。

⊖ 这个默认值是为了使对话框显示出来。

⊖ 但是，进度条对于连续的过程确实有意义，例如加载过程或者类似的过程。

程序清单 11.29 进度条和滑动条

```
01 $(function(){
02  $("#progressbar").progressbar();
03  $("#sl").slider({
04    min: 0, max: 100, value: 0,
05  });
06  $('#sl').bind('slide', function(){
07    $("#progressbar").progressbar("value",
08       $("#sl").slider("value"));
09  });
10 });
```

在第 2 行中，我们创建一个进度条，第 3 ～ 5 行中创建一个滑动条。后者用几个选项配置。

更为有趣的是第 6 ～ 9 行中的数据绑定。为了拖动滑动条控件（slide 事件），我们绑定了后面的回调函数。在该函数中，我们在第 7 ～ 8 行中将滑动条的 value 属性赋给进度条的 value 属性，这确保了两者的同步。如图 11-26 所示。

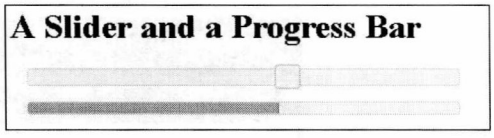

图 11-26 滑动条和进度条显示相同的值

Tabs

目前，我们看到许多网页中包含了通过选项卡细分的页面。这种结构也很容易通过 jQuery UI 实现。有趣的是，用 AJAX 加载内容甚至更容易。当然，也可以对这个窗口小部件进行广泛的配置，但是特别是在与 AJAX 联系时，基本设置已经非常强大，以至于很少需要进一步进行配置。

与 Accordion 组件类似，选项卡基于嵌入式的 div 结构。在这种结构中，将首先看到一个列表。这个列表负责选项卡的标签，并交互式地显示内容，使用户能够单击选项卡在内容之间切换。这包括被指定为列表项内容的超链接。这里要区分两种情况：

- 我们想要显示已经存在于网页中但目前不可见的内容。这些内容位于用某个 ID 标识的内部 div 区域中。链接使用一个锚引用来指向该 ID。
- 我们打算通过 AJAX 加载新内容。这样，我们只要指定新文件的 URL 即可。至于 MIME 类型，应该发送 text/html（ch11_12.html）。

程序清单 11.30 基本结构

```
...
18  <body>
19    <h1>Navigation via Tabs</h1>
20    <div id="tabs">
21      <ul>
22        <li><a href="#tabs-1">Home</a></li>
23        <li><a href="#tabs-2">Services</a></li>
24        <li><a href="notice.html">Legal Info</a></li>
25        <li><a href="gallery.html">Gallery</a></li>
26      </ul>
27      <div id="tabs-1">
```

```
28          <h3>Welcome to the RJS site for IT KnowHow</h3>
29        </div>
30        <div id="tabs-2">
31          <h3>Training - Consulting - Publishing</h3>
32        </div>
33      </div>
34    </body>
35  </html>
```

选项卡区域来自第 20 ～ 33 行。这是一个外部结构。在第 22 行中，可以看到指向第 27 行中的 ID 的一个链接，在第 23 行中则有指向第 30 行中 ID 的链接。这些链接应用的是包含网页中已有内容的 div 区域。ID 为 tabs-1 的 div 区域在网页加载时已经显示。在默认设置中，显示的总是第一个选项卡，但是也可以通过选项来更改。如图 11-27 所示。

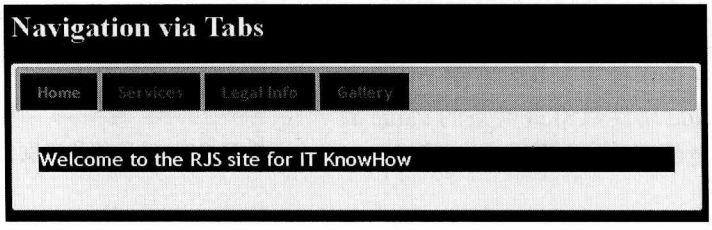

图 11-27 网页加载的时候显示第一个选项卡

应该注意，第 24 ～ 25 行中的链接指向 HTML 页面，而不是锚标记。单击这些链接通过 AJAX 加载引用的文件，这些文件应该具有 .html 扩展名，但是不包含一个基本结构。[⊖] 如图 11-28 所示。

图 11-28 数据已经通过 AJAX 加载

⊖ 用 AJAX 加载时总是如此。

注意，网页的硬编码结构中没有 div 区域或者类似区域来显示 AJAX 内容，这仍然由框架动态生成。

程序清单 11.31 展示了脚本 ch11_12_ready.js。

程序清单 11.31 div 区域被编程选项卡

```
01 $(function(){
02  $(function(){
03    $("#tabs").tabs({
04      ajaxOptions: {
05        error: function(xhr, status, index, anchor){
06          alert("Error on loading content");
07        }
08      }
09    });
10  });
11 });
```

本质上，我们并不需要第 4 ～ 9 行中的整个部分。要将 div 区域转变为选项卡，使用 tabs() 就足够了；即使我们通过 AJAX 在不刷新的情况下加载数据，也完全不需要参数。在此看到的只是通过选项进行的错误响应。这当然是正确的，但是对于基本功能来说并不是必需的。

注意

加载网页时显示的选项卡不应该通过 AJAX 加载，这没有意义。但是，对于其他所有选项卡，通过 AJAX 加载都可能是合适的，这取决于数据量。通过 AJAX 简化了固定的网页结构。

工具提示和 jQUery UI 1.9 的其他新特性

在 jQuery UI 1.9 中，我们可以自由地使用 Tooltip 类型的有趣组件，在用户的鼠标指针经过网页元素之上时显示上下文敏感的简短文字。如图 11-29 所示为新引进的小工具提示。

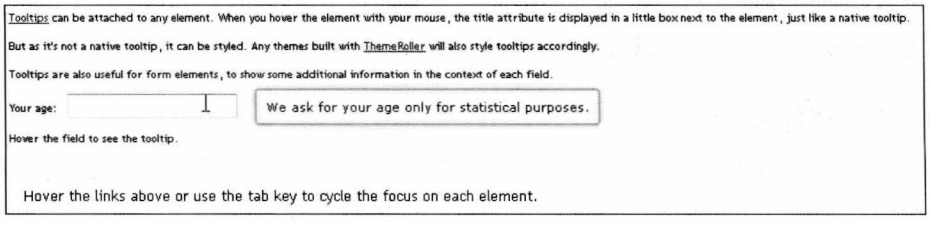

图 11-29 jQUery UI 1.9 中新引进的小工具提示

程序清单 11.32 创建一个工具提示

```
$("#myTip").tooltip();
```

新的版本还包含一个微调器（spinner）组件。这是一个具有两个箭头递增或者递减数字值的小组件。这个窗口小部件基于简单的输入元素。如图 11-30 所示。

图 11-30 微调器组件

还有一个基于无序列表的新 Menu 窗口小部件。如果想要构建嵌套式的菜单结构，可以简单地指定一个另一个列表的项目。如图 11-31 所示。

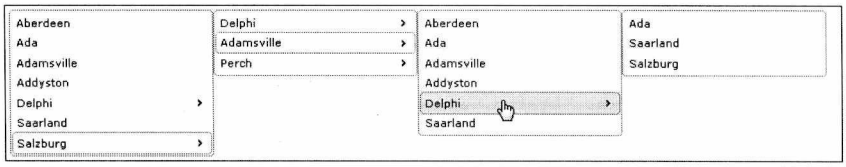

图 11-31 Menu 窗口小部件

11.5.3 实用工具

实用工具（Utilities）类别只包含一个类型为 position 的组件。可以用它可靠地以窗口、文档、特定元素或者鼠标指针的相对位置定位网页中的所有可见元素。严格地说，它是一个名为 position() 的辅助脚本，可以不带参数、使用一个函数引用或者用选项调用（例如，在 drag 事件的回调函数中）。

程序清单 11.33 定位

```
$( "#parent" ).draggable({
  drag: function() { position(); }
});
```

11.6 特效

有着完备文档的 UI 组件应用延续着 jQuery UI 的扩展特效。当然，大量特效需要许多选项和设置。应该转向文档，得到进一步的建议和信息。但是可以直接按照从 jQuery 得到的经验使用大部分方法。我们只打算稍微展开介绍两种特效。

11.6.1 effect() 方法

effect() 方法很有吸引力。它至少有一个描述特定效果的参数，可能值是“blind”、“bounce”、“clip”、“drop”、“explode”、“fold”、“highlighting”、“puff”、“pulsate”、“scale”、“shake”、“size”、“slide”、“transfer”。第四个可选参数中，可以指定一组选项[⊖]；第三个参数指定特效的时长，第四个参数是可选的回调函数。如果在一个元素上应用这些特效，它将会根据你的选择，产生爆炸、振动、摇晃等效果。可以使用文档中的交互式示例自行尝试。

⊖ 大部分特效可以在不指定选项的情况下很好地运行。只有少数意外情况需要指定选项。

11.6.2 使用 animate() 进行颜色动画

最后（但并非不重要），我们来看看扩展的 animate() 方法，通过 jQueryUI，它可以按照颜色建立动画。第一个参数可以指定想要设置动画的 CSS 属性，第二个参数是动画的时长（ch11_13.html）。

程序清单 11.34 包含一个 div 区域的基本结构

```
...
18  <body>
19    <h1>Color Animation</h1>
20    <button>Animate</button><hr />
21    <div id="i1">www.rjs.de</div>
22  </body>
23 </html>
```

网页中的 div 区域用外部 CSS 文件格式化，我们希望它随着按钮的单击动态变化；此外，我们希望颜色也出现变化。如图 11-32 所示。

程序清单 11.35 展示了颜色动画的方法（ch11_13_ready.js）。效果如图 11-33 所示。

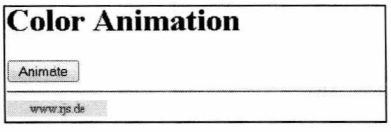

图 11-32 初始外观

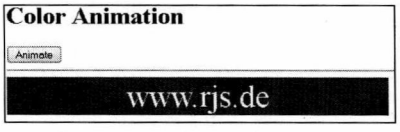

图 11-33 也可以根据颜色制作动画

程序清单 11.35 颜色动画

```
01 $(function(){
02   $("button:first").click(function(){
03     $("#i1").animate({
04       backgroundColor: '#aa0000',
05       color: '#fff',
06       width: 500,
07       fontSize: 42
08     }, 1000);
09     return false;
10   });
11 });
```

在第 3 ～ 8 行中，可以看到 animate() 方法。背景颜色和前景颜色的目标值在第 4 行中。

提示

本质上，可以指定所需的目标值，根据所有可观测的 CSS 属性值制作动画。

11.7 基于 jQuery UI 的完整网站

本章的最后，我们介绍一种用 jQuery UI 创建完整网站的基本方法。换言之，我将提供

一种使用 jQuery UI 组件和窗口小部件的网页结构。具体的布局只是一种建议，强烈建议大家在实践中采用自己的思路。

我希望以 AJAX 为后台技术，用 Accordion 组件构建一个本质上基于嵌套式选项卡的页面。正如我所说过的，这只是许多可能性中的一种，但是这种方法至少能作为你的一个起点。

我们先来看看完整的基本网页（ch11_14.html）。

程序清单 11.36 基本模板

```
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
   <title>jQuery UI</title>
   <link type="text/css" href=
      "lib/css/ui-lightness/jquery-ui-1.8.24.custom.css"
      rel="Stylesheet" />
   <link rel="stylesheet" type="text/css"
      href="lib/ch11_14.css" />
   <script type="text/javascript"
      src="lib/jquery-1.8.2.min.js"></script>
   <script type="text/javascript" src=
      "lib/js/jquery-ui-1.8.24.custom.min.js"></script>
   <script type="text/javascript" src=
      "lib/ch11_14_ready.js"></script>
 </head>
 <body>
   <div id="header"><img src="images/logo.gif"
      alt="Logo" /></div>
   <div id="content">
     <ul>
       <li><a href="#home">Home</a></li>
       <li><a href="sourcecode.html">
          Examples/Source Code</a></li>
         <li><a href="notice.html">
            Legal Info</a></li>
         <li><a href="more.html">More</a></li>
     </ul>
     <div id="home">
       <h3>Welcome to the RJS site for IT KnowHow</h3>
     </div>
    </div>
    <div id="footer">
    <a href="http://www.rjs.de/"
       title="RJS EDV-KnowHow"
       rel="home">RJS EDV-KnowHow</a>
    </div>
 </body>
</html>
```

如你所见，网页有 3 个 div 区域。页首和页脚都用纯 HTML 构造。在第 2 个 div 区域中，我们打算管理网页的实际内容，因此将这个区域转变为选项卡。只有起始页面随着网页加载。选项卡的其他内容在以后通过 AJAX 加载，结构很有趣。[⊖]首先看到的是 CSS 文件 ch11_14.css。

程序清单 11.37 CSS 格式

```
01 * {
02   background: white;
03 }
04 pre {
05  background: lightgray; color: blue;
06 }
07 #header {
08  width: 990px; height: 100px;
09  margin: auto;
10 }
11 #content {
12  width: 990px; min-height: 500px;
13  margin: auto;
14 }
15 #footer {
16  width: 990px; height: 100px;
17  margin: auto;
18 }
```

本质上，网页中的 3 个 div 区域被格式化为同样的宽度并被居中。下面是一个示例脚本 ch11_14_ready.js，用于创建选项卡。效果如图 11-34 所示。

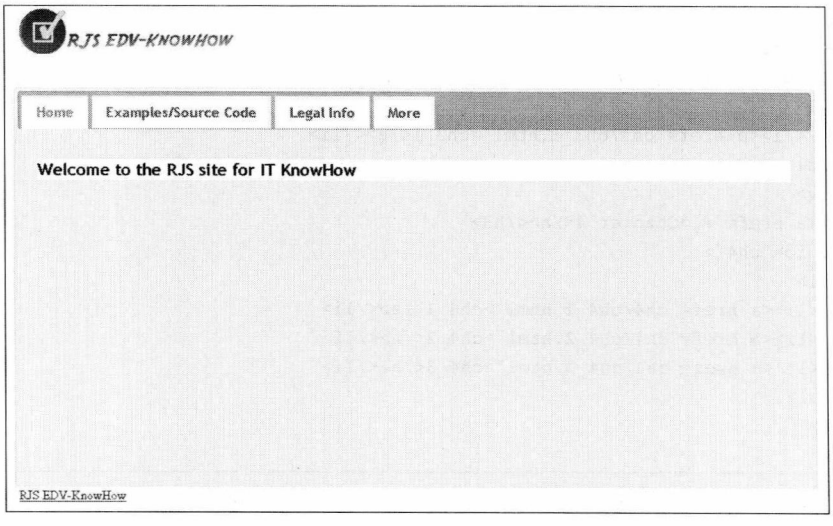

图 11-34 网页基本结构

⊖ 如果需要更多的选项卡，只需要相应地扩展结构。

程序清单 11.38 第二个 div 区域被转变为选项卡

```
01 $(function(){
02  $("#content").tabs();
03 });
```

以后通过 AJAX 加载的内容包含自己的结构。例如，文件 sourcecode.html 的结构如程序清单 11.39 所示。

程序清单 11.39 HTML 中的 Accordion 元素

```
01 <script type="text/javascript">
02  $(function(){
03    $("#ch2").tabs();
04    $("#ch3").tabs();
05    $("#ch4").tabs();
06    $("#accordion").accordion({
07      "fillSpace": true
08    });
09  });
10 </script>
11 <div id="accordion" style="min-height:400px">
12  <h3><a href="#">Chapter 2</a></h3>
13  <div id="ch2">
14    <ul>
15      <li><a href="ch2/ch2_1.html">ch2_1</a></li>
16      <li><a href="ch2/ch2_2.html">ch2_2</a></li>
17      <li><a href="ch2/ch2_3.html">ch2_3</a></li>
18    </ul>
19  </div>
20  <h3><a href="#">Chapter 3</a></h3>
21  <div id="ch3">
22    <ul>
23       <li><a href="ch3/ch3_1.html">ch3_1</a></li>
24    </ul>
25  </div>
26  <h3><a href="#">Chapter 4</a></h3>
27  <div id="ch4">
28    <ul>
29      <li><a href="ch4/ch4_1.html">ch4_1</a></li>
30      <li><a href="ch4/ch4_2.html">ch4_2</a></li>
31      <li><a href="ch4/ch4_3.html">ch4_3</a></li>
32    </ul>
33  </div>
34 </div>
```

可以看到，通过 AJAX 加载的文件本质上是一个作为外部结构的 accordion 组件。它包含一组用于各个内容区域的选项卡。可以用它轻松地嵌套内容。如图 11-35 所示。

