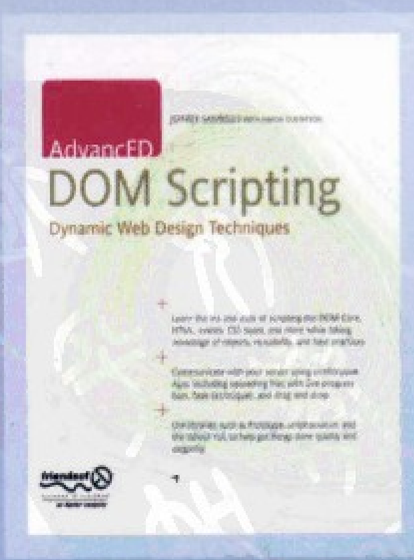
TURING 图灵程序设计丛书 Web开发系列

Apress®

# AdvancED DOM Scripting
## Dynamic Web Design Techniques

# JavaScript DOM 高级程序设计

[加] Jeffrey Sambells
[美] Aaron Gustafson 著
李松峰 李雅雯 等译




- 目前最深入的JavaScript力作之一
- 深入剖析Prototype、jQuery、YUI等JavaScript库的技术内幕
- 全景阐述JavaScript DOM程序开发的最佳实践

TURING 图灵程序设计丛书 Web开发系列

# AdvancED DOM Scripting
## Dynamic Web Design Techniques

# JavaScript DOM
# 高级程序设计

[加] Jeffrey Sambells
[美] Aaron Gustafson 著
李松峰 李雅雯 等译

人民邮电出版社
北 京

www.TopSage.com

图书在版编目（CIP）数据

JavaScript DOM 高级程序设计 /（加）桑贝斯（Sambells，J.），（美）古斯塔夫森（Gustafson，A.）著；李松峰等译．—北京：人民邮电出版社，2008.7
（图灵程序设计丛书）
书名原文：AdvancED DOM Scripting：Dynamic Web Design Techniques
ISBN 978-7-115-18109-1

Ⅰ. J… Ⅱ. ①桑…②古…③李… Ⅲ. JAVA 语言－程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2008）第067862号

## 内 容 提 要

本书注重理论与实践的结合，全面讲述高级的 DOM 脚本编程。全书分为 3 个部分：第一部分“深入理解 DOM 脚本编程”，涉及 W3C DOM 规范的各方面，包括非标准的浏览器支持和不支持的内容；第二部分“浏览器外部通信”，以 Ajax 和客户端-服务器端通信为主题；第三部分“部分高级脚本编程资源”，集中介绍了一批第三方脚本编程资源，包括库和 API。同时，每部分的最后一章都为案例研究，将学到的内容应用于实践。通过学习全书内容，读者将能构建起属于自己的 DOM 实用方法库。

本书适合有 Web 开发和设计经验的读者阅读和参考。

图灵程序设计丛书

JavaScript DOM高级程序设计

◆ 著　［加］Jeffrey Sambells　［美］Aaron Gustafson
译　李松峰　李雅雯等
责任编辑　杨　爽

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800×1000　1/16
印张：29.5
字数：710千字　2008年7月第1版
印数：1－4 000册　2008年7月北京第1次印刷

著作权合同登记号　图字：01-2008-2031号

ISBN 978-7-115-18109-1/TP

定价：59.00元

读者服务热线：(010)88593802　印装质量热线：(010)67129223
反盗版热线：(010)67171154

www.TopSage.com

# 计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

**Java** 一览无余：**Java** 视频教程 | **Java SE** | **Java EE**

**.Net** 技术精品资料下载汇总：**ASP.NET** 篇

**.Net** 技术精品资料下载汇总：**C#**语言篇

**.Net** 技术精品资料下载汇总：**VB.NET** 篇

撼世出击：**C/C++**编程语言学习资料尽收眼底 电子书**+**视频教程

**Visual C++(VC/MFC)**学习电子书及开发工具下载

**Perl/CGI** 脚本语言编程学习资源下载地址大全

**Python** 语言编程学习资料**(**电子书**+**视频教程**)**下载汇总

最新最全 **Ruby**、**Ruby on Rails** 精品电子书等学习资料下载

数据库精品学习资源汇总：**MySQL** 篇 | **SQL Server** 篇 | **Oracle** 篇

最强 **HTML/xHTML**、**CSS** 精品学习资料下载汇总

最新 **JavaScript**、**Ajax** 典藏级学习资料下载分类汇总

网络最强 **PHP** 开发工具**+**电子书**+**视频教程等资料下载汇总

**UML** 学习电子资下载汇总 软件设计与开发人员必备

经典 **LinuxCBT** 视频教程系列 **Linux** 快速学习视频教程一帖通

天罗地网：精品 **Linux** 学习资料大收集**(**电子书**+**视频教程**) Linux** 参考资源大系

**Linux** 系统管理员必备参考资料下载汇总

**Linux shell**、内核及系统编程精品资料下载汇总

**UNIX** 操作系统精品学习资料**<**电子书**+**视频**>**分类总汇

**FreeBSD/OpenBSD/NetBSD** 精品学习资源索引 含书籍**+**视频

**Solaris/OpenSolaris** 电子书、视频等精华资料下载索引


www.TopSage.com

# 译 者 序

读者的眼睛是雪亮的。为了让还没有成为这本书读者的朋友听听已经看过这本书的读者的意见，我一直在关注网上有关这本书的评论。

到中文版付梓时为止，Amazon上已经有4篇评论，总体上毁誉参半，两人给了5星，一个给出3星，另一个则给出1星。而业内几个知名的专业blog的评论中则普遍不乏溢美之辞。为什么会有这样大的差距呢？

Amazon上第一个发表评论的是该书的技术编辑Cameron Turner，评论题为Perfect in Every Way（一本十全十美的书），5星。他说："这是一本真正讲述构建下一代Web应用的书。到目前为止，这还是绝无仅有的。如果你需要为网站添加更多功能、灵活性和可访问性还有大势所趋的'耀眼的Web 2.0'特性，那么这本书是'必买'的。……需要提醒的是：如果你还是一个新手，可不要买这本书（因为它定位于'高级'这个层次上）。只有真正理解了CSS、JavaScript和HTML才能从本书中获益。相信本书将成为所有专业Web软件开发人员日常工作中时时查阅的必备图书。"

当然，Turner可以算是本书的参与者，有些偏爱在所难免。不过他指出了本书针对中高级读者，而不适合新手，这一点非常关键。实际上，Amazon上两位不满意的读者中，Richard（3星）就是因为看不太懂而发了牢骚。而另一位读者T.Dalmasso（1星）则对书中出现的错误非常恼火。幸运的是，他提到的错误在图灵公司给我的电子版文件里大部分已经修正。在翻译过程中，确实还发现了一些错误（主要是拼写和排版错误），但在我翻译过的书中已经算是比较少的了，远远没有多到令人生厌的程度，而且我基本上都已经解决了。

Amazon上最后发表评论的是资深.NET工程师David Betz "quantzai"（5星），他一上来就语出惊人："这是我见过的最好的一本现代JavaScript、DOM脚本编程和Ajax的书。这本书包括了JavaScript中从经常令人误解的变量作用域到与DOM深入交互等方方面面的内容。……本书是'研究生层次'的书，深入了Ajax的内幕，将使你成为专家，当然，也要求你一开始就具备思考力。"评论最后，David Betz "quantzai"对前面读者提出的拼写错误等问题给出了"反击"。他说："这又不是一本讲英语的书（出一两处错误在所难免），那些小错误根本无伤大雅；而且，即使没有这些错误你该理解不了，还是理解不了。"他甚至用"井蛙不可以语于海，夏虫不可以语于冰"，来表示对给出1星和3星的两位读者的不屑一顾。

在业内的知名blog圈里，瑞典哥德堡的资深Web工程师Roger Johansson这样说："我读了许多

www.TopSage.com

blog和书，想搞清楚作用域、闭包、面向对象等JavaScript概念，但是一直苦苦挣扎，读了本书后，我想问题终于解决了。”DOMAssistant库的作者Robert Nyman则评价：“如果你是一位中级JavaScript开发人员，还想更上一层楼，那么这将是使你梦想成真的绝妙好书。”Godbit项目的Nathan Smith也给予了很高的赞誉：“我要说，这是我读过的最好的JavaScript图书之一。”

看到这里，你有什么想法？没错。还记得小马过河的故事吗？无论是老黄牛，还是小松鼠，都有自己的角度和立场。因此，听别人的评论虽然能够大致了解一本书的内容，还是代替不了自己的判断。以我的经验，要购买一本自己感兴趣的、专注于某一技术领域的书，一是要听听网友的评论，二是仔细看一看书的目录，三是挑挑书的装帧，四是拿着书走向收银台（或者单击“放入购物车”按钮）。但是别忘了，对于一本外版书而言，译者也是中文版的第一读者。因此，听一听这位读者的看法也很重要（终于轮到我了，呵呵）。

作为译者，我来谈一谈自己对这本书的看法，供读者参考。

这本书在面向标准的Web编程领域是名副其实的扛鼎之作，也难怪它有些曲高和寡。全书的内容，都是作为一名专业的Web开发人员（或者真正的高手）所必须了解和掌握的高级知识，没一点多余的内容，洋洋550页中绝无浮华不实之辞。而且，书中对核心JavaScript原理的总结和概括（如常见陷阱、作用域链解析、闭包、面向对象等）、对最佳实践的倡导和践行（包括对面向未来的现代Web开发趋势的归纳和宣传，即脚本必须不唐突和增强而不是提供行为等）、对DOM规范讲解的提纲挈领（好像还没有哪本书这么详细地讲解过DOM）、对浏览器外部通信（Ajax）的反思与解决之道、对Web 2.0内容整合（Mashup）的分类与讲说等，无一不折射出这本书是作者博观约取、厚积薄发的心血力作。最后（最后说的往往最重要），如果你也醉心于Prototype、Base、jQuery、YUI、Ext、Mochikit、DOMAssitant、Script.aculo.us、Moo.fx等这些优秀的JavaScript库，不知道多少次被它们的魅力所倾倒，也想见微知著地真正理解这些库背后的工作原理，甚至于希望创建自己的库，那么这本书恰好适合你——一名JavaScript高手的需要，因为学习完这本书，你就会拥有自己跨平台的ADS库了（你必须Get your hands dirty——动手编写这个库的每一行代码），这还不够酷吗？应该说，本书是一本全景式的、沟通历史和未来的Web开发经典好书，是对现有JavaScript DOM程序开发最佳实践的一次大检阅和大放送，是推动Web标准化和向下一代Web开发挺进的里程碑式著作。而且，根据译者（就是我）的个人体会，这些话绝非溢美之辞，句句都言之有据，译者也愿意和读者就本书内容进行交流，互相学习。

然而，为什么在Amazon上本书不像*DOM Scripting*（中译本《JavaScript DOM编程艺术》，人民邮电出版社）那么广受关注，甚至大受欢迎呢——*DOM Scripting*有近60人评论，给1星的只有两人，3星三人，4星15人，其余均为5星？

比较了一下两书的内容，可以发现，*DOM Scripting*一书针对的是初学者，尤其是编程经验并不是很丰富的Web前端开发和设计人员，所以行文浅显，门槛比较低，而且学习曲线也非常平滑，对于这些读者，本书可以说是比较完美的。讲JavaScript非常好懂，而且字里行间渗透着现代的Web开发思想。这也是国内很多读者都嫌内容太浅的原因，他们往往都是已经有不少经验的程序员了。对于这些读者，本书才是他们真正想找的那本。我们有理由相信它会受到大家的欢迎。

这是我与图灵公司合作的第一本书，也是我最喜欢的一本书。在此，我要感谢刘江老师的热

www.TopSage.com

情邀请，感谢傅志红老师不厌其烦地回答我的问题和修改我的译稿，也感谢武卫东老师的悉心指导。同时，还要感谢本书的责任编辑杨爽，正是因为她创造性与我沟通，才使得本书在付梓前又消除了一些问题。不过，囿于个人水平和能力，翻译中的错误和不当之处在所难免。如果读者发现了书中的问题，请在我的个人网站http://www.cn-cuckoo.com中给予指出，或者将电子邮件发送到lsf.email@yahoo.com.cn。

本书由李松峰负责编译，参加翻译工作的还有李雅雯、程宝杰、宋会敏、闫建旺、左玉春、曹建辉、崔淑云、张井文、熊俊佳、左艳波、熊俊芹、刘英、赵淑云、贾爱华等。

译 者

2008年2月于北京

www.TopSage.com

# 前　言

DOM（Document Object Model，文档对象模型）脚本编程经常会被误解为Web上的某种脚本编程，实际上，纯粹的DOM脚本编程只包括W3C DOM规范中所涵盖的特性和方法。也就是说，不包括任何专有的浏览器特性。在理想的世界里，我们可以遵循标准，忽略专有特性，最终完成可以在任何设备中运行的脚本。但这个世界不是理想的世界——目前还不是。众所周知，并非所有设备或浏览器都合乎W3C标准，那像我们这样的程序设计人员要满足每个人的要求该怎么办呢，怎样才能继续严格遵守W3C DOM规范呢？

当试图回答这些问题并在保持真正的DOM符合性基础上处理多浏览器时，我们萌生了写这本书的想法。本书不仅对以上问题给出了答案，而且还涉及下列主题。

- 深入W3C DOM规范，并筛选出经常容易被误解的细节，同时仍然为非标准浏览器提供等价选项。
- 进一步探讨新方法，例如Ajax客户端-服务器端通信，冲破Ajax的局限性以提供更具交互性的体验。
- 体验一些主要的第三方资源，通过它们省掉一些平淡的日常工作。
- 理解并创建自己日常所用的DOM方法库。

这些新的能力也带来了许多诱惑。我们在进行DOM脚本编程时，往往因为热衷于一些华而不实的新特性，而背离了良好清晰的Web设计原则。因而，纵贯全书作者都会强调最佳实践的价值，提供很多强调可用性和可访问性的解决方案，这样对最终用户和你——开发者或设计者而言，都是有益的。

你可以把这本书放在计算机旁作为参考，也可以从头到尾读完它。无论采取哪种方式，只要你坚持学习完本书中的理论、代码、例子和案例研究，就会深刻地发现自己已经很好地理解了书中那些高级概念的含义，不仅知其然，而且更知其所以然。

## 本书读者对象

本书适合对DOM感兴趣并希望进一步提升自己的所有Web开发者和设计者。通过本书通俗易懂的讲解，读者能够轻松地理解高级的DOM编程概念。如果读者对DOM脚本编程和Web标准有一些基本的经验，那么通过学习本书收获会更大。

www.TopSage.com

## 本书组织方式

本书分三部分，通过学习全书内容，读者将能构建起属于自己的DOM实用方法库。书中的每一章都以前一章学习的概念为依托，因而本书的每一部分都是一个完整、独立的主题，而每一章则并非完全独立。

第一部分，“深入理解DOM脚本编程”，涉及W3C DOM规范的方方面面，包括非标准的浏览器支持和不支持的内容。从一开始就以最佳实践为榜样，然后你将了解到DOM2 HTML和DOM2核心规范，同时还有DOM2事件和DOM2样式规范。本部分中的每一章都会给出一些不针对特定浏览器的例子。而且，你也将着手构建自己的脚本程序库，并往其中添加各种方法去访问和操纵DOM、样式以及事件。这些方法将不针对特定的浏览器，因此你可以很容易地在公共方法（你将自己创建）的基础上建立自己的应用程序。第一部分最后的第6章将会完成一个案例研究，在这一章中，你将学会建立一个交互式裁剪和调整图像大小的工具。

在介绍了操纵和访问文档的各个方面知识之后，第二部分，“浏览器外部通信”，将以Ajax和客户端-服务器端通信为主题。在这一部分中，作者没有停留在介绍简单的做法上，而是深入解释了相应的内部工作机制，同时，也没有忘记介绍整合Ajax界面时可能遇到的麻烦。第二部分最后把这些技能用于实战检验，综合运用传统和当前的通信方法，创建了一个带有实时进度条的文件上传程序。

最后，在第三部分，“部分高级脚本编程资源”中，作者集中介绍了一批第三方脚本编程资源，包括库和API。你将在这一部分学习到如何利用主要的DOM脚本库来提高自己的开发效率，也包括使用一些视觉效果，为自己的Web应用程序添彩。同时，你还将学习如何通过可自由使用的API来整合交互式地图和项目管理工具。这些资源将为你提供高级编程能力，同时最大限度地减少你的重复性工作——但只有在对第一部分和第二部分内容深入理解的基础上，才能较好地体会到这些资源的价值。本书以Aaron Gustafson撰写的一个案例研究作为结尾，这个案例把`select`元素提高到了一个全新的水平。

作者没有提供附录，而是向读者公布了一个网站http://advanceddomscripting.com。在这个网站中，读者可以下载到本书的源代码及额外一些例子和参考文献。作者将在这个网站中发布与DOM脚本编程相关的最新的重要消息，读者可以经常访问这个网站，以便与时俱进。

## 本书约定

为保证本书内容清晰，容易理解，全书将遵守如下约定：

(1) 代码以等宽字体表示。

(2) 新加的或修改过的代码通常会以加粗的等宽字体表示。

(3) 伪代码和变量输入以斜体等宽字体表示。

(4) 对于需要提醒读者注意的内容，会像下面这样突出排版：

> 嗯，别说我没提醒过你！

www.TopSage.com

(5) 有时候某一行代码可能会超出书本的宽度，这时将使用像下面这样的箭头图标➡：

```
这一部分的代码非常、非常、非常地长，应该全都写在不带有换行符的➡
同一行中。
```

(6) 为了节省版面，许多例子代码中都会包含"……省略的代码……"这样的文本行。这行文本表示相应位置处省略了一部分代码。这样做可以使当前例子中的主要代码更突出。在有些情形下，也会存在代码前后都有"……省略的代码……"字样的情况，这也是为了更好地在上下文中显示代码。例如，下面的代码：

```
(function(){
window['ADS'] = {};

……省略的代码……

function helloWorld(message) {
    alert(message);
}
……省略的代码……

})();
```

表示函数`helloWorld()`处于以`(function(){`开始和以`})();`结尾的代码块中。如果代码在所在文件中的位置也很重要，同样会加以标识。

(7) 在要求读者按照作者的指示构建例子的情况下，会以一个带注释的结构开头，比如：

```
// 定义一个变量
// 通过helloWorld()函数给出警告
```

这样，读者就可以在每条注释语句后面添加一些代码，以便构成例子：

```
// 定义一个变量
var example = 'DOM Scripting is great!';

// 通过helloWorld()函数给出警告
helloWorld(example);
```

这种方式可以使读者轻松地编写完成每个例子，并有助于理解每一步的意图。

(8) 在下载的本书源代码中，读者可能也会注意到书中没有出现的一些注释。书中删除了这部分注释，保证印刷出来的代码不致于太冗长。

## 阅读本书的条件

创造力、兴趣和学习的渴望就是阅读本书的全部条件，熟悉一些DOM、JavaScript、Web标准和CSS的基础知识当然也是有益无害的。只要有可能，作者会尝试着讲解一些高级主题，而且尽力以一名Web初学者都能够理解的方式来进行。

本书中的例子代码已经在下列浏览器中测试通过了。在书中，作者还将针对每种浏览器给出相应的提示：

- Firefox 1.5+

www.TopSage.com

- Microsoft IE 6+
- Safari 2+
- Opera 9+

在其他兼容标准的浏览器中，同样可以运行本书的例子。但比较早的浏览器只能降级使用不含DOM脚本的版本。

## 如何下载代码

如果担心输入本书的全部代码会把手指头累得抽筋，作者建议你访问friends of ED的网站（http://friendsofed.com）并从中下载本书的源代码。同时在这个网站上，你还可以找到与Web标准、DOM、CSS以及Flash相关的其他好书。

如果你觉得不满足，那么还可以访问本书的站点（http://advanceddomscripting.com），其中除了源代码之外，还会提供与DOM脚本编程相关的更多信息。

## 如何联系作者

读者如果有任何问题、意见和想法，可以随时与作者联系，电子邮件地址是 jeff@advanceddomscripting.com。同样，读者也可以经常浏览一下本书的网站 http://advanceddomscripting.com，作者将在此发表一些更新的内容和勘误信息，以及其他对读者有帮助的第一手资料。

Aaron Gustafson 的电子邮件地址是ads-book@easy-designs.net。

## 致谢

这几年来，在我的人生旅途中，有许多人都在影响着我——家人、朋友、相识的和不相识的，正是所有这些人的帮助才使得本书得以出版。要提及每个人恐怕不易，所以这里就一并致谢了。

感谢Chris Mills给我这次机会，让我能够深入到自己热爱的主题中。如果没有他的指导，我根本无法理清头脑中那些混乱的想法。而且，也要感谢friends of ED和Apress出版社的团队成员，包括Kylie Johnston、Heather Lang、Laura Cheu以及隐身于幕后的所有人，正是来自你们的反馈才使得这本书更加完善，从而让我有了一番不同的感受。

感谢Aaron Gustafson撰写了一个极好的案例。学习完这个案例一定会让读者因为超越了基本知识而感到惬意。

感谢Cameron Turner和Victor Sumner，他们测试了我编写的代码。相信他们测试出的或没测试出的问题一定不少，但读者的评论和鼓励一定会让我更加努力。

感谢那些慷慨地共享他们知识和思想的每一个人。如果没有他们深刻的评论、博客、图书、谈话和讨论，不可能有DOM脚本编程、Web标准的进步和创新。谢谢大家。

特别感谢我的妻子Stephanie和她给我的爱。感谢她在照顾我们刚出生的孩子和我时所表现出的耐心及给予我的极大的理解。没有她我将一事无成。

最后，感谢亲爱的读者花时间看这本书。我衷心地祝愿你理解我所讲的一切内容。

Jeffrey Sambells

www.TopSage.com

有这么多人我得感谢，不过我只是撰稿人之一，所以还是简略点儿好。感谢Brothercake，他的表达天分着实给我的书稿增色不少。感谢Jeremy Keith，他让我说得更头头是道。感谢Shaun Inman，This is Cereal的设计真让人深受启发啊。感谢Jeffrey Zeldman，他坚定了我当一名好作者的信心。感谢Molly Holzschlag，我取得的成功皆应归功于她，在此，不管怎么表达谢意都是不够的（但我还是要表达我的谢意）。最后（也是最重要的），感谢Kelly，有多少个深夜和周末，我都在忙于编写程序，再写书分析程序，她都默默忍受了。我要对她说：我爱你。

Aaron Gustafson

www.TopSage.com

# 目 录


第一部分　深入理解 DOM 脚本编程

第 1 章　遵循最佳实践 ······ 2

1.1 不唐突和渐进增强 ······ 2
1.2 让 JavaScript 运行起来 ······ 4
1.2.1 把行为从结构中分离出来 ······ 4
1.2.2 不要版本检测 ······ 11
1.2.3 通过平稳退化保证可访问性 ······ 13
1.2.4 为重用命名空间而进行规划 ······ 14
1.2.5 通过可重用的对象把事情简化 ··· 17
1.2.6 一定要自己动手写代码 ······ 26
1.3 JavaScript 语法中常见的陷阱 ······ 27
1.3.1 区分大小写 ······ 27
1.3.2 单引号与双引号 ······ 27
1.3.3 换行 ······ 28
1.3.4 可选的分号和花括号 ······ 28
1.3.5 重载（并非真正的重载）······ 29
1.3.6 匿名函数 ······ 30
1.3.7 作用域解析和闭包 ······ 30
1.3.8 迭代对象 ······ 35
1.3.9 函数的调用和引用（不带括号）······ 36
1.4 实例：WYSIWYG JavaScript 翻转图 ····· 36
1.5 小结 ······ 43

第 2 章　创建可重用的对象 ······ 44

2.1 对象中包含什么 ······ 44
2.1.1 继承 ······ 45
2.1.2 理解对象成员 ······ 46
2.1.3 window 对象中的一切 ······ 48
2.1.4 理解作用域和闭包是根本 ······ 51
2.2 创建你自己的对象 ······ 52
2.2.1 一变多：创建构造函数 ······ 53
2.2.2 添加静态方法 ······ 54
2.2.3 向原型中添加公有方法 ······ 55
2.2.4 公有、私有、特权和静态成员真那么重要吗 ······ 58
2.2.5 对象字面量 ······ 59
2.3 this 是什么 ······ 61
2.4 try{ }、catch{ }和异常处理 ······ 66
2.5 实例：你自己的调试日志 ······ 67
2.5.1 为什么需要 JavaScript 日志对象 ······ 68
2.5.2 myLogger()对象 ······ 68
2.6 小结 ······ 76

第 3 章　DOM2 核心和 DOM2 HTML ······ 77

3.1 DOM 不是 JavaScript，它是文档 ······ 77
3.2 DOM 的级别 ······ 78
3.2.1 DOM 0 级 ······ 78
3.2.2 DOM 1 级 ······ 78
3.2.3 DOM 2 级 ······ 79
3.2.4 DOM 3 级 ······ 79
3.2.5 哪个级别适合你 ······ 81
3.3 创建示例文档 ······ 82
3.3.1 创建 DOM 文件 ······ 83
3.3.2 选择一个浏览器 ······ 84
3.4 DOM 核心 ······ 86
3.4.1 继承在 DOM 中的重要性 ······ 88
3.4.2 核心 Node 对象 ······ 89
3.4.3 核心 Element 对象 ······ 102
3.4.4 核心 Document 对象 ······ 104
3.4.5 遍历和迭代 DOM 树 ······ 106
3.5 DOM HTML ······ 108


www.TopSage.com

3.5.1 DOM2 HTML 的 HTMLDocument 对象……108
3.5.2 DOM2 HTML 的 HTMLElement 对象……109
3.6 实例：将手工 HTML 代码转换为 DOM 代码……110
3.6.1 DOM 生成工具的 HTML 文件……111
3.6.2 使用示例 HTML 片段进行测试……112
3.6.3 扩充 ADS 库……113
3.6.4 generateDOM 对象的框架……115
3.7 小结……127

第 4 章 响应用户操作和事件……128

4.1 DOM2 级事件……129
4.2 事件的类型……130
4.2.1 对象事件……130
4.2.2 鼠标移动事件……132
4.2.3 鼠标单击事件……134
4.2.4 键盘事件……136
4.2.5 表单相关的事件……136
4.2.6 针对 W3C DOM 的事件……142
4.2.7 自定义事件……143
4.3 控制事件流和注册事件侦听器……143
4.3.1 事件流……143
4.3.2 注册事件……151
4.3.3 在事件侦听器中访问事件对象……159
4.3.4 跨浏览器的事件属性和方法……160
4.4 小结……170

第 5 章 动态修改样式和层叠样式表……171

5.1 W3C DOM2 样式规范……171
5.1.1 CSSStyleSheet 对象……171
5.1.2 CSSStyleRule 对象……172
5.1.3 CSSStyleDeclaration 对象……173
5.1.4 支持的匮乏……173
5.2 当 DOM 脚本遇到样式……173
5.3 把样式置于 DOM 脚本之外……179
5.3.1 style 属性……179
5.3.2 基于 className 切换样式……182
5.3.3 切换样式表……185
5.3.4 修改 CSS 规则……192
5.4 访问计算样式……200
5.5 Microsoft 的 filter 属性……201
5.6 实例：简单的渐变效果……204
5.7 小结……207

第 6 章 案例研究：图像裁剪和缩放工具……208

6.1 测试文件……208
6.2 imageEditor 对象……212
6.2.1 调用 imageEditor 工具……216
6.2.2 imageEditor 载入事件……217
6.2.3 创建编辑器标记和对象……218
6.2.4 向 imageEditor 对象添加事件侦听器……224
6.2.5 缩放图像……227
6.2.6 裁剪图像……230
6.2.7 未完成的图像编辑器……234
6.3 小结……234

第二部分 浏览器外部通信

第 7 章 向应用程序中加入 Ajax……236

7.1 组合的技术……236
7.1.1 语义化 XHTML 和 DOM……237
7.1.2 JavaScript 和 XMLHttpRequest 对象……237
7.1.3 XML……244
7.1.4 一个可重用的对象……248
7.1.5 Ajax 是正确的选择吗……253
7.2 为什么 Ajax 会破坏网站及如何解决……253
7.2.1 依赖 JavaScript 生成内容……253
7.2.2 通过<script>标签绕过跨站点限制……254
7.2.3 后退按钮和书签功能……260
7.2.4 完成请求的赛跑……270
7.2.5 增加资源占用……278
7.2.6 问题解决了吗……278

www.TopSage.com

7.3　实例：Ajax 增强的相册 …… 278
7.4　小结 …… 285

**第 8 章　案例研究：实现带进度条的异步文件上传功能 …… 286**

8.1　信息载入时的小生命 …… 288
8.2　起点 …… 291
8.3　完成整合：上传进度指示器 …… 292
8.3.1　addProgressBar()对象的结构 …… 294
8.3.2　载入事件 …… 296
8.3.3　addProgressBar()对象 …… 296
8.4　小结 …… 308

## 第三部分　部分高级脚本编程资源

**第 9 章　通过库来提高生产力 …… 310**

9.1　选择合适的库 …… 311
9.2　增强 DOM 操作能力 …… 314
9.2.1　连缀语法 …… 314
9.2.2　通过回调函数进行过滤 …… 321
9.2.3　操纵 DOM 文档 …… 322
9.3　处理事件 …… 324
9.3.1　注册事件 …… 325
9.3.2　自定义事件 …… 327
9.4　访问和操纵样式 …… 329
9.5　通信 …… 329
9.6　小结 …… 334

**第 10 章　添加效果增强用户体验 …… 335**

10.1　自己动手实现效果 …… 335
10.1.1　让我看到内容 …… 336
10.1.2　提供反馈 …… 340
10.2　几个视觉效果库简介 …… 342
10.3　视觉盛宴 …… 343
10.3.1　MOO 式的 CSS 属性修改 …… 344
10.3.2　通过 Script.aculo.us 实现视觉效果 …… 353
10.3.3　通过 Moo.fx 实现逼真的运动效果 …… 356
10.3.4　圆角效果 …… 360
10.3.5　其他库 …… 362
10.4　行为增强 …… 362
10.5　小结 …… 374

**第 11 章　丰富的 Mashup!运用 API 添加地图、搜索及更多功能 …… 375**

11.1　API 密钥 …… 376
11.2　客户端 API：离不开 JavaScript …… 377
11.2.1　地图中的 Mashup 应用 …… 377
11.2.2　Ajax 搜索请求 …… 388
11.2.3　地图与搜索的 Mashup 应用 …… 397
11.3　服务器端 API：需要代理脚本 …… 400
11.3.1　通过 Basecamp 构建集成的 To-Do 列表 …… 403
11.3.2　通过 Flickr 取得个性头像 …… 412
11.4　小结 …… 416

**第 12 章　案例研究：用 DOM 设计选择列表 …… 417**

12.1　经典的感觉 …… 417
12.2　构建更好的选择列表 …… 418
12.3　策略？我们不需要臭哄哄的策略 …… 420
12.3.1　相关的文件 …… 420
12.3.2　FauxSelect 对象 …… 421
12.3.3　开始创建人造 select 元素 …… 423
12.3.4　查找 select 元素 …… 425
12.3.5　构建 DOM 元素 …… 427
12.4　添加事件——为人造 select 赋予生命 …… 431
12.5　让表单绽放光彩 …… 435
12.6　行为修正 …… 445
12.6.1　z-index 来救急 …… 447
12.6.2　键盘控制及其他细节 …… 449
12.6.3　select 太大了吗 …… 454
12.7　最后的细节 …… 455
12.8　继续替换 select 的冒险 …… 456
12.9　小结 …… 457

www.TopSage.com

# 计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 **|** 软考历年真是解析与答案

软考视频 **|** 考试机构 **|** 考试时间安排

**Java** 一览无余：**Java** 视频教程 **| Java SE | Java EE**

**.Net** 技术精品资料下载汇总：**ASP.NET** 篇

**.Net** 技术精品资料下载汇总：**C#**语言篇

**.Net** 技术精品资料下载汇总：**VB.NET** 篇

撼世出击：**C/C++**编程语言学习资料尽收眼底 电子书**+**视频教程

**Visual C++(VC/MFC)**学习电子书及开发工具下载

**Perl/CGI** 脚本语言编程学习资源下载地址大全

**Python** 语言编程学习资料**(**电子书**+**视频教程**)**下载汇总

最新最全 **Ruby**、**Ruby on Rails** 精品电子书等学习资料下载

数据库精品学习资源汇总：**MySQL** 篇 **| SQL Server** 篇 **| Oracle** 篇

最强 **HTML/xHTML**、**CSS** 精品学习资料下载汇总

最新 **JavaScript**、**Ajax** 典藏级学习资料下载分类汇总

网络最强 **PHP** 开发工具**+**电子书**+**视频教程等资料下载汇总

**UML** 学习电子资下载汇总 软件设计与开发人员必备

经典 **LinuxCBT** 视频教程系列 **Linux** 快速学习视频教程一帖通

天罗地网：精品 **Linux** 学习资料大收集**(**电子书**+**视频教程**) Linux** 参考资源大系

**Linux** 系统管理员必备参考资料下载汇总

**Linux shell**、内核及系统编程精品资料下载汇总

**UNIX** 操作系统精品学习资料**<**电子书**+**视频**>**分类总汇

**FreeBSD/OpenBSD/NetBSD** 精品学习资源索引 含书籍**+**视频

**Solaris/OpenSolaris** 电子书、视频等精华资料下载索引

# Part 1

第一部分

# 深入理解 DOM 脚本编程

本部分内容


- 第 1 章　遵循最佳实践
- 第 2 章　创建可重用的对象
- 第 3 章　DOM2 核心和 DOM2 HTML
- 第 4 章　响应用户操作和事件
- 第 5 章　动态修改样式和层叠样式表
- 第 6 章　案例研究：图像裁剪和缩放工具


www.TopSage.com

第 1 章

# 遵循最佳实践

你很兴奋，你的客户也很兴奋。你刚刚为客户安装启用了新版的网站，一切都很顺利。网站很花哨，它历经了数小时汗水和泪水的浇灌，每一处设计细节——扩展式菜单、交互的Ajax都精心调试过，所有新式花样无所不包。它看上去很不错，运行得也很完美，所有人都沉浸在喜悦中。然而，一周之后，噩梦开始了。客户慌里慌张地打来电话，好像是据他们的部分顾客来电话反应说主页上的链接打不开了，而另外一些顾客在填写反馈表单时也碰到了问题。但在你和你的客户那里却不存在这些的问题。还有一些人也打来电话，抱怨下载网站的每一个页面都要等很长时间，即使看上去网页内容并不很多，而你从来没有发现下载时间有问题。更糟糕的是，随后，你的客户发现网站在搜索引擎中的排名一落千丈。看来事情没有想象的那么乐观，但问题出在哪里呢？下面我们就一起来查找原因。

最佳实践是人们做事时应该遵循的、被公认和经过验证的模式。虽然不一定是唯一的，甚至不是最佳的方式，但这些方式是大多数人认同的做事方式。一般的书在最后部分会提到一些最佳实践，这更多地是一种提示，即在你已学会了每件事并按自己的方式行事时，告诉你还会有一种最适当的方式。我之所以把最佳实践放在前面来讲，就是为了让你在开始学习新知识之前，先明确正确的方向。如果有阳关大道，那又何必去走独木桥呢？

## 1.1 不唐突和渐进增强

XHTML（Extensible HyperText Markup Language，可扩展超文本标记语言）、CSS（Cascading Style Sheet，层叠样式表）和使用JavaScript的DOM（Document Object Model，文档对象模型）脚本是Web设计的三个主要部分。其中，XHTML用于提供文档结构的语义标记，CSS为文档布局提供定位和样式，而DOM脚本编程用于增强文档的行为和交互性。发现了吗？我刚才说DOM脚本“增强”，而不是为文档“提供”行为和交互性。“增强”和“提供”之间的差异暗示了一个重要区别。我们都学过XHTML语义，知道验证文档是否符合W3C规范，而且也都在用CSS来为严格型XHTML文档标记应用适当的样式（对吧？）。但是，作为第三个主要部分的DOM脚本，虽然它可以把事情做得格外漂亮，为我们的Web应用程序增光添彩，却有可能是一个唐突的家伙。

DOM脚本编程依赖于JavaScript。如果你在Web上搜索“Unobtrusive JavaScript”这两个关键词，将会发现网上泛滥着不同的描述，这只不过是因为你无处不需要考虑不唐突性（unobtr-

www.TopSage.com

usiveness)。不过,这并不意味着不唐突性是"创可贴(Band-Aid)",随便贴到你的代码上就万事大吉了。JavaScript中没有不唐突的对象(嗯……或许有?),而且也不可能下载到"不唐突"的库来解决代码中的问题。不唐突性只能来自于对Web应用程序正确的规划和准备。当需要用户和Web开发者彼此合作时就必须考虑不唐突性。你的脚本对用户来说必须是不唐突的,即消除不必要的行为和令人讨厌的功能(过去曾令JavaScript背负坏名声的一些东西)。脚本也必须是不唐突的,即在没有JavaScript的情况下,能使页面和标记持续有效,虽然不再那么优雅了。最后,不唐突的脚本必须在标记中容易实现,也就是通过ID和类属性来关联行为,进而保证脚本与标记的分离。要想更好地理解不唐突性,以及DOM脚本编程、XHTML和CSS整合的本质,需要考虑我们在前面情形中所看到的结果,并理解基本原理及其在代码中的应用。

我们经常会听到与不唐突性有关的两个术语——"渐进增强(progressive enhancement)"和"平稳退化(graceful degradation)"。某些技术能够实现,当浏览器支持相应功能时文档会得到增强(渐进增强),而当浏览器不支持相应功能时,文档被退化(平稳退化)。通过使用这些技术,不支持相应功能的浏览器也会获得同一文档的相同信息量但却不同的视图。这两个术语经常交替使用,但二者都承认现实:并非所有浏览器都遵循相同的标准创建,而且不能对所有浏览器一视同仁。同理,谁也不能为了迎合少数人而强迫所有人都接受一种低质量的服务。

Yahoo的Nate Koechley关于渐进增强必要性的论述,我特别赞同。下面是他在介绍浏览器支持的时候总结出来的相关内容(http://developer.yahoo.com/yui/articles/gbs/gbs.html):

"支持并不意味着每个人都会得到完全一样的结果。期望两个使用不同浏览器的用户拥有相同的体验就是不认同Web的异构本性(heterogeneous essence)。事实上,要求所有用户都具有相同的体验会对参与性造成障碍。应该把内容的有效性和可访问性作为首要目标。"

如果你在使用JavaScript时妨碍了"内容的有效性和可访问性",那么你的做法就是错误的。

同时,由于渐进增强并不是必需的,因而也就意味着要提供一种全有或全无JavaScript的方案。问题在于,JavaScript与那些在专用服务器或计算机中运行的编程及脚本语言不同,它是一种运行在Web浏览器中的解释型语言。因此,对于脚本需要处理的各种各样的软硬件和操作系统的组合我们无法控制。为了应对几乎无限种这样的组合,必须小心谨慎地基于浏览器的能力和可用技术来创建行为性的增强。这意味着要么提供一个仍然较少依赖于JavaScript脚本的平稳退化方案,要么提供一个借助于传统、没有JavaScript方法的方案。

虽然在本书中会强调不唐突性、平稳退化和渐进增强这些观点,但我也要声明自己并不是一个狂热的信徒。我知道你的网站确实需要支持具有与标准不兼容的特性的浏览器。在很多情况下,一些专有的方法仍然是必需的。这其中的关键是,要走与标准兼容的道路,只在必要时使用专有方法。

当使用DOM脚本编程并将它们整合到网站中时,你编写的脚本必须:

- **与标准兼容**。面向未来开发应用程序,确保Web应用程序能够在更新更好的浏览器中继续运行。
- **容易维护**。综合运用可重用和容易理解的方法,以便你和其他人能够集中关注业务逻辑,而不是反复重写代码。

www.TopSage.com

- **具有可访问性**。确保每个人都能简捷而有效地访问到你的信息，即使他们无法运行脚本或者禁用了JavaScript。
- **具有可用性**。那些在一种情况下非常有效但很难实现或者重用的脚本，在第二次或第三次使用时不会有太大的价值。可用性不仅适用于与最终用户的交互，也适用于与开发者的交互。

这些指导原则在实践中相互重叠、相互结合，牢记它们有利于减少困惑，长久受益。事实上，我建议你把这几条原则用大号的粗体字打印出来摆在计算机旁，以便让它们植根于你的头脑中。为此，我还制作了一份适合打印的PDF文件，放在了http://advanceddomscripting.com/remember.pdf上。请记住，如果XHTML和CSS的目标是要成为普适的标准，那么你的JavaScript脚本也应如此！

在本章接下来的内容中，我们将介绍一些有助于实现这些目标的方法，这些方法会让你的客户、同事，包括你自己都皆大欢喜。同时，你也将开始着手构建自己的常用方法库，书中会一直用到，你还将了解一些常见的JavaScript陷阱（gotchas）。

## 1.2 让JavaScript运行起来

下面你打算从哪里开始？要怎样才能让自己的艰苦工作符合以上原则呢？

如果以本章开始的那个假想的不成功的网站为例，你可能会发现像下面这样一些常见的缺陷：

- `<script>`标签出现在了文档主体（body）中。
- 依赖于浏览器嗅探[①]，而不是通过能力检测来测试JavaScript功能的兼容性。
- 在锚元素的`href`属性中使用了硬编码的`javascript:`前缀。
- 多余的、重复的、高度定制的JavaScript代码。

你可能想知道这些有什么问题。毕竟，我们都时不时这么干过。真正成问题的是这些方法的有效性和它们无法适应所有的情况。如果你回过头来再用批判的眼光看一看它们，就会发现要用正确的方法完成同样的任务并不需要费太多的力气，并且还会获得其他的好处。为了帮你避免这些问题和诸如此类的错误，下面我们就来分析一下这些问题背后的总体思路以及你在实现下一个网站时应该记住的东西。

### 1.2.1 把行为从结构中分离出来

对于任何涉及JavaScript的项目来说，要改进它们的头一件事就是把行为从标记中分离出来。这个关键性技术常常被人忽略，或者没有得到足够的重视。遵循用外部样式表将CSS从XHTML文档中分离出来的相同设计规则，JavaScript也应该将自己分离到外部文件中。对于CSS，你可以通过在标签上使用`style`属性把CSS应用到DOM对象，但这样做并不比到处乱用过气的`<font>`标签更容易维护。为什么会看到，一个漂亮的网站通过`id`和`class`选择符把XHTML代码标记了出来，所有CSS都被恰当地分离到一个外部文件中，但是，却把一团糟的嵌入式JavaScript代码丢

① 嗅探在这里专指对浏览器品牌和版本的检测。——译者注

www.TopSage.com

弃在文档中？现在这种做法必须要停止了！JavaScript应该遵循与CSS相同的分离规则。

### 1. 如何以正确的方式包含JavaScript

可以通过几种不同的方式把JavaScript代码添加到Web应用程序中。你可能见到过或者用过所有这些方法，但这些方法并不都是可取的。我不想从一开始就误导你，所以在犹豫要不要向你展示这些方法。然而，只有如此我们才能同步前进。下面我会分别列举每一种方法并指出一些问题。

首先，可以把嵌入式脚本与其他标记混合在一起添加到body标签中：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Inline JavaScript</title>
</head>
<body>
    <h1>Inline Example</h1>
    <script type="text/javascript">
        //JavaScript代码
    </script>
</body>
</html>
```

但是，这样在你的行为增强代码和结构化标记之间无法实现任何程度的分离。这种方式不仅需要反复地展开代码，而且也经常会导致不必要的代码复制。

> 聪明的读者可能会注意到，我没有为了对非常老的浏览器或XML解析程序隐藏JavaScript代码，而在script标签中添加任何典型的注释标签，如：`<!--`和`//-->`或者`<!--//--><![CDATA[//><!--`和`//--><!]]>`。如果在这个文档的<body>之间使用嵌入的JavaScript，那么你就可能陷入到注释和不注释的争论中。要不然，你也可以采纳我的建议而完全忽略嵌入的脚本。因此，我不会为了解释这些注释标签而把内容搞复杂。

嵌入式JavaScript唯一有用的时候，就是使用`document.write()`方法在相应的位置直接修改页面内容。但是，要修改页面内容还有更好的方案，这一点将在本书后面介绍。

其次，可以把嵌入式脚本添加到文档的<head>元素中：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Head JavaScript</title>
    <script type="text/javascript">
        //JavaScript代码
    </script>
</head>
<body>
    <h1>Head Example</h1>
</body>
</html>
```

毋庸置疑，这样更好一些。至少这样能够把所有代码都汇集到标记中适当的位置上。但是，

www.TopSage.com

这种方法仍然混合了结构和行为，只不过混合的程度轻了一些。

最后，可以通过外部源文件来包含JavaScript脚本：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>External File JavaScript</title>
    <script type="text/javascript" src="source.js"></script>
</head>
<body>
    <h1>External File Example</h1>
</body>
</html>
```

这种方式始终是正确的，也是最好的。作为一条通用的规则，多数人会说任何不直接向文档中写入内容的脚本（使用`document.write()`方法）都应该放在标记代码的<head>元素中。虽然这也没错，但我认为还要考虑得更深远一些。无论什么时候，没错，就是在任何情况下，都要把全部脚本包含在一个外部源文件中。就当你从来没有见过前两种方法，事实上，最好假定它们不存在。如果你的标记中有任何JavaScript代码，都应该转回头来把它们移走，以保证行为和结构完全分离（不存在无法将全部脚本放到源文件中的情况），而且这样做还会带来诸多好处。你会因此强迫自己重新考虑如何着手创建函数和对象：是创建可重用的、适应性强的代码来保持程序简单和可维护呢？还是创建相同逻辑的定制副本把事情搞复杂呢？外部文件提供的另一个好处就是减少整个页面的大小。它们通常会被客户的Web浏览器缓存起来，并且只下载一次，从而减少了每个后续页面的加载时间。

你需要跨越的唯一真正差别，就是你所钟爱的`document.write()`方法可能不会再取得预期的结果。然而，因为这是一本有关DOM脚本编程的书，我们将会使用常规的和不针对特定浏览器（browser-agnostic）的DOM方法，你最终将会发现使用`createElement()`和`appendChild()`这样的DOM方法，或者非官方的`innerHTML`属性其实更加游刃有余——不过，这些我们会在后面再讨论。

**2. 昔日的`javascript:`前缀**

你可能注意到我还遗漏了其他几个文档中可能会出现JavaScript的地方，特别是在元素的属性中。考虑一下打开新弹出窗口的情况，如果你在使用严格型DOCTYPE规范，那么锚标签中的`target`属性是无效的：

```
<a href="http://advanceddomscripting.com" target="_blank"> ➥
AdvancED DOM Scripting</a>
```

因而打开另一个窗口的唯一有效方式就是用JavaScript。在这种情况下，你可能（至少偶尔）会在<a>标签的`href`属性中使用特殊的`javascript:`前缀：

```
<a href="javascript:window.open('http://advanceddomscripting.com');"> ➥
AdvancED DOM Scripting</a>
```

但在这种情况下，你不会得到想要的结果。

www.TopSage.com

> 要测试这些嵌入脚本的例子，请打开包含在本书源代码中的例子页面chapter1/popup/examples.html，或者访问本书的网站http://advanceddomscripting.com。

javascript:前缀的一个问题是，它只能处理一个函数，而不能处理多个[①]。而且，如果函数有返回值，原先的页面也会被返回的结果覆盖。例如，在Firefox中单击前面带有javascript:window.open(...)的锚，会打开想要的新窗口，但原先的窗口却会被window.open()方法返回的结果（[object Window]）所覆盖，如图1-1所示。

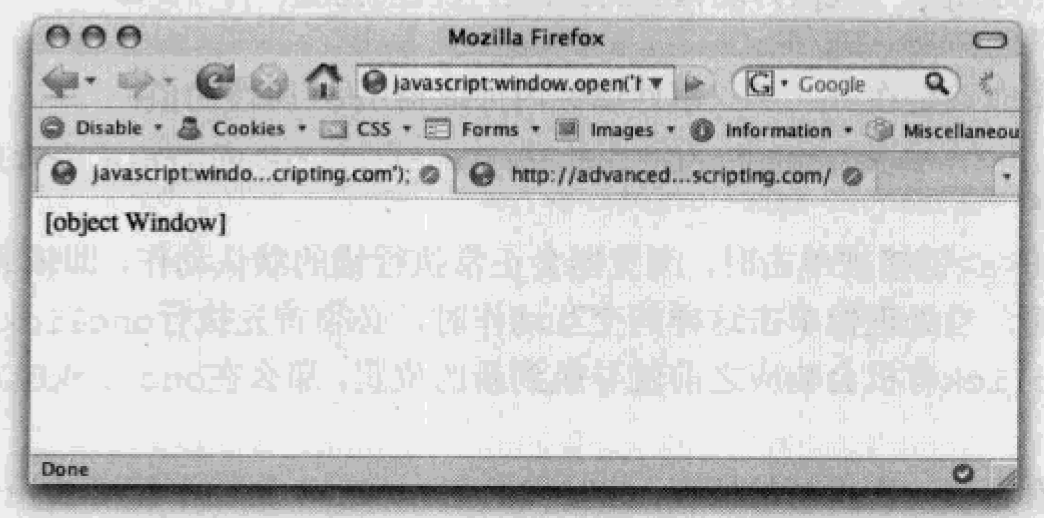



图1-1 在锚元素的href属性中使用javascript:前缀和返回值的方法时，返回的值会覆盖原先的页面

不过，这个问题可以通过如下方法来解决。在脚本中包含一个额外的函数：

```
function popup(url) {
    window.open(url);
    //不返回任何值！
}
```

然后引用这个函数而不直接引用window.open:

```
<a href="javascript:popup('http://advanceddomscripting.com');"> ➥
AdvancED DOM Scripting</a>
```

但这仍然使用了嵌入的javascript:前缀。为了避免对包装函数的依赖，你可能尝试过另一种不太令人讨厌的方式，即使用onclick事件属性将JavaScript代码附加到元素上以直接打开URL，同时在href属性中简单地放上一个#:

```
<a href="#"onclick="window.open('http://advanceddomscripting.com');"> ➥
AdvancED DOM Scripting</a>
```

虽然使用onclick事件属性的情况要好一点，但是这两种直接在文档中编码的方法，既没有提供任何行为与结构的分离，也不是不唐突的：在href属性中使用特殊的javascript:前缀是

---

① 此处作者说法可能有误，比如javascript:window.open('http://advanceddomscripting.com'); alert('hello,world!');是可以运行的。但毕竟，这种方法是不为本书所推荐的，所以读者大可假设这种方法也不存在。——译者注

www.TopSage.com

有问题的，因为它不是官方ECMAScript标准（http://www.ecma-international.org/publications/standards/Ecma-262.htm）的一部分，但它却通常被所有支持ECMAScript的浏览器所实现。而且，使用这个前缀或者前面的onclick属性，造成了在导航或激活锚时对JavaScript的直接依赖。

由于两种方式都不理想，所以最好的方案至少是通过嵌入的事件属性来修改或使用已有的属性，如href中的URL。这样，前面两个锚的例子可以被重写为修改已有的href属性值。这样当JavaScript被禁用时，锚元素仍然能够按照预期完成任务：

```
<a href="http://advanceddomscripting.com" onclick="this.href= ➥
'javascript:popup(\'http://advanceddomscripting.com\');'"> ➥
AdvancED DOM Scripting</a>
```

但更好的方案应该是在onclick事件属性内部取得href属性的值：

```
<a href="http://advanceddomscripting.com" onclick="popup(this.href); ➥
return false;">AdvancED DOM Scripting</a>
```

在一个常规的<a>链接被单击时，浏览器会正常执行锚的默认动作，即依照href中的路径打开一个页面。然而，当处理像单击这样的交互动作时，必须首先执行onclick事件属性。否则，如果浏览器在onclick有机会响应之前就导航到新的位置，那么在onclick中定义的动作也就没有意义了。

下面我们来看看第一次重写的情形，此时我们在onclick属性中使用了javascript:前缀和popup()函数：

```
<a href="http://advanceddomscripting.com" onclick="this.href= ➥
'javascript:popup(\'http://advanceddomscripting.com\');'"> ➥
AdvancED DOM Scripting</a>
```

运行这个脚本会导致下列动作：

(1) 发生onclick事件，将链接的href属性值替换为

```
href="javascript:popup('http://advanceddomscripting.com');"
```

该值使用javascript:前缀来执行被包含在文档其他地方的预定义函数popup()。

(2) 以新值执行锚的默认动作——打开href，进而执行弹出窗口函数并在其中打开指定的URL。

通过以这种方式使用onclick属性，可以完成与简单地使用javascript:前缀相同的结果，但不同的是，当JavaScript被禁用时，锚（链接）仍将起作用。

现在我们再看看第二次重写的情形，这次在onclick属性中通过引用锚的href值来使用popup()函数：

```
<a href="http://advanceddomscripting.com" onclick="popup(this.href); ➥
return false;">AdvancED DOM Scripting</a>
```

以下是在这种情况下发生的动作：

(1) 发生onclick事件，执行相同的popup()函数（或直接执行window.open()函数）：

```
popup('http://advanceddomscripting.com');
```

(2) 但是还必须返回false以阻止默认动作。

www.TopSage.com

当从嵌入的onclick事件属性中返回false时，就是在告诉浏览器停止并忽略执行链中其余的事件，包括默认动作。在这种情况下，浏览器会停止默认动作的执行，不再打开href属性中的链接。

> 大概你已经注意到了，我们在onclick事件属性的内部使用this来引用<a>标签。在JavaScript中，this用于控制函数的执行环境，引用的是对象的所有者。而在这个例子当中，<a>标签是onclick事件属性的所有者。在本书后面我们会更详细地讨论与this相关的内容。

既然你理解了编写嵌入式事件侦听器（event listener）的不同方式（避免直接在href属性中使用编码的javascript:，而是将适当的函数赋给事件属性），现在就忘掉我曾经告诉过你可以使用嵌入式事件属性。向你展示从javascript:到onclick事件属性这两种不同方法的演变是必要的，因为这样你才能看到下一个解决方案的好处。记住，你要将行为从结构化标记中分离出来，而通过嵌入的事件处理程序不会实现这个目标。最佳方案是使用不唐突的技术，在window载入时添加事件处理程序。你可以通过在文档中包含像popupLoadEvent.js这样的脚本文件，来应用与嵌入式脚本相同的逻辑，这个脚本文件中包含下列代码：

```
// 添加载入事件来修改页面
ADS.addEvent(window,'load',function(W3CEvent) {

    // 查找文档中带popup类的所有锚标签
    var popups = ADS.getElementsByClassName('popup', 'a');
    for(var i=0 ; i<popups.length ; i++ ) {

        // 为每个锚添加单击事件侦听器
        ADS.addEvent(popups[i],'click',function(W3CEvent) {

            // 使用href属性打开窗口
            window.open(this.href);

            // 取消默认的事件
            ADS.eventPreventDefault(W3CEvent);
        });
    }
});
```

然后，通过在class属性中加上popup类标记出相应的锚：

```
<a href="http://advanceddomscripting.com" class="popup"> ➥
AdvancED DOM Scripting</a>
```

> 在这个例子中，由于使用了很多ADS对象的方法，所以可能会有点不好理解。ADS对象是我们将在学习全书的过程中创建的一个公共库，但目前还没有开始创建，所以这个例子是不能运行的。我们将在本章稍后开始创建ADS库。

在这个例子中，行为与结构完全分离，因为文档中根本没有嵌入的JavaScript——除了在文档的head部分中使用<script>标签包含JavaScript源文件之外。如果JavaScript无效，那么文档将会

www.TopSage.com

平稳退化，因为锚仍然链接到http://advanceddomscripting.com。如果JavaScript有效，那么通过在所有带popup类的锚上添加单击事件侦听器，可以增强文档的行为。这一思想与Jeremy Keith在描述Ajax时（http://domscripting.com/blog/display/41）提到的Hijax[①]方法相似，只不过在这里我将它应用到了所有事件侦听器。

使用类（或id）属性来标识元素的另一个好处是，你可以通过相同的类名为锚添加独特的样式（使用.popup CSS选择符）：