警告

将用于生成jQuery UI窗口小部件集成到以后加载的文件，而不是基本文件，这一点很重要。

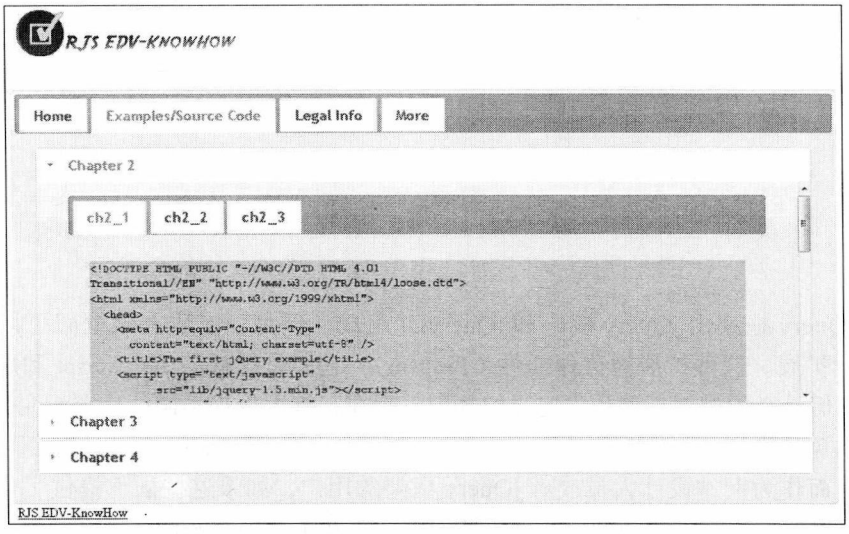

图 11-35 在折叠式组件中包含了下一级选项卡

如果需要更深的嵌套级别，可以相应地继续。

警告

记住，jQuery UI框架在某种程度上达到了自己的极限，特别是后续通过AJAX加载的片段和嵌套的窗口小部件。该框架以后不再能够正确地构造组件和窗口小部件，但是在那种情况下，你已经可以在加载网页时初始化组件或者窗口小部件，并简单地隐藏它们。在必要的时候，可以再次显示它们。

11.8 小结

本章大家见识了jQuery框架的第二个支柱——jQuery UI。它以大量强大的插件和用于构造图形界面的窗口小部件的形式提供了视觉控制元素。此外，它还提供了包含预设主题的CSS程序库和设计Web应用程序的强大工具。至少，jQuery UI扩展了jQUery的默认效果。尽管大部分组件都很强大且可以进行极其广泛的配置，使用jQuery UI的预设结构相对很简单。

第 12 章

插　　件

整个 jQuery 框架由 jQuery 核心和 jQueryUI 组成。但是，实质上该框架可以按照我们的喜好进行扩展。这种扩展被称作插件（Plug-in）。它们是纯粹的 JavaScript 和层叠样式表（CSS）库，但是必须遵循某种预先定义的规则。现在，有大量这类插件，它们往往可以通过 jQuery 网站免费得到（有些插件是商业化的）。本质上，任何 jQuery 开发人员都可以在那里发布插件。而作为网站设计人员或者 jQuery 框架的用户，如果这些插件提供了你不想自己创建、原生框架又未提供的功能，就可以使用它们。在本章中，我们将更仔细地研究这些插件，阐述使用外部插件，以及编写和发布自己的插件的方法。

12.1　jQuery 插件页面

为了更好地了解这一主题，应该观察 Web 上的一些插件。在 http://plugins.jquery.com/ 上你会找到 jQuery 官方插件页面。也可以通过 jQuery 网站页面顶部的 Plugins 链接进入该页面。在那里，你会找到许多编程人员创作发布的不同类型插件。当然，也可以用搜索功能找到特定的插件。插件页面如图 12-1 所示。

12.1.1　搜索和使用现有插件

假定你想要使用一个 jQuery 插件网站上可能有的插件。例如你正在寻找一个具有树形结构的导航菜单。目前，jQuery UI 和 jQuery 都未提供这类功能。你可能知道这种交互式导航树应该属于 Navigation（导航）类别。[⊖]如果搜索该类别列出的插件，那么或早或晚能找到一个合适的插件。也可以通过网页上对应的输入框直接搜索。

例如，可以输入搜索词“ Treeview”。[⊖]从匹配列表中，可以单击对应的超链接选择最满意的插件。

⊖ 严格地说，插件编程者应该将其归到合乎逻辑的类别中，否则，只有少数用户能够找到该插件。

⊖ 搜索词一般应该使用英语。

下一个网页描述使用该插件所需要知道的一切——如果编程人员提供了这些有用信息的话；从下载说明开始，到相关性、潜在许可条款直到要求的 jQuery 或者 jQuery UI 版本。在网页中通常还会找到一个演示插件的预览（尝试演示），只要插件编程者提供了这些页面。插件的文档完全集成到正常的 jQuery 文档中，所以它包含了概要说明、列出可用选项等。插件文档如图 12-2 所示。

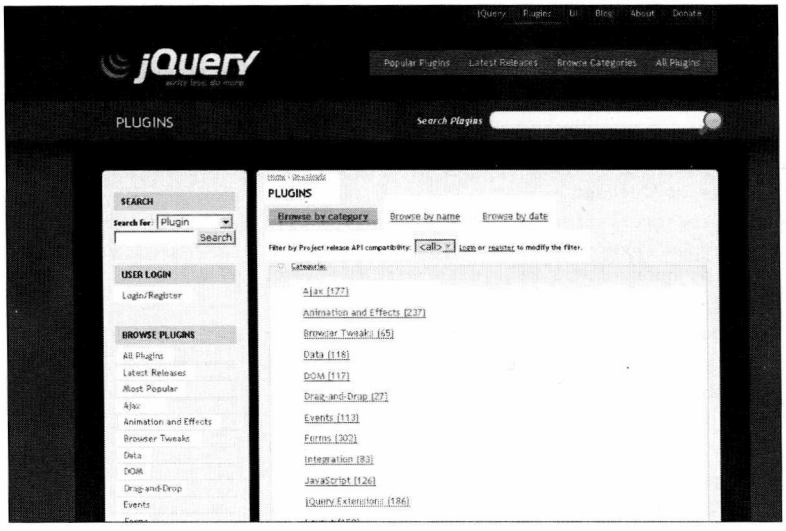

图 12-1 插件页面

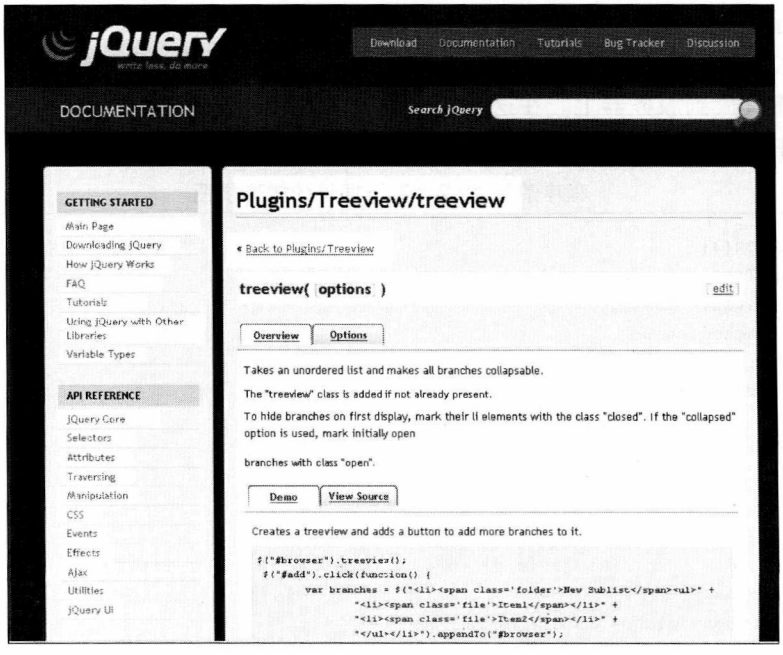

图 12-2 插件文档

如果已经下载了插件（必要的 JavaScript 和适用的 CSS 文件），只需要按照文档将其集成到网页，然后按照文档和演示的说明调用其中的方法——当然，这里假定插件编程者已经负责任地全面提供这些信息。

我们来仔细地看看用作例子的 Treeview 插件。在下面的例子中，我们只使用简单的导航树版本。但是这些插件通常提供大量的功能。流行插件的使用方法有很好的文档，所以使用更复杂的变种也很容易。

程序清单 12.1 引用插件的网页（ch12_1.html）

```
...
06    <link rel="stylesheet" type="text/css"
07      href="lib/ch12_1.css" />
08    <link rel="stylesheet"
09      href="lib/jquery-treeview/jquery.treeview.css" />
10    <script type="text/javascript"
11      src="lib/jquery-1.8.2.min.js"></script>
12    <script type="text/javascript" src=
13      "lib/jquery-treeview/jquery.treeview.js"></script>
14    <script type="text/javascript"
15      src="lib/ch12_1_ready.js"></script>
16  </head>
17  <body>
18    <h1>Navigation menu</h1>
19    <div id="output"></div>
20  </body>
21 </html>
```

在第 8 行和第 9 行中，可以看到交互式树形结构所用的特定 CSS 文件的集成。在第 12 和第 13 行中，可以看到对实现插件的 JavaScript 程序库的引用，你已经下载了该程序库和 CSS 文件并将其放到服务器上。在这个特例中，这些文件位于 lib/jquery-treeview 目录。导航菜单自动在 div 区域中生成（ch12_1_ready.js）。

程序清单 12.2 基于插件的树形视图

```
01 $(function(){
02   var myTree = $("#output").treeview({
03     animated: "fast",
04     collapsed: true
05   });
06   var newSublist = $(
07       "<li><span class='folder'>Websites</span><ul>" +
08     "<li><span class='file'>" +
09     "<a href='http://www.rjs.de'>www.rjs.de</a>" +
10     "</span></li>" +
11     "<li><span class='file'>" +
12     "<a href='http://blog.rjs.de'>blog.rjs.de</a>" +
13     "</span></li>" +
14     "<li><span class='file'>" +
```

```
15      "<a href='http://www.ajax-net.de'>" +
16      "www.ajax-net.de</a>" +
17      "</span></li>" +
18      "<li><span class='file'>" +
19      "<a href='http://www.javafx.cc'>www.javafx.cc</a>" +
20      "</span></li></ul></li>").appendTo(myTree);
21    myTree.treeview({
22      add: newSublist
23    });
24  });
```

创建树形视图很简单，只需要调用 treeview() 方法。当然，可以指定哪个参数取决于具体的编程，必须搜索插件的相关文档。在上述特例中，我们指定了树形结构展开和收起的速度。这里，插件编程者明显遵循了 jQuery 常用的标记法。第二个参数规定树形结构在初始阶段显示为折叠视图（第 2 ～ 5 行）。如图 12-3 所示为收起的树视图。

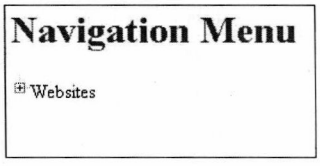

图 12-3 收起的树

迄今为止，树中还没有项目。它们通过在第 19 ～ 21 行中再次调用 treeView() 和选项 add 添加。但是，变量 newSublist 提供的内容是必要的。这种版本的树形视图中的具体项目基于 CSS 所改写的列表。 元素被赋予特定的类，定义主项目（folder）和子项目（file）。在第 5 ～ 16 行中，可以看到我们是如何在主项目下创建 4 个项目的。如图 12-4 所示为展开的菜单。

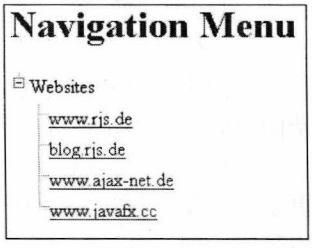

图 12-4 展开的菜单

正如你所看到的，只要插件的创建者正确地开发和编写文档，插件通常非常容易使用。

12.1.2 验证插件

在我看来，验证插件有着特殊的地位。这些插件根据某些规则验证用户的输入。在 http://jquery.bassistance.de/jquery-plugin-validation/ 下载的插件更为特殊。这个插件被 Microsoft 用于 Visual Studio 中的客户端验证，在我眼中，它不只是一个插件，甚至可以将它看作一个独立的小框架。

提示

在 http://docs.jquery.com/Plugins/Validation，你将找到完整的文档和许多演示应用程序。因为空间的限制，本文只提供该插件的一个非常简单的概述。

用户输入合法性验证或者检查是使用 JavaScript 最重要的方法之一。检查 Web 表单的合法性意味着在数据发送之前验证用户输入是否有意义。如果根据必要的预设数据，输入没有意义，那么就要采取措施进行更正。由于动态超文本标记语言（DHTML）和异步 JavaScript 及 XML（AJAX）现在得到了广泛的使用，输入有些时候在用户退出一个输入字段之后立即

检查，有时甚至在输入一个字符之后马上开始。[⊖]整个验证过程相当复杂，重点不是编程本身。Web 表单验证只在作为全局概念时才有意义，它必须包含发送之后对数据的进一步使用和不同的其他因素。在验证 Web 表单时，必须意识到许多潜在的问题。这可以用几个通用的问题来总结：

- 需要验证哪些输入？这本质上意味着：需要检查哪些表单字段？需要通过选项创建反映到插件中的规则。
- 用户输入和表单字段之间存在哪些相关性？
- 需要怎么样的验证？这包括一些问题，例如一个字段只要被填写就能满足要求（必填字段），还是字段内容必须有特定的形式（例如，必须是数字）？这一点也必须明确地在规则中或者更精确的方法中规定。
- 何时进行验证？例如，是直接在输入字段上检查用户输入还是只在 Web 表单发送的时候进行检查？如果数据在发送到服务器之前需要在多个网页中进行收集，是否在离开一个网页时进行检查？
- 数据将在何时、如何验证？在服务器还是客户端上验证？是不是在两端进行冗余验证？大部分情况下，冗余检查是必不可少的，但是我们只专注于 jQuery 环境中的客户端验证。

正如你所看到的，对验证的概念需要进行很多的思考。权衡必要的检查和实现的难度以及涉及的工作量并不容易。在任何情况下，规划具有用户交互的专业网页和合法性检查所花费的时间总是多于最终投入实践的时间。

最终，在用户输入错误信息时如何作出反应取决于你。通常，我们希望阻止表单数据在更正之前发送，为此给用户提供正确的反馈是很有意义的。为了提供这种反馈，同样可以使用某些特别适合于该任务的 jQuery 技术或者 jQuery UI 窗口小部件。

回到插件的话题。本质上，我们需要集成插件程序库，并以特殊的方式标记想要检查的表单字段。在应用插件提供的 validate() 方法时，合适的预设规则就已经生效。如果这些规则还不能满足你的需求，你可以定义自定义规则，将其作为选项传递给验证方法。我们先来看一个演示应用程序，它只包含默认的验证和消息（ch12_2.html）。如图 12-5 所示为一个 Web 表单。

程序清单 12.3 我们希望通过插件验证的一个表单

```
...
10      <script src="lib/jquery-validate/jquery.validate.js"
11        type="text/javascript"></script>
12      <script type="text/javascript"
13        src= "lib/ch12_2_ready.js"></script>
14    </head>
15    <body>
16      <h1>Form Validation</h1>
17      <form class="cmxform" id="myForm"
18        method="get" action="">
```

⊖ 但是后者很少成为合适的方案。

```
19      <fieldset>
20        <legend>
21          A validated form with default messages
22          and validity checks
23        </legend>
24        <p><label for="sname">Surname</label>
25          <em>*</em>
26          <input id="sname" name="sname" size="25"
27            class="required" minlength="2" />
28        </p>
29        <p><label for="fname">First name</label>
30          <em>*</em>
31          <input id="fname" name="fname" size="25"
32            class="required" minlength="2" />
33        </p>
34        <p><label for="email">E-Mail</label>
35          <em>*</em>
36          <input id="email" name="email" size="25"
37            class="required email" />
38        </p>
39        <p><label for="url">Website</label>
40          <em style="visibility:hidden">*</em>
41          <input id="url" name="url" size="25"
42            class="url" value="" />
43        </p>
44        <p><label for="comment">Comment</label>
45          <em>*</em>
46          <textarea id="comment" name="comment"
47            cols="25" rows="5"
48            class="required"></textarea>
49        </p>
50        <p>
51          <input class="submit" type="submit"
52            value="OK"/>
53        </p>
54      </fieldset>
55    </form>
56  </body>
57 </html>
```

正如你所看到的，我们已经在第 10 行和第 11 行中集成了插件的核心 JavaScript 文件（jquery.validate.js）。注意为单独的字段指定的类和最小长度所用的特性。这些特性是通过该插件进行验证的基础。验证方法与这些特性相连，在后台通过 validate() 调用。JavaScript 代码仍然非常简短[⊖]（ch12_2_ready.js）。

⊖ 如果你将其与手工编制的验证例程相比的话。

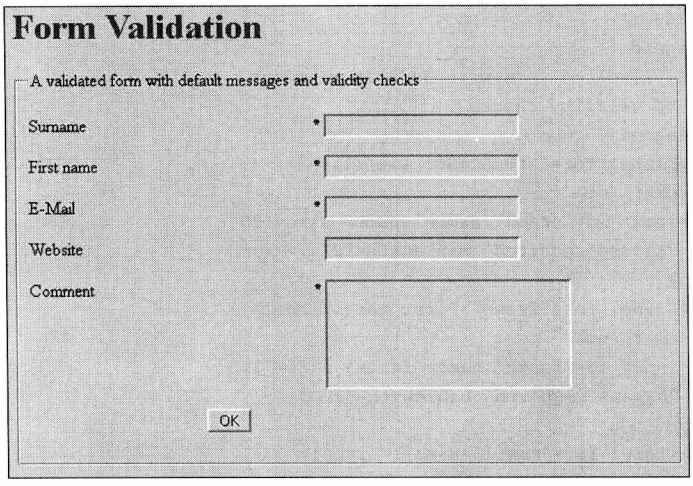

图 12-5 Web 表单，必填字段用星号标记

程序清单 12.4 使用插件

```
01 $(function(){
02   $("#myForm").validate();
03 });
```

要对表单进行全面验证，只需要对表单应用方法。类的选择和 length 规格确保了所有必填字段的填写。在错误的时候，表单数据的发送被阻止，输入字段后出现默认的错误信息，描述出现的问题。在所有错误被修复之后，发送表单。如图 12-6 所示为一些必填字段。

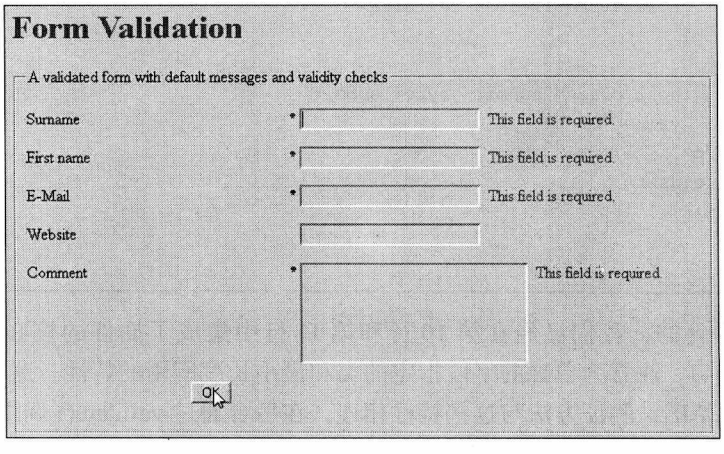

图 12-6 必填字段

提示

插件目录包含了本地化子目录，其中包含了国家特定错误信息的 JavaScript 文件。如果

打算将错误信息改成另一种语言，可以打开相关语言的文件（例如，messages_de.js 用于德语）将其中列出的选项（在花括号中）插入 jquery.validate.js 文件中的 defaults 选项，作为 messages 的值。然后，这些错误信息将以对应的语言显示。

如果在指定了最小长度的输入字段中输入的字符数不足，就会得到一个提醒。但是这种逻辑更进一步，因为甚至可以根据某些限制检查输入是否有效。在例子中，我们检查是否输入一个有效的电子邮件地址和 URL。输入字段的对应类规定了需要验证的这些内容。如图 12-7 和图 12-8 所示。

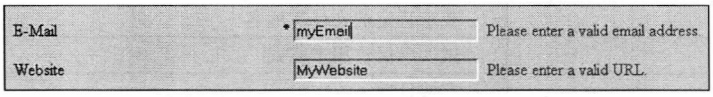

图 12-7 电子邮件没有 @，URL 的结构明显也不正确

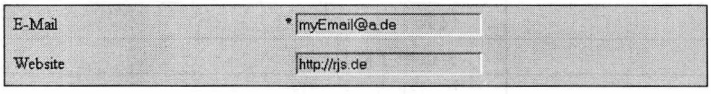

图 12-8 现在，插件的条件得到满足

注意

如果打算发送表单数据且检测到多个错误，所有错误将被标示出来。通常，第一个无效字段会获得焦点。顺便提一句，验证检查只发生在第一次试图发送数据时。只要用户没有试图发送表单，他就可以在字段之间跳转，即使输入了无效数据，也不会触发错误通知。但是一旦在试图发送时检测到错误并标示，每个字段在输入值之后立刻进行检查，错误信息在输入之后相应地更新或者删除。验证检查的这种动态行为是为了给用户提供合理的表单可用性。

validate() 方法是整个插件的核心。[⊖]它在默认配置下的工作效率已经很高——刚才你已经看到了。可以根据自己的需要，指定该方法的许多选项，如表 12-1 所示。

表 12-1 validate() 的一部分可能选项

选　项	描　述
rules	以关键字 – 值对形式出现的规则定义。关键字是一个元素或者一组复选框、选项框的名称。值是包含规则或者参数对、字符串的对象
messages	包含改写的错误信息的关键字 – 值对
各种事件	可以为各种不同事件触发验证。这些事件包括 onsubmit、onfocusout、onkeyup 和 onclick。可以和平常一样在选项中指定回调函数
errorClass	指定单独的错误类
highlight	指定无效字段高亮显示的方式

如果你打算进一步规定验证，插件提供了两个核心元素：

⊖ 但是正如我们稍后看到的，这并不是唯一的方法。

- 实现验证逻辑的验证方法。例如，这些逻辑可以更正电子邮件的结构。已经有整组的标准方法，其中一些已经在我们的例子中后台工作。但是也可以创建自定义的方法，然后在规则中指定它们。
- 将验证方法与元素相关联的规则。

表 12-2 描述了大部分重要的标准验证方法，它们已经自动可用，将在为元素指定同名的类时自动应用。

表 12-2 标准验证方法

方 法	描 述
accept(extension)	要求特定的文件扩展名
creditcard()	试图确保信用卡格式
date()	有效日期
dateDE()	有效德国格式日期
dateISO()	有效 ISO 日期
digits	仅数字
email()	有效电子邮件格式
equalTo(other)	用对比值进行测试
max(value)	指定最大值
maxlength(length)	指定最大字符数
min(value)	指定最小值
minlength(length)	指定最小字符数
number()	要求十进制数格式
numberDE()	要求德国格式的十进制数。除了德语，其他语言格式也是可能的
rengelength(range) 或者 range(range)	指定取值范围
require()	指定某个字段为必填。该方法有多个变种
url()	有效 URL

除了 validate() 之外，插件还包含多个其他方法，见表 12-3。

表 12-3 插件方法

方 法	描 述
validate (options)	表单已经使用的验证
valid()	返回一个布尔值，表示检查的表单是否有效
rules()	返回第一个选中元素的验证规则
rules ("add", rules)	添加指定规则，并返回第一个匹配元素的所有规则。例如，可以使用如下规则： $("#name").rules("add",{ minlength:5 }); 该方法的需求是父表单前面已经用 validate() 检查过
rules ("remove", rules)	删除指定的规则，并返回第一个匹配元素的所有规则
removeAttrs (attributes)	删除第一个匹配元素的指定特性，并返回这些特性

现在，我们输出描述前一个例子中描述电子邮件字段规则的选项。为此目的，我们添加一个按钮和一个用于输出的 div 区域（ch12_3.html）。程序清单 12.5 展示了输出这些规则的方法（ch12_3_ready.js）。

程序清单 12.5 字段规则

```
01 $(function(){
02   $("#myForm").validate();
03   $("button:first").click(function(){
04     var rule = $("#email").rules();
05     for (i in rule) {
06       $("div:first").append(i + ": " + rule[i]
07           + "<br />");
08     };
09   });
10 });
```

如前所述，规则纯粹是选项，所以利用脚本中的循环，我们将获得每个关键字和相关的值。如图 12-9 所示。

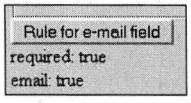

图 12-9 电子邮件字段的两条规则

正如你所看到的，默认配置下的电子邮件字段是必填字段，要求一个合适的格式。对应于这个结构，也可以选项形式创建自定义规则。此外，可以将上面列出的验证方法与合适的事件关联。

12.2 创建自定义插件

现在，我们转向创建自己的插件。但是，首先需要理解，为什么要这么做。

12.2.1 为什么创建自定义插件

如果你经常需要某些功能，通常会创建一个函数。在某些场合，只要在 JavaScript 之类技术中可行，我们还会创建类或者对象。相同的思路对于插件也适用：

现有功能的可重用性以及最大限度的广泛应用。

正如你在外部插件中已经看到的那样，用这种方式能为用户提供使用简单的复杂功能。创建插件与面向对象编程或者模块化思想有很多相关之处，在这种编程思想中，内部结构被加以封装，通过尽可能通用和简单的方式调用。

jQuery 插件机制使我们可以在一个包（JavaScript 文件）中添加方法和功能，这些包可以直接作为框架的一部分。jQuery 的许多组件本身也由插件组成。

12.2.2 创建你的第一个插件

编写插件分为两步。首先，编写打算提供的方法和函数。中心点是将这些功能分配给

特定的命名空间：用于方法的 jQuery.fn[⊖]和用于函数的 jQuery。而且，这一功能应该以单独 JavaScript 文件的形式提供，用户以后只需要集成它，该文件的命名遵循固定的规则：jquery.[插件名称].js。

程序清单 12.6 展示了插件 JavaScript 文件的一个示例，我们为其命名为 DragWithStatusLight（jquery.dragwithstatuslight.js）。

程序清单 12.6 插件源代码

```
01 jQuery.fn.dragwithstatuslight = function(){
02  return this.each(function(){
03    $(this).css({
04      border: "5px outset", cursor: "move"
05    });
06    $(this).draggable({
07      start: function(event, ui){
08        $(this).css({
09          opacity: 0.5
10        });
11      },
12      stop: function(event, ui){
13        status = "";
14        $(this).css({
15          opacity: 1
16        });
17      },
18      drag: function(event, ui){
19        status = "X/Y-Coordinates: " +
20        event.pageX + "/" + event.pageY;
21      }
22    });
23  });
24 };
```

这个插件的功能很简单。应用 dragwithstatuslight() 方法，使网页中的一个元素或者一组元素可以拖动。目前为止，这一功能与 draggable() 方法所提供的相同，也可以在第 6 行中看到我们使用了这个方法。

但是，我们的插件使这种功能更进一步。每个可拖动元素用一个外框标记，如果鼠标移过可拖动元素，光标也将改变，它变成十字形状——这在图形界面中常用于表示元素可拖动。我们使用 css() 方法来实现。

我们的插件的另一个特性是，被拖动元素在位置改变时被设置为透明的。我们通过覆盖 draggable() 方法的 start 选项来实现。如果用户拖动元素，元素就变成透明的。在元素被放下时，消除透明性。为此，我们覆盖了 draggle() 方法的 stop 选项。

在第 13 行，可以看到 status 属性值被设置。更准确地说，这个属性在拖动过程结束时

⊖ 我们已经详细地讨论了 jQuery 命名空间的思想。

被设置为空。你可能知道，我们用这个属性处理浏览器的状态行。但是为什么我们在此将状态行清空呢？这是因为我们打算在用户用鼠标拖动一个对象时，持续地显示更新信息；也就是说，显示被拖动对象的 X 和 Y 坐标。为此，我们覆盖 draggable() 的 drag 选项。这样，我们即使在最小幅度的鼠标动作中，也在拖动事件期间处理包含所需信息的事件对象。

警告

通过浏览器状态栏输出信息有一些矛盾。例如，状态栏绝对不在用户的视野中，大部分用户完全忽略状态栏信息。所以，只应该在这个区域中放入可选的附加信息。然而，更大的问题是，通过 JavaScript 访问状态栏在某些浏览器的默认设置中可能被禁用，也可能被用户禁用。因此，通过插件访问状态信息只在有限的程度上有用。但是，这不是本文的重点，我们希望借此展示创建插件的方法。

在第 2 行中可以看到，我们的插件返回类型为 jQuery 的对象或者该类型的一个集合。⊖ 这通常是插件的核心原则！

因为我们的新函数被分配到 jQuery.fn 命名空间，它可以通过如下的语法在所有 jQuery 对象上使用。

程序清单 12.7 使用自定义插件

```
.dragwithstatuslight();
```

例如，如果想让网页中的所有图像变成可拖动的，就可以按照下例使用插件。

程序清单 12.8 根据新功能，网页中的所有图像都可以拖动

```
$("img").dragwithstatuslight();
```

我们将这个插件作为完整示例的一部分（ch12_4.html）。

程序清单 12.9 使用插件

```
...
06    <link type="text/css" href=
07       "lib/css/ui-lightness/jquery-ui-1.8.24.custom.css"
08       rel="Stylesheet" />
09    <link rel="stylesheet" type="text/css"
10       href="lib/ch12_4.css" />
11    <script type="text/javascript"
12       src="lib/jquery-1.8.2.min.js"></script>
13    <script type="text/javascript" src=
14       "lib/js/jquery-ui-1.8.24.custom.min.js"></script>
15    <script type="text/javascript" src=
16       "lib/jquery-dragwithstatuslight/
         jquery.dragwithstatuslight.js" ></script>
```

⊖ 我们用 each() 循环读取所有前导对象。

```
17    <script type="text/javascript" src=
18       "lib/ch12_4_ready.js"></script>
19  </head>
20  <body>
21    <h1>Using the Plug-In DragWithStatus</h1>
22    <img src="images/pic1.jpg" />
23     <img src="images/pic2.jpg" />
24     <img src="images/pic3.jpg" />
25     <img src="images/pic4.jpg" />
26  </body>
27 </html>
```

正如你所看到的，我们必须在网页中集成许多 CSS 和 JavaScript 文件。这包括 jQuery 和 jQuery UI 的资源（因为我们在插件中使用了它的一个组件——否则就没有必要），当然还有插件本身。如图 12-10 所示。

在第 15 和第 16 行中，可以看到对插件文件的引用。程序清单 12.10 展示了如何将其应用到网页的所有图像（ch12_4_ready.js）。效果如图 12-11 所示。

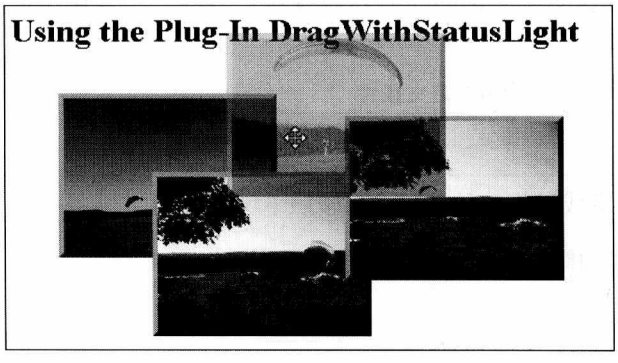

图 12-10 可拖动对象有一个外框。刚刚被拖动的图像是透明的，鼠标指针变成十字形

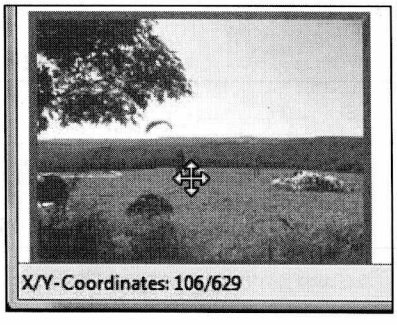

图 12-11 如果浏览器支持，拖动元素的坐标将被显示

程序清单 12.10 插件方法很容易使用

```
01 $(function(){
02   $("img").dragwithstatuslight();
03 });
```

12.2.3 创建简单插件的主要原则

下面总结了创建简单插件的主要原则和几种技巧：

- 最重要的一点：将你的方法赋予 jQuery.fn 对象，将所有函数赋予 jQuery 对象。
- JavaScript 文件的名称遵循如下模式：jquery.[插件名称].js。
- 在方法中，可以通过 this 访问当前 jQuery 对象。

- 每个方法或者函数必须以分号“；”结束；在某些情况下，常规的 JavaScript 中无需如此。
- 方法必须返回一个或者多个 jQuery 类型对象，除非明确地指定不同的返回值。
- 如果要枚举一组匹配元素，最清晰的方法是使用 this.each()。
- 为了避免无法解决的命名冲突，应该将插件连接到 jQuery 而不是 $。如果发生冲突，用户可以通过 noConflict() 定义一个别名。

12.2.4 创建较为复杂的插件的原则

如果打算创建更复杂的插件，就要坚持另外几条原则以避免出现问题。下面简短地概述这些原则。

在对象中收集静态函数

如果需要多个公用的静态方法，可以用单个对象声明式地收集它们，如程序清单 12.11 所示。

程序清单 12.11 在对象中收集

```
jQuery.log = {
  success : function() { ... },
  error : function() { ... },
  warning : function() { ... },
  debug : function() { ... }
};
```

这样做可以避免命名空间被分隔为太多的片段。

隐藏变量

隐藏变量是面向对象编程的核心原则之一。在创建插件时，也应该应用这一原则。在可能的情况下，不应该定义全局变量，而应该将其隐藏在函数（当然，这不是在严格的面向对象编程中完成的——在这种编程方法中没有函数）和方法中。通过传递值和返回值，可以在必要时传递变量的内容。

用于设置默认值的 jQuery.extend 或者 jQuery.fn.extends 选项

较为复杂的插件通常需要不同的设置选项。可以通过参数将这些设置选项传递给插件，但是最好的途径是使用以合适的默认值初始化的选项。最好的解决方案是通过 extends() 或者 jQuery.fn.extends() 预设选项，如果指定了选项，这些选项将取代预设值。例如，程序清单 12.12 展示了这种方法（jquery.dragwithstatus.js）。

程序清单 12.12 调用插件时的可选设置

```
jQuery.fn.dragwithstatus = function(options) {
  defaultvalues = jQuery.extend({
    border: "5px outset",
    cursor:"move" ,
```

```
    opacity : 0.5,
    statusinfo:true
  }, options);
  // rest of plug-in (*)
}
```

提示

除了选项之外，也可以用选择的语法将参数传递给函数。如果不需要参数，也可以通过 jQuery.fn.extends() 而不是 jQuery.extend() 扩展 jQuery 对象。

现在，我们可以使用带选项和不带选项的插件，如程序清单 12.13 所示。

程序清单 12.13 插件的不同使用方法

```
$("img").dragwithstatus({border:"50px", opacity:0.1,statusinfo:false});
$("#i1").dragwithstatus();
$("div").dragwithstatus({cursor:"cursor" });
```

可改编的动画

如果在插件中为某些事件使用动画，将这类动画设置为可改编的，对用户很有帮助。

避免导致命名冲突

为了不限制别名的使用，不应该在插件中使用 $。在插件中，如果有必要可以使用自己的别名。

12.2.5 带有选项的插件示例 1

现在，我们来创建第一个插件的扩展。这次，我们打算使其可以用选项配置。为此，我们只需要将上面的 jquery. dragwithstatus.js 模板与 jquery.dragwithstatuslight.js 的第 2 ～ 23 行组合起来，使得在没有指定选项时能够使用预设值，如果指定了选项，则覆盖预设值。