```
.popup {
    padding-right: 25px;
    background: transparent url(images/popup.png) ➥
no-repeat right center;
}
```

该样式预示这个锚的链接将在弹出窗口中打开，如图1-2所示。

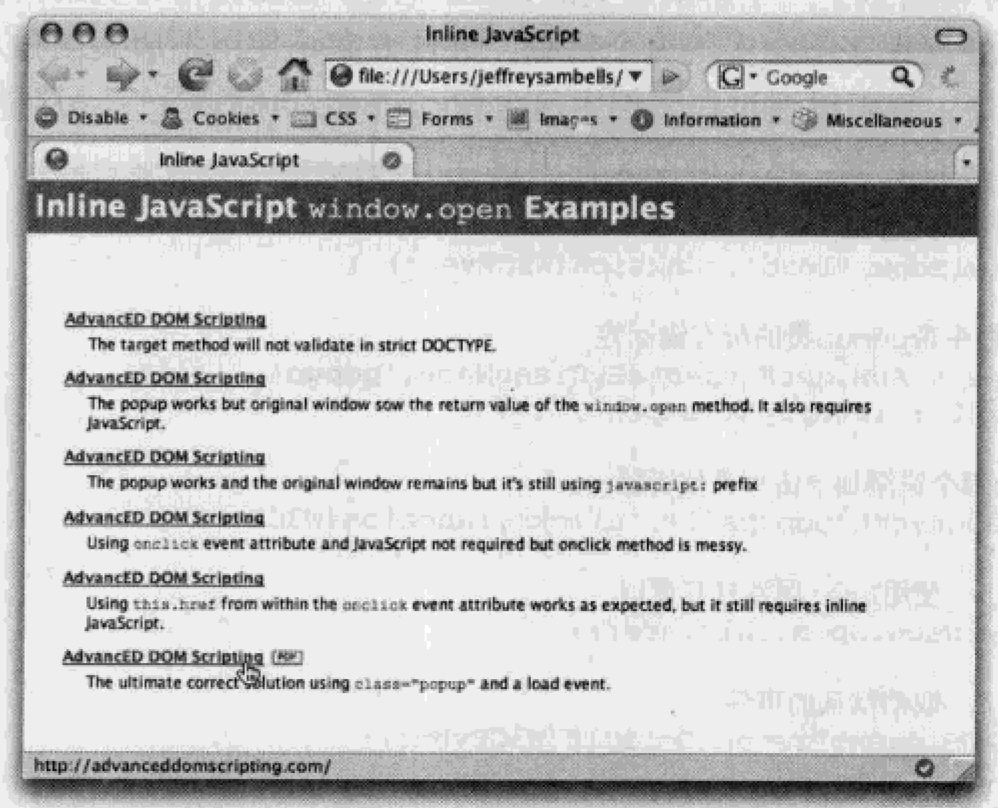



图1-2　一个应用了样式的、表示链接将在弹出窗口中打开的锚

> 我在这里举的使用window.open()的例子是相当简单的。而使用载入事件侦听器来增强页面行为的方法，几乎在本书每个修改文档的例子中都会涉及到。

此处只是对事件、事件侦听器其及正确实现的简单介绍。在第4章中，我们还将更详细地讨论有关事件的内容，因此对事件的讨论到此先告一段落。

为了示范不唐突性在实践中的应用（同时增加一点“惊喜”的因素），我要提到由

① 读者可以参考译者网站中的一篇文章（http://www.cn-cuckoo.com/2007/11/14/the-origin-of-the-concept-of-unobtrusive-144.html），或阅读《Bulletproof Ajax中文版》（人民邮电出版社）了解与Hijax相关的内容。——译者注

www.TopSage.com

http://www.huddletogether.com的Lokesh Dhakar编写的Lightbox JS。这个Lightbox图像查看解决方案堪称行为与结构分离的典范。如果你还没听说过或者没有用过Lightbox JS脚本，可以在http://www.huddletogether.com/projects/lightbox2/中看到这个脚本的应用示例，如图1-3所示。

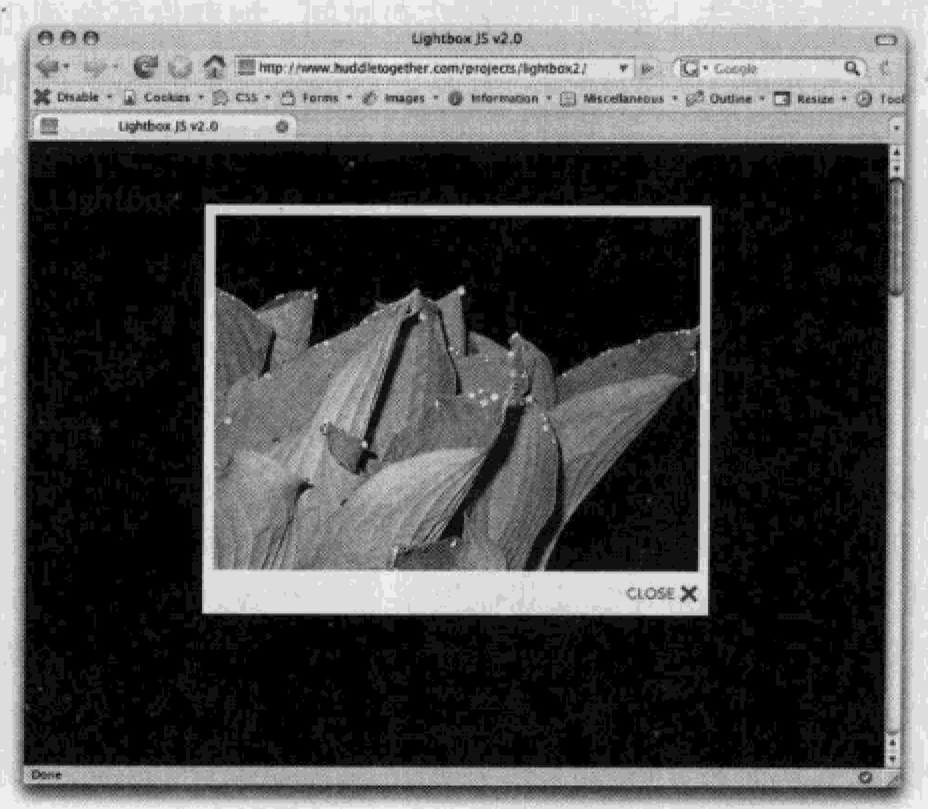



图1-3 不唐突的Lightbox JS 2.0的实际应用

Lightbox是一个不唐突的脚本，它能够取得指向图像文件的普通链接，并直接在浏览器中创建交互式的幻灯片。要调用这个脚本，只需在文档`<head>`部分中加上适当的JavaScript和CSS链接，同时在标记中为每个你希望通过Lightbox查看的包围缩略图的锚元素添加`rel="lightbox"`属性即可：

```
<a href="myphoto.jpg" rel="lightbox">
    <img src="myphoto_thumb.jpg" alt="My Photo"/>
</a>
```

仅此而已，不需要任何嵌入的脚本。它不仅有力地增强了你的网站行为和交互性，而且还对网站的用户及Web开发者都保持了可访问性、可用性以及不唐突性。

## 1.2.2 不要版本检测

什么？不要版本检测？是的，没错！就是不要那么做。相反，请考虑一下适应未来（future-proofing）。当谈到最佳实践时，“版本检测”或“浏览器嗅探”通常被认为是错误的做法。对我们大多数人而言，通过检测所希望得到的结果无非是下列浏览器中的一种：

- Microsoft IE 6和7
- Firefox 1.5和2.0
- Safari 2.0
- Opera 9

www.TopSage.com

可是，这并不意味着只有这些浏览器才能查看网页。即使有人使用古董级的浏览器尝试浏览你的网站，也不应该让网页四处蹦错。同时，可能会有一些新的不知名的浏览器跻身于市场并席卷一批用户。要编写未可知（agnostic）的代码，就需要迎合所有类型和所有版本的浏览器。但是，也不必对市场上每一款前途未卜的浏览器都花费时间。根据你所检测出的浏览器标识（要是这些标识的确没错的话）来完成不同的任务，类似于只根据投手的名字来判断棒球应该投出曲线球还是快球。借用这个棒球的类比，你的代码可能会像下面这样：

```
if ( pitcher.name == 'Addison' ) {
    pitcher.screwball.throw();
} elseif ( pitcher.name == 'Stephanie' ) {
    pitcher.fastball.throw();
} else {
    alert( 'Sorry, you can't throw the ball.' );
}
```

这样就把脚本限制为只对Addison或Stephanie有效，但要是有其他选手加入你的球队这个脚本的用处就不大了。如果有一个名叫Bobby的新投手想要投球怎么办？你的脚本会简单地假定Bobby根本不能投。要使代码对Bobby以及其他投手们都有效，则必须添加一连串if/else逻辑判断语句，将每个队员的名字以及将来可能加入球队的选手都计算在内。你真希望在每次增加新球员时添加一个新的if/else语句吗？大概不是吧，况且即便你愿意，到时候都不一定能搞清谁能干什么，谁不能干什么。只有在发现了某个具体问题的前提下增加检测才是适当的，而且不考虑将来。因此，最佳的解决方案是能力检测。

**1. 使用能力检测**

能力检测，通常也叫做对象检测，指的是在执行代码之前检测某个脚本对象或者方法是否存在，而不是依赖于你对哪个浏览器具有哪些特性的了解。如果必需的对象或方法存在，那么说明浏览器能够使用它，并且代码也可以按照预期执行，从而不必理会浏览器的版本或标识。同样，再使用棒球作类比，你可以使用能力检测将代码重写如下：

```
if ( pitcher.screwball ) {
    pitcher.screwball.throw();
} elseif ( pitcher.fastball ) {
    pitcher.fastball.throw();
} else {
    alert( 'Sorry, you can't throw the ball.' );
}
```

现在，当Bobby（他能投出快球）出现时，这些代码仍然有效，因为你没有检测他的名字，只是检测他是否能投出快球。对浏览器的检测也同样如此。如果你的代码需要document.body，就应该像下面这样进行检测：

```
if (document.body) {
    // 依赖document.body的代码
}
```

如果你的代码在不支持document.body的浏览器（例如Netscape 4或IE 3）中运行，那些令人讨厌的部分将被忽略，而如果你写的代码能够平稳退化（正如我们后面将要讨论的），那么浏

www.TopSage.com

览器仍然能够访问到信息，只不过页面中会少一些闪光点罢了。在所有其他浏览器（即使是较新的和将来出现的浏览器最终都会支持document.body）中运行这些代码，则不会出现差错，或者说无须修改。

如果需要检测多个对象，甚至可以使用逻辑操作符。要检测document.body.getElementsByTagName，你可以使用下列代码：

```
if ( document.body && document.body.getElementsByTagName) {
    // 使用document.body.getElementsByTagName的代码
}
```

但是，如果你只检测了下面这一项

```
if (document.body.getElementsByTagName) {
    // 使用document.body.getElementsByTagName的代码
}
```

那么，当浏览器中不存在document.body时，就会返回错误。

当然，也不必每次都对将要使用的函数和方法进行检测，否则你会疯掉的。只需在脚本开始处对你打算使用的所有对象和方法进行一次检测，而在万一遇到所需功能不可用时退出即可。

```
var W3CDOM = document.createElement && ➥
document.getElementsByTagName;

function exampleFunctionThatRequiresW3CMethods() {
    if(!W3CDOM) return;
    // 使用document.createElement()
    // 和document.getElementsById()的代码
}
```

在进行能力检测时要记住一件重要的事情，即你在检测中使用的JavaScript对象应该与要执行的代码中的对象和方法紧密相关。不要在检测中使用无关的方法，因为那样将会构成不符合实际需要的虚假的比较关系。

**2. 什么时候使用浏览器版本嗅探**

无论何时，你都应该尽可能地使用基于对象的能力检测，但有一种情况下，你也将被迫进行浏览器标识的检测——在涉及产品独有的特性时。如果每个浏览器都实现了同一个对象，但与这个对象及其方法或属性的交互方式不同，也不可能根据一个特殊对象或方法的存在来修改脚本代码。在这些情况下，你就必须使用浏览器嗅探，因为这是基于同一对象检测产品特有差异的唯一方法。至于如何检测具体的浏览器，将取决于你要进行测试的产品特性。我们将在第7章中更深入地讨论这一问题，届时你将看到浏览器产品的特性会发生冲突，而为了化解这些冲突，你必须拿出一些有创造性的解决方案。

### 1.2.3 通过平稳退化保证可访问性

在编写DOM脚本时的第1条规则，就是不能依赖于JavaScript的可用性。在编写JavaScript代码时的第2条规则，也是不能依赖于JavaScript的可用性。知道第3条规则是什么吗？你猜对了。在前面两小节中，我们讨论了包含JavaScript以及检测适当的兼容性的最佳方式。适当的包含和能力检

www.TopSage.com

测将在很大程度上帮助网站应对各种可能的浏览器及不同的版本，但如果根本没有JavaScript又会怎样呢？

在本章前面讲过脚本应该增强（而不是提供）行为和交互性。要牢记这一点，因为你必须要照顾到访问你网站的人们或者程序不能使用JavaScript，或者JavaScript功能受限制，或者只有部分JavaScript功能。根据W3C（http://www.w3schools.com/browsers/browsers_stats.asp）的统计，截止到2007年1月份，还有6%（2006年1月是10%）的人使用的浏览器不支持JavaScript，或者禁用了JavaScript。然而，事实上正如W3C所指出的，不应该僵化地看待这些世界范围内的统计数据。如果你的开发针对的是局部性市场，而该市场的统计数据很可能迥异于全世界的平均水平。同样，除非你的目标市场是熟悉Web的（web-savvy）用户，否则多半用户都不一定知道他们的浏览器中能运行JavaScript，更不用说要禁用它了。

如果你的脚本是使用不唐突的技术来编写的，那么没有JavaScript的浏览器也将像启用了JavaScript的浏览器一样方便地访问到等量的信息。同样，对于JavaScript能力受限或具有非标准JavaScript能力的设备，说“欢迎您，请禁用JavaScript以获得更一致的体验”一定会比“本站需要JavaScript支持，否则请离开”更加友好。

**不要使用JavaScript生成内容——结束了**

不管谁有还是没有JavaScript，但有一类你需要依赖但却根本不支持JavaScript的程序，即搜索引擎的爬虫（bot）和蜘蛛（spider）。在因特网中爬行的爬虫会搜索你的全部页面和内容，但不会执行JavaScript代码。如果你想被发现，那么对于导航和内容来说最好不要依赖于JavaScript——这一点无论怎么强调都不过分。

> 如果你经常关注有关搜索引擎爬虫的最新消息，大概听说过Google的爬虫程序能够下载网页及其中的CSS和JavaScript文件的传闻。这表明Google可能正在尝试将JavaScript导航整合到他们的爬虫程序中，或者仅仅是为了丰富http://google.com/codesearch而搜索你的源代码。但不论怎样，假如Google真的找到了一种理解和操纵JavaScript界面的方法，那它的爬虫程序对这条规则来说也只是个例外，所以还是不要依赖它。

对于公共网页中要给所有人看的重要内容，你不能完全依赖JavaScript来生成。因为当搜索引擎进入你的网站时，你必须要保证它能得到适当的信息。为此，当你的网站开发即将结束时，最好能在禁用JavaScript的情况下再仔细检查一遍每个页面。假如你没有看到应该看到的内容，那说明还没有到宣布结束的时候。

> 做个小试验：在你的浏览器中禁用JavaScript，然后在网上冲浪一天。你将会惊讶地发现当在自己心爱的网站中浏览、导航时，怎么会碰到那么多的障碍。

### 1.2.4 为重用命名空间而进行规划

有没有朋友和你同名？如果有，那么你应该很快就能认识到命名空间在脚本中的价值。可以

www.TopSage.com

想象一下这个情景：当你要问某人一个问题时，你只叫出了一个名字，结果有好几个人回答，或者根本就没人回答。在你的代码中，要避免这种情况可以使用命名空间——这个JavaScript中最被人忽视但却容易实现的特性之一。不过，这里我要限定一下前面的说法：至少在JavaScript 2.0被广泛使用之前，我所说的都不是真正的命名空间，只是能在脚本内部营造一个属于你自己的小空间的小技巧而已。

正如后面将要学习到的，JavaScript不会抗议多次声明同一个函数，它只是使用最后声明的版本。当你要使用几个自行其事的库时，必须确保它们不会与你自己编写的代码发生冲突。而要避开这些问题困扰，只需记住简单的两点：保持唯一性和不共享。

要保持唯一性，首先要为自己的空间挑选一个不会在别处被使用的名字。例如，Yahoo的库使用的是YAHOO，而Google Maps则在所有标识符中都添加了G前缀。很快你就会看到，我会在本书所有的例子中用ADS来表示Advanced DOM Scripting[①]，不过你也可以将其修改为自己认为更具唯一性的名字。如果你使用自己的名字，那么即使同时加载本书的源代码和你自己的源代码，也可以互不干扰地运行它们。

而“不共享”则是所有魔力的发源地。通过“不共享”（我的意思是什么都不共享），即可实现你的目标。当你在编写脚本时，可能经常会需要一些内部的、专用的函数，这些函数不一定是整个脚本或网站的一部分，但这样反倒会使这些函数的编写及维护更容易。假设你要在自己的网站中创建一个所有人都可以访问的名为`alertNodeName()`的函数。该函数只用于修改一个HTML元素的`nodeName`属性。同时，你可能也会创建一个通过`id`捕获元素的名为`$()`的函数，以便在`alertNodeName()`以及你库中的其他函数内部使用它：

```
function $(id) {
    return document.getElementById(id);
}
function alertNodeName(id) {
    alert($(id).nodeName);
}
```

还不错，但是由于你的`$()`函数与来自Prototype库（http://prototypejs.org）的众所周知的Prototype `$()`函数并不相同，所以这样一来，其他一些期待不同功能的公共库或脚本可能会因此而停止运行。要保证只有你自己使用这个`$()`函数，可以使用一些JavaScript技巧。这里所说的技巧指的是一个自执行的匿名函数，相应的代码如下所示：

```
(function(){
    //代码
})();
```

可能你以前看到过这样的代码，并且也曾奇怪过它能做什么——应该承认，这是相当优雅的代码。包围函数（`function(){}`）的第一对括号向脚本返回未命名的函数，随后的一对空括号立即执行返回的未命名函数。要是我向其中放入一个参数，那么理解起来大概会更容易一些：

```
(function(arg){
    alert(arg);
```

① 本书英文名。——译者注

www.TopSage.com

```
})('This will show in the alert box');
```

> 与此相同的伪命名空间的思想，也可以通过稍有不同的对象语法来实现。但它们要完成的任务都是相同的，即保证你的代码被包含在它自己的小空间内。

本质上来看，这也没有做什么特别的，但是，只要你把自己所有的代码都写在这个特殊的函数包装内，那么将没有人能够在这个特殊的包装之外访问到你的任何自定义函数或对象——除非你允许访问。为了避免冲突，你需要像下面这样将自己的$()和alertNodeName()函数放在这个伪命名空间内：

```
(function(){

    function $(id) {
        return document.getElementById(id);
    }

    function alertNodeName(id) {
        alert($(id).nodeName);
    }

    window['myNamespace'] = {};
    window['myNamespace']['showNodeName'] = alertNodeName;

})();
```

使用这种伪命名空间可以封装并保护自己的所有函数、对象和变量。而且，由于它们位于同一个函数之中，所以它们之间仍然可以互相访问。不过，脚本其他部分中的代码将无法使用你的函数。现在，你的$()函数以及位于包装中的一切都是你自己的了。

为了对你受保护的部分脚本进行全局化，随后的一对括号告诉浏览器立即执行返回的匿名函数，而且在执行期间，最后几行代码将alertNodeName()赋值给了window的一个方法：

```
window['myNamespace']['showNodeName']
```

然后，就可以在位于这个命名空间外部的其他脚本中引用这个方法了。为了方便说明这一点，你会注意到我把alertNodeName()重命名为了window方法中的showNodeName()。这样，在包装的外部，将无法直接执行alertNodeName()。但通过将alertNodeName()函数赋值给window ['myNamespace']['showNodeName']，实际上创建了带有下面这样优雅的myNamespace命名空间的通用方法，当执行这个方法时就如同是在执行受保护的alertNodeName()函数一样：

```
myNamespace.showNodeName('example');
```

由于存在作用域和闭包，实现前面的内部赋值和外部调用是完全可能的，有关作用域和闭包的内容我们将在本章后面讨论。简言之，当你调用myNamespace.showNodeName()时，这个方法是在你的匿名函数的命名空间中执行的，因此它仍然能够访问你专用的$()函数和该命名空间中的其他对象，以及作用域链中位于这个包装外部的所有对象。

www.TopSage.com

> 如果你对所有这些都感到有点困惑，别担心。本章剩下的内容以及第2章应该可以澄清许多你所关心的JavaScript中对象的问题。因此，请允许我稍后再为你介绍有关JavaScript对象构成的更高级特性。

使用匿名函数封装的另外一个好处是，你不需要将自己的命名空间前缀逐一添加到众多的函数和对象上。唯一需要添加命名空间，同时也是你担心可能会因疏忽而覆盖其他标识符的地方，就是给window对象赋值的语句。当然，如果在你的伪命名空间内部覆盖了任何函数（像$），也将会因为明显的异常而引起你对变化的注意，所以这也算不上一个问题。

### 1.2.5 通过可重用的对象把事情简化

知道了如何使用最佳实践来整合你的代码还只是第一步，如果你认真研究过因特网上那些卓越而且令人惊叹的JavaScript库（详细内容见第9章），应该会注意到它们中多数都具备的一些共性特征。几年来，很多使用JavaScript的人写出了一些我们都能从中受益的方便的小函数。尽管不是官方JavaScript语言的一部分，但使用这些经过验证的可靠的函数会使你的代码更清晰、更易读，而且当别人需要调试或搞懂你的开发成果时也将更容易被人理解。如果你在因特网上搜索“Top 10 JavaScript”这几个关键词，将会得到一串大多数开发者赖以生存的常用函数。

库也是一个饱受争议的热门话题。一种观点认为它们是非常棒的工具，是任何开发者都不可或缺的；而另一种观点则认为在不理解库的内部工作原理的情况下对库形成依赖，会助长懒惰的风气从而导致开发者素质下降。从个人角度来说，我认为应该针对具体问题或者具体的库进行具体分析。但不管怎样，好像每个人都赞同编写自己的库，况且把自己日常要用的东西合并到一起本身就是一件有意义的事。那么，下面就根据这个观点来建立一个你自己的库，将在本书后面的章节中随时用到并不断丰富它的内容。

#### 1. 开始构建ADS库

首先，你需要一个唯一的命名空间。在本书其余章节中，我将使用ADS来表示Advanced DOM Scripting，而现在你也使用这个名称从而轻松地与本书代码保持一致。本书中大多数例子都将引用本书源代码根文件夹中的文件ADS-final-verbose.js。这个文件是一个完整的库，包含了你将要向其中添加的全部代码，而且也包含描述每个方法用途的注释。我建议你不要仅仅复制所提供文件中的代码，而应该花点时间独立地创建这个文件，以便理解其中的内容——而这正是本书的全部价值所在。不过，假如你遇到了麻烦并且需要能够运行的版本，可以将本书提供的文件作为参照。

其次，创建一个名为ADS.js的JavaScript文件，并以下列命名空间和方法框架作为你自己库的起点（下面的起点文件代码可以在本书的源文件chapter1/ADS-start.js中找到）：

```
(function(){

//ADS命名空间
if(!window.ADS) { window['ADS'] = {} }
```

www.TopSage.com

```
    function isCompatible(other) { };
    window['ADS']['isCompatible'] = isCompatible;

    function $() { };
    window['ADS']['$'] = $;

    function addEvent(node, type, listener) { };
    window['ADS']['addEvent'] = addEvent;

    function removeEvent(node, type, listener) { };
    window['ADS']['removeEvent'] = removeEvent;

    function getElementsByClassName(className, tag, parent){ };
    window['ADS']['getElementsByClassName'] = getElementsByClassName;

    function toggleDisplay(node,value) { };
    window['ADS']['toggleDisplay'] = toggleDisplay;

    function insertAfter(node, referenceNode) { };
    window['ADS']['insertAfter'] = insertAfter;

    function removeChildren(parent) { };
    window['ADS']['removeChildren'] = removeChildren;

    function prependChild(parent, newChild) { };
    window['ADS']['prependChild'] = prependChild;

    })();
```

这将是你自己的公共库的起点，其中包含的一些函数签名都是可以在因特网中找到的受欢迎的可重用JavaScript方法——我们将在你向这些方法中添加内容时再讨论它们。代码开始处的命名空间对象也被包装在了一个if语句中，以便能够在多个文件中使用相同的命名空间：

```
if(!window.ADS) { window['ADS'] = {} }
```

虽然不同文件中的方法不能以相同的方式相互访问，但通过使用不同的文件将不常用的方法分离出来，可以使你的库更加模块化。当引用来自另一个文件中的ADS方法时，唯一的技巧可能就是需要包含ADS这个对象前缀。

> 这些还只是你在学习本书期间要向ADS.js文件中添加的众多方法的开始部分。我们的目标是让你在阅读本书的同时构建你自己的库，这样在学习完本书后，你不仅会拥有一些得力的工具，而且也将对本书及你的库中的内容有一个透彻的理解。

#### 2. ADS.isCompatible()方法

ADS.isCompatible()用于确定当前浏览器是否与整个库兼容。可以先按照下面的代码来构造这个方法，不过将来可能还需要对它进行更多的改造：

```
function isCompatible(other) {
    // 使用能力检测来检查必要条件
    if( other===false
        || !Array.prototype.push
```

www.TopSage.com

```
            || !Object.hasOwnProperty
            || !document.createElement
            || !document.getElementsByTagName
            ) {
            return false;
        }
        return true;
    }
    window['ADS']['isCompatible'] = isCompatible;
```

这个方法通过在一个简单的if语句中包装代码，为我们提供了一个确定浏览器是否能够使用库中所有方法的简便而快捷的方式：

```
if(ADS.isCompatible()) {
    // 使用ADS库的代码
}
```

就如同你要在ADS库的所有方法中都进行能力检测一样，其实这种检测并非对所有脚本都是绝对必需的。但是，有一些你将在本书中使用的代码依赖于JavaScript 1.5，而非常古老的浏览器并不支持JavaScript 1.5。在这种情况下，使用前面的检查通过寻找一些普通的对象来保证页面能够迅速而平稳地退化，胜过抛出JavaScript错误或者没有节制地对每一行代码都进行能力检测。

我不会在本书全部的代码例子中都使用这种检测包装，因为所有例子都将借助载入事件来修改页面。而你将在本章稍后添加的ADS.addEvent()方法和在第4章中添加的ADS.addLoadEvent()方法中都会包含这样的isCompatible()检查，因此如果检查发现不兼容就不会运行载入事件处理程序。

如果你要向这个库中增加你自己的使用其他较新的JavaScript特性的方法，那么就应该根据需要修改你的ADS.isCompatible()方法。

3. **ADS.$()方法**

我最喜欢的流行函数是由Prototype JavaScript框架（http://prototypejs.org）推而广之的$()，它可以为你节省很多输入工作量。而从本质上看，$()只是document.getElementById()的替代函数。为创建这个方法，请把下列代码添加到你的新ADS命名空间中：

```
function $() {
    var elements = new Array();

    // 查找作为参数提供的所有元素
    for (var i = 0; i < arguments.length; i++) {
        var element = arguments[i];

        // 如果该参数是一个字符串那假设它是一个id
        if (typeof element == 'string') {
            element = document.getElementById(element);
        }

        // 如果只提供了一个参数，
        // 则立即返回这个元素
        if (arguments.length == 1) {
            return element;
        }
```

www.TopSage.com

```
            // 否则，将它添加到数组中
            elements.push(element);
        }

        // 返回包含多个被请求元素的数组
        return elements;
    };
    window['ADS']['$'] = $;
```

编写完以上代码后，你就可以在自己所有的脚本中只通过包含下面的引用来取得与id对应的DOM元素了：

```
var element = ADS.$('example');
```

以上这行代码等价于

```
var element = document.getElementById('example');
```

> Prototype 1.5中添加了一个更高级的$()方法，该方法给DOM元素增加了许多新方法。我们这里的ADS.$()方法只用于取得一个常规的DOM元素。

在ADS命名空间内部，可以只使用$()而不必给它添加前缀。这虽然是一个简单的想法，但要是用在一个大型复杂的库中，它就能节省大量的时间。另外，你刚才实现的ADS.$()函数也支持请求多个元素，它会在这种情况下返回一个包含那些元素的数组。然后，可以轻松地对返回的结果数组进行循环遍历，并对其中的元素进行操作：

```
var elements = ADS.$( 'a' , 'b' , 'c' , 'd' );
for ( e in elements ) {
    // 执行某些操作
}
```

而且，对于需要向其中传递一个DOM元素引用的库方法来说，ADS.$()也是得力的帮手。如果将下面这行代码：

```
if(!(obj = $(obj))) return false;
```

添加到你的ADS库的方法中：

```
function exampleLibraryMethod(obj) {
    if(!(obj = $(obj))) return false;
    // 对obj进行某些操作
}
window['ADS']['exampleLibraryMethod'] = exampleLibraryMethod;
```

那么，在指定参数时，既可以使用代表对象id的字符串：

```
var element = ADS.exampleLibraryMethod('id');
```

也可以使用对象的引用：

```
var element = ADS.exampleLibraryMethod(ADS.$('id'));
```

要记住的一点是，在命名空间内编写代码不需要包含ADS前缀，但要在命名空间之外编写代码，

www.TopSage.com

就必须包含这个前缀。

Prototype库的$()函数也为元素应用了一些方法，以便开发者能够像下面这样通过点记号连缀方法：

```
// Prototype库中添加了像getElementsByClassName()这样的方法
var elements = $('element-id').getElementsByClassName('className');
```

为了简化例子，你的库不会遵循这种模式，但如果你愿意也可以将来自己添加。Prototype库中也有许多其他有用的函数和对象，这里不会逐一对它们进行介绍，如果你想了解更多内容，可以跳到第9章，它在更详细地介绍Prototype库的同时，也介绍了其他几个库。

4. **ADS.addEvent()和ADS.removeEvent()方法**

我们在前面讨论过使用JavaScript源文件和不唐突的方法为标记结构中的对象添加事件。但是，这样做不仅本身很麻烦，而且会导致代码体积增大。为了简化操作并提高可读性，可以使用一个在前面弹出窗口的例子中看到过的addEvent方法。现在就在你的ADS命名空间中加上下面的addEvent()和removeEvent()方法：