程序清单 12.14 修改后的插件 JavaScript 文件

```
01 jQuery.fn.dragwithstatus = function(options){
02  // Defining default values and
03   // overriding with options
04  var defaultvalues = jQuery.extend({
05    border: "5px outset",
06    cursor: "move", opacity: 0.5,
07    statusinfo: true
08  }, options);
09  return this.each(function(){
10    $(this).css({
11      border: defaultvalues.border,
12      cursor: defaultvalues.cursor
13    });
14    $(this).draggable({
15      start: function(event, ui){
```

```
16        $(this).css({
17          opacity: defaultvalues.opacity
18        });
19      },
20      stop: function(event, ui){
21        status = "";
22        $(this).css({
23          opacity: 1
24        });
25      },
26      drag: function(event, ui){
27        if (defaultvalues.statusinfo)
28          status = "X/Y-Coordinates: " +
29          event.pageX + "/" + event.pageY;
30      }
31    });
32  });
33 };
```

在第 4 ～ 7 行中，可以看到 4 个选项的预设值是如何设置的。第 2 个参数是选项名称，这些选项可以声明式地传递给插件，那样它们将会覆盖预设值。

在相关的 JavaScript 文件中，我们使用 defalutvalues 对象，并使用预设属性包含的属性或者用户指定的选项。

程序清单 12.15　使用插件的新变种（ch12_5_ready.js）

```
01 $(function(){
02   $("img:first").dragwithstatus({
03     border: "10px inset",
04     opacity: 0.1,
05     statusinfo: false
06   });
07   $("img:eq(1)").dragwithstatus({
08     border: "1px outset",
09     opacity: 0.7,
10     statusinfo: false
11   });
12   $("img:eq(2)").dragwithstatus();
13   $("img:eq(3)").dragwithstatus({
14     border: "5px outset",
15     opacity: 0.2,
16     statusinfo: true
17   });
18 });
```

该方法用不同的选项应用到 3 幅图像上，其中一幅图像没有使用选项。如图 12-12 所示。

12.2.6　带有选项的插件示例 2

为了让你更加清楚地了解，我们再来看一个小插件。我是在与我的乐队一起练习的时候

产生了这个思路。我们乐队通常以蓝调开始练习，这么多年之后，没有一个人希望负责选择调性。[⊖]所以，我建议使用一个随机数生成器来选择音乐的调性，并将其制作成 jQuery 插件（jquery.keys.js）。

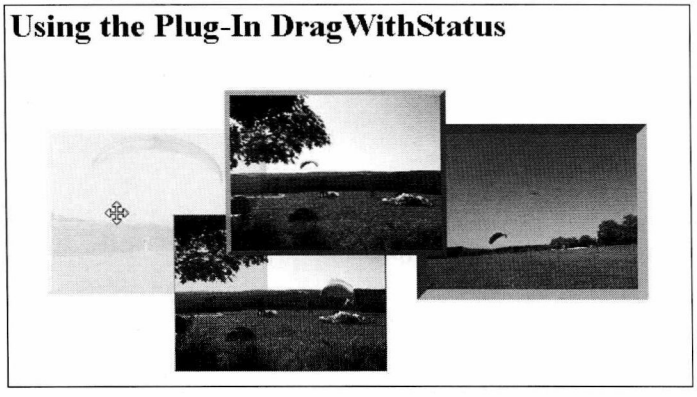

图 12-12 用不同选项调用方法

程序清单 12.16 在插件中使用一个随机数生成器

```
01 jQuery.fn.keys = function(options){
02      // Inner function
03     function keyGenerator(){
04           var key = ["C","C#", "D", "D#", "E", "F",
05              "F#","G","G#", "A", "B", "H"];
06           var index=Math.floor(Math.random()*key.length);
07         return key[index];
08     }
09     var defaultvalues = jQuery.extend({
10         border: "1px outset",
11         width: "150px",
12         opacity: 0.5,
13         height: "60px",
14         fontSize: "42px",
15         textAlign: "center",
16         paddingTop: "5px",
17         background: "black",
18         color: "white"
19     }, options);
20     return this.each(function(){
21         $(this).css({
22             border: defaultvalues.border,
23             width: defaultvalues.width,
24             height: defaultvalues.height,
25             opacity: defaultvalues.opacity,
```

⊖ 我们感觉每个调性都已经演奏了无数次。

```
            fontSize: defaultvalues.fontSize,
            textAlign: defaultvalues.textAlign,
            paddingTop: defaultvalues.paddingTop,
            background: defaultvalues.background,
            color: defaultvalues.color
        });
        $(this).html(keyGenerator());
    });
};
```

同样，可以看到预设的选项和用户覆盖它们的可能性。第 3 ～ 8 行中实现的逻辑简单地从一个包含调性值的数组中随机选择一个调性并返回。然后，可以简单地在按照预设值或者选项格式化的 div 区域中输出。如图 12-13 所示。

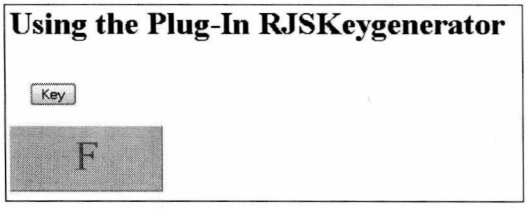

图 12-13 从数组中输出随机值

程序清单 12.17 使用带有选项的插件

```
01 $(function(){
02  $("button:first").click(function(){
03    $("#key").keys({
04      color: "blue",   background: "gray"
05    });
06  });
07 });
```

12.3 发布插件

我们现在来看看如何发布插件。为了通过 jQuery 网站发布插件，你必须拥有一个账户。如果已经注册，则需要登录。如图 12-14 所示。

登录之后，可以在个人区域中看到 Add Plugin（添加插件）链接。如图 12-15 所示。

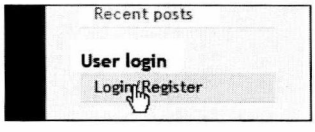

图 12-14 在发布插件之前首先需要登录

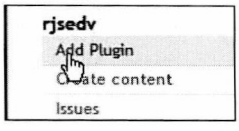

图 12-15 添加插件

单击这个链接之后，可以在右侧输入插件的所有重要细节。如图 12-16 所示。

这些细节包括类型、需要的应用编程接口（API）版本、描述、标签和对你的首页的引用。还可以指定一个插件的实际演示以及提供其他信息。在输入所有信息并发送数据之后，你的插件就正式可用了。

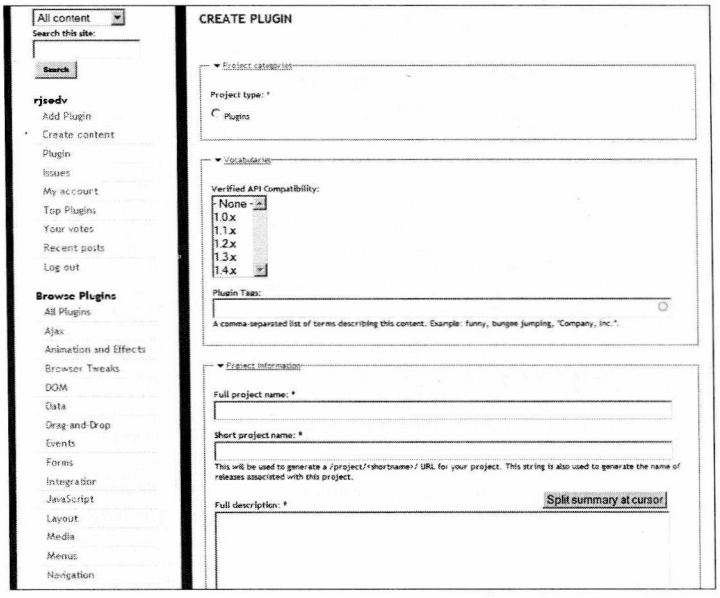

图 12-16 插件细节

12.4 小结

在关于插件的本章中，我们已经见识了 jQuery 编程中最有趣和最费力的部分。特别要说的是，在不远的未来这一领域将发生很多变化。插件对已经很强大的 jQUery 框架的扩展几乎是没有止境的。我们可以毫无问题地使用这些功能。但是，也可以通过 jQuery 的网页非常轻松地向广大公众介绍自己的想法。正如你所看到的，插件简化了交换，从编程的角度看，这是 Web 2.0 最大的意义——参与和加入。

第 13 章
jQuery Mobile

移动设备上的 Web 技术目前已经做好了面对竞争的准备，这不仅归功于超文本标记语言第 5 版（HTML5），还归功于层叠样式表第 3 版（CSS 3），尤其是与 JavaScript 及强大的框架结合的情况。将 PC 平台上的常规 Web 浏览器所用的传统富互联网应用程序（RIA）及相关的环境移植到移动设备上，或者相应地进行改编是令人神往的（因为大部分现代手机、平板电脑和智能手机都可以连接互联网，并且具有标准的浏览器）。你甚至不需要上线，就可以在运动中使用这些应用程序。只要设置得当，可以简单地将应用保存在移动设备上，从这些设备上加载并离线使用。在我们的环境中，有一个框架有助于开发基于 RIA 技术的这类移动应用：jQuery Mobile。在本书的最后一章里，我们将仔细研究这个框架。它的潜力巨大，特别是还能使用移动类别的大量 jQuery 插件。

13.1 基础知识

可以访问 jQuery Mobile 框架的网站：http://jquerymobile.com/。也可以在 jQuery 和 jQuery UI 网站的顶部找到 jQuery Mobile 的链接。如图 13-1 所示。

注意

在本书编著期间，jQuery Mobile 的最新稳定版本是 1.2.0。近期预计不会有重大的改变，但是为了安全起见，总是可以访问该网站，寻找最新版本的详情。

本质上，这个移动框架背后的想法很简单，且遵循常规 RIA 的模式：基于 JavaScript，在大部分重要的平板电脑、手机和智能手机或者它们的浏览器上构建统一的用户界面（UI）。jQuery Mobile（以及其他针对移动设备的技术）的主要问题是适应广泛的移动平台。需要考虑大部分移动设备上较小的屏幕尺寸和较低的分辨率，以及仍然很低的色深。在此之上，我们要面对不同的输入选项和特殊的效果，例如，自动旋转屏幕对设备的旋转做出反应。

图 13-1 jQuery Mobile 网站

为了采用纯粹标准化的 Web 技术，必须使用纯粹的语义 HTML 作为移动 RIA 的基础。至于逻辑，只有 JavaScript 能够在 HTML 上进行操作，而 CSS 承担所有的格式化工作（就像桌面的 RIA 那样）。[⊖]尽管也可以在桌面上创建与此背道而驰的变通手段或者简单地用拙劣的方法进行，但是在移动世界中这绝对是灾难性的。jQuery Mobile 框架采用了彻底的方法，明确要求对 HTML5 和 CSS3 的支持，确保使用中不出现错误。特别是，网页应该基于 HTML5 创建[⊖]，因为该框架一致地使用这个标准，从一开始就确保能够实现动态特性和窗口小部件。该框架也一致地在 jQuery 核心框架的基础上构建。

13.1.1 平台

jQuery Mobile 是一个为触摸界面做了优化的 Web 框架，它基于图形化的窗口小部件，这些窗口小部件试图在所有流行的移动设备平台上提供统一的 UI。该框架的目标不是为不同设备和操作系统开发不同的应用程序，而是让它们只需要一个应用程序。目前，该框架支持如下操作系统：

- iOS
- Android
- BlackBerry

⊖ 即便如此，对于大量不同的平台，仍然需要做很多大的工作；只要想象一下不同的屏幕大小或者不同的输入及控制选项就不难发现。

⊖ 至少应该避免使用老旧的标记，特别是用于规定布局格式的那些。

- Bada
- Windows Phone/Mobile
- Firefox Mobile
- Chrome for Android
- Opera Mobile
- Kindle 3 and Fire
- Nook Color
- Tizen
- Palm WebOS
- Symbian
- MeeGo

加上 Opera，浏览器支持包括了移动领域最重要的浏览器，以及 Fennec、Ozone、Netfront 和 PhoneGep。全面支持的浏览器被称作 A 级浏览器（这里指的是移动领域），部分支持的版本被称作 B 级或者 C 级浏览器；当然，这和具体的版本相关。可以在 http://jquerymobile.com/gbs/ 上找到完整的列表，告诉你哪些平台受到支持，以及支持的程度。注意，新的设备提供更好的 HTML5 和 CSS3 支持，所以框架的窗口小部件在更新的设备上可能比在旧的版本上运行得更好。当前支持的浏览器和操作系统如图 13-2 所示。

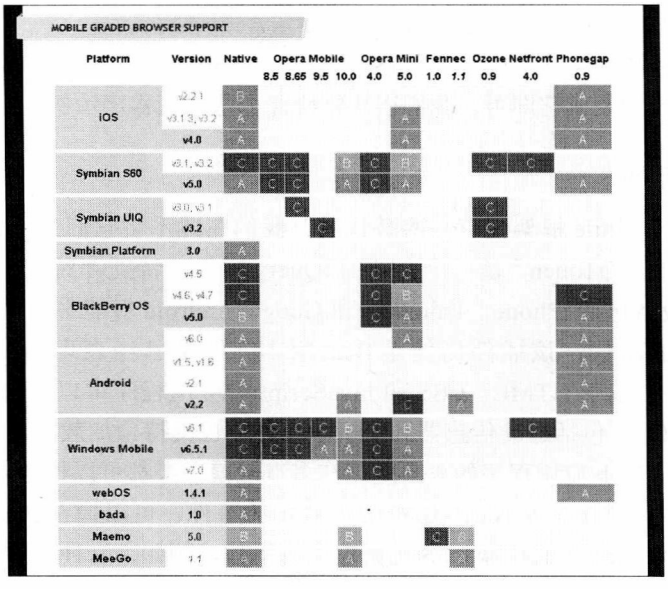

图 13-2 当前支持的浏览器和操作系统

警告

在创建移动 RIA 时要记住，许多并不过时的手机和智能手机也没有达到需要的标准。

如果移动设备不支持必要的标准或者不使用 A 级浏览器，网页将根据容错原则显示。在“最糟糕”的情况下，你将会看到一个不包含网页创建者需要的窗口小部件的网页。这个网页通常仍然起着部分作用，但是用户必须滚动很多次才能看到，而且看起来不怎么好看。还有一些移动设备没有完全支持 CSS3，但是在其他方面与 jQuery Mobile 框架配合得很好。在这种情况下，可能无法正常显示圆角或者无法正常显示动画效果，但是窗口小部件能够正常工作。

说到窗口小部件，jQuery Mobile 框架提供了大家熟悉的 jQuery UI 组件（只要它们适合于移动设备）。它们为使用触摸屏进行了优化，并增添了一些新的事件。jQuery UI 的 CSS 框架已经被扩展并为移动目标平台做了改编。前面已经提到，jQuery Mobile 大量使用了 CSS3。通过特殊的 data-role 特性，可以在后台进行自动的内在化（Internalization），对此不应低估。该框架还为屏幕阅读器提供了可访问性功能如 WAI-ARIA 的支持。移动环境中的性能局限在框架设计中也被考虑在内。这反映在需要下载的文件尺寸上。整个框架的文件尺寸很小；整个移动功能被包含在一个大约 24KB 的压缩版本中。

13.1.2 下载和集成框架

可以从 http://jquerymobile.com/download/ 下载框架的当前版本。Zip 文件包含所有必要的资源，但是 jQuery Mobile 也通过 CDN 提供，而且现在有一个新的下载构建器，可以构建只包含所需组件的自定义安装包。

对于移动应用程序来说，只要使 CSS 文件和 JavaScript 文件在 Web 服务器上可用，并按照 jQueryUI 的规则集成它即可。我们很快将研究一些应用程序示例。

13.1.3 替代方案

当然，jQuery Mobile 框架也有一些替代品。我们在此不将它们全部列出，只打算简单地介绍一些。例如，jQTouch[⊖]是一个有趣的 jQuery 插件，特别针对使用 WebKit 浏览器的移动设备。这类设备包括 iPhone、Palm Pre 和 Google Android 智能手机（如 G1）。

在写作本书的时候，该插件仍然是 Beta 版本，但是已经表现出了它的方向。利用 jQTouch 和 jQuery，基于 HTML、CSS 和 JavaScript 的应用程序可以支持 WebKit 浏览器和手机硬件的特殊属性（包括运动传感器和手机触摸屏，也支持预加载图像、页面过渡和新的全屏模式）。因此，Web 应用程序的方向变化和多点触摸手势都可以得到处理。

对于上述的两种情况，jQTouch 还提供了适合的事件，可以像在 jQuery 中一样绑定回调。除此之外，jQTouch 中的特殊 CSS 选择器支持手势。例如，这种选择器确保了列表元素中的一个链接可以对指按手势做出反应。.back、.cancel 或者 .goback 等选择器可以和 Apple 的 Back 按钮等硬件组件配合工作。如图 13-3 所示。

⊖ jQTouch 是采用 MT 许可证的开源项目，网址是 http://jqtouch.com/。

图 13-3 jQTouch 项目网站

在 CSS 方面，应用程序也可以通过 jQuery 风格的主题直观地改写，插件也包含了自己的主题。

jQTouch 插件的一个有趣特征（尤其是对于 iPhone 来说）是可以轻松地创建有创造力的服务，而不需要受到 App Store 的阻碍。

13.2 角色系统和 data-role

让我们回到最后一章所要讲述的框架上来。在 HTML5 中，我们试图将注意力完全集中到页面结构上，而将设计全部交给样式表。在现代的网页中，我们尽可能地将自己限制在几个描述结构的标记上。特别重要的是，除了各种新元素之外，旧的 div 元素得到了具有专用功能或角色的 HTML5 新专用块元素的支持。这种想法也得到了框架的一致反映，但是它将自己的功能主要限制在已有的块元素（div）中。

jQuery Mobile 框架使用一个网页元素的角色系统以标记窗口小部件。通过 data-role 特性，每个完成特定任务的元素用标准化的值加以标记。例如，用户通常不需要在标记窗口小部件时直接编程，就像 jQuery UI 中一样。[⊖]例如，可以简单地用一个特性和一个特定值标记 div 元素，将其转换为按钮。因此，不需要像在 jQuery UI 中那样有一个明确的脚本区域和对应的语句。在框架中使用的所有窗口小部件都可以这样创建。还可以通过这种方法创建

⊖ 但是，后台中发生的大体上和使用 jQUery UI 方法时相同。

页面过渡、链接和页面，完全不需要手工编程。

13.3 移动网页的基本结构

我们希望从 jQuery Mobile 中得利的网页基本结构通常遵循相同的模式。前面已经提到过，在整个框架中使用 data-role 特性，并且在 DOCTYPE 中明确地说明将以 HTML 作为基础。框架词汇表中所称的整个网页可能意味着一个页面，但是页面（Page）这个术语常常指的是其他的意思。它代表网页中的一个单独的片段。典型的网站结构代表一个单独的页面（如果没有太多内容）或者内部链接的页面。

在单个 HTML 文件的主体中，我们希望作为移动屏幕上的一个页面的视图或者元素（通常是一个 div 元素）被赋予特性 data-role="page"。在这个容器中，可以使用任何有效的 HTML 语句（除了基本框架本身的语句之外）。通常用单独的标题、内容和页脚区域来构造这种页面分段，这种思路也贯穿在 HTML5 中的多个新元素中。这些区域用框架中的 data-role 特殊值进行标记（换句话说，和 HTML5 相反，使用特性而不是单独的元素如 footer 或者 header）。

程序清单 13.1 页面的基本结构

```
<div data-role="page">
  <div data-role="header">...</div>
  <div data-role="content">...</div>
  <div data-role="footer">...</div>
 </div>
```

下面是一个满足以上需求的完整网页，只包含一个页面，并且被集成到我们通常的项目结构中（ch13_1.html）。

程序清单 13.2 典型的基本结构

```
01 <!DOCTYPE HTML>
02 <html>
03  <head>
04    <meta http-equiv="Content-Type"
05       content="text/html; charset=utf-8" />
06    <title>jQuery Mobile</title>
07    <link rel="stylesheet" type="text/css" href=
08       "lib/jquery.mobile-1.2.0/
         jquery.mobile-1.2.0.min.css" />
09    <link rel="stylesheet" type="text/css"
10       href="lib/kap13_1.css" />
11    <script type="text/javascript" src=
12       "lib/jquery-1.8.2.min.js"></script>
13    <script type="text/javascript" src=
14       "lib/jquery.mobile-1.2.0/
         jquery.mobile-1.2.0.min.js"></script>
```

```
15    <script type="text/javascript" src=
16       "lib/kap13_1_ready.js"></script>
17  </head>
18  <body>
19    <div data-role="page">
20      <div data-role="header">
21        <h1>The Page Title</h1>
22      </div>
23      <div data-role="content">
24        <p>
25          The normal content
26        </p>
27      </div>
28      <div data-role="footer">
29        <h4>The Footer</h4>
30      </div>
31    </div>
32  </body>
33 </html>
```

在第 7 行和第 8 行中，可以看到指向框架样式表的链接。应该将这个文件作为第一个集成的 CSS 文件。如果有必要，应该在这之后集成自己的 CSS 文件。

注意

在本章的示例中，我们包含对自己的 CSS 和 JavaScript 文件的引用，即使在示例中没有明确地使用它们也是如此。

在第 11 行和第 12 行中，我们集成了常规的 jQuery 程序库。在此之后，可以在第 13 行和第 14 行中看到对移动 JavaScript 程序库的引用。如果打算使用 jQUery UI，[⊖]可以将引用放在移动 JavaScript 程序库的前面或者后面。在此之后，才能放入自己的 JavaScript 文件（如果有必要的话）。

如果在 HTML 文件中标记 div 元素，框架将确保在合适的平台上，该区域根据所选择的主题格式化。如前所述，第 19 ～ 31 行的 page-block 被视为一个页面。效果如图 13-4 所示。

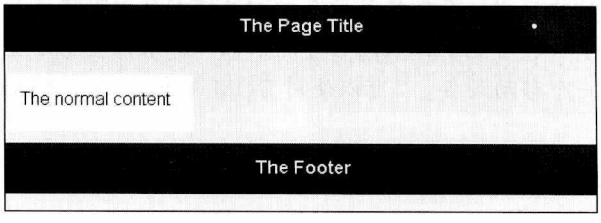

图 13-4 该页面已经由框架格式化

⊖ 在移动应用中，这实际上没有必要；移动窗口小部件应该用于这个目的。

13.4 链接页面

基本上，jQuery Mobile 框架支持所有标准的 HTML 链接类型。在链接页面时，必须在框架中区分外部和内部链接，这来自于页面组成的不同解释。

13.4.1 通过 Hijax 实现外部链接

在默认设置中，只要引用的是同一个域，jQuery Mobile 在链接到外部页面时使用 AJAX 请求（Hijax）。和桌面应用程序不同，移动应用的光标改变，使我们能够在文件稍后加载时看到（旋转的煮蛋计时器或者纺锤）。如果请求成功，新数据被集成到文档对象模型（DOM），移动窗口小部件被重新初始化。然后，以动画形式显示新页面。

如果链接指向另一个域或者用 rel="external" 或者 data-ajax="false" 标记，或者有一个 target 特性，则通过常规方式加载新页面。

jQuery Mobile 框架也支持用于其他链接类型的超文本传输协议（HTTP），其中有些是移动环境的结果。例如，它支持电子邮件所用的 mailto:[⊖]和用于电话的 tel:、wtai: 和 dc:。

13.4.2 内部链接和页面特殊解释

在 jQuery Mobile 中，从框架的意义上来说，单个 HTML 文档可以包含多个页面。这种方法很有意义，因为传统的网页对于较小的手机和智能手机屏幕来说太大了。在常规网页中，使用移动设备的访问者几乎总是必须滚动才能看到整个网页的内容。网页创建者将网页内容分为具有逻辑分组信息的固定段落是完全正确的。预先下载这些段落在带宽仍然很小的移动环境中也是合乎逻辑的。这些预先加载但是尚未显示的段落可以通过特殊的链接连接起来，在用户从一个页面转向另一个页面时快速显示，因为不同的页面都已经下载完毕。

提示

不应该将设计用于将为常规 PC 开发的 Web 应用转换为形式完全相同的移动应用。即使链接 HTML 文件中的单独的页面段落，对移动访问者来说也太多。精简内容并压缩多媒体文件是非常明智的。为此，如果自定义了 CSS 规则，当然可以使用改编的 CSS 文件。最好的选择是创建两个单独的应用，在加载起始页面时通过浏览器嗅探选择。例如，可以使用屏幕分辨率，对于普通手机和智能手机来说它可能很低。如果检查屏幕宽度，就能够在该值低于 800 时可靠地确定一个移动设备。[⊖]可以使用 DOM 对象 screen 进行这项检查。

程序清单 13.3 自动识别和重定向移动设备的浏览器嗅探器

```
if(screen.width < 800) {
  location.href="indexMobileApp.html";
```

⊖ 包括 cc:、bcc:、主题和预设正文。

⊖ 即使在这里错误地标识了移动设备，也不会造成问题，因为它能够应付常规的尺寸。

```
}
else {
 location.href="indexPCApp.html";
}
```

当然，也存在其他可能性，但是这种方法能够很好地工作。在本书的配置网站上，也能够找到这样的简单浏览器嗅探器，它以 browersniffer.html 的形式存在，使用 testresolution.html 文件可以检查设备分辨率。

提示

如果你的移动设备支持自动屏幕旋转，你可能看到 screen.Width 的值也发生改变（这当然有意义）。

所以，在 jQuery Mobile 中页面就是用 data-role="page" 标记的网页区域，它们已经被一起加载（作为整个 HTML 页面的一部分）。现在，每个 page 块都需要唯一的 ID，以便能够正确应用。这个 ID 用于内部链接，正如你在 jQuery 中所看到的那样（href="#myId")。单击一个链接时，对应的标记区域被移动到屏幕上。

警告

如果链接到通过 AJAX 加载且包含多个 page 段的页面，则需要在链接中添加 rel="external" 或者 data-ajax="false" 特性。

程序清单 13.4 链接到具有多个页面片段的页面

```
<a href="multipage.html" rel="external">Link</a>
```

在历史中回退

从 JavaScript 中你已经知道了 history.back() 的功能。这个方法将你带回浏览器历史中的最后一个页面。利用 data-rel="back" 特性可以精确地使用这一功能。相反，如果不想使用历史，可以使用 data-direction="reverse"。

程序清单 13.5 链接页面片段和历史中的回退

```
...

16   <body>
17     <div data-role="page" id="p1">
18       <div data-role="header">
19         <h1>Page 1</h1>
20       </div>
21       <div data-role="content">
22          <h2>Welcome on the Mobile Page</h2>
23               <a href="#p2">Go to Page 2</a>
24       </div>
```

```
25        <div data-role="footer">
26          <h4>(c) www.rjs.de</h4>
27        </div>
28      </div>
29          <div data-role="page" id="p2">
30        <div data-role="header">
31
32          <h1>Page 2</h1>
33        </div>
34        <div data-role="content">
35          <p>
36            <a data-rel="back" href="#">Back</a>
37          </p>
38        </div>
39        <div data-role="footer">
40           <h4>(c) www.rjs.de</h4>
41        </div>
42      </div>
43    </body>
44 </html>
```

这里，可以看到一个包含两个页面的 HTML 文档。如果加载这个 HTML 文件，在正确支持框架的浏览器中，开始时只显示第一个 page 页面。如图 13-5 所示。

第 23 行的超链接指向一个 ID，可以打开第 29 行中以这个 ID 标记的第二个 page 段。如图 13-6 所示为第二个页面。

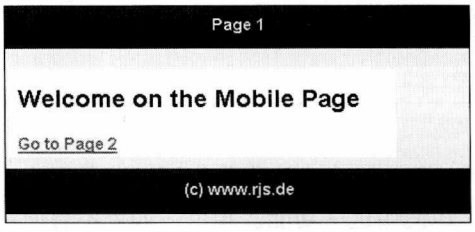

图 13-5 加载 HTML 文件时，只显示第一个页面

图 13-6 第二个页面

在第二个页面上，可以找到回第一个页面的链接和自动生成的图形化 Back(后退）按钮。注意，第 37 行中的 href 值被设置为 #。这个链接通过 data-rel 触发，但是 # 也能触发到第一个 page 段的跳转。[⊖]

13.5 过渡

jQuery Mobile 框架目前有 9 种基于 CSS 的页面间过渡效果，或者可以赋予页面 change 事件，用于显示新元素的对象。在默认设置中，过渡从右向左进行。如果想改变这种行为，

⊖ 双重标记是有意义的，因为那样就可以确保在某些不完全兼容的平台仍然可以正常地跳转。

可以对触发过渡的链接使用 data-transition 特性。

程序清单 13.6 明确指定过渡效果

```
<a href="index.html" data-transition="pop">Home</a>
```

警告

过渡效果大量使用 CSS3，目前在某些旧平台上没有得到完全的支持。在那种情况下，新内容会立即显示。

可能的取值有 slide、slideup、slidedown、pop、fade 和 flip。从 jQuery 本身 和 jQuery UI 中可以了解这些过渡。

提示

可以通过 data-direction="reverse" 反向进行相同的过渡。

13.6 对话框

在 jQuery Mobile 框架中，对话框也可以通过特性以最简单的方式生成。很少需要在指向新页面或者新的 page 段的链接中指定 data-rel="dialog"。框架自动确保将新内容显示在带有圆角和合适间距的“对话窗口”中。这些对话框通常被创建为模态窗口，在显示期间背景变暗，应用程序暂停。

程序清单 13.7 打开对话框

```
<a href="dialog.html" data-rel="dialog">New Dialog</a>
```

注意

为了清晰起见，我们再次重申，页面是否是对话框并不在页面中确定，而是由打开的链接决定。这是一个不同寻常的概念，必须了解这一点。

页面和一个对话框之间的过渡可以用与标准页面之间的变动相同的方式指定。框架在对话框中的一个链接被单击时自动关闭对话框。可以轻松地创建一个关闭按钮；只需要放置一个链接，指向打开对话框的页面（例如，通过链接中的 data-rel="back"）。也可以使用 dialog() 方法中的 close 参数，从外部关闭对话框。

程序清单 13.8 关闭对话框

```
$('#myDialog').dialog('close').
```

13.7 按钮

PC 上的经典 Web 应用程序中控制主要基于鼠标指针，而移动应用越来越多地是用手指控制（关键字是“触摸屏”）。这使得点击文本形式的经典超链接狭窄的敏感区域更为困难（或者更不准确）。但是，即使能用特殊键控制移动设备上的光标，这种控制通常需要相当高的技巧。这意味着必须使敏感元素变得更大，让移动用户更舒服地进行控制。按钮很适合用于这一目的。除了 PC 应用中完成的常规任务之外，它们还能够承担基于文本的超链接的大部分工作。

按钮的创建和对话框一样简单。所需要做的只不过是将一个链接的 data-role 赋值为 button。

程序清单 13.9 按钮

```
<a href="index.html" data-role="button">Click</a>
```

作为替代，框架也能将类型为 input，type 特性值为 submit、reset、button 或者 image 的元素转换为自定义按钮。

注意

注意对话框和按钮之间的概念差异。尽管代表对话框的页面没有明确的标记，但是在按钮中，超链接被明确地转换为一个按钮（其他窗口小部件也是如此）。

13.7.1 具有图标的按钮

可以轻松地为按钮添加图形化符号。jQuery Mobile 框架提供了一组移动应用常用的图标。这些图标被优化为较小的尺寸和最大的对比度。要使用这些图标，只需要为按钮添加 data-icon 特性。

程序清单 13.10 带有图标的按钮

```
<a href="index.html" data-role="button" data-icon=
"delete">Delete</a>
```

表 13-1 列出目前可用的图标。

表 13-1 按钮图标

图 标	描 述
arrow-l，arrow-r，arrow-u，arrow-d	左、右、上、下箭头
delete	删除
plus，minus	加号和减号
check	对勾
gear	齿轮图标
refresh	刷新

（续）

图　标	描　述
forward，back	前进和后退
grid	# 号
star	星形
alert	警告图标
info	信息图标
home	主页
search	搜索放大镜

提示

我们甚至可以定义自定义图标，并在框架中使用。需要使用一个唯一名称作为特性值，然后框架生成一个以 ui-icon- 开头的新 CSS 类。这个类被赋予该按钮。然后，修改类的主体，指向单独的图像。图像的格式应该为 18×18 像素的 PNG-8 格式（具有 alpha 透明度）。

图标的位置

图标的默认位置是按钮文本的左侧。如果想覆盖这一设置，可以使用 data-iconpos 特性。然后，可以将图标放在文本右边（right）、顶部（top）或者下方（bottom）。使用 data-iconpos="notext"，可以完全消除按钮上的文字。

程序清单 13.11　在按钮中定位一个图形

```
<a href="index.html" data-role="button" data-icon="delete"
data-iconpos="right">Delete</a>
```

13.7.2　块元素或者行内元素

在默认设置中，所有按钮被作为块元素生成（至少在页面的中心内容区域是如此）。[⊖]因此，它们占据整个页宽[⊜]并生成一个换行。如果为这个按钮设置 data-inline="true" 特性，就会创建一个行内元素，只占据内容所需的空间，避免增加换行。相应地，false 值生成一个块元素。

13.7.3　分组

也可以从视觉上将按钮合并成组。它们被放入一个指定 data-role="controlgroup" 特性的容器中。在默认设置中，按钮垂直组合，按钮之间的所有空白都被移除。阴影效果也会重新设计，以便使该组形成一个视觉单元。

⊖ 在页脚中，这个设置也可以不同于默认设置，取决于所选择的主题。

⊜ 请牢记，页面通常指的是移动设备相当小的屏幕，这个敏感的按钮必须用一个手指操作（在手机是触摸屏的情况下），因此，建议使用一个“大”按钮。

程序清单 13.12 组合按钮

```
<div data-role="controlgroup">
  <a href="one.html" data-role="button">Yes</a>
  <a href="two.html" data-role="button">No</a>
  <a href="#" data-role="button">Cancel</a>
</div>
```

提示

通过 data-type="horizontal"，也可以水平放置元素。按钮也就会被格式化为行内元素。

13.7.4 实用示例

现在，又到了用一个完整的使用示例说明前面的解释的时候了。记得我们在上一章中提出的自定义插件——音乐调性生成器吗？让我们将其用在一个移动应用中。[⊖]这也意味着，我们现在将在移动框架标记基础上，明确引入 JavaScript 和 jQuery 功能。在实践中，除了窗口小部件和界面之外，我们将在移动应用中提供程序逻辑。

我们首先来看看网页 ch13_3.html。我们打算研究整个网页（除了最开始的几行之外），因为标题的结构也和少数几个位置相关。

程序清单 13.13 移动应用的 HTML 代码

```
...
15    <script type="text/javascript"
16       src="lib/jquery-rjskeygenerator/jquery.keys.js"
17       ></script>
18    <script type="text/javascript"
19       src="lib/ch13_3_ready.js"></script>
20  </head>
21  <body>
22    <div data-role="page" id="p1">
23      <div data-role="header">
24        <h1>The RJS Key Generator</h1>
25      </div>
26      <div data-role="content">
27        <a id="generator" href="#p2"
28           data-rel="dialog" data-role="button"
29           data-inline="true" data-icon="forward"
30           data-transition="slide">To the key</a>
31      </div>
32       <div data-role="footer">
33        <a href="http://rjs.de"
34           data-role="button">(c) www.rjs.de</a>
```