```
function addEvent( node, type, listener ) {
    // 使用前面的方法检查兼容性以保证平稳退化
    if(!isCompatible()) { return false }

    if(!(node = $(node))) { return false; }

    if (node.addEventListener) {
        // W3C的方法
        node.addEventListener( type, listener, false );
        return true;
    } else if(node.attachEvent) {
        // MSIE的方法
        node['e'+type+listener] = listener;
        node[type+listener] = function(){
            node['e'+type+listener](window.event);
        }
        node.attachEvent( 'on'+type, node[type+listener] );
        return true;
    }

    // 若两种方法都不具备则返回false
    return false;
};
window['ADS']['addEvent'] = addEvent;

function removeEvent(node, type, listener ) {
    if(!(node = $(node))) { return false; }

    if (node.removeEventListener) {
        // W3C的方法
        node.removeEventListener( type, listener, false );
        return true;
    } else if (node.detachEvent) {
        // MSIE的方法
        node.detachEvent( 'on'+type, node[type+listener] );
        node[type+listener] = null;
```

www.TopSage.com

```
        return true;
    }
    // 若两种方法都不具备则返回false
    return false;
};
window['ADS']['removeEvent'] = removeEvent;
```

有各种不同风格的addEvent()方法散见于因特网中，例如Dean Edwards编写的addEvent()方法（http://dean.edwards.name/weblog/2005/10/add-event）。这里，我介绍的是在John Resig的addEvent()和removeEvent()方法（http://ejohn.org/projects/flexible-javascript-events）基础上稍作修改而成的更详细的版本。这些addEvent()方法都实现了同一个目标——在元素上注册事件侦听器。之所以介绍John Resig的版本，是因为他的方法简单，不会退化不同事件方法的各种特性，而且还修复了一些W3C和Microsoft事件注册模型之间的不一致性。

> 我们将在第4章更详细地介绍与事件有关的内容。

要使用这个ADS.addEvent()方法，只需像你在前面学习弹出窗口的例子时看到的那样，简单地指定想要组合的元素、事件类型和侦听器函数即可：

```
ADS.addEvent(window,'load',function(W3CEvent) {
    // 查找文档中所有带popup类的锚标签
    var popups = ADS.getElementsByClassName('popup', 'a');
    for(var i=0 ; i<popups.length ; i++ ) {

        // 给每个锚添加一个单击事件侦听器
        ADS.addEvent(popups[i],'click',function(W3CEvent) {

            // 使用href属性打开窗口
            window.open(this.href);

            // 取消默认事件
            ADS.eventPreventDefault(W3CEvent);
        });
    }
});
```

> 我们将在第4章添加这个例子用到的ADS.eventPreventDefault()方法。该方法用于阻止锚打开href属性中的链接。

甚至可以使用这个方法为一个窗口添加多个载入事件：

```
function sayHello() {
    alert('Hello');
}
ADS.addEvent(window,'load',sayHello);

function sayGoodbye() {
    alert('Goodbye');
}
```

www.TopSage.com

```
ADS.addEvent(window,'load',sayGoodbye);
```

这样，当窗口载入时，你会看到两个警告框：一个显示Hello，另一个显示Goodbye（但不一定是按照这个顺序）。如果你要编写多个库，而每个库都需要自己的载入事件，那么拥有多个载入事件侦听器就非常有用了。因为这样你只需使用`ADS.addEvent(window,'load',...)`为每个库添加它自己的载入事件即可，而不必编写新的、组合而成的`window.onload`函数，并在该函数中包含来自每个对象的事件了。

> 请记住第二个参数不包含on前缀（你通常会在DOM元素的事件属性中看到on前缀）。onload这样的标识符表示的是一个对象的具体属性，而简化术语load则是W3C的addEventListener()方法识别事件所必须的。因此，要用ADS.addEvent (window,'load',initPage)，而不能用ADS.addEvent(window,'onload',initPage)。

### 5. ADS.getElementsByClassName()方法

当编写需要大量操纵DOM的代码时，你会非常频繁地用到`getElementById()`方法，或者现在的`ADS.$()`方法。使用`getElementById()`的问题在于，它基于id取得DOM元素，而同一个文档中的所有id都必须是唯一的。那么当要取得一组具有公共属性的DOM元素时怎么办呢？虽然可以使用`getElementsByTagName()`来取得一组具有相同标签的元素，但这个方法却无法取得一组具有不同标签的元素。我们的方案是，可以为要选择的一组元素指定相同的类属性，然后使用你在ADS命名空间中添加的`ADS.getElementsByClassName()`方法：

```
function getElementsByClassName(className, tag, parent){
    parent = parent || document;
    if(!(parent = $(parent))) { return false; }

    // 查找所有匹配的标签
    var allTags = (tag == "*" && parent.all) ? parent.all : ➥
parent.getElementsByTagName(tag);
    var matchingElements = new Array();

    // 创建一个正则表达式。来判断className是否正确。
    className = className.replace(/\-/g, "\\-");
    var regex = new RegExp("(^|\\s)" + className + "(\\s|$)");

    var element;
    // 检查每个元素
    for(var i=0; i<allTags.length; i++){
        element = allTags[i];
        if(regex.test(element.className)){
            matchingElements.push(element);
        }
    }
    // 返回任何匹配的元素
    return matchingElements;
};
window['ADS']['getElementsByClassName'] = getElementsByClassName;
```

这个方法中大部分的代码是由Jonathan Snook（http://www.snook.ca/jonatha）和Robert Nyman

www.TopSage.com

（http://www.robertnyman.com）编写的，它也是基于相同思想的许多不同实现中的一种。为了使这个方法更详尽也更容易理解，我重写了他们的代码[①]。不仅通过前面提到的ADS.$()方法对传递的参数是不是DOM对象的引用进行检查，而且还反转了原来方法中参数的次序，即将类名从最后一个参数变成了现在的第一个。调整参数的次序是因为Firefox 3将引入一个核心的getElementsByClassName()方法，而该方法把类名作为第一个参数（与Firefox 3保持一致，意味着如果将来你想转而使用新引入的内部方法，那么代码的改动量会较少）。

从本质上说，这个方法的工作原理是在DOM文档树中查找类属性包含某个className、可选的节点和标签过滤器的元素，找出所有与指定的类、节点和标签相关的DOM元素。以下面的文档为例：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>ADS.getElementsByClassName() Example</title>
</head>
<body>
    <h1 class="findme">ADS.getElementsByClassName() Example</h1>
    <p class="findme">This is just an <em class="a">example</em>.</p>
    <div id="theList">
        <h2 class="findme">A list!</h2>
        <ol>
            <li class="findme">Foo</li>
            <li class="findme">Bar</li>
        </ol>
    </div>
</body>
</html>
```

如果像这样使用该方法：

```
var found = ADS.getElementsByClassName('findme','*',document);
```

将会得到一个由下列元素组成的数组。

- <h1>元素
- <p>元素
- <em>元素
- <h2>元素
- 列表中的第一个<li>元素
- 列表中的第二个<li>元素

同样，如果像下面这样使用该方法：

```
var found = ADS.getElementsByClassName(
    'findme', 'li', ADS.$('theList')
```

① Robert Nyman在2007年5月11日发布了这个方法的最终修订版http://www.robertnyman.com/2005/11/07/ the-ultimate-getelementsbyclassname/，除了没有添加$()方法检查参数之外，几乎与作者这里的改写版完全相同。

——译者注

www.TopSage.com

```
);
```

那么得到的数组中将只包含位于相应div中的两个<li>元素。

#### 6. ADS.toggleDisplay()方法

另一种你经常会实现的操作是切换DOM树中元素的可见性。虽然切换可见性的代码量很少，但将其提取到一个公共函数中同样会减少不必要的重复输入。在你的ADS命名空间中添加这个方法的其余代码：

```
function toggleDisplay(node, value) {
    if(!(node = $(node))) { return false; }

    if ( node.style.display != 'none' ) {
        node.style.display = 'none';
    } else {
        node.style.display = value || '';
    }
    return true;
}
window['ADS']['toggleDisplay'] = toggleDisplay;
```

然后，当像下面这样调用该方法时：

```
ADS.toggleDisplay(ADS.$('example'));
```

则相应元素的display属性值将在none和空值（默认值）之间来回切换。如果为了实现不同的显示类型[①]而重用这个函数，也可以在调用它时包含可选的第二个参数，以定义期望的默认显示属性，比如：

```
ADS.toggleDisplay(ADS.$('example'),'block');
```

或

```
ADS.toggleDisplay(ADS.$('example'),'inline');
```

重用这么一点代码看起来好像不足以说明问题，但这样做也将极大地增强代码的可读性——因为只需通过方法的名称就能看出要切换什么。

#### 7. ADS.insertAfter()方法

在你的浏览器对W3C DOM规范的实现中，提供了很多可以操纵文档结构的对象，第3章将详细介绍这些对象。但是，在DOM核心规范中好像缺少一个方法，即element.insertAfter()方法。不过，通过组合使用已有的DOM方法也能实现相应的功能。如果你在自己的ADS命名空间中实现了这个简单的包装函数，那么你的代码也将具有更好的可读性：

```
function insertAfter(node, referenceNode) {
    if(!(node = $(node))) return false;
    if(!(referenceNode = $(referenceNode))) return false;
    return referenceNode.parentNode.insertBefore(
        node, referenceNode.nextSibling
```

① 这里指CSS中的inline和block两种元素显示类型，这两种显示类型会使元素分别成为页面中的行内元素和块级元素。但对于这个方法定义而言，只有当要操纵的DOM元素当前的显示属性为none时这种切换才会实现。——译者注

www.TopSage.com

```
    );
};
window['ADS']['insertAfter'] = insertAfter;
```

有了这个方法事情就更好办了，比如下面这行代码：

```
ADS.insertAfter(ADS.$('example'),domNode);
```

要比这行代码容易理解得多：

```
ADS.$('example').parentNode.insertBefore(ADS.$('example'),domNode);
```

8. **ADS.removeChildren()和ADS.prependChild()方法**

本章最后两个要添加到ADS命名空间中的方法是ADS.removeChildren()方法：

```
function removeChildren(parent) {
    if(!(parent = $(parent))) { return false; }

    // 当存在子节点时删除该子节点
    while (parent.firstChild) {
        parent.firstChild.parentNode.removeChild(parent.firstChild);
    }

    // 再返回父元素，以便实现方法连缀
    return parent;
};
window['ADS']['removeChildren'] = removeChildren;
```

和ADS.prependChild()方法：

```
function prependChild(parent,newChild) {
    if(!(parent = $(parent))) { return false; }
    if(!(newChild = $(newChild))) { return false; }

    if(parent.firstChild) {
        // 如果存在一个子节点，则在这个子节点之前插入
        parent.insertBefore(newChild,parent.firstChild);
    } else {
        // 如果没有子节点则直接添加
        parent.appendChild(newChild);
    }
    // 再返回父元素，以便实现方法连缀
    return parent;
}
window['ADS']['prependChild'] = prependChild;
```

与ADS.insertAfter()方法类似，这两个方法也只是通过包装现有的DOM方法，实现了对开发Web应用时常见操作的封装。

### 1.2.6　一定要自己动手写代码

许多具有设计背景的开发者，乃至一些新开发者经常使用WYSIWYG（What You See Is What You Get，所见即所得）编辑器编码。虽然类似的编辑器有很多种，但除非你使用这些编辑器独有、内置的源代码管理方式，否则通过它们来整合JavaScript将永远得不到想要的结果。我强烈建议你抛开WYSIWYG编辑器，更不要依赖它对Web应用程序的行为层进行最终整合。WYSIWYG

www.TopSage.com

编辑器只适合实现初级的概念和想法，在真正涉及为站点创建代码时，它们对你不会有任何帮助。如果你想真正获得可访问的、语义化的、不唐突的经验，那么必须要自己动手编写代码。

只有通过自己编写代码才能学到更多东西，而且随着经验的增多，才能进一步理解JavaScript提供的强大特性。但是，这并不意味着你必须重新开发一个新的JavaScript库。现成的JavaScript库已经有很多了，但其中既有非常出色的，也有非常差劲的。本书第9章会讨论其中几个JavaScript库。但在此之前，我们先简短地重温在编写代码时经常会遇到的一些JavaScript陷阱。

## 1.3 JavaScript语法中常见的陷阱

即使你是一位具有十多年脚本编写经验的Web开发老手，也可能会遇到被某个问题难住几小时都想不通的时候。而大多数这种时候，问题都出在被你忽略的一个简单的错误上。因此，我认为最好提前指出一些你在学习本书例子时可能会遇到的常见陷阱。如果在使用本书中介绍的方法时遇到了问题，可以到本书的网站（http://advanceddomscripting.com）上看看是否别人也碰到了类似的问题，如果没有，再回到本节看看你的问题有没有在这里提到过。

### 1.3.1 区分大小写

你创建的所有函数和变量都是区分大小写的，因此

```
function myFunction() { }
```

不同于

```
function MyFunction() { }
```

这一规则也适用于JavaScript核心对象，如`Array`和`Object`。

### 1.3.2 单引号与双引号

单引号（'字符串'）和双引号（"字符串"）在JavaScript中没有特殊的区别，都可以用来创建字符串。但作为一般性的规则，大多数Web开发者都选择使用单引号而不是双引号，因为XHTML规范（http://www.w3.org/TR/xhtml1）要求所有XHTML属性值都必须使用双引号括起来。这样，在JavaScript中使用单引号，而对(X)HTML属性使用双引号，会使混合两者的代码更方便也更清晰。

例如，无论是读还是写下列代码：

```
var html = '<h2 class="a">A list!</h2>'
    + '<ol>'
    + '<li class="a">Foo</li>'
    + '<li class="a">Bar</li>'
    + '</ol>';
```

都比在下面例子中转义所有的双引号来得更容易：

```
var html = "<h2 class=\"a\">A list!</h2>"
    + "<ol>"
    + "<li class=\"a\">Foo</li>"
    + "<li class=\"a\">Bar</li>"
    + "</ol>";
```

www.TopSage.com

但是，对于行内（inline）的单引号仍然必须转义：

```
var html = '<p class="a">Don\'t forget to escape single quotes!</p>';
```

### 1.3.3 换行

不论你使用哪种引号来创建字符串，字符串中间都不能包含强制换行符：

```
var html = '<h2 class="a">A list!</h2>
    <ol>
        <li class="a">Foo</li>
        <li class="a">Bar</li>
    </ol>';
```

如果这样做，就会导致解析错误，因为换行符将被解释为分号（;）。如果想把字符串分割到多行中定义，要通过反斜杠来转义换行符[①]以告知浏览器该行是连续的：

```
var html = '<h2 class="a">A list!</h2>\
    <ol>\
        <li class="a">Foo</li>\
        <li class="a">Bar</li>\
    </ol>';
```

要注意的是，这样做虽然能够在字符串中保留空白和缩进，但如果你打算使用第三方压缩工具来压缩代码，那它可能会带来负面效果。

另一个可选的方案，是使用字符串连接操作符（+）并将每一行用引号括起来：

```
var html = '<h2 class="a">A list!</h2>'
    + '<ol>'
    + '<li class="a">Foo</li>'
    + '<li class="a">Bar</li>'
    + '</ol>';
```

通过使用引号和连接操作符，结果字符串的标签之间不会存在空白。

### 1.3.4 可选的分号和花括号

用分号来结束一条语句或一行代码并不是必需的。换行符通常也会被假定为分号，除非换行符处于某个控制结构中，因此以下代码

```
alert('hello')
alert('world')
alert('!')
```

会被解释为：

```
alert('hello');
alert('world');
alert('!');
```

然而，以下代码

① 换行符通常属于不可见字符，所以要转义换行符只需在每行结尾最后一个字符（这里的>）之后添加反斜杠即可。——译者注

www.TopSage.com

```
if(a==b)
alert('true!')
alert('false?')
```

不会被解释成：

```
if(a==b);
alert('true!');
alert('false?');
```

而是会按照if控制结构被解释为：

```
if(a==b) {
    alert('true!');
}
alert('false?');
```

结果是只有a等于b的情况下才会提示“true!”，而不管前面是什么条件都会提示“false?”。这样的代码解释机制有时候无论对新老开发者都会造成困扰，特别是当代码中混合了不同的标记（如HTML标签）时更是如此。为避免这种困扰，我强烈建议你无论在什么情况下都要使用分号和花括号：

```
if(a==b) {
    alert('true!');
}
alert('false?');
```

编写出能够清晰地反映出自己意图的代码，不仅方便自己将来查看，而且方便接手你项目的其他人遵从和理解。

### 1.3.5 重载（并非真正的重载）

JavaScript不支持重载，因此这里所说的重载实际上更像是替换。所谓重载，指的是一门编程语言根据传递给函数或方法的参数的数据类型[①]，区别不同的函数或方法的能力。例如，利用重载可以像下面这样声明两个同名函数：

```
function myFunction(a,b) { alert(a+b); }
function myFunction(a) { alert(a); }
```

根据它们参数的个数不同（也包括每个参数的数据类型，而这在JavaScript中是无效的）这一事实，可以将它们视为两个单独的函数。因此，在支持真正重载的语言中，调用下面的函数：

```
myFunction(1);
```

将提示“1”，而调用下面这个

```
myFunction(1,2);
```

则会提示“3”。然而，如果在JavaScript中尝试以上过程，那么第二个声明的myFunction()将会替换第一个，因而接下来的两次调用都将执行同一个函数，并且都提示“1”。

在JavaScript中，脚本在执行时不会顾及函数定义时的参数，而是直接使用在作用域链中最后

① 及参数的个数。——译者注

www.TopSage.com

定义的那个函数。这意味着，相同名称的函数永远只存在一个实例，所以当创建自己的函数或方法时，要确保不会覆盖现有的JavaScript核心元素——除非你是有意那么做。也就是说，可以通过定义自己的同名对象来轻易地覆盖任何JavaScript对象。如果编写下面这个函数：

```
function alert(message) {
    ADS.$('messageBox').appendChild(document.createTextNode(message));
}
```

那么浏览器将不会再像往常一样提示信息，而是会按照新定义的alert函数执行规定动作——此处会将信息添加到指定的<div>。在某些情况下，这可能是你期望的，不过一定要记住，如果你在同一页面中组合使用了多个库和不同来源的代码，而这些脚本恰好需要使用被你覆盖的核心函数完成相应的功能，那么就可能会得到意外的结果。

### 1.3.6　匿名函数

你经常能在JavaScript中看到而又可能会感觉莫名其妙的一个优雅的特性就是匿名函数。它是一种在定义时不带名称的函数，而且本章中已经出现过很多次了。匿名函数对于在DOM对象上注册事件侦听器或者将函数作为参数传递给其他方法时特别有用。在下面的例子中，我们使用一个命名函数和前面的ADS.addEvent()方法在锚上注册了一个click事件：

```
function clicked() {
    alert('Linked to: ' + this.href);
}
var anchor = ADS.$('someId');
ADS.addEvent(anchor, 'click', clicked);
```

这里，先行定义了clicked()函数，然后又将它赋值给锚的click事件侦听器。结果相当不错——以上代码不仅能够很好地运行，而且也支持将来多次重用clicked()函数。

现在再看一个完成同一个目标的替代方式，但此时我们直接传递一个匿名函数：

```
var anchor = ADS.$('someId');
addEvent(anchor, 'click', function () {
    alert('Linked to: 'this.href);
});
```

在这个例子中，直接传递到ADS.addEvent()方法中的函数没有名称，也就是说这个函数是匿名的。匿名函数非常适合于只应用到特殊元素上的非常简单或者非常有针对性的代码，但这并不是它唯一的用途，本书中很多地方都体现了这一点。

> 要了解有关JavaScript对象的更多内容，请看第2章。

### 1.3.7　作用域解析和闭包

因为下一章将会详细讨论JavaScript的对象模型，所以这里只简单介绍一下作用域和闭包的问题。

首先，我们看作用域。作用域是指对某一属性（变量）或方法（函数）具有访问权限的代码空间。在JavaScript中，作用域是在函数中进行维护的，即在下面这个函数中

```
function myFunction() {
```

www.TopSage.com

```
    var myVariable = 'inside';
}
```

变量myVariable的作用域会被限制在myFunction()中。如果你在这个函数之外访问myVariable，结果将是无效的：

```
function myFunction() {
    var myVariable = 'inside';
}
myFunction();
alert(myVariable);
```

同样地，在这种情况下执行这个函数不会影响到外部作用域：

```
function myFunction() {
    var myVariable = 'inside';
}
// 定义变量
var myVariable = 'outside';
// 执行前面的函数
myFunction();
alert(myVariable); // 将提示"outside"
```

在这个例子中，因为在函数内部使用了var关键字来维护作用域链，所以执行myFunction()函数对位于其外部作用域中的myVariable没有影响。如果在myFunction()内部去掉为myVariable赋值时使用的var关键字，那么myVariable的作用域将会解析到myFunction()的外部，因此就会修改外部的变量：

```
function myFunction() {
    // 没有使用var
    myVariable = 'inside';
}
// 定义变量
var myVariable = 'outside';
// 执行函数
myFunction();
alert(myVariable); // 将提示"inside"
```

作用域链是用来描述一种路径的术语，沿着该路径可以确定变量的值（或者当函数被调用时要使用的方法）。当给myVariable赋"inside"值时，myFunction()的作用域中没有使用var关键字，因此赋值操作会沿着作用域链查找到执行myFunction()的作用域中（在这种情况下是window对象内部），并修改其中myVariable实例的值。从本质上说，var关键字决定了哪个函数是特定变量的作用域链的终点。同样的逻辑也适用于取得变量值的情况。

> 除了var关键字之外，像下面这样将变量包含在定义函数时的圆括号内，与使用var关键字的效果是相同的，即该变量也会被包含在函数的作用域内：function myFunction(**myVariable**) { ... }

闭包是与作用域相关的一个概念，它指的是内部函数即使在外部函数执行完成并终止以后，仍然可以访问其外部函数的属性。当引用一个变量或方法时，JavaScript会沿着由对象执行路径构

www.TopSage.com

成的作用域链对作用域进行解析，查找变量最近定义的值，一旦找到，即使用该值。

作用域解析和闭包中的隐含问题往往并不是一开始就显而易见的，而且通常是我遇到过的“残缺（broken）”脚本中大部分问题的根源所在。为了说明这一点，我们来看下面的例子：

```
function initAnchors(W3CEvent) {
    for (var i=1 ; i<=3 ; i++ ) {

        var anchor = document.getElementById('anchor' + i);

        ADS.addEvent(anchor,'click',function() {
            alert('My id is anchor' + i);
        });
    }
});
ADS.addEvent(window, 'load', initAnchors);
```

这个例子通过在窗口载入完成时运行`initAnchors()`函数，为页面中的锚注册事件侦听器。现在，我们假设文档中有3个锚元素，它们的ID值分别为`anchor1`到`anchor3`，那么单击这几个锚的结果会怎样呢？你可以自己试一下，打开本书源文件中的chapter/scopechain/example1.html，并单击其中的几个锚。

虽然代码中的语法正确，但其中的逻辑却存在缺陷。大多数新手，甚至某些经验丰富的程序员，都会设想当单击每个锚时，会提示My id is anchorX（其中X对应着指定`click`事件侦听器时`i`的值）。如果真是这样，单击第一个锚应该提示My id is anchor1，而单击第三个应该提示My id is anchor3。但这种设想是错的。事实上单击每个锚都会提示相同的信息，如图1-4所示。

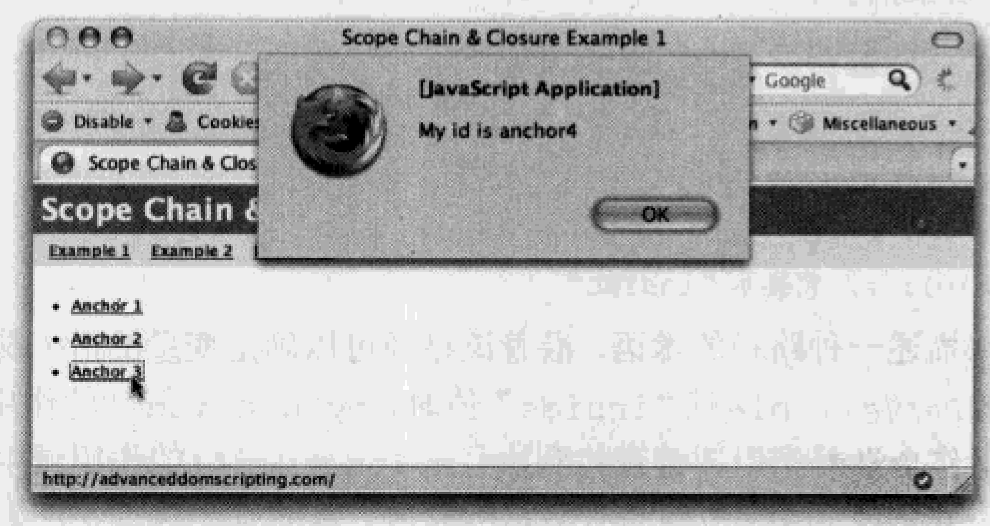



图1-4　警告窗口中显示了单击后的实际结果

为什么会这样呢？因为`i`的值实际上是在单击事件发生时才从作用域链中取得的。当单击事件发生时，`initAnchors()`已经执行完毕，因此`i`的值等于4（因为`i`要递增为比3大的数才能使循环停止），所以每个`alert()`都会显示相同的信息。

具体来说，当`click`事件侦听器被调用并在它的内部作用域中查找`i`的值时，结果没有找到[①]，

① 因为`i`的值在作为`click`事件侦听器的匿名函数中没有定义。——译者注

www.TopSage.com

因此它会到包含自己的外部作用域即initAnchors()函数中去查找。而在initAnchors()函数中i的值是4，所以它就从该处取得了这个值。图1-5说明了这一过程。

要得到正确的结果，需要把事件侦听器的注册转移到一个独立的函数中，并通过该函数的参数传递适当的值：

```
function registerListener(anchor, i) {
    ADS.addEvent(anchor, 'click', function() {
        alert('My id is anchor' + i);
    });
}
function initAnchors(W3CEvent) {
    for (i=1 ; i<=3; i++ ) {
        var anchor = document.getElementById('anchor'+i);
        registerListener(anchor,i);
    }
}
ADS.addEvent(window, 'load', initAnchors);
```

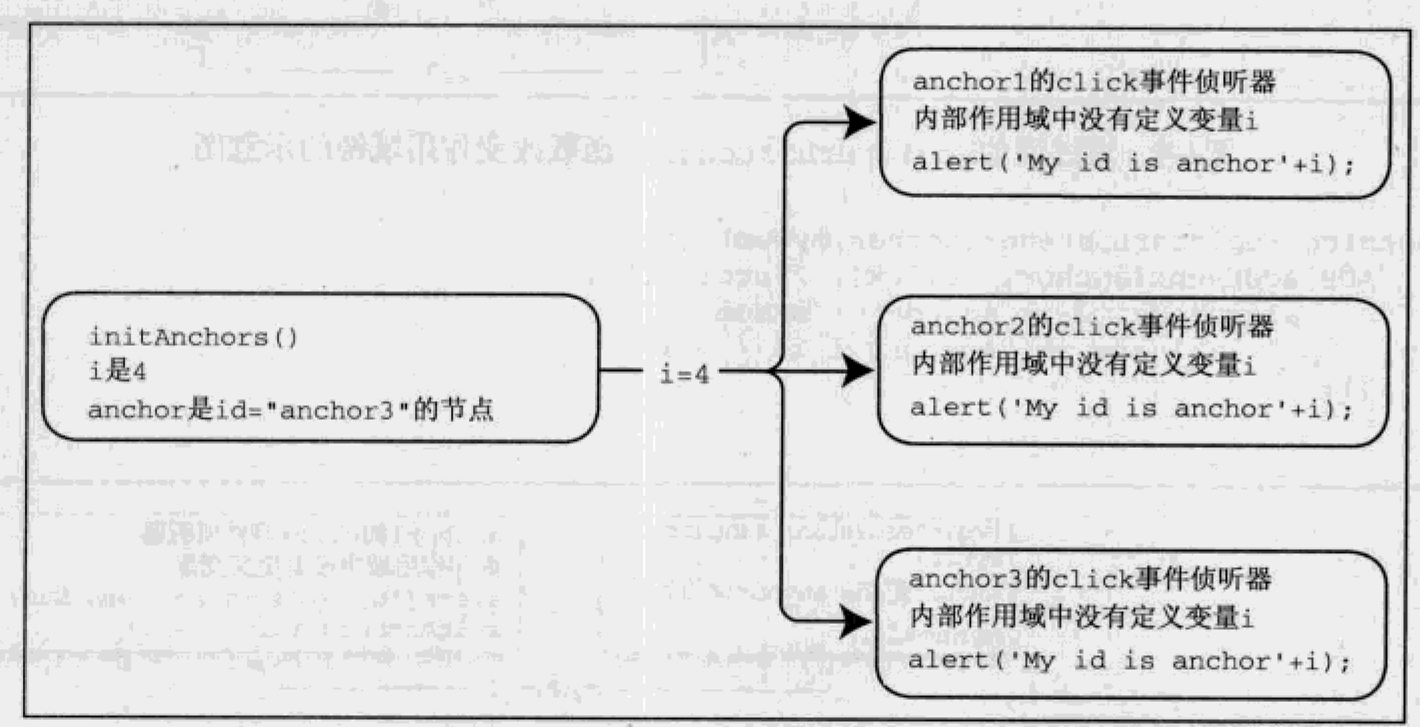



图1-5 单击事件侦听器与initAnchors()方法之间的作用域链示意图

由于在作用域链中定义了额外的函数和变量，正如你在运行chapter1/scopechain/ example2.html时看到的，提示信息中保持了正确的值。因为click事件侦听器现在的外部作用域变成了registerListener()函数，该函数在其每个实例[①]的内部作用域中都为i维护了一个唯一的值，这一过程如图1-6所示。