⊖ 在这个例子中，我还受到了个人兴趣的激励：因为在我们的练习期间这个程序可以从手机上获得，我们就不再需要争论谁在乐队热身的时候负责选择调性。

```
35      </div>
36    </div>
37    <div data-role="page" id="p2">
38      <div data-role="header">
39        <h1>The RJS Key Generator</h1>
40      </div>
41      <div data-role="content">
42        <p>
43          <div id="key"></div>
44          <div data-role="controlgroup"
45             data-type="horizontal">
46            <a data-rel="back" href="#"
47               data-role="button" data-icon="delete"
48               data-iconpos="right">Close</a>
49            <a data-role="button" data-icon="refresh"
50               data-iconpos="right" id="new">New</a>
51          </div>
52        </p>
53      </div>
54      <div data-role="footer">
55        <a href="http://rjs.de"
56           data-role="button">(c) www.rjs.de</a>
57      </div>
58    </div>
59  </body>
60 </html>
```

在网页的开头（在此没有列出），你将再次看到对所用框架的 CSS 和 JavaScript 程序库的引用。在第 15 ～ 17 行，可以看到对我们使用的插件的引用。

另外，该网页仍然由两个页面组成。起始页面在标题区域中为用户提供了一个标题、在页脚提供了版权信息（在本例中很敏感），并在主内容区域提供了一个按钮。该按钮使用了前面描述的多种标记。

在第 27 行中，可以看到指向 ID 为 p2 的页面的链接。该页面是一个对话框。但是在此我们要重申，链接页面是对话框并不是在目标页面中确定的，而是在打开链接的时候确定的。可以在第 28 行中看到这一切。该行还指定框架应该将链接表示为按钮。

在第 29 行，我们指定一个内联按钮，并显示图标。第 30 行中的最后一项配置确定了到打开的对话窗口的过渡效果。如图 13-7 所示为起始页面。

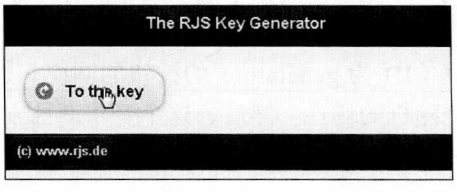

图 13-7 起始页面

在第 33 行和第 34 行，我们将页脚内容转换为一个指向外部资源的按钮。如果查看屏幕截图，就会看到这个按钮被当作内联元素，尽管我们并没有明确地指定。在这种情况下，为页脚区域选择的主题决定了这一点。如图 13-8 所示为

图 13-8 页脚的一个敏感按钮

页脚的一个敏感按钮。

在第 37 ～ 58 行中描述了第二个页面。我们已经在第 27 行将其标记为对话框。在第 43 行，可以看到一个 DIV 区域。该区域完全是“正常”的。这意味着，移动框架的标记在此没有产生效果。我们打算在这个区域显示插件选择的音乐调性。所以，这里和“经典”的 JavaScript 或者 jQuery 编程取得了联系。

在第 44 ～ 51 行中，可以看到按钮的水平组合。这里组合了两个按钮。第 46 ～ 48 行定义的按钮关闭对话框。第二个按钮不使用 href 和 data-rel 特性。它明显不使用移动框架的默认响应。但是，我们当然也可以在移动应用中使用常规的 jQuery 事件处理。这也就是我们在例子中所做的（注意第 50 行中按钮的 ID），将很快在对 JavaScript 文件 ch13_3_ready.js 文件进行讨论时看到这一点。但是首先应该注意两个组合按钮有安排在右侧的图标。

下面是 JavaScript 文件 ch13_3_ready.js。

程序清单 13.14　自定义 JavaScript 功能

```
01 $(function(){
02  $("#generator").click(function(){
03    ta();
04  });
05  $("#new").click(function(){
06    ta();
07  });
08 });
09 function ta(){
10  $("#key").keys({
11    color: "blue",
12    background: "gray",
13    width: "100px",
14    height: "30px",
15    fontSize: "20px"
16  });
17 }
```

在上述 JavaScript 文件中可以看到已经很熟悉的 jQuery 编程。在第 9 ～ 16 行中可以看到一个函数，用某个选项调用插件并输出返回值。

在 ready 方法中，指定了对两个单击事件的响应；其中之一是对第一个页面上打开对话框（ID 为 generator）的按钮的响应。这意味着在打开对话框时，调用插件并选择和返回一个随机的调性。然后，这个音乐调性立刻显示在对话窗口中。单击 ID 为 new 的按钮重复这一选择过程，但是这个按钮在对话窗口中。从可用性的角度来说，这意味着用户可以选择一个音乐调性而无需关闭和重新打开对话框。

在本节的最后，我们来简单地看一下 CSS 文件 ch13_3.css，因为我们在这个文件中也指定了少数细节。

程序清单 13.15 CSS 文件

```
#key {
  margin:auto;
}
```

这里，输出音乐调性的 div 区域居中显示。无法在插件的选项中指定这个样式。[⊖]对话框如图 13-9 所示。

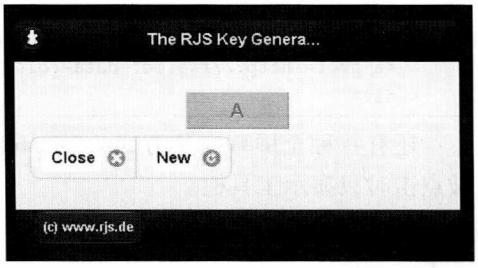

图 13-9 对话框

13.8 工具栏和导航栏

工具栏是框架中窗口小部件的一种特殊形式。它们用于页眉和页脚，[⊖]正如我们在之前的页面中所做的那样。页眉是页面实际开始的地方，在那里除了标题行之外，有时也会出现一两个按钮。页脚是页面结束的地方。在这些区域添加水平导航或者选项卡很方便。jQuery Mobile 框架包含一个导航栏窗口小部件（data-role="navbar"），将包含链接的无序列表转换为带有按钮的水平栏。为了将其中一个链接设置为预设的（活动），可以为锚元素添加 class="ui-btn-active"。

程序清单 13.16 带有预设链接的导航栏

```
<div data-role="footer">
  <div data-role="navbar">
    <ul>
      <li><a href="http://rjs.de"
        class="ui-btn-active">Home</a></li>
      <li><a href="http://blog.rjs.de">Blog</a></li>
    </ul>
  </div>
 </div>
```

提示

如果嵌套元素适合小屏幕，那么导航栏也适合它们。

页眉和页脚区域的定位默认采用行内（inline）方式。这意味着这些元素处于网页的正常流向中。这样做的好处是它们在所有设备中可见（因为它独立于 JavaScript/CSS 的定位）。这些元素简单地按照可用空间编排。这是一个优势，特别是在移动环境具有许多不同分辨率的情况下。

在某些情况下，固定位置也可能合适。在那种情况下，工具栏在用户滚动时保留在屏幕

⊖ 这没有意义，因为和插件没有直接的关联，而是影响返回值的使用方式。

⊖ 或者和实用工具栏在一起。

区域的开始和结束位置。为了在必要时给实际内容留下更多空间，框架确保在必要时可以通过轻按 / 单击屏幕切换工具栏的可见性。要实现这种行为，可以为工具栏容器指定 data-position="fixed"。

程序清单 13.17 固定式页脚

```
<div data-role="footer" data-position="fixed">
  <a href="http://rjs.de" data-role="button">(c) www.rjs.de</a>
</div>
```

还有一种全屏幕定位方式（data-fullscreen="true"），表现与固定定位类似，但是在页面被点击时只显示工具栏。

警告

在某些平台上，工具栏的固定和切换无法正常工作。

我们来研究一个完整的例子，它对我们的最后一个例子进行了小修改，HTML 文件中的标记根据上面的解释进行了修改（ch13_4.html）。

程序清单 13.18 使用导航栏和特殊格式的工具栏

```
...
21  <body>
22    <div data-role="page" id="p1">
23      <div data-role="header" data-position="fixed">
24        <h1>The RJS Key Generator</h1>
25      </div>
26      <div data-role="content">
27        <a id="generator" href="#p2" data-rel="dialog"
28           data-role="button" data-inline="true"
29           data-icon="forward"
30           data-transition="slide">To the key</a>
31      </div>
32      <div data-role="footer">
33        <div data-role="navbar">
34          <ul>
35            <li>
36              <a href="http://rjs.de"
37                 class="ui-btn-active">(c) www.rjs.de</a>
38            </li>
39            <li>
40              <a href="http://blog.rjs.de">Blog</a>
41            </li>
42          </ul>
43        </div>
44      </div>
45    </div>
46    <div data-role="page" id="p2">
47      <div data-role="header" data-fullscreen="true">
```

```
48        <h1>The RJS Key Generator</h1>
49      </div>
50      <div data-role="content">
51        <p>
52          <div id="key">
53          </div>
54          <div data-role="controlgroup"
55              data-type="horizontal">
56            <a data-rel="back" href="#"
57               data-role="button" data-icon="delete"
58               data-iconpos="right">Close</a>
59            <a data-role="button" data-icon="refresh"
60               data-iconpos="right" id="new">New</a>
61          </div>
62        </p>
63      </div>
64      <div data-role="footer">
65        <div data-role="navbar">
66          <ul>
67            <li>
68              <a href="http://rjs.de"
69                 class="ui-btn-active">(c) www.rjs.de</a>
70            </li>
71            <li>
72              <a href="http://blog.rjs.de">Blog</a>
73            </li>
74          </ul>
75        </div>
76      </div>
77    </div>
78  </body>
79 </html>
```

我们格式化工具栏，并在页脚创建一个带有两个按钮的导航栏。图 13-10、图 13-11 和图 13-12 分别为相应的效果示意图。

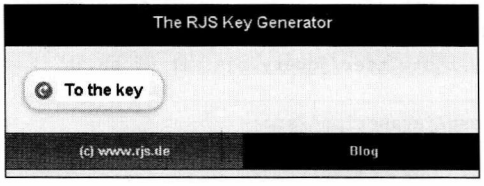

图 13-10　页脚包含一个带有两个按钮的导航栏，左侧的按钮被预先选中

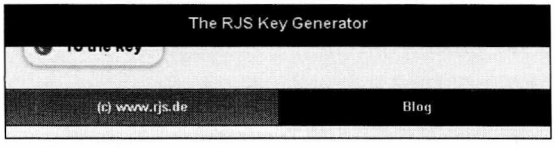

图 13-11　如果浏览器支持，页眉在内容移动时位置保持不变

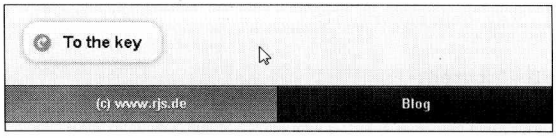

图 13-12 轻按 / 单击屏幕切换标题的可见性

13.9 列表

列表可以清晰地表示数据，这对于移动环境中的小屏幕更为重要。导航也可以用列表设计。jQuery Mobile 框架提供大量默认列表，可以用许多种方式格式化。该框架支持嵌套列表、编号列表、非交互性（只读）列表、带图标的列表、缩进列表和多种其他变种。

本质上，只需要使用通常的 HTML 标记构建一个列表，并为父元素添加特性 data-role="listview"。jQuery Mobile 框架自动添加必要的样式和事件支持。必要时，它完全在后台组织用于新内容的 AJAX 请求，并将其集成到文档对象模型（DOM）中。基本的列表如程序清单 13.19 所示。

程序清单 13.19 列表

```
<ul data-role="listview" data-theme="g">
  <li><a href="http://rjs.de">Home</a></li>
  <li><a href="http://blog.rjs.de">Blog</a></li>
  <li><a href="http://www.ajax-net.de">AJAX-Net.de</a></li>
  <li><a href="http://safety-first-rock.de">
    jQuery Examples</a></li>
 </ul>
```

让我们再次扩展移动参考应用。我们添加第三个页面，在这个页面上使用另一个插件，可以在配套网站上找到这个 rjsTwitter 插件[⊖]（ch13_5.html）。

程序清单 13.20 使用列表

```
...
15    <script type="text/javascript"
16      src="lib/jquery-rjstwitter/jquery.rjstwitter_en.js" >
17     </script>
18     <script type="text/javascript" src=
19       "lib/ch13_5_ready.js"></script>
20   </head>
21  <body>
22     <div data-role="page" id="p1">
23      <div data-role="header" data-position="fixed">
24        <h1>Mobile RJS PlugIns</h1>
25      </div>
```

⊖ 你在目录找到的 jquery-rjstwitter 插件是我们的 Twitter 客户端的插件变种，它通过 JSONP 加载 Tweeter 上的 Tweet，并在输出区域中显示。

```
26      <div data-role="content">
27        <ul data-role="listview" data-theme="g">
28          <li>
29            <a id="generator" href="#p2"
30               data-rel="dialog" data-transition="slide">
31               RJS Key Generator</a>
32          </li>
33          <li>
34            <a id="generator" href="#p3"
35               data-rel="dialog" data-transition="slide">
36               RJS Twitter Client</a>
37          </li>
38        </ul>
39      </div>
40      <div data-role="footer">
41        <div data-role="navbar">
42          <ul>
43            <li>
44              <a href="http://rjs.de"
45                 class="ui-btn-active">(c) www.rjs.de</a>
46            </li>
47            <li>
48              <a href="http://blog.rjs.de">Blog</a>
49            </li>
50          </ul>
51        </div>
52      </div>
53    </div>
54    <div data-role="page" id="p2">
55      <div data-role="header" data-fullscreen="true">
56        <h1>The RJS Key Generator</h1>
57      </div>
58      <div data-role="content">
59        <p>
60          <div id="key"></div>
61          <div data-role="controlgroup"
62            data-type="horizontal">
63            <a data-rel="back" href="#"
64               data-role="button" data-icon="delete"
65               data-iconpos="right">Close</a>
66            <a data-role="button" data-icon="refresh"
67               data-iconpos="right" id="new">New</a>
68          </div>
69        </p>
70      </div>
71      <div data-role="footer">
72        <div data-role="navbar">
73          <ul>
74            <li>
75              <a href="http://rjs.de"
```

```
76                class="ui-btn-active">(c) www.rjs.de</a>
77            </li>
78            <li>
79              <a href="http://blog.rjs.de">Blog</a>
80            </li>
81          </ul>
82        </div>
83      </div>
84    </div>
85    <div data-role="page" id="p3">
86      <div data-role="header" data-fullscreen="true">
87        <h1>The RJS Twitter Client</h1>
88      </div>
89      <div data-role="content">
90        <p>
91          <div id="twitter"></div>
92        </p>
93      </div>
94      <div data-role="footer">
95        <div data-role="navbar">
96          <ul>
97            <li>
98              <a href="http://rjs.de"
99                 class="ui-btn-active">(c) www.rjs.de</a>
100           </li>
101           <li>
102             <a href="http://blog.rjs.de">Blog</a>
103           </li>
104         </ul>
105       </div>
106     </div>
107   </div>
108 </body>
109 </html>
```

在第 85 ～ 108 行中，定义了新的第三个页面。它的结构大家都很熟悉，我们在应用程序中主要修改的是起始页面上的导航。我们在第 27 ～ 38 行中用一个有两个项目的列表代替按钮。两者都启动一个对话框，对话框的特性也和前面的按钮相同（除了明确标记按钮的特性之外）。敏感列表如图 13-13 所示。

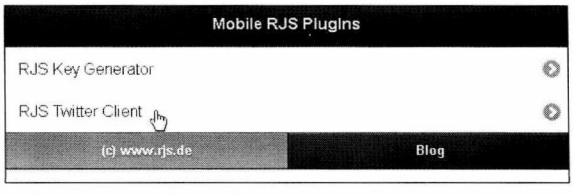

图 13-13　敏感列表

如果用户单击列表的第一个项目，就会进入包含音乐调性生成器的对话框。如果单击第

二个项目，就会调用包含 Twitter 客户端的新对话框。如图 13-4 所示。

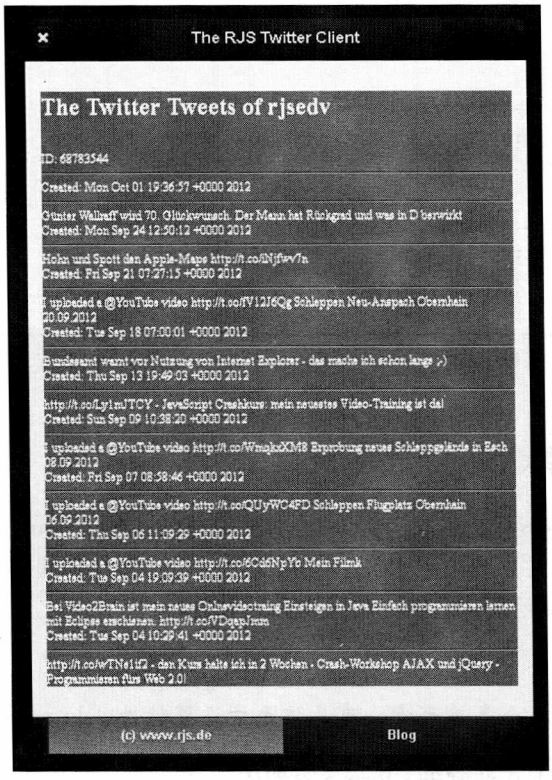

图 13-14 第 3 页是一个包含 Twitter 客户端的对话框

我们来看看 JavaScript 文件 ch13_5_ready.js，其中集成了新的插件。

程序清单 13.21 使用少数选项的新插件

```
01 $(function(){
02   $("#generator").click(function(){
03     ta();
04   });
05   $("#new").click(function(){
06     ta();
07   });
08     twitter();
09
10 });
11 function ta(){
12   $("#key").keys({
13     color: "blue",
14     background: "gray",
15     width: "100px",
```

```
16      height: "30px",
17      fontSize: "20px"
18    });
19 }
20 function twitter(){
21    $("#twitter").rjsTwitter({
22      background: "gray",
23      color: "white",
24      fontSize: 12,
25      imageDisplayed: false,
26      idDisplayed: true
27    });
28 }
```