同样地，如果将registerListener()替换成下面的函数，如你在运行chapter1/scopechain/example3.html时所见，提示的信息将是My id is anchorX and initAnchors i is 4。同理，这也是因为i的值不是在registerListener()函数及其包含的匿名事件侦听器中定义

① 每次调用registerListener()函数都会生成该函数的一个副本，以维护正确的变量作用域。——译者注

www.TopSage.com

的，因此最终还是要从initAnchors()函数的作用域中取得i的值，这一过程如图1-7所示。

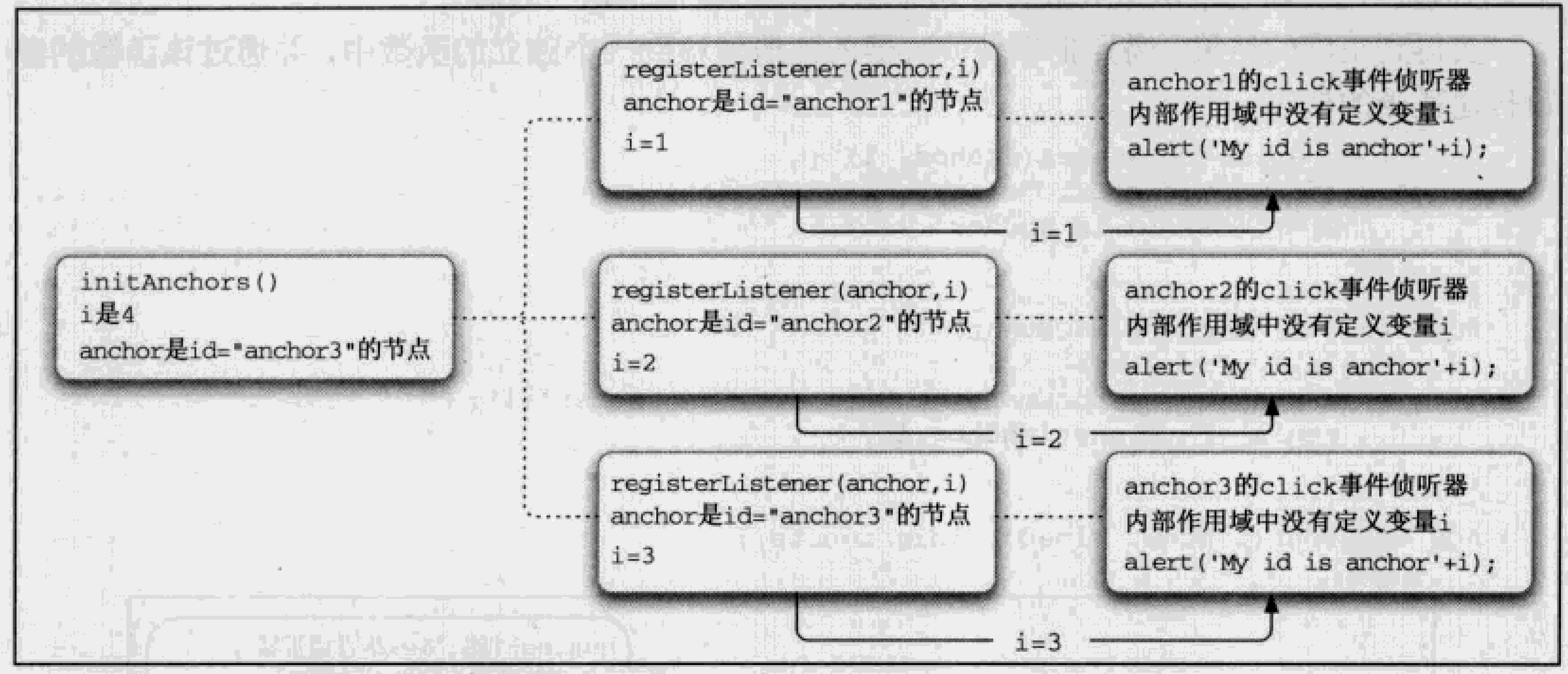



图1-6 通过添加registerListener()函数改变作用域链的示意图

```
function registerListener(anchor,myNum) {
    ADS.addEvent(anchor, 'click', function() {
        alert('My id is anchor' + myNum
            + ' and initAnchors i is ' + i);
    });
}
```

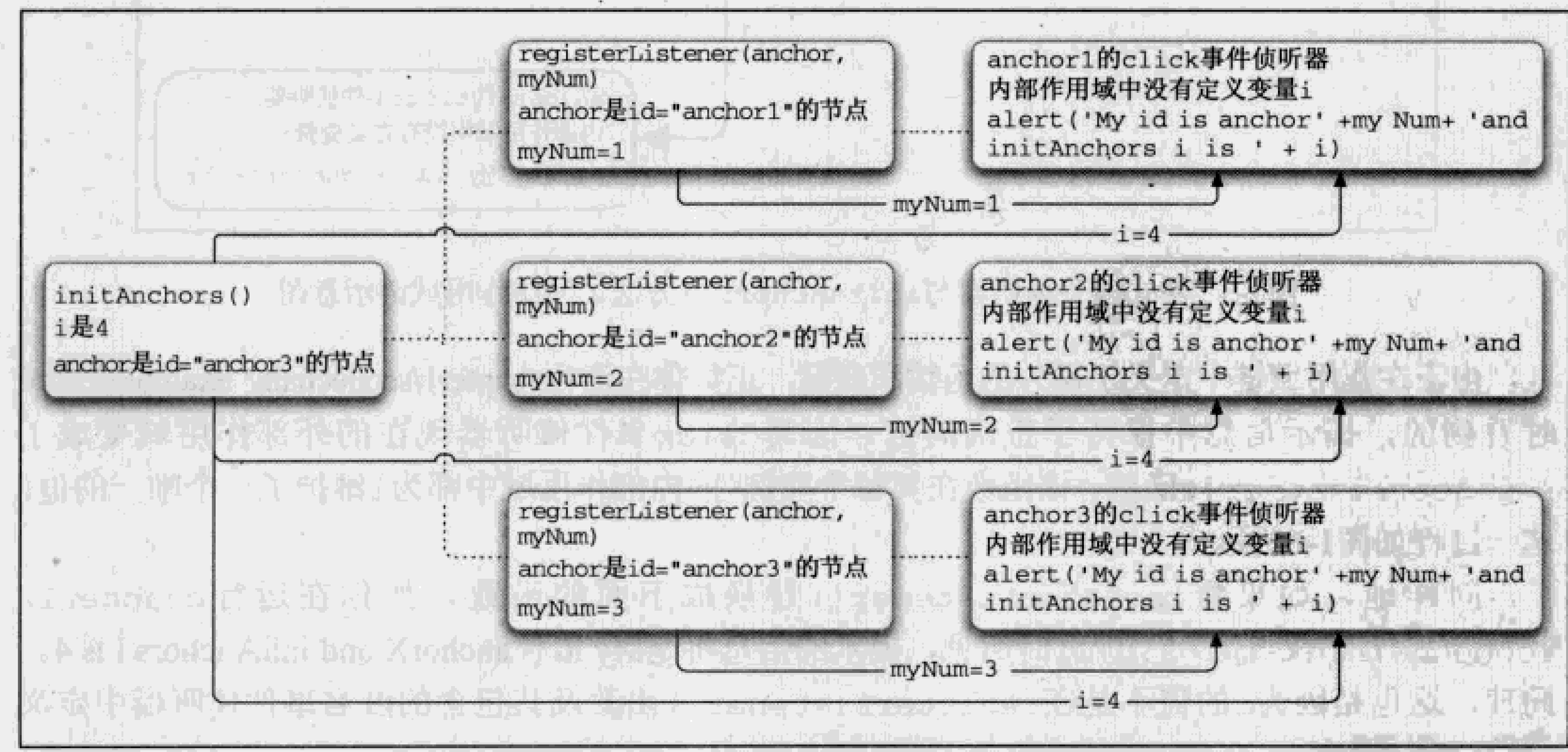



图1-7 添加额外的registerListener()函数并修改其输入变量后改变作用域链的示意图

www.TopSage.com

这种类似的情况在本书中并不少见，即在将方法动态赋值给对象时，必须使用额外[①]的函数才能维护适当的变量作用域。

### 1.3.8 迭代对象

在编写脚本时经常用到迭代，比如使用for循环对数组中的所有元素进行迭代：

```
var list = [1,2,3,4];
for( i=0 ; i<list.length ; i++ ) {
    alert(list[i]);
}
```

使用这种递增控制法或许效果还不错。另一种可供选择的迭代方法是使用for循环遍历位于（in）list中的每个属性：

```
var list = [1,2,3,4];
for( i in list ) {
    alert(list[i]);
}
```

此时，得到的是与使用前一种迭代方法相同的结果，因为list是一个Array对象。

但是，当使用for(i in item)方法操纵类似数组而又不是数组的对象时一定要格外小心。例如，由getElementsByTagName()返回的NamedNodeMap对象与数组类似而且具有length属性，也就是说可以像对一个常规数组那样对它进行迭代：

```
var all = document.body.getElementsByTagName('*');
for( i=0 ; i<all.length ; i++ ) {
    // 对all[i]元素进行某些操作
}
```

可是，如果使用下面的迭代方法，那么循环中还将包含NamedNodeMap对象的附加方法：

```
var all = document.body.getElementsByTagName('*');
for( i in all) {
    // 对all[i]元素进行某些操作
}
```

在这次的迭代过程中，i的值也会分别等于length、item和namedItem，而这很可能会导致代码中出现意外错误。在某些情况下，可以使用对象的hasOwnProperty()方法来避免这个问题。

如果对象的属性或方法是非继承的，那么hasOwnProperty()方法返回true。即这里的检查不涉及从其他对象继承的属性和方法，只会检查在特定对象自身中直接创建的属性，比如分配给数组的元素。因此，如果在for循环中使用了这种检查，那么循环将会跳过length这样的属性，因为length不是数组all的直系属性，而是从派生数组all的NamedNodeMap对象中继承的属性：

```
var all = document.body.getElementsByTagName('*');
for( i in all) {
     if(!all.hasOwnProperty(i)) { continue; }
    // 对all[i]元素进行某些操作
}
```

① 之所以要使用额外的函数，主要是因为无法给匿名函数传递参数直接维护内部作用域。——译者注

www.TopSage.com

> 我们将在第3章研究两个规范之间的相互关系，进而学习有关继承的更多内容。

### 1.3.9 函数的调用和引用（不带括号）

最后一个你在本书中随处可见的JavaScritp陷阱，涉及对函数的引用和调用。JavaScript中的函数有许多独到之处，但这里只要求你理解以下两种方式的区别，一是调用函数并将它的返回结果赋给一个值：

```
var foo = exampleFunction();
```

另一个是将函数的引用赋给一个值：

```
var foo = exampleFunction;
```

发现不同了吗？区别在于结尾处执行函数的圆括号——它对结果的影响极大。没有圆括号，赋值给变量的是函数本身，而非结果。这在需要将函数作为数据赋值或者作为参数传递给其他方法（如window的load事件侦听器）时是非常重要的：

```
function loadPage() {
    // 载入脚本
}
// 没有圆括号
ADS.addEvent(window, 'load', loadPage);
```

如果像下面代码中所示的在函数的结尾添加了圆括号，那么就会把loadPage()函数的执行结果，而不是loadPage()函数本身赋值给window的load事件侦听器：

```
// 带圆括号不会得到想要的结果
ADS.addEvent(window, 'load', loadPage());
```

## 1.4 实例：WYSIWYG JavaScript翻转图

在深入讨论第2章的JavaScript对象模型和第3章的DOM之前，我们先来看一个实际的例子，通过这个例子你将切实体会到遵循本章介绍的最佳实践和忠告，会节省很多项目开发时间。

> 在你们给我大发邮件抱怨下面JavaScript翻转图的例子应该使用CSS实现（这一点我完全赞同）之前，我要解释一下。之所以选择这个例子，是因为大多数（差不多是全部）Web开发者对它都很熟悉，而且这个例子也非常适合说明本章的诸多要点。

除非这是你第一次尝试进行Web设计和开发，否则你很可能对在WYSIWYG编辑器中使用JavaScript实现多状态按钮时，必需而又混乱的鼠标事件比较熟悉。一般来说，WYSIWYG都会生成如下所示的代码：

```
<a href="http://example.com"
    onmouseover="swapImage(...)"
    onmouseout="swapImage(...)"
    onmousedown="swapImage(...)"
    onmouseup="swapImage(...)">
```

www.TopSage.com

```
<img src="images/button.gif" width="10" height="40" ➡
border="0" alt="Click Me!"></a>
```

问题是这样一来在一个包含许多锚的大型文档中，标记中的主要部分都将是嵌入的JavaScript代码，并且最终使得结构化标记被“淹没”在嵌入式的代码中。

作为一个例子，可以看一下图1-8所示的页面。

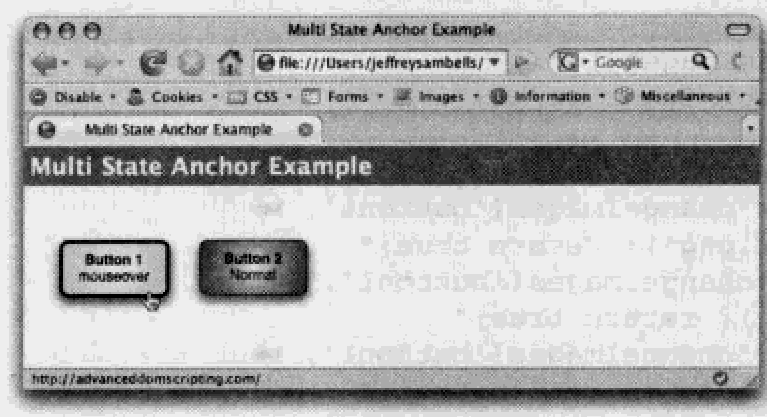



图1-8 带两个翻转按钮的简单网页

如果使用Adobe ImageReady生成图1-8所示的页面，那么页面中可能会包含如下标记和代码：

```
<html>
<head>
<title>ImageReady Example</title>
    <link rel="stylesheet" href="style.css" ➡
type="text/css" media="screen" />

    <script type="text/javascript">
    <!--

    function newImage(arg) {
        if (document.images) {
            rslt = new Image();
            rslt.src = arg;

            return rslt;
        }
    }

    function changeImages() {
        if (document.images && (preloadFlag == true)) {
            for (var i=0; i<changeImages.arguments.length; i+=2) {
                document[changeImages.arguments[i]].src =
                    changeImages.arguments[i+1];
            }
        }
    }

    var preloadFlag = false;
    function preloadImages() {
        if (document.images) {
            button1_over = newImage("images/button1-over.png");
            button1_down = newImage("images/button1-down.png");
            button2_over = newImage("images/button2-over.png");
```

www.TopSage.com

```
            button2_down = newImage("images/button2-down.png");
            preloadFlag = true;
        }
    }

    // -->
    </script>

</head>
<body onload="preloadImages();">
<h1>Multi State Anchor Example</h1>
    <div>
    <a href="http://advanceddomscripting.com"
        onmouseover="changeImages('button1', ➥
'images/button1-over.png'); return true;"
        onmouseout="changeImages('button1', ➥
'images/button1.png'); return true;"
        onmousedown="changeImages('button1', ➥
'images/button1-down.png'); return true;"
        onmouseup="changeImages('button1', ➥
'images/button1-over.png'); return true;">
        <img name="button1" src="images/button1.png"
            width="113" height="67" alt="button1"></a>
    <a href="http://advanceddomscripting.com"
        onmouseover="changeImages('button2', ➥
'images/button2-over.png'); return true;"
        onmouseout="changeImages('button2', ➥
'images/button2.png'); return true;"
        onmousedown="changeImages('button2', ➥
'images/button2-down.png'); return true;"
        onmouseup="changeImages('button2', ➥
'images/button2-over.png'); return true;">
        <img name="button2" src="images/button2.png" ➥
width="113" height="67" alt="button2"></a>
    </div>
</body>
</html>
```

类似地，使用Adobe（原Macromedia）Dreamweaver将生成如下代码：

```
<html>
<head>
<title>Dreamweaver Example</title>
    <link rel="stylesheet" href="style.css" ➥
type="text/css" media="screen" />

    <script type="text/JavaScript">
    <!--

    function MM_preloadImages() { //v3.0
        var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
        var i,j=d.MM_p.length,a=MM_preloadImages.arguments;
        for(i=0; i<a.length; i++)
        if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image;
        d.MM_p[j++].src=a[i];}}
    }
```

www.TopSage.com

```
    function MM_swapImgRestore() { //v3.0
        var i,x,a=document.MM_sr;
        for(i=0;a&&i<a.length&&(x=a[i])&&x.oSrc;i++) x.src=x.oSrc;
    }

    function MM_findObj(n, d) { //v4.01
      var p,i,x; if(!d) d=document;
      if((p=n.indexOf("?"))>0&&parent.frames.length) {
          d=parent.frames[n.substring(p+1)].document;
          n=n.substring(0,p);
        }
        if(!(x=d[n])&&d.all) x=d.all[n];
        for (i=0;!x&&i<d.forms.length;i++) x=d.forms[i][n];
        for(i=0;!x&&d.layers&&i<d.layers.length;i++)
            x=MM_findObj(n,d.layers[i].document);
        if(!x && d.getElementById) x=d.getElementById(n); return x;
    }

    function MM_swapImage() { //v3.0
      var i,j=0,x,a=MM_swapImage.arguments; document.MM_sr=new Array;
      for(i=0;i<(a.length-2);i+=3)
        if ((x=MM_findObj(a[i]))!=null){document.MM_sr[j++]=x;
        if(!x.oSrc) x.oSrc=x.src; x.src=a[i+2];}
    }

    //-->
    </script>

</head>
<body onload="MM_preloadImages('images/button1-over.png')">
<h1>Multi State Anchor Example</h1>
<div>
    <a href="http://advanceddomscripting.com"
        onmouseover="MM_swapImage('button1','', ➥
'images/button1-over.png',1)"
        onmouseout="MM_swapImgRestore()"
        onmousedown="MM_swapImage('button1','', ➥
'images/button1-down.png',1)"
        onmouseup="MM_swapImgRestore()">
        <img id="button1" src="images/button1.png"
            width="113" height="67" alt="Button1"></a>
    <a href="http://advanceddomscripting.com"
        onmouseover="MM_swapImage('button2','', ➥
'images/button2-over.png',1)"
        onmouseout="MM_swapImgRestore()"
        onmousedown="MM_swapImage('button2','', ➥
'images/button2-down.png',1)"
        onmouseup="MM_swapImgRestore()">
        <img id="button2" src="images/button2.png" ➥
width="113" height="67" alt="Button2"></a>
</div>
</body>
</html>
```

这个页面中包含两个链接的图像（请参考chapter1/rollovers/中的源代码）；但从前面代码中可以看出，其中的JavaScript代码远远超过了其他标记。可以确定的是，上面的每个例子都使用了一

www.TopSage.com

些可重用的函数，而这些函数都可以轻易地被提取到一个外部源文件中。但对于ImageReady的例子来说，有些变量被硬编码到了`preload()`函数中，因此我们无法在不让JavaScript源文件依赖特殊页面的条件下将该函数从此网页中提取出来。这无论对用户还是对开发者都不是一种非常友好的情形。更好的方案是让页面标记中根本不包含任何嵌入的脚本代码，而是在文档头部包含必需的JavaScript源文件。然后，由这个JavaScript源文件通过`load`事件来按照需要增强文档的结构：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Multi State Anchor Example</title>
    <link rel="stylesheet" href="style.css" type="text/css" media="screen" />
    <script src="../../ADS-final-verbose.js" type="text/javascript"></script>
    <script src="unobtrusiveRollovers.js"
        type="text/javascript"></script>
</head>
<body>
<h1>Multi State Anchor Example</h1>
<div>
    <a href="http://advanceddomscripting.com" ➥
class="multiStateAnchor">
        <img src="images/button1.png" width="113" ➥
height="67" alt="Button 1" />
    </a>
    <a href="http://advanceddomscripting.com" ➥
class="multiStateAnchor">
        <img src="images/button2.png" width="113" ➥
height="67" alt="Button 2" />
    </a>
</div>
</body>
</html>
```

这个方案使用了更小也更容易理解的文档。而且，如果你遵循下列两条规则，它也能以完全相同的方式实现与前两个WYSIWYG版本的例子相同的功能：

(1) 在图像文件名中添加`-over`和`-down`，用以表示`mouseover`和`mousedown`这两个状态。

(2) 在每个包含要翻转的单个图像的锚中添加`class="multiStateAnchor"`。

如果遵循这些规则，那么就可以使用下面不唐突而且可重用的`unobtrusiveRollovers.js`脚本文件，加上你的ADS库来修改页面中同样多的锚，从而实现与WYSIWYG版本的例子中相同的效果：

```
function registerMultiStateAnchorListeners ➥
(anchor,anchorImage,path,extension) {
    // 载入鼠标悬停状态的图像
    // 载入过程与其余的脚本
    // 异步进行
    var imageMouseOver = new Image()
    imageMouseOver.src = path + '-over' + extension;

    // 当鼠标悬停时变换图像的源文件
    ADS.addEvent(anchor, 'mouseover', function (W3CEvent) {
```

www.TopSage.com

```
        anchorImage.src = imageMouseOver.src;
    });

    // 当鼠标移出时将图像变换为原始文件
    ADS.addEvent(anchor, 'mouseout', function (W3CEvent) {
        anchorImage.src = path + extension;
    });

    // 载入鼠标按下时的图像
    var imageMouseDown = new Image()
    imageMouseDown.src = path + '-down' + extension;

    // 当鼠标按下时将图像变换为按下状态的源文件
    ADS.addEvent(anchor, 'mousedown', function (W3CEvent) {
        anchorImage.src = imageMouseDown.src;
    });

    // 当鼠标放开时将图像变换为原始文件
    ADS.addEvent(anchor, 'mouseup', function (W3CEvent) {
        anchorImage.src = path + extension;
    });
}

function initMultiStateAnchors(W3CEvent) {

    // 查找页面中所有的锚
    var anchors = ADS.getElementsByClassName('multiStateAnchor', ➥
'a', document);

    // 循环遍历列表中的所有锚元素
    for (var i=0; i<anchors.length ; i++) {

        // 找到锚中的第一个子图像元素
        var anchorImage = anchors[i].getElementsByTagName('img')[0];

        if(anchorImage) {

            // 如果存在图像元素，解析其源路径
            var extensionIndex = anchorImage.src.lastIndexOf('.');
            var path= anchorImage.src.substr(0, extensionIndex);
            var extension= anchorImage.src.substring(
                extensionIndex,
                anchorImage.src.length
            );

            // 添加各种鼠标处理程序
            // 同时预先加载图像
            registerMultiStateAnchorListeners(
                anchors[i],
                anchorImage,
                path,
                extension
            );
        }
    }
}
// 当文档载入完成时修改具有特定标记的锚
```

www.TopSage.com

```
ADS.addEvent(window,'load',initMultiStateAnchors);
```

自下而上地分析前面的代码，可以看出该文件所做的几件事。首先，脚本使用initMultiStateAnchors()函数和本章前面讨论的ADS.addEvent()函数注册了window的load事件侦听器：

```
ADS.addEvent(window,'load',initMultiStateAnchors);
```

其中的initMultiStateAnchors()函数负责查找带有multiStateAnchor类的所有链接，然后在每个链接上注册相应的鼠标事件侦听器。为此，initMultiStateAnchors()函数使用本章前面讨论的ADS.getElementsByClassName()函数，相对于文档取得了<a>标签的class属性中带有multiStateAnchor类名的所有锚元素的列表：

```
var anchors = ADS.getElementsByClassName('multiStateAnchor', ➡
'a', document);
```

然后，循环遍历这个列表并解析每个锚元素中的第一个<img>标签的src属性就很简单了：

```
// 循环遍历列表中的所有锚元素
for (var i=0; i<anchors.length ; i++) {

    // 找到锚中的第一个子图像元素
    var anchorImage = anchors[i].getElementsByTagName('img')[0];

    if(anchorImage) {

        // 如果存在图像元素，解析其源路径
        var extensionIndex = anchorImage.src.lastIndexOf('.');
        var path= anchorImage.src.substr(0, extensionIndex);
        var extension= anchorImage.src.substring(
            extensionIndex,
            anchorImage.src.length
        );

        // 添加各种鼠标处理程序并预先加载图像
        registerMultiStateAnchorListeners(
            anchors[i],
            anchorImage,
            path,
            extension
        );
    }
}
```

这里，我们使用了由ADS.getElementsByTagName('img')返回的数组中的第一个元素，该元素引用的是包含在相应锚元素中的图像元素。当然，可以通过对以上源文件进行简单的修改来影响包含在锚元素中的多个图像元素，或者其他关联的标签组——这就取决于你的需要了。

最后，预先加载了mousedown和mouseover状态的图像，而且使用“第三方”函数registerMultiStateAnchorListeners()为每个锚注册了事件侦听器，并维护了每个锚的适当的作用域和值：

```
function registerMultiStateAnchorListeners ➡
(anchor,anchorImage,path,extension) {
```

www.TopSage.com

```
        var imageMouseOver = new Image()
        imageMouseOver.src = path + '-over' + extension;

        ADS.addEvent(anchor, 'mouseover', function (W3CEvent) {
            anchorImage.src = imageMouseOver.src;
        });

        ……省略的代码……
    }
```

运行以上组合的脚本也会得到与WYSIWYG编辑器生成的脚本相同的行为，但只要能够遵守我们在开始时设置的命名约定，那么通过手工编码的方案就会更容易维护，而且百分之百可以重用。假如你想添加更多带有翻转效果的链接，只要像往常一样添加带有适当类的链接，并确保你的图像在服务器上可用就行了。

## 1.5 小结

当你在Web应用中集成自己的DOM脚本和行为增强时，在所有的代码中遵循公认的最佳实践，可以最大限度地节省时间、金钱和精力。最佳实践也可以帮你避免常见的问题，比如难以访问和功能残缺的站点以及超载的源代码等。许多先行者在你之前就遇到了类似的问题，因此从他们的范例中汲取营养，可以使你集中精力进行创造而不是承受挫败。

本章也揭示了在测试和开发网站的过程中，虽然WYSIWYG编辑器、嵌入的JavaScript、浏览器版本检测以及基于脚本生成内容看起来好像是简单直观，而且容易上手，但是你不应该被这些表面现象所迷惑，更不应该让肤浅的做事方式取代正确的做事方式。有时候，遵循最佳实践和标准需要你在短期内投入更多的努力。然而，你的付出终将得到回报，因为正确的做事方式始终会有生命力，而错误的方式则一定会让你在某些时候尝到苦头——只是时间早晚的问题。

虽然使用像`addEvent()`和`getElementsByClass()`这样的公共函数会使你的代码更容易理解，但不要仅仅为了使用库而使用库，更不要让自己因此而失去自我。理解为什么要使用每个函数和理解每个函数的工作原理是同等重要的。有关使用和滥用JavaScript库的问题将在第9章中详细讨论。

本章也介绍了在使用JavaScript这门语言的过程可能会遇到的许多常见的陷阱。大概你也发现了JavaScript集强大和难解于一身的特点，虽然匿名函数和作用域链的概念有些费解，但只要掌握了使用它们的窍门，那么多数问题都会迎刃而解。

在本书其余的章节中，我会经常提到本章介绍的概念，因此你也会进一步理解它们。你还会向在本章创建的ADS库中添加更多的方法。接下来，让我们在第2章深入到JavaScript的内部，并且学习如何创建你自己的对象。

www.TopSage.com

第2章

# 创建可重用的对象

JavaScript中一切都是对象。对象是所有一切的基础，如果你还不熟悉对象，别担心，我们很快就会介绍到。本书的目标不是为你提供JavaScript或者DOM代码的参考，而是要保证你理解我要介绍的诸多例子和概念。本章我们就花点时间讨论一下对象。透彻地理解对象（特别是JavaScript中的对象）的工作过程和基本原理，将对你创建节省时间和金钱的一般可重用代码大有裨益。

在本章中，我将向你介绍操纵对象时需要记住的如下重要内容：

- 什么是对象，如何构建对象
- 静态、公有、私有以及特权成员之间的区别
- `this`引用什么
- 有关作用域链的更多内容
- 简单介绍对象的环境

在本章最后，我们要构建一个自定义的调试日志对象，在将上述所有内容付诸实践的同时，向你展示通过适当的成员定义为你的对象创建一个清晰、公有的API的过程。

## 2.1 对象中包含什么

或许你还不了解对象，但可能已经在使用它了。对象，简而言之，就是包含一组变量（称为属性）和函数（称为方法）的集合的实例。对象通常由类派生而来，而类中定义了对象拥有的属性和方法。如果你的脚本中都是对象之间的交互操作，那么就可以称之为OO（Object Oriented，面向对象）的脚本。特别要说明的是，JavaScript是一种原型式（prototype-style）的OO语言，没有类的概念，所有一切都派生自现有对象的一个副本。在JavaScript中，从函数到字符串实际上都是对象，而这也正是JavaScript既强大又令人费解的根源所在。JavaScript中的大多数对象可以分为如下两种类型。

- `Function`对象，例如`alert()`函数，可以使用参数来改变这类对象的功能：