在第 8 行中，插件于网页加载时被调用。第 20 ～ 27 行中定义了该函数。

13.10 表单元素

图形用户界面中必不可少的交互组件是基于表单元素的。jQuery Mobile 框架提供为移动设备进行优化的所有常见表单元素。它特别针对手指控制。但是本质上，所有表单元素都基于原生的 HTML 表单元素，框架在后台初始化和改写它们。

提示

如果想让框架改写某个表单元素，可为该元素设置特性 data-role="none"。

该表单元素应该包围在通常的 <form> 容器中。

警告

如果在表单（或者元素）中使用 ID，应确保这些 ID 在网页中是唯一的。这一规则因为在网页中构造多个页面的选项而变得模糊。但是即便如此，ID 仍然应该在整个网页中保持唯一性。

从根本上说，所有表单元素都被指定一个动态宽度，以适应不同的屏幕尺寸。不同元素之间的编排也可能根据屏幕宽度而有所不同（上下排列，或者左右排列）。

13.10.1 字段容器

为了确保标签和表单元素适应较宽的屏幕，框架的文档中建议将它们包含在一个 div 或者 fieldset 元素中，并添加特性 data-role="fieldcontain"。然后，框架将这些元素放到一起管理，并添加外框等合适的设计。

13.10.2 各种不同的表单元素

大家已经熟悉了不同的表单元素，所以我们现在将观察一个完整的示例，它包含所有当

前可用的表单元素（ch13_6.html）。

程序清单 13.22 jQuery Mobile 中的不同表单元素

```
...
22  <body>
23    <div data-role="page">
24      <div data-role="header">
25        <h1>Form Elements</h1>
26      </div>
27      <div data-role="content">
28        <form action="#" method="get">
29        <div data-role="fieldcontain">
30          <label for="name">
31            Text input:
32          </label>
33          <input name="name" id="name" type="text" />
34        </div>
35        <div data-role="fieldcontain">
36          <label for="textarea">
37            Text area:
38          </label>
39          <textarea cols="30" rows="5"
40              name="textarea" id="textarea"></textarea>
41        </div>
42        <div data-role="fieldcontain">
43          <label for="search">
44            Search input:
45          </label>
46          <input type="search" name="password"
47              id="search" />
48        </div>
49        <div data-role="fieldcontain">
50          <label for="slider">
51            Slider:
52          </label>
53          <input type="range" name="slider"
54              id="slider" value="50" min="0" max="100" />
55        </div>
56        <div data-role="fieldcontain">
57          <label for="um">
58            Switch
59          </label>
60          <select name="um" id="um" data-role="slider">
61            <option value="off">On</option>
62            <option value="on">Off</option>
63          </select>
64        </div>
65        <div data-role="fieldcontain">
66          <fieldset data-role="controlgroup">
67            <legend>
```

```
68              Options
69            </legend>
70            <input type="radio" name="radio-choice-1"
71               id="radio-choice-1" value="choice-1"
72               checked="checked" />
73            <label for="radio-choice-1">
74              Male
75            </label>
76            <input type="radio" name="radio-choice-1"
77               id="radio-choice-2" value="choice-2" />
78            <label for="radio-choice-2">
79              Female
80            </label>
81          </fieldset>
82        </div>
83        <div data-role="fieldcontain">
84          <fieldset data-role="controlgroup">
85            <legend>
86              Checkboxes:
87            </legend>
88            <input type="checkbox" name="checkbox-1"
89               id="checkbox-1" />
90            <label for="checkbox-1">
91              OK
92            </label>
93          </fieldset>
94        </div>
95        <div data-role="fieldcontain">
96          <label for="select-choice-1" class="select">
97            Selection menu
98          </label>
99          <select name="select-choice-1"
100             id="select-choice-1">
101           <option value="beer">Beer</option>
102           <option value="wine">Wine</option>
103           <option value="coke">Coke</option>
104         </select>
105       </div>
106           </form>
107     </div>
108     <div data-role="footer">
109       <h4>jQuery Mobile</h4>
110     </div>
111   </div>
112 </body>
113 </html>
```

可以看到，整个应用的内容区域实际上是一个表单（第 28 ～ 106 行）。在这个表单中，可以看到标记为 data-role="fieldcontain" 的容器中的表单元素。第 60 ～ 63 行中的开关值得注意，它必须用 data-role="slider" 加以标记。此外，第 66 行中的 fieldset 必须用 data-

role="controlgroup" 标记为单选按钮，第 84 行中则用 data-role="controlgroup" 标记为复选框。包含所有标准元素的表单如图 13-15 所示，菜单在被启用时以弹出方式打开如图 13-16 所示。

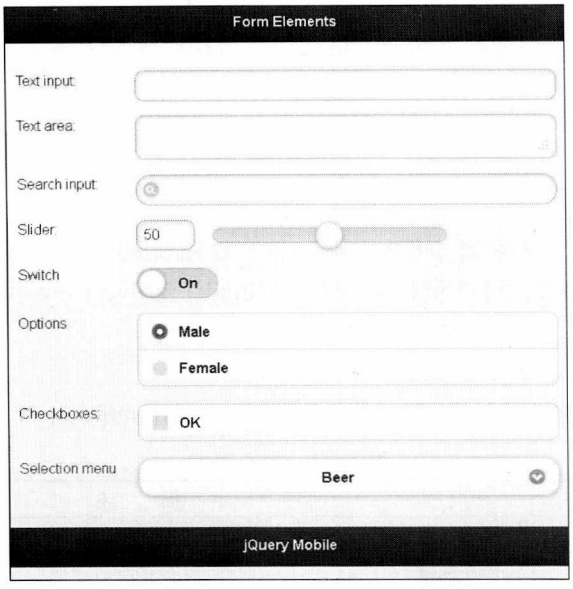

图 13-15 包含所有标准元素的表单

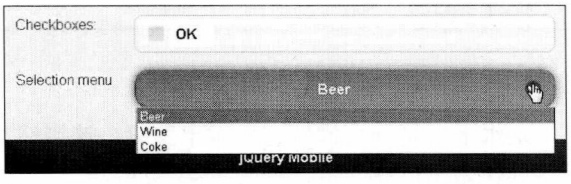

图 13-16 选择菜单在被启用时以弹出方式打开

13.10.3 表单元素的插件方法

在 jQuery Mobile 相应地生成或者改写表单元素之后，可以用多种集成方法与它们取得联系。

用 .selectmenu('open') 可以打开菜单，用 .selectmenu('close') 可将其关闭，用 .selectmenu ('refresh') 可以刷新菜单，用 .selectmenu('disable') 可以禁用菜单，而用 .selectmenu('enable') 可以启用菜单。

文本输入也可以用 .textinput('enable') 启用，用 .textinput('disable') 禁用。

同样的方法也适用于复选框——.checkboxradio('enable')、.checkboxradio('disable') 和 .checkboxradio ('refresh')，对滑动条则使用 .slider('enable') 、.slider('disable') 和 .slider

('refresh')；对按钮使用 .button('enable') 和 .button('disable')。

13.10.4 发送表单数据

在 jQuery Mobile 中，发送表单数据在可能的情况下将通过 AJAX 完成。但是如果数据通过 GET 方法发送，结果数据可能通过框架的后台操作保存为书签。URL 在数据到达时相应做出修改。

13.11 特殊事件

移动框架提供了对一些特殊事件的支持。这些事件根据原生事件创建，但是不完全可用于全部移动设备。这些事件可以和平常一样，和 bind() 及 live() 一起使用。

13.11.1 触摸事件

表 13-2 描述了特别针对触摸屏上不同类型的触摸操作的事件。

表 13-2 触摸事件

事　件	描　述
tap	轻按操作是快速而完整的触摸动作
taphold	轻按并保持
swipe	在一秒钟内水平拖动 30 或者更多个像素，在垂直方向上少于 75 个像素
swipeleft	向左轻扫
swiperight	向右轻扫

13.11.2 方向变化

许多移动设备对我们是水平还是垂直握持它们做出反应。这当然也会触发一个事件，可以对其做出响应。对设备旋转做出反应的事件如表 13-3 所示。

表 13-3 对设备旋转做出反应

事　件	描　述
orientationchange	如果设备被旋转，可以在回调函数中通过第二个参数和值“portrait”或者“landscape”求取方向

13.11.3 滚动事件

也可以对内容滚动相关的各种事件做出响应。各种滚动事件如表 13-4 所示。

表 13-4 滚动事件

事　件	描　述
scrollstart	滚动开始
scrollstop	滚动结束

13.11.4 页面事件

页面的初始化、显示和隐藏与表 13-5 中列出的事件相关联。

表 13-5 页面事件

事 件	描 述
pagebeforecreate	在创建页面之前触发的初始化事件
pagebeforehide	开始隐藏。这描述了页面实际隐藏之前触发的事件
pagebeforeshow	显示开始（通常是在动画中）。这描述了在页面实际显示之前触发的一个事件
pagecreate	在页面创建之后触发的一个初始化事件
pagehide	页面完全隐藏
pageshow	页面完全显示

我们用一个包含两个页面分段的网页作为响应事件的例子，对起始页面的隐藏和显示做出响应（ch13_7.html）。

程序清单 13.23 响应事件

```
...
18  <body>
19     <div data-role="page" id="p1">
20      <div data-role="header">
21        <h1>Page 1</h1>
22      </div>
23      <div data-role="content">
24         <h2></h2>
25               <a href="#p2">To Page 2</a>
26      </div>
27      <div data-role="footer">
28        <h4>(c) www.rjs.de</h4>
29      </div>
30    </div>
31         <div data-role="page" id="p2">
32      <div data-role="header">
33        <h1>Page 2</h1>
34      </div>
35      <div data-role="content">
36        <p>
37          <h2></h2>
38          <a data-rel="back" href="#">Back</a>
39        </p>
40      </div>
41      <div data-role="footer">
42         <h4>(c) www.rjs.de</h4>
43      </div>
44    </div>
45  </body>
46 </html>
```

特定的响应发生在 JavaScript 文件 ch17_7_ready.js 中。加载后的页面如图 13-17 所示。

程序清单 13.24 对页面的隐藏和显示做出响应

```
01 $(function(){
02  $('#p1').live('pageshow', function(event, ui){
03    $("h2:eq(0)").html(
04       "The first page is now displayed again. " +
05       "Triggering element: " + event.target);
06  });
07  $('#p1').live('pageshow', function(event, ui){
08    $("h2:eq(1)").html(
09       "The first page is now hidden. " +
10       "Triggering element: " + event.target);
11  });
12 });
```

对于隐藏和显示[⊖]，我们通过 live() 方法绑定了两个事件。在隐藏起始页面之后，我们在第二个类型为 h2 的标题（在第二个页面上）中显示信息。效果如图 13-18 所示。

反之亦然，我们在页面再次显示时，于第一个类型为 h2 的标题（在第一个页面上）显示不同的信息。效果如图 13-19 所示。

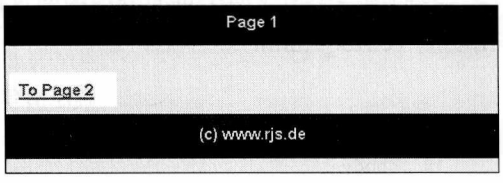

图 13-17 加载之后的页面

图 13-18 起始页面被隐藏

图 13-19 起始页面再次显示

13.12 主题框架和通用内容设计

根本上，在 jQuery Mobile 中应该尽量将内容设计留给浏览器，尽可能少地传输布局开销，甚至强制统一的设计，完全按照 HTML 第一个版本的传统。在默认设置下，框架使用 HTML 的标准样式和尺寸规范。前面已经提到过，应该避免不灵活的面向设计标记（如 <font>）。基本上，框架只在表格和字段集上添加一些这一级别的样式，使它们更容易管理。其他任何样式都被转移到精炼的 CSS 类中。

在 jQuery Mobile 下，还有一个丰富的主题（Theme）框架。CSS 规则的集中分配通过

⊖ 只适用于在初始创建之外的时候隐藏或者显示元素的情况。

data-theme 特性进行。可以将这个特性赋予标题或者页脚，改变其显示。也可以将这个特性赋予页面的实际内容区域，但是框架文档建议将该特性赋予单独的页面——换言之，所有用 data-role="page" 标记的容器。但是，也可以单独设计其他窗口小部件。

注意

框架的默认主题的目标是得到最优的对比度。

移动 CSS 主题框架的整个概念基于 jQUery UI 的 ThemeRoller，但是添加了几个源于特殊移动条件的扩展。特别是，它大量使用了 CSS3 的属性（例如，实现圆角和梯度以及最大程度压缩的图标）。和使用图形的实现相比，这显著地减少了传输数据量。而且，每个主题都提供多种颜色组合和不同的设计。

移动主题系统将结构样式（如尺寸和缓存）的颜色和文本单独分开。所以，可以定义这些规则，并在闲暇的时候混合或者重新组合它们。

框架中的每个主题都包含字体、阴影效果、颜色和圆角半径等全局设置。

提示

jQuery Mobile 1.2 提供了单独的 ThemeRoller 工具，可以用它构建主题，然后下载自定义 CSS 文件，这些文件可以很简单地放入项目中。

也可以用框架创建纯粹基于 CSS 的网格布局，但是不建议这么做。用于这种布局的类以 ui-grid 开头。可以在文档中找到更多细节。

jQuery Mobile 的 5 个默认主题简单地按照字母“编号”（a、b、c、d、e）。a 主题使用最大的对比度，e 主题使用最多的颜色。我们来看看示例 ch13_5.html 中的一个变种，其中指定了主题 e（ch13_8.html）。各种不同的主题如图 13-20 和图 13-21 所示。

程序清单 13.25 使用不同的主题

```
...
21  <body>
22      <div data-role="page" id="p1" data-theme="e">
...
27          <ul data-role="listview" data-theme="d">
...
40      <div data-role="footer" data-theme="e">
...
55      <div data-role="header" data-fullscreen="true"
          data-theme="b">
...
85    <div data-role="page" id="p3">
...
94      <div data-role="footer" data-theme="e">
...
```

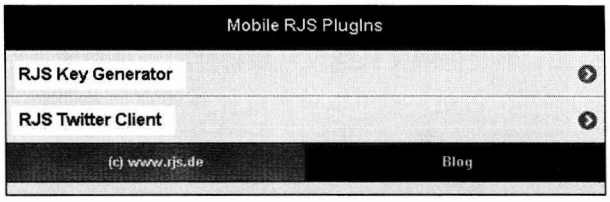

图 13-20 如果选择不同的主题，起始页面将得到一个新的颜色主题

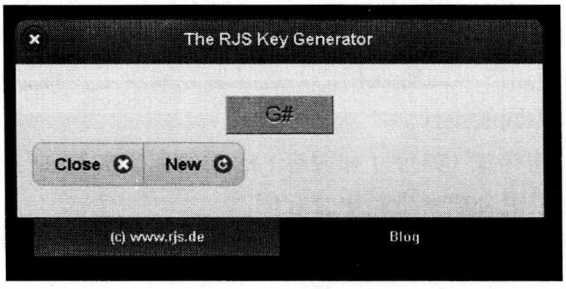

图 13-21 也可以嵌套或者混合主题

提示

如果你打算创建自定义主题，应遵循使用字母的命名约定。

13.13 收起和展开内容

在 jQuery Mobile 中，可以轻松地创建用户单击时展开和收起的内容，只需对容器添加 data-role="collapsible" 即可。在容器中，可以编写所有 HTML 标题。框架设计标题，使它们看上去像可点击的按钮，并在内容收起时显示一个加号。如果内容被展开，你将看到一个减号。在默认设置中，内容加载时展开，但是如果设置特性 data-collapsed="true"，内容在加载时就会收起（ch13_9.html）。

程序清单 13.26 包含可展开和收起的内容的页面

```
...
18   <body>
19    <div data-role="page">
20      <div data-role="header">
21        <h1>Famous Quotes</h1>
22      </div>
23      <div data-role="content">
24        <p>
25          <div data-role="collapsible">
26            <h3>Aristotle</h3>
```

```
27            <p>
28              The whole is greater than
29               the sum of its parts.
30            </p>
31          </div>
32          <div data-role="collapsible"
33             data-collapsed="true">
34            <h3>Aristotle</h3>
35            <p>
36              Hiding a mistake through a lie
37               means replacing a stain
38               through a hole.
39            </p>
40          </div>
41          <div data-role="collapsible"
42             data-collapsed="true">
43            <h3>Archimedes</h3>
44            <p>
45              Give me place to stand,
46               and I shall move the Earth.
47            </p>
48          </div>
49        </p>
50      </div>
51      <div data-role="footer">
52        <h4>Aristotle and Archimedes </h4>
53      </div>
54    </div>
55  </body>
56 </html>
```