```
alert('argument');
```

- `Object`对象，例如下面代码片断中的`obj`，这类对象不能像函数那样被调用，而且具有固定的功能——除非它们包含额外的`Function`对象（这一点我们将在2.1.2节中介绍）：

www.TopSage.com

```
var obj = new Object();
obj('argument'); // 会出错，因为obj不是Function对象
```

Function对象也可以分成如下两个子类别。

- Function实例，例如alert()，可以使用参数来调用。
- 作为构造函数的Function，必须通过new操作符进行实例化。

为提高效率，JavaScript也提供了下列内置对象。

- Object是通用基础对象，可以使用它来创建简单的静态对象。
- Function是被所有使用参数的对象复制的对象，也是在脚本中定义函数时所创建的对象。
- Array是一种特殊的属性和方法的集合，比如使用其length属性可以通过循环迭代操纵这类对象，而且使用方括号也可以访问它们的属性。
- String、Boolean和Number则分别用于表示字符串、布尔值以及数字。
- Math、Date、RegExp以及其他内置对象，也都有各自独特的用途，但在这里我们就不一一介绍了。

所有内置对象都可以通过new关键字或者其他特殊的语法来创建，例如function关键字用于创建Function对象，花括号（{}）是Object的简写形式，而方括号（[]）则是Array的简写形式。这些对象的共同特点就是提供一组属性和方法，以便根据每个对象的设计用途通过不同方式来操纵这些对象。

> 当你在本章后面创建自己的对象时，我们再讨论new操作符和实例化这两个概念。

### 2.1.1 继承

对象继承是面向对象编程的一个重要组成部分。当创建自己的对象时，你可以扩展或者继承现有对象的属性和方法。继承为重用现有对象的功能提供了便利的途径，这样你就可以把精力完全集中于新的改进的代码中。

与传统的基于类的面向对象语言不同，JavaScript中没有从一个类扩展出另一个类的底层类结构。在JavaScript中，继承是通过简单地从一个对象原型向另一个对象原型复制方法而实现的，但最终的思想都是相同的。

```
// 创建一个person对象的实例
var person = {};
person.getName = function() { ... };
person.getAge = function() { ... };

// 创建一个employee对象的实例
var employee = {};
employee.getTitle = function() { ... };
employee.getSalary = function() { ... };

// 从person对象中继承方法
employee.getName = person.getName;
employee.getAge = person.getAge;
```

www.TopSage.com

如图2-1所示，高层的对象从低层的对象中继承了所有属性和方法。

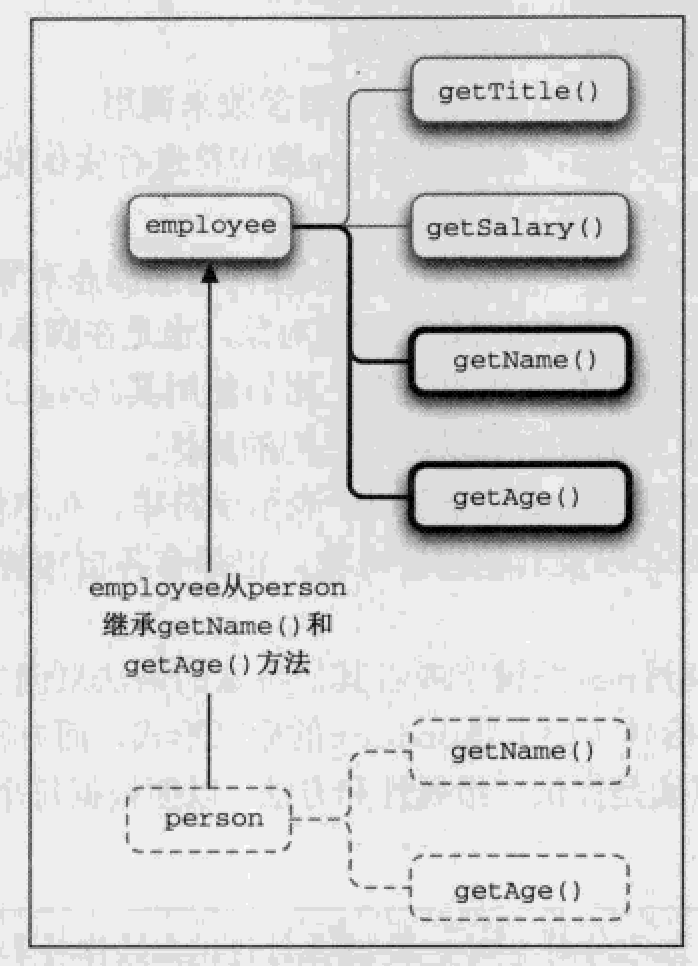



图2-1　继承的一般思想

要了解有关继承的具体细节和一些与经典继承方法非常类似的奇特方法，我建议你看一看Douglas Crokford在"Classical Inheritance in JavaScript"（http://www.crockford.com/javascript/inheritance.html）一文中的解说。

## 2.1.2　理解对象成员

虽然你已经熟悉了一些简单的老函数如alert()，而且也知道函数是为避免冗余代码而构建的一种简单的可重用的容器。但是，你对也同样熟悉的对象、属性和方法，却不一定真正理解。事实上，当你使用document的body属性：

```
document.body
```

或者document的getElementById()方法时：

```
document.getElementById('example');
```

你是在访问document对象的一个成员。以上属性和方法都被称为对象的成员，因为它们都从属于父对象，即上面例子中的document。其中，body成员是一个属性，它只是引用了一个值；而getElementById()成员是一个方法，因为它接受参数并且可以操纵对象的内部状态。

www.TopSage.com

实际上，属性自身只是Object对象或者另一个扩展自Object的对象（例如String或Number）的实例。同样，方法也扩展自Object对象，但因为它们接受参数，所以它们是Function对象的实例，而且方法也可以返回值。

通过运行测试文档chapter2/types/types.html，其中load事件中包含的下列代码会告诉你body和getElementById的对象类型：

```
ADS.addEvent(window,'load', function() {
    alert('document.body is a: ' + document.body);
});
```

以上代码在Firefox和Opera中将显示document.body是一个[object HTMLBodyElement]对象，如图2-2所示。

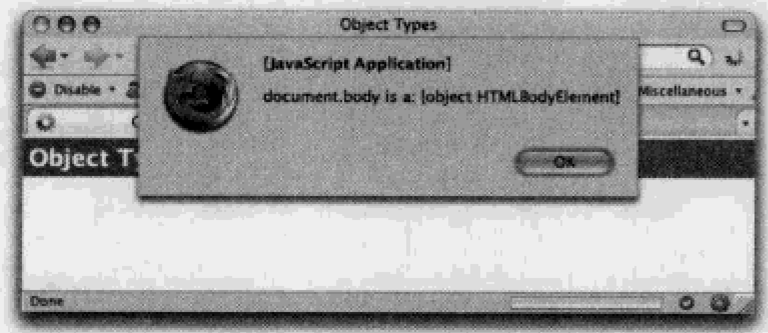



图2-2 在Firefox中警告框显示document.body是一个[object HTMLBodyElement]

同样的代码在Safari中将显示一个[object BODY]对象，如图2-3所示。

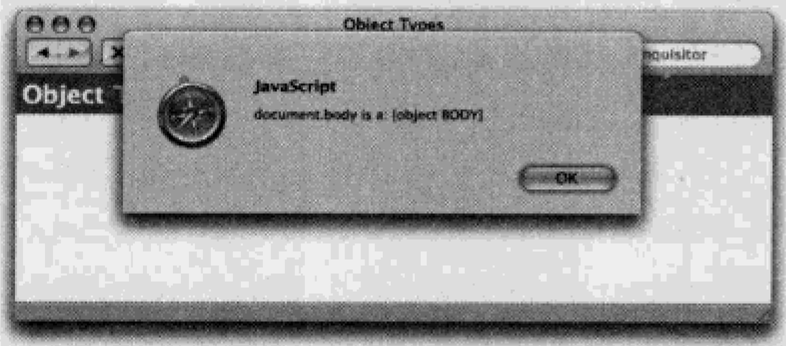



图2-3 在Safari中警告框显示document.body是一个[object BODY]

而在Microsoft IE中，则会显示[object]，如图2-4所示。

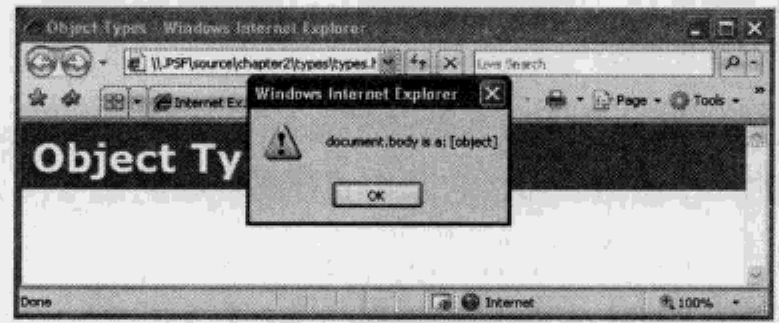



图2-4 在Microsoft IE中警告框显示document.body是一个[object]

www.TopSage.com

以上浏览器之间的差别是由处理DOM对象的不同方式所导致的。如果浏览器遵守W3C DOM核心和DOM HTML规范的命名约定，那么document.body应该是DOM HTMLBodyElement对象的实例。而实际上，Safari用BODY对象表示document.body，而在IE中，只是一个简单的object。Safari和IE各自的对象中虽然也包含了许多HTMLBodyElement的方法，但它们与官方定义的HTMLBodyElement对象仍然不是一回事。

如果查看document.getElementById方法（不带圆括号，否则会执行该方法），那么在所有浏览器中都会显示document.getElementById是一个函数，见图2-5：

```
ADS.addEvent(window,'load',function() {
    alert('document.getElementById is a: ' + document.getElementById);
});
```

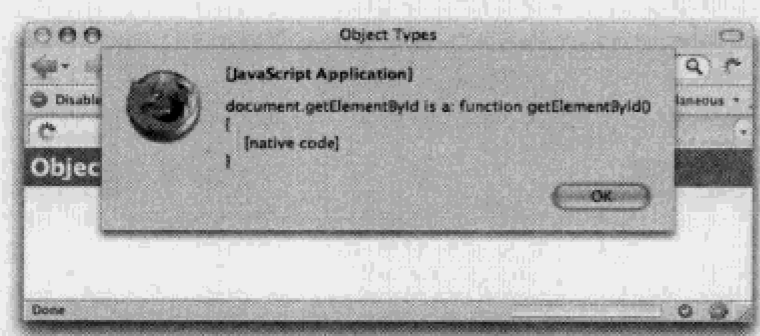



图2-5 警告框显示document.getElementById是一个原生函数

> 你可能会对在前面两个例子中要求报告对象自身，但结果报告的却是对象的类型感到奇怪。事实上，当对象被传递到期望一个字符串参数值的alert()函数中时，Object对象的toString()方法就会被调用，以取便得对象的字符串表示。toString()方法来源于基础的Object对象，该方法会将对象转换成一个字符串表示，这个字符串表示通常就是对象类型的名称。而对于document.body而言，相应的字符串表示是一个简单的短语：[HTMLBodyElement]（或者其他字符串），该短语正是由document.body.toStrinig()方法返回的字符串。如果对象是String或HTMLAnchorElement，那么它们的toString()方法则会返回不同的值，比如String对象中包含的字符串值或者锚元素href属性的内容。

我们将在第3章讨论各种DOM方法和对象。现在，只要知道它们与别的对象一样，都是由Object对象和Function对象构成的就可以了。

### 2.1.3 window 对象中的一切

不知道你有没有意识到，你所写过的许多函数实际上都是window对象的方法。如果在你的脚本中，只包含下面所示的一个函数，那么它和内置函数（如alert()）实际上都是全局window对象的方法：

```
function myFunction(message) { alert(message); }
```

如果上面的函数是你在JavaScript文件最顶层编写的没有被其他对象包含的代码（如chapter2/window/with-without.js文件中所展示的那样），那么通过下面这行代码执行它：

www.TopSage.com

```
myFunction('Without window object');
```

与执行下面这行代码的结果是相同的：

```
window.myFunction('With window object');
```

但是，如果你是在另一个对象中创建的这个函数，那么由于作用域链的关系，情况就会变得复杂一些。要证明这一点，可以进行下面的练习。

**练习：覆盖作用域链中的对象**

在这个简单的练习中，你会看到如何覆盖JavaScript的`alert()`方法，而且在某些情况下，仍然可以通过window对象直接访问到原始的版本。

(1) 创建一个简单的HTML文档并命名为override.html，然后在头部中包含override.js文件：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Override the alert() method</title>
    <link rel="stylesheet" href="style.css"
        type="text/css" media="screen" />
    <script type="text/javascript" src="override.js"></script>
</head>
<body>
    <h1>Override the alert() method</h1>
    <p>Interesting isn't it?</p>
</body>
</html>
```

(2) 创建override.js文件，其中包含下面的override()函数：

```
function override() {
    // 覆盖alert函数
    var alert = function(message) {
        window.alert('overridden:' + message);
    };
    alert('alert');

    // 在override()函数的作用域中调用原始的alert()函数
    window.alert('window.alert');
}
override();

// 在window的作用域中调用原始的alert()函数
alert('alert from outside');
```

(3) 在浏览器中载入网页。

> 如果是在IE 7中运行，而且设置了很高的安全级别，那么你可能会注意到与覆盖alert()有关的警告。虽然浏览器对你覆盖JavaScript函数发出了警告，但因为这也正是本例的意图所在，所以单击同意忽略它即可。

www.TopSage.com

在页面载入完成后，先后会看到3个警告框。第1个警告框（如图2-6所示）表明alert()方法已经被覆盖了。

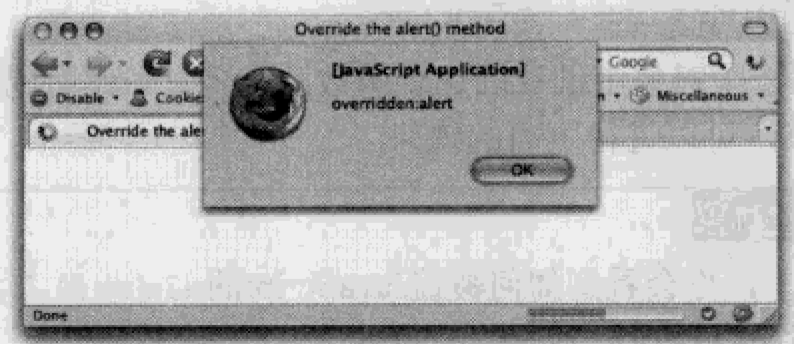



图2-6 警告框显示了覆盖alert()后的结果

第2个警告框（如图2-7所示）显示了仍然可以通过window.alert()访问常规的方法。

图2-7 警告框显示这是在覆盖后访问原始的alert()方法的结果

而第3个警告框（如图2-8所示）显示了在override()函数外部，常规的alert()方法仍然有效。

我们通过声明新的alert()方法已经覆盖了window的alert()方法，但这次覆盖只在override()函数的作用域内部才有效：

```
var alert = function(message) {
    window.alert('overridden:' + message);
};
```

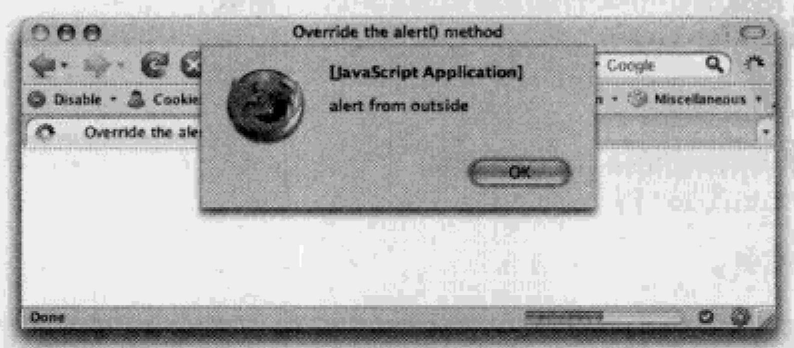



图2-8 警告框显示在override函数外部可以访问原始的alert()方法

如图2-6所示，var关键字将自定义的新alert方法的作用域维持在了override()函数的内部。而在新alert()方法的内部，通过引用window.alert()我们仍然可以访问到原始的、

www.TopSage.com

未被修改的、属于全局window对象的alert()方法，相应的作用域链解析过程如图2-9所示。

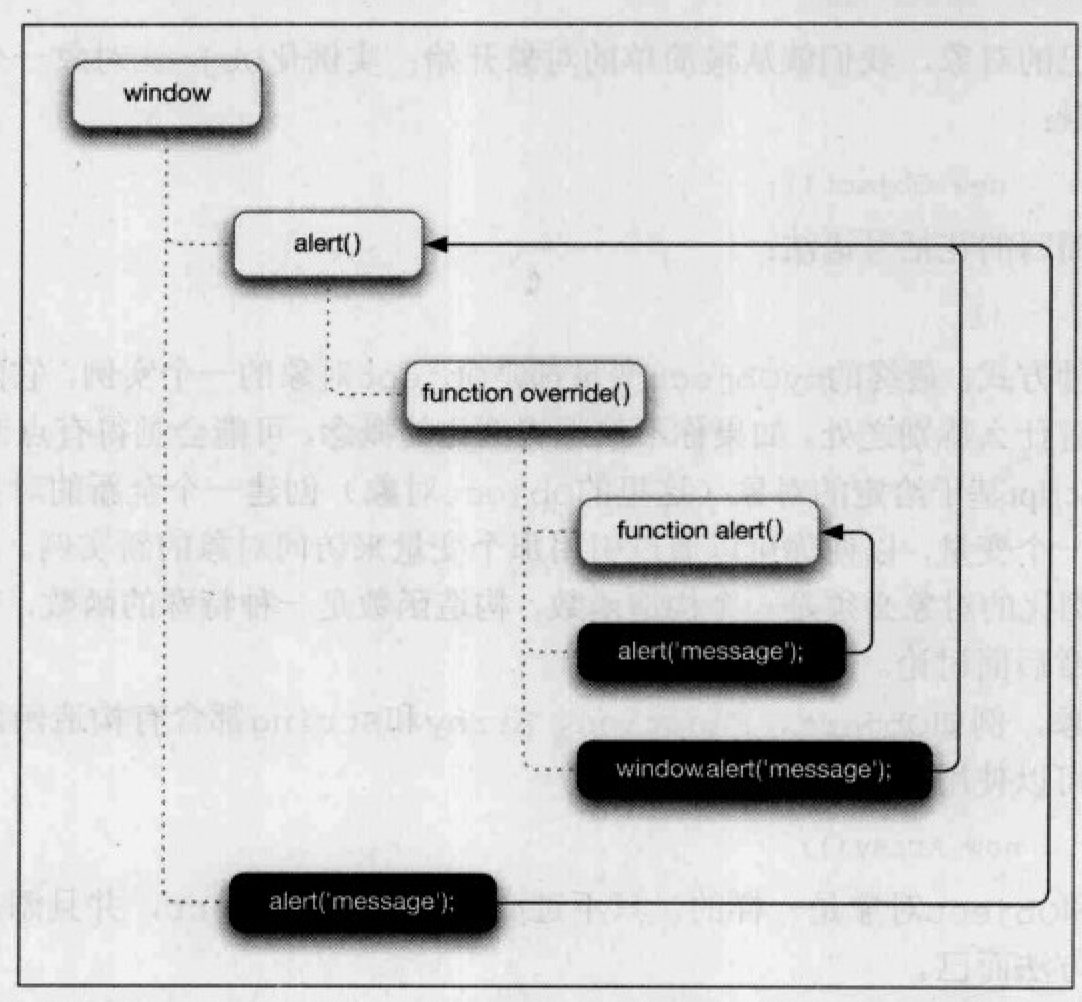



图2-9 override()函数的结构和访问alert()方法的示意图

当然，这个结构只在如图2-9所示的情形下才有效，因为我们是在override()方法内部覆盖并调用alert()方法的。假如在override.js文件中像下面这样没有使用包装函数：

```
// 在window的作用域中覆盖alert函数
var alert = function(message) {
    window.alert('overridden:' + message);
};

alert('test1');
```

那么脚本将会因为过度递归而终止运行。因为这次是在脚本的最顶层覆盖的alert()方法，相当于用你自己的方法替换了window.alert()方法，所以在alert()方法内部调用window.alert()（真正的window.alert()），就会导致无限递归循环。

## 2.1.4 理解作用域和闭包是根本

在函数中覆盖方法和调用方法取决于你的应用程序和对象结构内部的作用域链。在第1章讨论最佳实践和JavaScript陷阱的时候我们已经介绍了作用域链和闭包。如果你还没有阅读第1章，我建议你在继续往下阅读之前先翻回到介绍作用域解析和闭包的那一节，因为下面要跳过作用域和闭包的内容而深入介绍对象的内部结构。

www.TopSage.com

## 2.2 创建你自己的对象

要创建你自己的对象，我们就从最简单的对象开始：实例化Object对象一个新实例，并将它赋值给一个变量：

```
var myObject = new Object();
```

也可以使用简写的花括号语法：

```
var myObject = {};
```

无论通过哪种方式，最终的myObject变量都是Object对象的一个实例，它除了作为一个对象存在之外并没有什么特别之处。如果你不熟悉实例化的概念，可能会觉得有点奇怪。首先，new关键字告诉JavaScript基于给定的对象（这里的Object对象）创建一个全新的对象。然后新创建的实例被赋值给一个变量，以便你可以通过引用那个变量来访问对象的新实例。不过，为了做到这一点，你所实例化的对象必须是一个构造函数，构造函数是一种特殊的函数，有关构造函数的内容我们将在本章后面讨论。

每个核心对象，例如Object、Function、Array和String都含有构造函数。例如，要创建一个新数组，可以使用下面的语法：

```
var myArray = new Array();
```

Array对象和Object对象是一样的，只不过Array扩展自Object，并且添加了额外的属性（如length）和方法而已。

当完成了对象实例化之后，不能再基于新实例使用new操作符创建另外的实例。如果试图运行下列脚本，那么将会得到如图2-10所示的错误提示：

```
var anotherObject = new myObject();
```

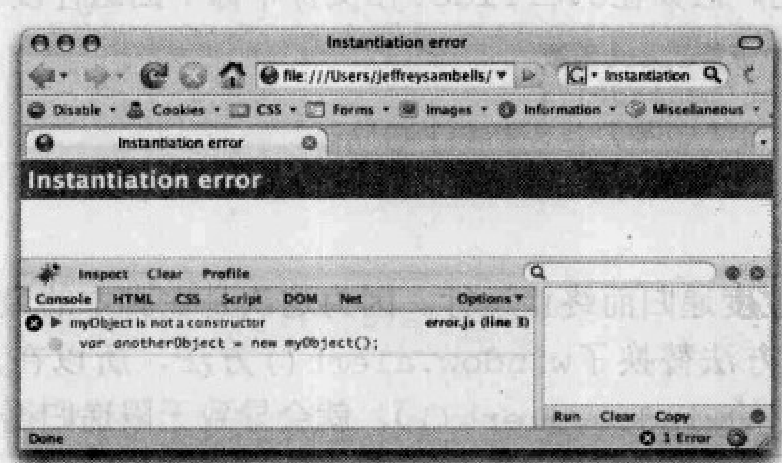



图2-10　错误信息表明myObject()不是一个构造函数

由于myObject已经是一个实例了，所以在这个实例上使用new操作符创建另一个实例的操作是无效的。不过，可以创建原始的Object对象的任意多个新实例。

除非为新创建的对象添加自己的属性或方法，否则默认的Object对象本身用处并不大。为了让你自己的对象适合某些用途，而不仅仅作为一个空的容器存在，就需要向其中添加属性或方

www.TopSage.com

法。同样地，如果想让自己的对象能够被多次复制，那么就需要将其作为一个构造函数来创建，同时对访问其内部的属性和方法设置不同等级的权限，这正是接下来我们要做的。

## 2.2.1 一变多：创建构造函数

Function对象是创建构造函数的起点。使用function关键字可以创建下面的myConstructor函数：

```
function myConstructor(a) {
    // 某些代码
}
```

对上面的代码你可能并不陌生，因为我们已经使用过很多次函数了。而且，大概你也曾看到过另外一种定义函数的语法：

```
var myConstructor = function(a) {
    // 某些代码
}
```

以上两种定义函数的语法从功能上都等价于下面这行代码：

```
var myConstructor = new Function('a','/* 某些代码*/');
```

不过，通过这种方式以new关键字来创建函数会导致性能问题，因此最好还是使用function关键字。

Function对象的特殊之处在于，它的实例也能作为构造器方法，因而可以用来创建函数的新实例。使用前面任何一个myConstructor Function对象，并通过new操作符对其进行实例化都是完全合法的：

```
var myObject = new myConstructor();
```

此时，myConstructor函数就如同基于类的OO语言中的构造器方法。当对象被实例化之后，构造函数会立即执行它所包含的任何代码，比如：

```
function myConstructor(message) {
    alert(message);
    this.myMessage = message;
}
var myObject = new myConstructor('Instantiating myObject!');
```

在实例化myObject之后，浏览器会立即弹出警告框，如图2-11所示。

图2-11 在页面载入期间弹出警告框说明对象的构造函数在实例化完成后会立即被执行

www.TopSage.com

在这个例子中，传递的message参数同时也使用this关键字赋值给了myObject的myMessage属性（现在，我只会告诉你this引用的是myObject实例，而在本章后面我还会更详细地解释this关键字）。

也就是说，通过将message参数赋值给this.myMessage，使myObject拥有了一个可以随时访问的名为myMessage的属性。要从该实例中取得message的值，直接访问其myMessage属性即可：

```
var message = myObject.myMessage;
```

对你而言，关键是要记住myMessage属性只在被实例化的myConstructor的实例中可用，而在myConstructor函数自身中是无效的。

### 2.2.2 添加静态方法

在讨论使用prototype属性添加公有方法之前，我想先指出一个人们经常会误解的问题。当你在阅读和研究其他JavaScript的例子时，可能见过类似下面这样的代码：

```
// 创建一个Object对象的实例
var myObject = new Object(); // 或 var example = {};

// 添加一个属性
myObject.name = 'Jeff';

// 添加一个方法
myObject.alertName = function() {
    alert(this.name);
}

// 执行添加的方法
myObject.alertName();
```

在这里，通过直接在myObject对象实例上使用点号操作符，把name属性和alertName()方法作为静态成员添加到了对象实例中。而这正是容易导致误解的地方。因为这里的静态成员只存在于对象的一个具体实例而不存在于构造函数中。虽然对这个实例而言，相应的代码是合法的而且也可以正常运行，但也仅仅是对Object的这个特定实例myObject能够正常运行。

如果以Function对象为起点，也存在同样的问题，只不过此时的对象实例同时也是一个构造函数：

```
// 创建一个Function对象的实例
var myConstructor = function() {
    // 某些代码
}

// 添加一个静态属性
myConstructor.name = 'Jeff';

// 添加一个方法
myConstructor.alertName = function() {
    alert(this.name);
}
```

www.TopSage.com

```
// 执行添加的方法
myConstructor.alertName();
```

同样，这些代码也能够正常运行，因为myConstructor既是一个实例也是一个构造函数，但是name和alertName成员却同样不会应用到myConstructor的任何新实例中。如果创建myConstructor的一个实例，然后像下面这样尝试访问以上成员：

```
var anotherExample = new myConstructor();
anotherExample.alertName();
```

将会导致下面的错误：

```
TypeError: anotherExample.alertName is not a function
```

理解实例与构造函数之间的区别有助于消除许多已有的问题。

### 2.2.3 向原型中添加公有方法

如果想在实例化新对象时使其包含公有方法，则需要修改构造函数的原型（prototype）。这里所说的原型，指的不是Prototype这个JavaScript框架（http://prototypejs.org），而是指对象的prototype属性。prototype属性是用来定义对象自身内部结构的一个特殊成员。它与其他传统的面向对象语言中的类相似，但又不同于类。如果你不熟悉面向对象的编程语言和（或）原型架构，那么你可以把对象的原型想象为对象的蓝图，而这个蓝图一旦被修改，则会立即改变基于它派生的对象和实例。

当修改一个对象的原型时，任何继承该对象的对象和该对象已经存在的所有实例都会立即继承同样的变化。根据用法不同，这一特性既强大也可能会导致问题，因此当你修改已有的但不是你的对象的原型时一定要谨慎从事。

要在myConstructor的新实例中包含公有方法，只需使用点号操作符向它的原型添加方法即可：

```
// 创建构造函数
function myConstructor(message) {
    alert(message);
    this.myMessage = message;
}

// 添加一个公有方法
myConstructor.prototype.clearMessage = function(string) {
    this.myMessage += ' ' + string;
}
```

与本章前面使用点号向一个对象的已有实例中添加成员不同，向prototype中添加成员将会把新方法添加到myConstructor的底层定义中，而不是添加到myConstructor实例自身！

因此，在新实例上面调用clearMessage()方法将会得到预期的结果：

```
var myObject = new myConstructor('Hello World!');
myObject.clearMessage();
```

但是，你不能直接在myConstructor上面调用这个方法，因为myConstructor是Function

www.TopSage.com

对象的一个实例，而不是myConstructor对象的实例：

```
myConstructor.clearMessage();
// TypeError: myConstructor.clearMessage is not a function
```

公有方法对于要添加到对象中的大多数功能来说都是非常合适的，但在某些情况下，你可能会考虑添加一个只能在对象内部使用的方法或属性，也就是说这个方法或属性不能被任何人公开地访问到。在这些情况下，你希望添加的成员就是私有成员或者特权成员，而这正是下一节我们要讨论的主题。

**通过私有和特权成员控制访问**

现在，你已经知道如何向对象实例中添加公有成员了。全部以公有成员的身份提供属性和方法，在所有成员和方法都必需与对象进行交互的多数情况下都是没有问题的。但是，如果你需要添加一个仅供你自己内部使用的私有或特权成员该怎么办呢？这种措施对于在库中防止别人以你无法预测的方式使用内部方法特别有用。

如果我说你对创建私有成员已经相当熟悉并且也曾使用过私有成员，你可能会感到惊讶。事实上，私有成员就是在另一个函数中定义的变量和函数。如果要给前面的myConstructor函数添加一个私有的alertMessage()方法和一个私有的separator属性，那只需在该构造函数中使用普通的var和function关键字定义它们即可：

```
function myConstructor(message) {
    this.myMessage = message;

    // 私有属性
    var separator = ' -';
    var myOwner = this;

    // 私有方法
    function alertMessage() {
        alert(myOwner.message);
    }
    // 在实例化时显示信息
    alertMessage();
}
```

你可能注意到了，这个例子代码中也包含另外一个私有属性myOwner，它引用的是this。通过将this赋值给myOwner，你的私有方法就可以通过引用myOwner来访问myConstructor的实例。私有方法是存在于构造函数作用域中的自包含的（self-contained）对象，它们实际上并不是prototype的方法，因此在私有方法内部this引用的只是私有方法的实例，而非myConstructor的实例。而在作用域链中，私有成员内部的myOwner将会解析为上层的myConstructor的实例，如图2-12所示。我会在2.3节中更详细地讨论与this关键字相关的内容。

同样也不能在对象外部访问这些私有成员，因为它们被限制在了构造函数的作用域中。如果试图在myObject实例中访问它们，无论是调用myObject.alertMessage()还是myObject.separator都将失败。而且，由于这些私有成员被严格限制在了构造函数的作用域中，所以也不能通过对象自己的公有方法来访问它们。如果像下面这样通过prototype中的公有方法来访问

www.TopSage.com

separator:

```
myConstructor.prototype.appendToMessage = function(string) {
    this.myMessage += separator + string; // 将出错
}
```

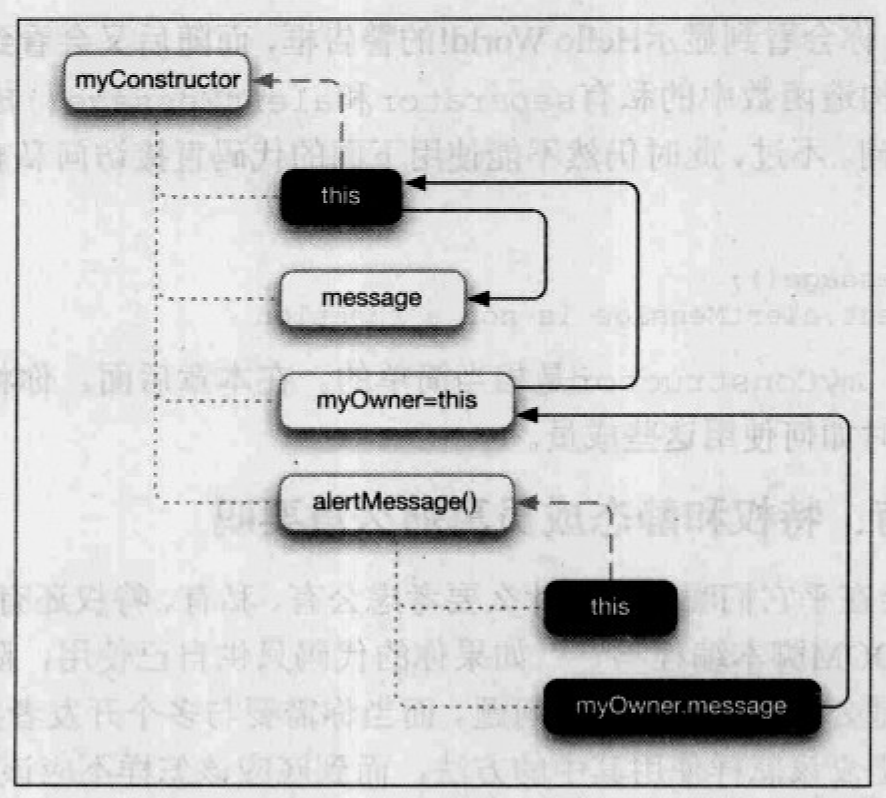



图2-12 使用myOwner变量来指代外部作用域的this关键字

你会得到一个指示separator未定义的错误，因为私有的separator属性只存在于对象的构造函数之中。而要避开这个限制，就需要用到特权成员了。

与私有方法不同，特权方法能够被公开访问，而且还能够访问私有成员。特权方法是指在构造函数的作用域中使用this关键字定义的方法：

```
function myConstructor(message) {
    this.myMessage = message;
    var separator = ' -';
    var myOwner = this;

    function alertMessage() {
        alert(myOwner.myMessage);
    }
    alertMessage();

    // 特权方法
    this.appendToMessage = function(string) {
        this.myMessage += separator + string;
        alertMessage();
    }
}
```

以这种方式创建特权方法后，myConstructor同样拥有了和前面使用prototype的例子中相同的appendToMessage()方法，但此时的appendToMessage()方法位于构造函数的作用域

www.TopSage.com

中，因而具有通过作用域链访问私有的separator成员的权限。为了验证这一点，可以实例化一个对象，使用chapter2/myConstructor/privleged.html试验一下：

```
var myObject = new myConstructor('Hello World!');
myObject.appendToMessage('Jeff');
```

在实例化过程中，你会看到显示Hello World!的警告框，而随后又会看到显示Hello World!-Jeff的警告框，这是因为构造函数中的私有separator和alertMessage()成员可以通过appendToMessage()方法访问。不过，此时仍然不能使用下面的代码直接访问私有的alertMessage()方法：

```
myObject.alertMessage();
// Error: myObject.alertMessage is not a function
```

从这一点上来看，myConstructor是相当简单的。在本章后面，你将看到在创建一个有用的JavaScript调试对象时如何使用这些成员。

## 2.2.4　公有、私有、特权和静态成员真那么重要吗

你可能会问"谁会在乎它们呢？我为什么要考虑公有、私有、特权还有静态成员等这些概念，它们还会影响到我的DOM脚本编程吗？"如果你的代码只供自己使用，除了你谁也看不到它，那么即使你的方法都是公有成员也不会有问题。而当你需要与多个开发者共享代码时，问题就出现了。只有你自己知道应该怎样使用其中的方法，而到底应该怎样不应该怎样别人则一概不知。将所有属性和方法都定义为公有成员，对象的公有部分与内部处理之间也就没有了差别。对他人来说，可以使用哪个成员，而哪个成员又是供对象自身使用的都不好确定。假如你还没有提供适当的文档说明，那么问题就更难解决了。因特网上有许多库，要分清这些库中哪些部分是公有的API，哪些部分是私有的API几乎是不可能的。

但是，通过组合使用各种成员，你的myConstructor()对象就可以清晰地将公有和私有成员分开，因而就不会造成混乱了：

```
// 构造函数
function myConstructor(message) {
    this.myMessage = message;

    // 私有属性
    var separator = ' -';
    var myOwner = this;

    // 私有方法
    function alertMessage() {
        alert(myOwner.myMessage);
    }
    alertMessage();

    // 特权方法（也是公有方法）
    this.appendToMessage = function(string) {
        this.myMessage += separator + string;
        alertMessage();
    }
```

www.TopSage.com

```
}

// 公有方法
myConstructor.prototype.clearMessage = function(string) {
    this.myMessage = '';
}

// 静态属性
myConstructor.name = 'Jeff';

// 静态方法
myConstructor.alertName = function() {
    alert(this.name);
}
```

理解了创建对象的适当方式，将会使你自己和其他人在使用你的对象时都更加容易。同样地，记住以下几条规则可以保证你对所有成员的身份作出适当地界定：

- 由于私有和特权成员在函数的内部，因此它们会被带到函数的每个实例中①。
- 公有的原型成员是对象蓝图的一部分，适用于通过new关键字实例化的该对象的每个实例。
- 静态成员只适用于对象的一个特殊实例②。

## 2.2.5 对象字面量

到目前为止，本章的例子都是使用点号来创建对象和成员的。我有意在一开始介绍对象模型时选择这种语法，是因为你可能对使用点号操作符比较熟悉。举一个熟悉的例子，比如下面这样使用点号访问元素：

```
var h1Elements = document.getElementById('example).➥
getElementsByTagName('h1');
```

或使用点号完成赋值操作：

```
document.getElementById('example).style.color = 'green';
```

而当提到对象时，我们也曾使用同样的点号操作符在myConstructor的prototype中定义过方法：

```
myConstructor.prototype.clearMessage = function() { }
```

然而，对象字面量作为另一种语法则更清晰也更便于阅读，特别是在向对象中添加多个成员时更是如此。在对象字面量的语法中，使用花括号表示对象的结构：

```
var myObject = {
    propertyA:'value',
    propertyB:'value',
    methodA:function() { },
    methodB:function() { },
    methodC:function() { },
    methodD:function() { }
```

① 即由构造函数创建的每个实例中都会包含同样的私有和特权成员的副本，因而实例越多占用内存也就越多。——译者注

② 这个特殊的实例就是作为Function对象实例的构造函数本身。——译者注

www.TopSage.com

```
}
```

以上代码与下列“复制粘贴[①]”的方法原理相同，但却更优雅：

```
var myObject = new Object();
myObject.propertyA = 'value';
myObject.propertyB = 'value';
myObject.methodA = function() { };
myObject.methodB = function() { };
myObject.methodC = function() { };
myObject.methodD = function() { };
```

也就是说，下面的简写形式：

```
var myObject = {};
```

等价于以下代码：

```
var myObject = new Object();
```

{键:值,键:值}语法中的“键/值”对会成为对象的静态成员。如果给某个“键”指定的值是一个匿名函数，那么该函数就会变成对象的静态方法；否则，该值就是对象的一个静态属性。这种语法结构与我们将在第7章中讨论的JSON（JavaScript Object Notation，JavaScript对象表示法）的语法结构相同。只要记住对象字面量语法会自动创建Object对象的实例即可，也就是说不能使用new关键字对其再次进行实例化。

如果要使用同样的对象字面量语法构建一个带有公有方法的构造函数，仍然需要从作为构造函数的Function对象开始：

```
function myConstructor() {
    // 私有和特权成员
};
```

或者，也可以使用下面的替代语法：

```
var myConstructor = function() {
    // 私有和特权成员
};
```

> 对于前面例子中使用的第2种语法需要注意一下。当浏览器在解析你的脚本并遇到使用第一种语法定义的函数，比如function example(){...}时，example()函数会在脚本执行之前立即被声明。这意味着你可以在脚本中任何地方调用example()，即使对函数的调用发生在函数的定义之前也没问题。而对于第2种替代语法来说，如var example2 = function(){...}，在脚本执行到该赋值语句之前example2()函数是不存在的。如果你在该定义发生之前调用example2()，结果会失败[②]。

① 意指myObject在代码中多次重复出现。——译者注

② 浏览器中的解释程序在执行JavaScript代码之前，首先要对代码进行变量初始化，即初始化window对象之下的所有顶级变量。由于第2种定义函数的语法是通过赋值语句实现的，所以在初始化过程中不会被立即声明。只有在解释器开始执行代码并执行到相应的赋值语句时，该函数才会被声明。因此调用该函数的语句只能出现在定义该函数的语句之后。并且，如果使用第2种语法来定义构造函数，那么为其prototype属性中添加公共成员的代码，也应该放在该函数定义之后。——译者注

www.TopSage.com

然后，就可以使用对象字面量语法来向prototype属性中添加所有公有成员了：

```
function myConstructor() {
    // 私有和特权成员
};

// 公有成员
myConstructor.prototype = {
    propertyA:'value',
    propertyB:'value',
    methodA:function() { },
    methodB:function() { },
    methodC:function() { },
    methodD:function() { }
}
```

同样，以上代码与使用点号操作符的原理也是相同的，唯一的区别就是表现形式。至于使用哪种方法完全取决于你，不过从个人的观点来说，我更喜欢对象字面量语法。因为这种语法不仅冗余代码少，而且也更容易理解。因此，我在本书的多数例子中都会使用对象字面量语法。

如果你也使用对象字面量，那么还必须小心结尾处的逗号。如果你在定义对象时，像下面这样在最后一个项的结尾放了一个逗号，那么最后一个项的值会变成null：

```
var badObjectLiteral = {
    a:1,
    b:2, // 多余的逗号
}
```

Firefox和Mozilla浏览器会忽略值为null的项目，但是IE则会在它试图解析这个对象时报错。我认为在处理这种情况时IE的做法是正确的，因为它没有试图修复你不明确或者不完整的代码。但是，当我在复制粘贴大段的代码时，也曾不止一次地发现在Firefox中能够正常运行的代码，到了IE中却成了残缺代码。

## 2.3 this 是什么

this是一个难以捉摸的家伙。大多数程序员都认为this是在当前脚本或对象的作用域中引用一个普通元素的标识符。然而，对于JavaScript来说，却不一定是这种情况——this在JavaScript中是一个依赖于使用它的执行环境被解析的关键字。然后呢？请看下面的例子：

```
var sound = 'Roar!';
function myOrneryBeast() {
    this.style.color='green';
    alert(sound);
}
```

其中的this关键字引用的是包含它的函数作为某个对象的方法被调用时的那个对象。那么这对于myOrneryBeast()函数意味什么呢？我们在前面曾经讨论过，你的全部脚本，包括每一点代码都包含在一个名为window的全局对象中。就如同前面的appendToMessage()和clearMessage()函数是myConstructor的方法一样，myOrneryBeast()函数也是window的方法。因此，在脚本的最顶层，this引用的是window对象。你可以自己运行下面这行没有任何

www.TopSage.com

包装方法或对象的代码试验一下，运行结果如图2-13所示。

```
alert('Does window === this? ' + (window === this));
```

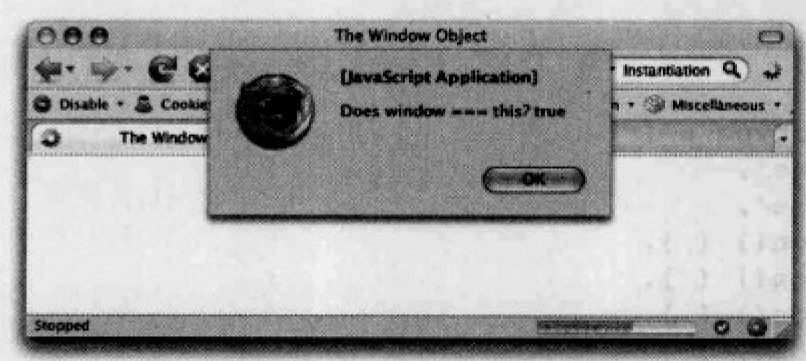



图2-13 警告框显示在脚本的最顶层环境中window对象等同于this

的确，警告框证实了this与window对象等同。但是，在myOrneryBeast()函数的环境中使用的this是如何解析为window对象的呢？

因为就目前的情况而言，myOrneryBeast()函数是作为window对象的一个方法存在的。如果你只是照原样调用这个函数

```
myOrneryBeast();
```

那么，函数中的this关键字将会解析为包含它的函数作为方法被调用时所属的对象，因为myOrneryBeast()是在window的环境中被调用的，所以this.style.color将会引用window.style.color，相应的解析过程如图 2-14所示。

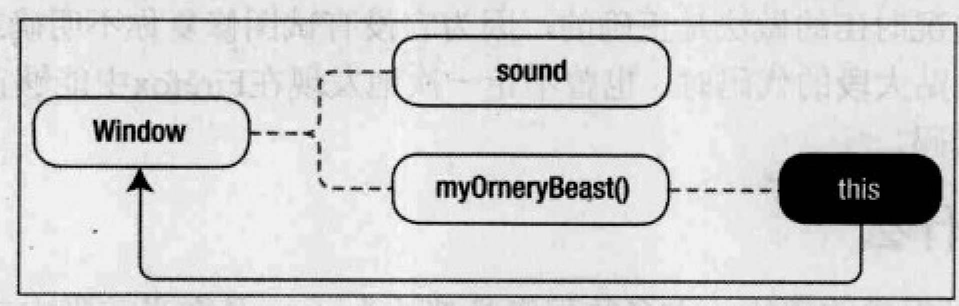



图2-14 this关键字的作用域链求值过程

然而，在这个例子中myOrneryBeast()方法有可能会不开心，因为sytle不是window对象的合法属性。不过，this的环境可以随着函数被赋值给不同的对象而改变。比如直接或者使用第1章中定义的ADS.addEvent()方法将函数赋值给一个事件侦听器属性：

```
var sound = 'Roar!';
function myOrneryBeast() {
    this.style.color='green';
    alert(sound);
}

function initPage() {
    var example = ADS.$('example');

    // 使用事件属性方法
```

www.TopSage.com

```
    example.onclick = myOrneryBeast;

    // 或者使用ADS.addEvent方法
    ADS.addEvent(example,'mouseover', myOrneryBeast);
  }
ADS.addEvent(window,'load',initPage);
```

> 与少数优秀的开发者一样，我们这里的测试使用了一个被包含的脚本文件和一个不唐突的load事件侦听器。对本例中测试页面的唯一要求就是，它必须包含一个id="example"的元素，而且ADS.addEvent()函数也要存在。为了运行这个例子，你也可以仅仅使用<script>标签在<head>中以嵌入的方式添加这些代码。但是，我始终认同实践出真知，因此为什么不将你的沙箱环境设置成与你的生产环境一样呢？如果你能坚持正确的做事方式，就会挡住坏习惯的诱惑！

无论使用哪种事件注册模型，相应的元素都将在被单击时变成绿色，而且也会看到显示“Roar!”的警告框，对不对？确定吗？是的！你是正确的。但是，真正的问题是为什么？

回忆一下，在第1章我们讨论作用域和闭包的例子时，由于id引用的是外部函数作用域中的闭包，所以每个链接的id都没有被正确地设置。而在本例中，如果对变量sound求值的结果仍然是“Roar!”，那么this不也仍然应该引用window对象吗？虽然所有迹象都表明应该如此，但事实上结果并不像你所期望的那样。记住，JavaScript会在事件侦听器被调用的环境中将this作为一个关键字来解析。具体到这个例子，无论是onclick属性还是click事件，都是example对象的方法。因此，this会被解析为将函数作为其方法的对象，或者更确切地说，将事件作为方法的HTML元素，即this引用的是example这个HTML元素。

对于sound变量而言，它仍然会引发我们在第1章前面讨论过的同样的作用域和闭包的问题。但由于sound始终都一样，即它在initPage()函数的作用域中没有改变，所以它在作用域链中始终会被解析为“Roar!”，如图2-15所示。

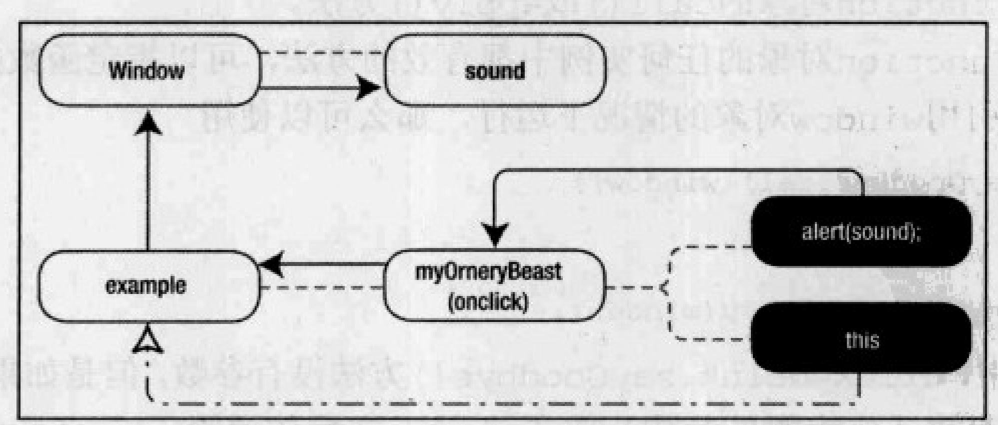



图2-15 通过作用域链解析sound变量的过程

如果你在myOrneryBeast()方法中或者脚本的其他地方，赋给sound别的值，那么通过作用域链就会反映出相应的变化。你可以试验一下，看在不同的位置为sound赋不同的值会得到什么结果。

## 通过call()和apply()重新定义执行环境

到目前为止，你已经看到了如何定义自己的对象，并且试验了this在方法和对象构造函数的内部所引用的对象。而且，也知道了this在简单的老函数中引用的是全局window对象。这些不算什么，你大概都已经烂熟于心了。除此之外，你还看到了this的环境是如何随着函数被赋值给其他对象（比如HTML元素）而相应改变的。如果你完全理解了以上内容，说明你已经前进到了一个好的起点，但在前面最后的例子中还有一个问题有待解决，这个问题涉及环境的改变。如果改变环境并非我们希望看到的，那又该怎么办呢？

下面举一个例子，这个例子位于chapter2/doubleCheck/without-context.js中：

```
function doubleCheck() {
    this.message = 'Are you sure you want to leave?';
}
doubleCheck.prototype.sayGoodbye = function() {
    return confirm(this.message);
}
initPage() {
    var clickedLink = new doubleCheck();
    var links = document.getElementsByTagName('a');
    for (var i=0 ; i<links.length ; i++) {
        ADS.addEvent(links[i], 'click', clickedLink.sayGoodbye);
    }
}
ADS.addEvent(window,'load',initPage);
```

这个例子中载入事件的预期结果是循环遍历页面中所有的链接，然后将clickedLink.sayGoodbye()方法指定为每个链接的click事件侦听器。看起来好像很简单，但是如果你回忆一下前面学过的this引用的对象会发生改变的事实，就会发现其中存在问题。根据上一个例子我们知道，当这个例子中的sayGoodbye()方法在<a>这个HTML元素的环境中执行时，this所引用的就是这个HTML元素，而不是你期望的clickedLink对象。嗯……那你怎么解决这个环境问题呢？可以使用Function对象的call()或apply()方法。

通过这两个在Function对象的任何实例中都有效的方法，可以指定函数的执行环境。假如你想让方法在this引用window对象的情况下运行，那么可以使用

```
clickedLink.sayGoodbye.call(window);
```

或者

```
clickedLink.sayGoodbye.apply(window);
```

在前面的例子中，clickedLink.sayGoodbye()方法没有参数，但是如果这个方法有参数，那么应该将方法的参数（作为额外参数）放在window之后传递给call()或者apply()。对于call()而言，每个参数都应该位于在对象之后，比如：

```
functionReference.call(object, argument1, argument2, ...)
```

对于apply()，则应该将方法的参数作为一个数组放在第2个参数的位置上传递：

```
functionReference.apply(object, arguments)
```

www.TopSage.com

这是call()与apply()之间唯一的区别。

当为了调整方法的执行环境而生成匿名包装函数时，apply()方法特别有用。比如由下面bindFunction()方法返回的函数，会取得通过func参数传递进来的函数，然后将其应用到由obj参数传递进来的对象上面：

```
function bindFunction(obj, func) {
    return function() {
        func.apply(obj,arguments);
    };
}
```

以上代码的关键在于，由binkFunction()返回的匿名函数使用了内部作用域中特殊的arguments参数，作为以外部作用域中的obj和func调用apply()时传递的额外参数。执行以上函数主要是为了给原始的函数（由func参数传递）创建一个新的环境，之后虽然原始函数仍然接受同样的参数，但它却具有了不同的环境，如图2-16所示。

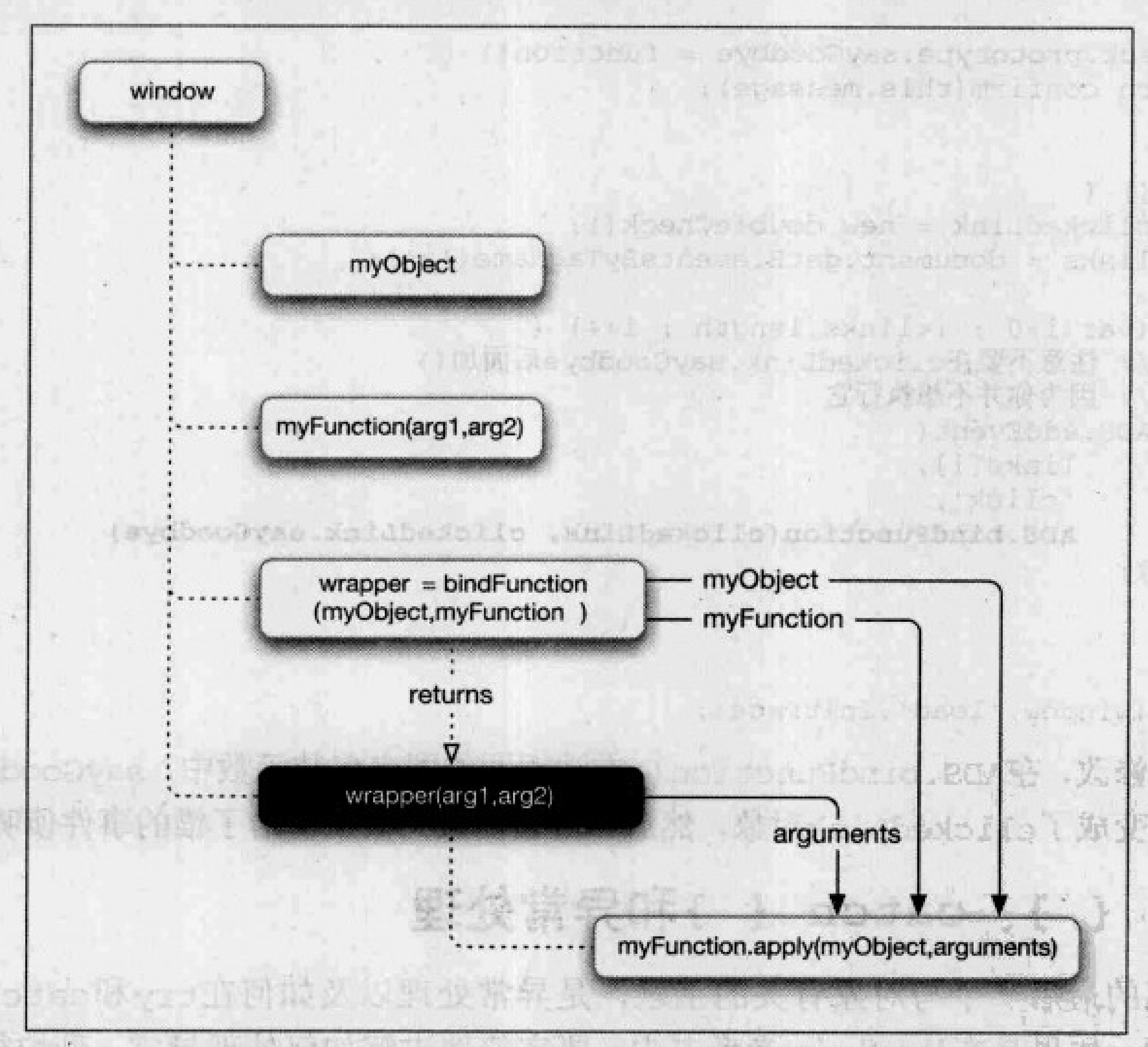



图2-16 在另一个函数中包装一个函数，以便使用apply()方法改变这个函数的环境

花点时间把bindFunction()方法添加到你的ADS库中吧，因为在本章后面还要用到它：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上库中已有的内容……
```

www.TopSage.com