在例子中，可以看到 3 个可展开和收起的区域。第一个区域在加载时展开，其余收起。效果分别如图 13-22 和图 13-23 所示。

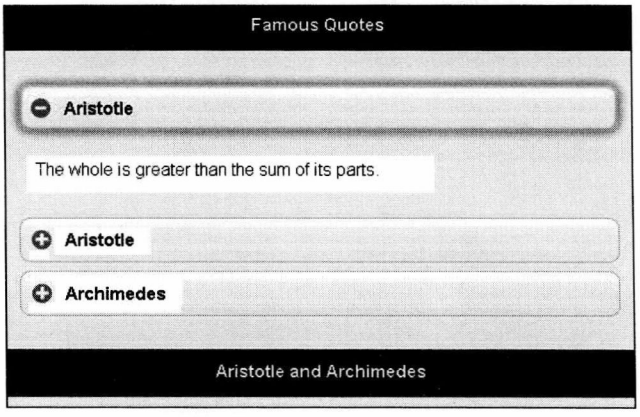

图 13-22 加载应用之后

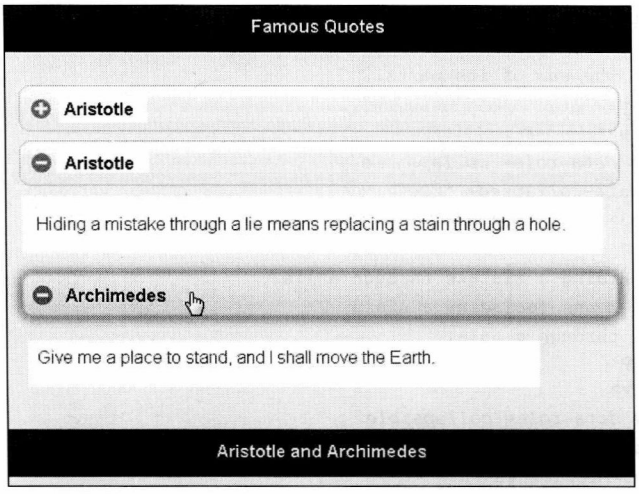

图 13-23 现在，第一个区域被收起，其他两个区域展开

13.14 小结

在本书的最后一章中，我们学习了如何将 jQuery 带入移动应用的世界里。jQuery Mobile 框架是一种极有前途的方法，它高度封装了移动环境中不同平台和条件带来的无数困难。它很容易且非常可靠（在它可能的范围内），得到了广泛的支持。可以肯定的是，在新的移动设备上，支持水平会继续提高。这意味着，基于 HTML5、CSS3 和 JavaScript 的移动 RIA 正在成为专属方法的替代品。我相信，未来就在于此。但是现在，我希望你利用 jQuery 和在本书中学到的所有知识去得到快乐和成功。

附　录

A.1　JavaScript 基本信息概述

有效地用 jQuery 编程需要 JavaScript 语法和概念方面的知识。本附录提供了对最重要的 JavaScript 技术的简短介绍，以便帮助大家理解 jQuery 的工作原理和方法的使用。对于完全理解 JavaScript 的读者，应该寻找其他的 JavaScript 资源。

A.1.1　大小写敏感性

JavaScript 在变量、关键字、函数标识符和方法标识符中区分大小写。

A.1.2　变量、字面量和数据类型

当然，JavaScript 也有变量、数据类型和字面量（Literal）。然而，JavaScript 与“真正”的编程语言有一些不同的特殊性。

注意

字面量是一种清晰定义和不能修改的值，例如数字或者文本。而变量代表着可以临时存储值的内存命名位置。字面量和变量都需要数据类型，它规定了变量或者字面量在主存中的大小、信息的类型以及操作如何进行。

在 JavaScript 中，数据类型不是在声明（在第一次出现的时候）中通过关键字明确规定的。数据类型是通过变量赋值的类型隐含规定的。如果后来为变量赋了不同类型的值，变量的数据类型甚至可以在脚本运行期间改变。数据类型由 JavaScript 在内部进行管理。所以，JavaScript 是弱类型语言，只有少数数据类型。

尽管 JavaScript 是弱类型语言，仍然实现了多种数据类型（被称作内建数据类型）。这些数据类型的功能和更强大的编程语言（如 Java 或者 C#）中的数据类型近似。

注意

jQuery 扩展了 JavaScript 中的数据类型范围，添加了自己的数据类型。这些类型被称作

虚拟类型和伪类型。

经典 JavaScript 数据类型

当然，在 jQuery 中可以使用所有经典的 JavaScript 数据类型。我们在此不作详细的讨论，只是简单地介绍一下。

字符串

字符串是包围在双引号和单引号中的文本。利用两种不同的字符串形式，可以嵌套字符串。在 JavaScript 中，字符串也被看作一个对象，该对象提供一些有趣的方法（例如，转换为大写以及搜索文本）。

提示

除了原生的字符串方法之外，jQuery 提供了多种字符串处理方法。例如，jQuery.trim() 删除字符串开始和结尾的多余空格。

Number、Integer、Float 和 Math 类

Number 类型代表 JavaScript 中的数值。Integer 是代表整数的子类，Float 则代表浮点数。可以对这种类型的变量和字面量应用通常的数学运算。这种表现形式没有必要进行任何的解释。除了纯粹的数据类型之外，JavaScript 以 Math 类的形式为数值处理提供了多种工具（属性和方法）。例如，利用这些工具，可以计算正弦或者生成随机数。

在处理变量或者字面量时，有时候结果可能不是数字或者表现为无穷大。对于这两种情况，JavaScript 提供了定义的标志 NaN（非数字）和 Infinity。

Object

Object 数据类型代表对象。在 JavaScript 组中，这样的对象可以用构造方法或者声明方式创建，或者根据环境存在。

布尔类型

JavaScript 中的 Boolean 值由 true 或者 false 表示。

警告

在 JavaScript 中，其他数据类型也可以表示布尔值，这和 Java 或者 C# 等专业或者强类型语言形成对比。例如，空串或者数字 0 被解释为 false；相应地，其他值被解释为 true。但是对象永远不会被解释为 false，总是代表 true。后者通常用于浏览器嗅探的编程中。（用于测试对象是否存在，如果不存在，就会被解释为 false。）

A.1.3 函数和方法

函数和方法是将一条或者多条指令放到一个代码块（或者子例程）中，为其取一个名称或者标识符。JavaScript 的一个重要特征是组成子例程会使某些命令步骤在网页中加载脚本

时不被执行。如果不这么做而是简单地在网页中将命令写入 <script>，由于顺序处理，这些命令将被执行。这往往不是你所希望或者正确的做法。只有在条件成立（例如，如果用户单击某处、离开网页或者结束某些数据的输入）时，函数或者方法才会通过其标识符被调用，并执行其命令序列。最后（但并非不重要），组成函数使我们可以创建程序库，这也是 jQuery 程序库的构成方式。

可以用 JavaScript 的 function 关键字跟上函数或者方法的标识符，最后加上一对圆括号，在其中包含所有传递给函数的必要参数，来声明函数或者方法。参数是可以在函数中局部使用的变量。如果需要多个参数，它们在括号中用逗号分隔。声明后面是包围在花括号中的指令块，没有前导的引号。可以选择通过关键字 return 指定函数的返回值。

程序清单 A.1 展示了 JavaScript 中函数或者方法的典型声明。

程序清单 A.1 典型的函数声明

```
function [function identifier]([optional parameters]) {
..any statements
  return [return value];
}
```

注意

对于专家来说，JavaScript 中的函数通过 this 某个类型的对象声明中的函数引用或者构造方法的匿名标记法就可以变成一个方法。注意，在第一种情况下，它仍然可以被当作一个纯函数（没有前导的对象）调用，这在严格的面向对象语言中是不可能的。

函数调用与函数引用的对比

现在，我们转向函数调用和函数引用的不同之处。如果声明了一个函数，它必须被调用才能具有生命力。但是要怎么做呢？方法之一是通过函数调用。一般来说，函数调用的方式是函数标识符后面跟上一对括号，其中放入打算传递的参数值。函数调用以分号结束，和脚本中的正常语句一样。当脚本处理到这个时点，函数被执行。

与函数调用相比，纯粹从形式上讲，函数引用没有任何括号。这只是对函数的一个引用；它并没有调用函数。那么，对于函数引用有一个问题：何时执行被引用的函数？答案很简单：函数应用通常与时间处理器组合使用，在特定的事件发生时调用引用的函数。

函数参数：arguments

前面已经提到过，可以向函数传递任何值或者参数。在函数中，有一个特殊的变量 arguments。这个变量总是可用，包含参数的具体信息。特别是参数的数量，因为 arguments 是一个变量，它具有 length 属性。

命名函数与匿名函数的对比

在声明一个函数时，可以（但是并不是必需）省略函数名称。这被称作匿名函数（与命名函数相对）。

匿名函数声明适合于只在某个时点可用的函数。这里不需要函数标识符。这样的匿名函数可以赋予一个变量或者传递给一个方法。

程序清单 A.2 匿名函数作为方法的参数

```
$("#toggle2").click(function(){
  $("img").attr({src:"http://rjs.de/bilder/ducky.gif"});
});
```

注意

jQuery 非常密集地使用匿名函数。在应用编程接口（API）和用 jQuery 创建自己的脚本中都是如此。

回调

回调函数（Callback Function，简称回调）是一个 JavaScript 函数（可能有也可能没有返回值）。这就是我们在前面的示例中看到的，在例子中，匿名函数被传递给 click() 方法作为参数。可以看到，有些回调就是在某些状态出现时出发的事件。

注意

jQuery 的整个事件系统都使用回调（非常频繁和一致）。

大部分回调提供参数和一个上下文，对于大部分事件回调，参数是一个事件对象，上下文被设置为处理的元素。

递归函数调用

JavaScript 支持递归——也称为递归调用。这意味着函数简单地在执行的某个时点调用自身。

A.1.4 JavaScript 中的对象

JavaScript 原生编程的几乎所有能力基于一个事实：可以使用对象提供某些功能。考虑一下文档对象模型（DOM）的概念，jQuery 等框架的潜力完全基于对象。

为了能够使用 JavaScript 的对象，可以采用不同的方法。在 JavaScript 中，对象可以直接使用。这些对象是运行时环境的标准对象，或者浏览器在加载网页时作为网页 XHTML 结构的结果自动创建和提供的对象。后者适用于 DOM 概念。jQuery 等程序库也可以不同的方式提供对象。

构造方法

在其他情况下，必须明确地自己创建对象。为了在 JavaScript 中从属性的说明和对象（类）的功能创建一个新的对象实例，通常使用保留的关键字 new 加上类标识符，然后加上一对括号（如果必要，其中包含参数）。也可以使用对象声明或者构造器方法（或者简称构造器）。使用构造方法创建一个对象在语法上通常如程序清单 A.3 所示。

程序清单 A.3 在 JavaScript 中使用构造方法

```
new [object declaration]([optional parameters]);
```

注意

可以使用大量的 JavaScript 默认类创建对象。但是，在许多情况下如果可以通过对象声明描述脚本的某些功能，那将是很有帮助的。尽管 JavaScript 不是真正面向对象的，也可以用 JavaScript 创建自己的构造方法或者对象声明。

通过对象字面方式进行声明性的对象创建

作为使用构造方法创建对象的一个替代，也可以通过对象字面语法声明式地创建 JavaScript 对象。不需要构造方法，这简化了对象的创建，特别是对初学者来说。

程序清单 A.4 通过对象字面语法声明式创建，并直接初始化两个属性

```
var x = {
   name: "Felix", age: 11
 };
```

注意

创建对象的这种方法在使用 jQuery 方法时是首选方法。

在声明式地创建对象时，只要指定希望对象拥有的属性和方法即可。但是因为没有构造方法，无法从一个对象的描述中创建另一个对象，而是必须再次描述另一个对象，即使结构完全相同。

警告

注意，对象字面语法有一个用于赋值的冒号和一个用于分隔属性的逗号。前面已经提到过，这是一种声明性的语法而不是经典的赋值。

访问对象成员

为了访问对象的各个成员，通常使用句点标记法。作为替代，也可以使用数组标记法。这种标记法通过数据字段标记法读写属性，可以用有意义的名称作为索引。

程序清单 A.5 通过数组标记法访问

```
var x = {
  name: "Thomas",
  age: 11
};
var index = "name";
document.write(x[index]); // "Thomas"
```

这种方法也称作关联数组。在 JavaScript 中，这些数组不是直接可用的，但是可以看到，如果必要，这种访问方法很容易重现。

类元素

在 JavaScript 中还有类元素。可以用前导类名使用这些元素，而不必先用构造方法闯进一个对象。例如，JavaScript Math 类的不同属性和方法就是类元素。

原型化、扩展、创建新数据类型

JavaScript 不完全是面向对象的语言，没有严格的面向对象语言中的那种类和继承机制。然而，仍然可以编写自己的对象定义，用这些定义创建自定义对象。用 JavaScript 创建自己的对象实际上需要两步：

1. 必须按照前面的描述创建自己的对象声明。

2. 通过对象声明，可以创建一个具体的对象实例，就像预定义的对象声明一样。

这种方法是 jQuery 框架的支柱之一。对于 jQuey 来说，纯 JavaScript 的功能通过添加许多自己的对象而得到扩展，这在许多情况下意味着编写新的对象声明。

原型化（Prototyping）是另一个支柱。利用它，可以根据现有数据类型创建新的数据类型。但是，它的工作原理到底是什么？答案首先将我们引向继承机制，要应用原型化必须知道和了解这种机制。 继承的细节超出了本书的范围。尽管只为了使用 jQuery 并不需要完全理解原型化和新数据类型的创建，但是如果你对这一领域有所了解仍然是很有好处的。

A.1.5 数组

数据字段或者数组指的是可以通过名称访问的变量集合。如果想要保存一些同类信息，数组总是有着很大的优势。数组的特性在于，可以通过名称或者索引访问包含的元素。这通过在标识符后面的方括号中放入索引来实现。

程序清单 A.6 访问数组元素

```
x[0] = 1;
document.write(x[2]);
```

前面已经看到，在 JavaScript 中也可以通过文本索引创建数据字段。

声明

在经典方法中，JavaScript 中创建数组是通过 Array 类的一个实例完成的（换句话说，使用对应的构造方法）。

程序清单 A.7 创建一个数据字段

```
myDatafield = new Array();
```

注意，在 JavaScript 中通常不指定数组的大小，因为数组元素在必要时“即时”创建。但是，也可以使用数组字面量创建数组。

程序清单 A.8 用数组字面量创建数组

```
var x = [];
var primenumbers = [1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29];
```

还要注意一个事实——数组也被视为对象。数组的大部分属性和方法也可以应用到对象或者对象集合上。反之亦然，数组可以拥有任何对象的基本属性和方法。在 jQuery 中，这个事实被大量利用。

JavaScript 和数组通常可以和列表一样，被当作键 – 值对记录，这意味着用对象字面量进行的声明性对象创建也可以用于数组。

程序清单 A.9 用对象字面量进行声明性创建和直接初始化

```
var company = {
   designation: "RJS EDV-KnowHow",
   website: "www.rjs.de",
   contact: "ralph.steyer@rjs.de"
 };
```

访问数组元素

一般通过标识符和方括号中的索引访问数组元素。数组的最大好处之一是可以通过循环访问它们的元素。而且如前所述，因为数组也可以视为对象，因此也就实现了一些方法和属性。数组循环读取中最重要的属性是 length，该属性包含数组中元素的数量。这个信息可以用作数组循环中的条件（或者对象中元素的数量）。

提示

size() 方法提供了相同的信息，但是性能不及 length。

A.2 可用的 DOM 对象

在 DOM 的概念下，可使用表 A-1 列出的对象。注意，并不是所有浏览器都支持所有对象，而且，即使对于 DOM 的概念，支持的类型也各不相同。

表 A-1 重要的 DOM 对象

对象 / 类	描　述
all	这个类型的对象原则上可以访问网页的所有元素。这不是 DOM 官方标准的一部分，而是 Internet Explorer 的专有实现。该对象现在几乎完全被更新的对象 node 所替代，node 对象在现代浏览器中得到了更好的支持。应该避免使用这个对象
document	这个对象代表网页本身，网页的组件作为它的子对象。特别地，可以通过这个对象的方法写入网页并选择元素
event	由网页中的事件创建的一个对象，可以用于 JavaScript 中的（集中式）事件处理。但是，JavaScript 中的这种事件处理在不同的浏览器中实现不一致。在这方面，jQuery 等框架能够提供很多帮助

（续）

对象 / 类	描 述
element	Web 表单中的一个元素通过这种类型的对象表示
form	该对象包含对代表 HTML 页面中表单的对象的引用
frame	代表 window 下一个框架的对象
history	通过 history（window 的直接子对象），可以访问用户浏览器中以前加载的页面
image	这个对象包含对网页中一幅图像的引用
location	location 对象（window 的直接子对象）代表链接到一个 window 对象（换句话说，浏览器的地址栏）的完整 URL
navigator	通过 navigator，可以访问用户浏览器的信息
node	在 DOM 概念的新变种中，node 对象提供了对树形结构文档中单独元素的访问。这是使用现代 DHTML 的关键因素。通过 node 对象，可以访问网页的每个组件，但是不能直接写下 node。要使用 node 对象（一个节点），可以使用间接方法。例如，可以通过 getElementById() 和 getElementByName() 以及 getElementByTagName() 方法（或者通过对象字段即直接名称标记，这是一种古老的方法，在某些情况下有效）访问一个节点或者包含多个节点的数据字段。[⊖]但是，最终也可以通过 jQuery 函数和方法（如 $()、find() 或者 eq()）获得节点或者包含节点的整个数据字段
screen	通过 screen 对象，可以访问关于用户屏幕的各种平台相关信息
style	网页中一个对象的 CSS 属性的对象表现形式
window	这个对象包含关于整个浏览器窗口的状态信息。每个浏览器窗口使用自己的 window 对象。window 对象是直接影响浏览器的对象层次结构中最高的对象

⊖ 例如表单。