```
function bindFunction(obj, func) {
    return function() {
        func.apply(obj,arguments);
    };
};
window['ADS']['bindFunction'] = bindFunction;

……以下是库中已有的内容……

})();
```

使用这个方法，可以修改任何方法的环境，而只需像使用原始的函数一样使用返回的匿名函数即可。具体到doubleCheck()的例子中，可以将原来的代码修改如下（注意粗体），完整的代码位于chapter2/doubleCheck/with-context.js中：

```
function doubleCheck() {
    this.message = 'Are you sure you want to leave?';
}
doubleCheck.prototype.sayGoodbye = function() {
    return confirm(this.message);
}

initPage() {
    var clickedLink = new doubleCheck();
    var links = document.getElementsByTagName('a');

    for (var i=0 ; i<links.length ; i++) {
        // 注意不要在clickedLink.sayGoodbye后面加()
        // 因为你并不想执行它
        ADS.addEvent(
            links[i],
            'click',
            ADS.bindFunction(clickedLink, clickedLink.sayGoodbye)
        );
    }

}
addEvent(window,'load',initPage);
```

经过此番修改，在ADS.bindFunction()方法返回的匿名包装函数中，sayGoodbye()方法的调用环境如期变成了clickedLink对象，然后该匿名函数又被赋值给了锚的事件侦听器。太棒了！

## 2.4　try { }、catch { }和异常处理

我要介绍的最后一个与对象有关的主题，是异常处理以及如何在try和catch控制结构中进行异常处理。如果是在JavaScript参考书中，要完整地讲解如何处理异常，恐怕得需要一章的篇幅。但是，因为我们要学习的是有趣的DOM脚本编程，所以我只为你提供了一页的浓缩版。在很多情况下，对象都会以抛出异常的方式报告错误，而不是默默地接受错误或者简单地返回false。如果你在window对象的环境中运行下面的myOrneryBeast()方法（位于chapter2/myOrneryBeast/exception.js中），它会抛出一个异常：TypeError: this.style has no properties，因为window对象中不存在style属性：

www.TopSage.com

```
var sound = 'Roar!';
function myOrneryBeast() {
    this.style.color='green';
    alert(sound);
}
myOrneryBeast();
```

这个异常将被沿着作用域链反向发送，进而导致方法立即停止执行。如果你没有进行任何额外的错误检测，那么这个错误将被记录到JavaScript错误日志窗口中。如果你安装了Firefox和Firebug，那么错误会显示在Firebug控制台窗口中。然而，作为一种选择，我们可以通过try/catch控制结构来捕获异常错误，并像chapter2/myOrneryBeast/exception-catch.js中那样对其进行处理：

```
var sound = 'Roar!';
function myOrneryBeast() {
    this.style.color='green';
    alert(sound);
}

try {
    myOrneryBeast();
} catch(theException) {
    alert('Oops, we caught an exception! Name: '
        + theException.name
        + ', message: '
        + theException.message
    );
}
```

运行以上脚本将显示如图2-17所示的警告信息。

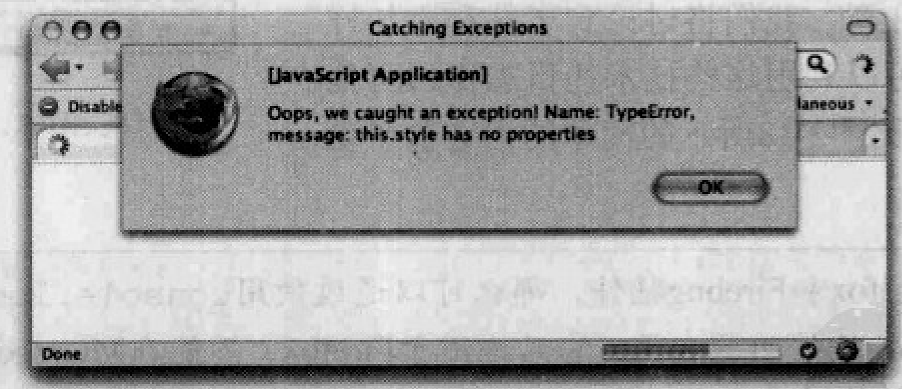



图2-17　警告框显示了在页面载入期间捕获到的异常信息

通过使用适当的try/catch控制结构，你可以自己处理程序中发生的错误，并且采取相应的补救措施。

当我们在第4章中讨论事件和在第7章中讨论Ajax时，对异常的处理会显得格外重要。但在这里我想提醒你一下，因为某些对象（包括你自己的）中同样可能会含有异常，所以你也应该能够解决这些问题。

## 2.5 实例：你自己的调试日志

到本章目前为止，你已经学习了有关对象的很多知识。为了更好地利用所学的知识，我们要

www.TopSage.com

创建一个简单易用的调试日志对象，以便代替使用`alert()`进行调试的技术。在本书其余的章节中，当需要进行试验或者调试DOM元素时，也可以使用这个对象。

### 2.5.1 为什么需要 JavaScript 日志对象

虽然使用警告框调试应用程序是一个不错的起点，但是，如果你进行过某些DOM脚本编程，很可能遇到过警告框的限制。当使用`alert()`在一个大型的循环中调试某个变量的值时，你很可能要坐在电脑前长时间反复不停地单击浏览器警告框上面的“确定”按钮，除非强制退出浏览器并重新启动。再比如，在研究DOM时，如果你希望看看`document`对象都包含哪些方法和属性：

```
for (i in document) { alert(i); }
```

在Firefox中使用上面的循环会生成大约140次警告框供你检查，但这么多的警告框的确非常令人讨厌。我们现在要创建的则是一个更好的解决方案，即使用浮动的调试日志随心所欲地记录日志信息。也就是说，不必再使用`alert()`，而是使用下面的对象：

```
for (i in document) { ADS.log.write(i); }
```

使用了这个对象后，你可以在调试日志窗口中通过滚动条方便地查看140个条目的列表——再也不必强制退出了——如图2-18所示。

本节创建的日志窗口只是一个添加到文档主体中被定位的固定大小的`<ul>`元素，因而它能够悬浮在浏览器窗口的中央。在后面几章，我们将讨论到事件和添加样式化的内容，之后你可以利用这些技术让日志窗口可拖放、可调整大小，实现比图2-18所示的简单的黑边盒子更有趣的效果。

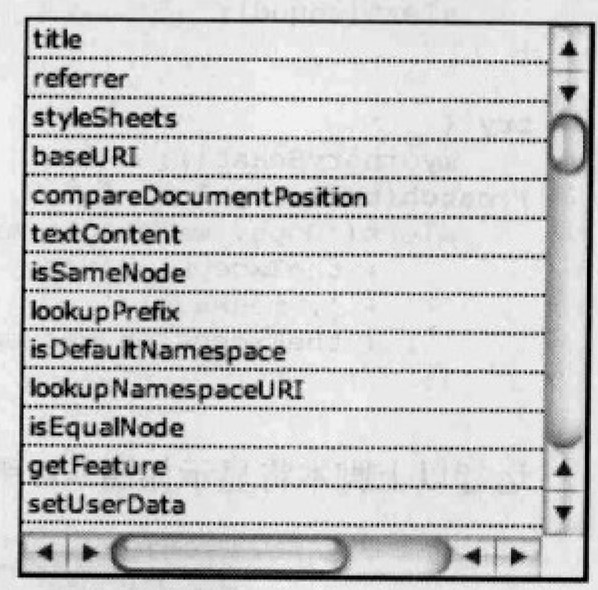



图2-18 `ADS.log()`窗口的外观

> 如果你在使用Firefox和Firebug插件，那么可以通过使用`console.log()`进行多种有效的调试。但使用Firebug扩展的问题在于，它只适用于Firefox，不能在Microsoft IE、Safari、Opera或者其他浏览器中使用。虽然有一个可以跨浏览器使用的Firebug Lite库（http://www.getfirebug.com/ lite.html）也可以实现类似的调试功能，但通过创建自己的调试对象能够学习到这个对象的各种细节，而且还能根据自己的意愿自定义日志。

### 2.5.2 `myLogger()`对象

在开始创建日志对象之前，需要先创建一个简单的HTML页面，其中包含你的ADS库和一个名叫`myLogger.js`的新的JavaScript文件。除此之外，不需要对这个测试页面进行任何修饰。然后，将下列代码结构复制到myLogger.js文件中。本书的源文件中包含了创建日志对象的初始文件，可以在chapter2/myLogger-start/目录下找到它们。此外，我还建议像下面代码中所示的那样，将

www.TopSage.com

myLogger对象的一个新实例赋值给ADS.log成员。一般来说，你会把所有日志记录添加到同一个对象中，所以通过已经实例化的ADS.log成员，可以随时使用ADS.log.write('log message')方法来记录日志信息。虽然也可以像我们前面介绍的那样，使用静态方法来简单地定义ADS.log对象，但我们在这里选择使用构造函数的方法，以便你能看到对象中各类成员的工作机制。而且，如果将来你希望为不同的需求而创建独立的日志，这样也便于创建日志窗口的多个实例。

```
function myLogger(id) {
    id = id || 'ADSLogWindow';
    var logWindow = null;
    var createWindow = function () { };
    this.writeRaw = function (message) { };
}

myLogger.prototype = {
    write: function (message) { },
    header: function (message) { }
};

if(!window.ADS) { window['ADS'] = {}; }
window['ADS']['log'] = new myLogger();
```

前面定义的myLogger对象具有几个特征。首先，myLogger是一个拥有受保护的、特权的以及公有的属性和方法的对象。而且，myLogger对象也是一个构造函数，因此当为ADS.log成员赋值时需要使用new关键字对它进行实例化。

myLogger对象的每个成员都有特定的、规划好的目的：

- logWindow是一个受保护的属性，该属性将在内部被对象用来引用日志窗口的DOM节点。
- createWindow()是一个受保护的方法，可以用这个方法在DOM树中创建logWindow节点。
- writeRaw()是一个特权方法，该方法用于向日志窗口中添加一条新记录。这个方法之所以具有特权，是因为如果logWindow尚未定义，它还要调用受保护的createWindow()方法。
- write()是一个在记录日志时使用最频繁的公有方法。它会对message中的尖括号进行编码以便在日志窗口中显示HTML源代码。而且当message是一个对象（并存在toString()方法）时，它也会调用对象的toString()方法。该方法最终会将编码后的消息字符串传递给writeRaw()方法。
- header()是另一个公有方法，如果你像在第3章中那样需要调试脚本中的多个部分，可以使用它向日志窗口中添加加粗的、红色的条目来充当标题。

在刚刚创建的代码结构中，公有的prototype方法是使用对象字面量语法定义的：

```
myLogger.prototype = {
    write: function(message) {},
    header: function(message) {}
```

www.TopSage.com

```
};
```

当然，也可以使用下面的语法：

```
myLogger.prototype.write = function(message) {};
header.prototype.write = function(message) {};
```

但是，如果你打算向日志对象中添加更多的公有方法，比如link方法，那么使用对象字面量会减少冗余代码：

```
// 不使用对象字面量
myLogger.prototype.write = function (message) {};
myLogger.prototype.header = function (message) {};
myLogger.prototype.link = function (link) {};

// 使用对象字面量
myLogger.prototype = {
    write: function (message) {},
    header: function (message) {},
    link: function (link) {}
};
```

以上任何一种方式都有效，具体使用哪种由你自己决定。

虽然这些受保护的、特权的和公有成员的设置初看起来好像没有什么必要，但正如我前面所说，在创建可重用的对象时这些成员的设置始终都应该考虑到。你可能会假设任何使用你的日志对象的人，永远只会调用ADS.log.write()来记录他们所有的信息。虽然完全有理由相信这一点，但他们同样可以使用ADS.log.writeRaw()把原始的HTML代码写入日志窗口中。因为这两个方法都是公有方法，所以无论是哪个方法被调用都是意料之中的，而且也是在API中事先安排好的。把createWindow()方法也设计成一个公有方法，虽然不一定会导致问题，但假如开发者偶然发现了这个公有的createWindow()方法，那他们就可能在自己觉得合适的时候调用它——毕竟你已经将它作为有效的公有方法了。如果后来你又显著地改变了对象的内部工作方式，那么引用新的或缺少的createWindow()方法的任何代码都可能会意想不到地中断。为此，你可能会辩解说任何没有适当地使用你的对象的代码都有可能中断。但请记住，你是想让编写的代码可维护而且容易使用，所以，不应该在代码中打开任何不必要的后门。现在，我们就来分别看一看每个方法。

#### 1. myLogger.createWindow()方法

私有的createWindow()方法将负责创建保存日志条目列表的DOM元素，并将其添加到文档主体中。因为日志以项列表形式表现，所以使用无序列表（如果你愿意，也可以使用有序列表）将具有语义学上的意义，因而可以简单地将每个日志条目作为列表项来添加。在myLogger构造函数内部，createWindow()方法大致应该是这样的：

```
// 用受保护的方法创建日志窗口
createWindow = function () {

    // 取得新窗口在浏览器中
    // 居中放置时的左上角位置
    var browserWindowSize = ADS.getBrowserWindowSize();
    var top = ((browserWindowSize.height - 200) / 2) || 0;
```

www.TopSage.com

```
    var left = ((browserWindowSize.width - 200) / 2) || 0;
    // 创建作为日志窗口的DOM节点
    // 使用受保护的logWindow属性维护引用
    logWindow = document.createElement('UL');

    // 指定ID值，以便必要时在DOM树中能够识别它
    logWindow.setAttribute('id', id);

    // 在屏幕中居中定位日志窗口
    logWindow.style.position = 'absolute';
    logWindow.style.top = top + 'px';
    logWindow.style.left = left + 'px';

    // 设置固定的大小并允许窗口内容滚动
    logWindow.style.width = '200px';
    logWindow.style.height = '200px';
    logWindow.style.overflow = 'scroll';

    // 添加一些样式以美化外观
    logWindow.style.padding= '0';
    logWindow.style.margin= '0';
    logWindow.style.border= '1px solid black';
    logWindow.style.backgroundColor= 'white';
    logWindow.style.listStyle= 'none';
    logWindow.style.font= '10px/10px Verdana, Tahoma, Sans';

    // 将其添加到文档主体中
    document.body.appendChild(logWindow);
};
```

代码中以粗体显示的部分是createWindow()方法中的两个关键部分。

首先，使用DOM方法createElement()创建一个无序列表元素并将其赋值给私有的logWindow属性。为私有的logWindow赋值之所以有效，是因为createWindow()方法的内部作用域中没有定义logWindow的实例。当赋值语句执行时，代码会到外部作用域中查找，于是恰好找到了私有的logWindow属性。

第二个关键部分是把logWindow添加到document.body中，这样会使它在浏览器中可见。私有的logWindow属性将维护一个指向相应DOM节点的引用，因此可以在myLogger对象中通过引用logWindow找到文档主体中相应的无序列表元素。

createWindow()方法也使用传递到构造函数中的可选的id参数为logWindow指定了一个ID值。这样做，只是为了万一需要别的脚本在DOM树中识别日志窗口时方便。

至于要给日志窗口添加什么样式完全取决于你。我在这个例子中将它设置成了200×200像素，并且把它定位在了当前窗口的中央位置。你可能注意到我调用了ADS.getBrowserWindowSize()方法，该方法应该放在你的ADS.js命名空间中，其代码如下：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……
```

www.TopSage.com

```
function getBrowserWindowSize() {
    var de = document.documentElement;
    return {
        'width':(
            window.innerWidth
            || (de && de.clientWidth )
            || document.body.clientWidth),
        'height':(
            window.innerHeight
            || (de && de.clientHeight )
            || document.body.clientHeight)
    }
};
window['ADS']['getBrowserWindowSize'] = getBrowserWindowSize;

……以上是库中已有的内容……

})();
```

专门在日志对象中包含一个取得浏览器窗口大小的方法并没有太大的意义。同样，这个方法之所以必要是因为浏览器间的差别，因此当需要在不同的对象中重用这个相同的方法时它就有意义了。而把这个方法放在你的ADS库的命名空间中，不仅可以实现重用，而且也可以避免与别人定义的不一样的getBrowserWindowSize()函数发生冲突。

**2. myLogger.writeRaw()方法**

接下来是特权方法writeRaw()，它负责真正地向logWindow中添加新条目。遵循与createWindow()相同的模式，writeRaw()方法也应该是创建一个DOM列表项元素、为其添加样式、生成日志条目，然后再把它添加到logWindow对象中，该方法的代码如下：

```
this.writeRaw = function (message) {

    // 如果初始的窗口不存在，则创建它
    if(!logWindow) createWindow();

    // 创建列表项并适当地添加样式
    var li = document.createElement('LI');
    li.style.padding= '2px';
    li.style.border= '0';
    li.style.borderBottom = '1px dotted black';
    li.style.margin= '0';
    li.style.color= '#000';
    li.style.font = '9px/9px Verdana, Tahoma, Sans';
    // 为日志节点添加信息
    if(typeof message == 'undefined') {
        li.appendchild(document.createTextNode( ➡
'Message was undefined'));
    } else if(typeof li.innerHTML != undefined) {
        li.innerHTML = message;
    } else {
        li.appendchild(document.createTextNode(message));
```

```
    }

    // 将这个条目添加到日志窗口
    logWindow.appendChild(li);

    return true;
};
```

> 你会注意到writeRaw()方法和其他方法都使用了style属性来包含嵌入的样式。作为一个通用的规则，应该使用其他技术来保持样式与行为和标记相互分离（我们将在第5章中介绍这一技术）。然而在这里，日志窗口是一个开发工具，它与站点的整体风格和主题无关。假如使用类名和一个外部CSS文件为它定义样式，那么在网页中包含myLogger.js的同时还要包含相应的CSS文件，而这对于调试用的页面元素是没有必要的。

在创建和添加条目之前，需要先检测一下是否已经创建了日志窗口，如果没有，则调用私有的createWindow()方法。你当然可以使用window的load事件侦听器或者其他机制来提前创建日志窗口，但是对于显示日志而言，如果没有任何条目而只显示一个空日志窗口没有实际意义。而通过在添加条目时创建日志窗口，你也创建了一个伪onWhenAnEntryIsAdded事件①。

没有必要在私有的createWindow()方法前面加上this前缀。实际上，如果加上this前缀，该方法反而会无效。使用createWindow()要遵循与使用私有的logWindow成员相同的逻辑。当该方法被调用时，首先会检测函数的内部作用域，然后再检测外部作用域，这里的外部作用域是myLogger对象的构造函数，也就是私有的createWindow()方法所在的构造函数。而使用this.createWindow()也就相当于在对象的实例上调用createWindow()，但因为logWindow是私有的，所以会导致该方法无效。

取决于你的沟通对象，这个方法中使用的另一个元素也可能会被视为“有害的”，但这个元素却能服务于我们的目的，它就是innerHTML。相关的争论中存在两派，一派认为innerHTML是有害的，它不属于W3C DOM规范，而且这种专有浏览器推行非标准特性的草率行为又把我们拖回了DHTML时代。而另一派则这么说“嘿，它的确有效。所有浏览器都支持它，而且DOM代码太长也太复杂了。为什么不利用它呢？”我的观点介于这两派之间。innerHTML的确有很多优点，比如：

- 使用它比使用等价的DOM方法要少很多代码，而且也简单得多。
- 使用它可以使代码更清晰、更易读，因为它允许使用HTML字符串。
- 它的速度更快也更容易实现，因此效率更高。

但innerHTML也有不少缺点，比如：

- 它可能会缺乏长远的支持，因为它不是W3C规范中的内容。
- 它不能维护对一个新创建的节点的引用。
- 它是一个针对浏览器中HTML的特性，在多数针对XML的情况下都无效。
- 仅仅因为使用它更容易而且速度更快，并不意味着就能开发出好的应用程序；很可能只是为自己懒惰找借口而已。

① onWhenAnEntryIsAdded就是所谓的自定义事件，将在第9章中讨论。——译者注

www.TopSage.com

通过使用适当的DOM方法分别创建每个节点，会使你对应用程序的运行机制具有更好的控制和更清晰的理解，因此我优先使用DOM方法。但我也会谨慎地使用innerHTML，以便使它像在本例中那样更有意义。

> 稍微吊吊你的胃口，在第3章末尾的实例研究中，我们将建立一个将HTML转换为必需的DOM代码的工具。虽然在这种特殊（即HTML是未知）的情况下无法使用它，但你可以在其他项目的WYSIWYG组件中使用这个转换工具来快速地创建必需的DOM代码。

在这种情况下使用innerHTML会为你提供很多便利。当记录日志时，你可能希望记录原始的HTML并在日志窗口中将其作为HTML进行解析。这样，日志中的HTML将成为logWindow的DOM结构的组成部分（在调试时你必须知道这一点）。但此时要记录什么样的HTML却是未知的，因此也就无法简单地使用DOM方法来创建相应的结构。不过，从保险的角度考虑，你应该检测节点中是否存在innerHTML属性，如果不存在，则使用createTextNode()方法：

```
if(typeof li.innerHTML != undefined) {
    li.innerHTML = message;
} else {
    li.appendchild(document.createTextNode(message));
}
```

虽然使用createTextNode()不会让浏览器将原始的HTML解析为DOM，但仍然能够保证将信息作为文本字符串显示在日志窗口中。

到目前为止，你已经为myLogger对象创建了完整的功能，也可以开始使用writeRaw()方法来调试你的代码了：

```
ADS.log.writeRaw('This is raw.');
```

但是，如果你像下面这样在日志条目中包含了标记代码，那么这些代码将与日志窗口的标记代码组合到一起，如图2-19中粗体的条目所示：

```
ADS.log.writeRaw('<strong>This is ➡
bold!</strong>');
```

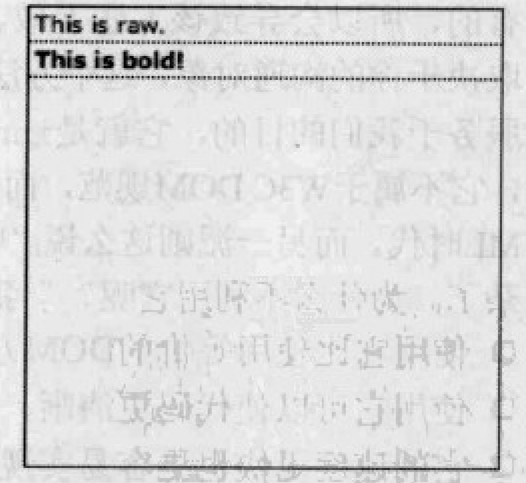


图2-19　ADS.log日志窗口中显示未经处理的普通条目和<strong>标签的效果

遗憾的是，在你添加其余的方法时，不能使用自己新创建的日志对象来调试日志本身。如果你愿意，可以根据需要创建一个不同名称的副本并使用它来调试对象的剩余部分，或者继续使用alert()和JavaScript控制台。

为了显示日志条目中包含的源代码，同时也为了通过对象实现一些其他功能，还需要添加write()和header()方法。

### 3. myLogger.write()和myLogger.header()方法

write()方法只是将writeRaw()方法包装起来，同时执行了一些额外的检测，以防止代码

www.TopSage.com

记录某个对象的实例，并且将左右尖括号（<和>）转换成`<`和`>`。这样，传递到`ADS.log.write()`中的任何HTML代码都将以源代码形式显示在日志窗口中，从而不会像使用`writeRaw()`方法那样显示为被解析的HTML了。同样地，使用`header()`方法，可以添加一个精致的固定样式的条目，如图2-20所示。

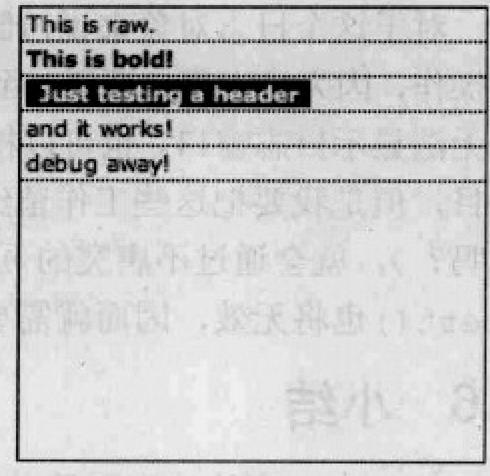



**图2-20** `ADS.log`窗口中显示标题和转义后的HTML代码的效果

为创建公有的`write()`和`header()`方法，要把下列代码添加到`myLogger`对象的原型中：

```
myLogger.prototype = {

    write: function (message) {
        // 警告message为空值
        if(typeof message == 'string' && message.length==0) {
            return this.writeRaw('ADS.log: null message');
        }

        // 如果message不是字符串，则尝试调用toString()
        // 方法，如果不存在该访问则记录对象类型
        if (typeof message != 'string') {
            if(message.toString) return this.writeRaw(message.toString());
            else return this.writeRaw(typeof message);
        }

        // 转换<和>以便.innerHTML不会将message
        // 作为HTML进行解析
            message = message.replace(/</g,"&lt;").replace(/>/g,"&gt;");

            return this.writeRaw(message);
        },

        // 向日志窗口中写入一个标题
        header: function (message) {
            message = '<span style="color:white;background-color:black; ➥
    font-weight:bold;padding:0px 5px;">' + message + '</span>';
            return this.writeRaw(message);
        }

    };
```

现在，可以将如图2-20所示的多种不同的信息记录到日志窗口中了。

到此结束！你的日志对象大功告成。把对`ADS.js`的引用包含在HTML文档的头部，就可以调试相关的内容了。

> 聪明的读者可能会想到`write()`方法提供的功能很容易整合到`writeRaw()`方法中。我之所以把一个功能分成不同的方法来实现，就是为了向你展示私有的、特权的和公有的方法之间的区别。同时，也为你提供一个清晰的`writeRaw()`方法，它除了向日志中添加条目，不对输入作任何技巧性处理。这样，你可以在空闲时添加执行各种技巧性处理的其他代码。

www.TopSage.com

对于这个日志对象来说，唯一需要注意的是它要求页面必须在载入完成后才能执行调用日志的操作，因为日志窗口要添加到文档的主体中。如果页面没有载入完成，页面主体尚不存在，也就无法显示日志窗口。也可以检测页面主体是否载入完成，如果没有，则使用警告框来显示日志条目，但是我要把这些工作留给你来完成。如果你遵循我们在第1章介绍的最佳实践（你会的，对吗？），就会通过不唐突的页面载入来修改文档，因此除非JavaScript完全失效（如果是这样，alert()也将无效，因而就需要采取其他方式来进行调试了），否则你不会遇到太多的问题。

## 2.6　小结

JavaScript中的一切都是对象，但对象却是棘手的小家伙。本章为你介绍了创建包含公有的、私有的、特权的以及静态方法的对象的各种方式。尽管这些对象的成员一开始看起来没那么重要，但你也清楚地看到了如何控制对象方法的访问权限并提供清晰的API，从而使最终用户从中受益。同时，通过控制公有的API并保持额外的辅助方法和属性只在内部有效，也会使你（开发者）从中受益。当你在本书后面的章节中构建自己的ADS库并整合大量公有及私有对象时，还将用到在本章介绍的很多知识。

至此我们已经充分地展示了你应该知道的有关JavaScript的各种细节，下一章我们就来深入了解W3C规范，介绍与DOM2核心和DOM2 HTML相关的更多内容。

www.TopSage.com

第3章

# DOM2核心和DOM2 HTML

在日常生活中，我们对很多事情都是只知其然而不知其所以然。你真的知道汽车引擎的工作原理，或者知道它是怎么从一个地方行驶到另一个地方的吗？你知道当你给朋友打电话时会发生什么事，或者你的声音是怎样在瞬间传送到地球另一端的吗？你知道DOM脚本是怎样使用JavaScript识别并与HTML文档的每一部分交互的吗？你可能对大多数这类事情都持想当然的态度。好，本章就来帮你弄明白最后一个问题的所以然。

如果你自认为是一名设计者，那现在就请你转换角色，用一名程序员的思维来想问题。本章，我们将学习W3C DOM2核心和DOM2 HTML规范，这两个规范为网页文档的DOM表现提供了基础。如果你一谈到规范就恐惧或者对这个概念比较烦，别担心，我们不会面面俱到地讲述每个小细节，因为那样的确很乏味。你大概对上述规范已经有所了解了，但掌握规范的工作原理而不仅仅是如何使用它们，则会使你的理解更上一层楼，比如`Node`和`Element`之间的区别，以及`document.getElementById()`这样的方法怎样同文档进行交互等。而且，你可能还会发现一些原来不知道的新知识。

## 3.1 DOM不是JavaScript，它是文档

众所周知，网页是一种结构化的文档，使用一组预定义的XML或HTML标签进行标记。当浏览器接收到网页文档时，会根据文档类型（doctype）和关联的样式表对其进行解析，然后以可视化形式显示在屏幕上。在与W3C标准兼容的浏览器中，网页文档也可以在遵循DOM规范（http://www.w3.org/DOM）的指导方针下使用JavaScript对象来引用。这些JavaScript对象可以通过脚本获得，而且提供了一种标准的、不针对特定浏览器的文档交互方式。

DOM是一组用来描述脚本怎样与结构化文档进行交互和访问的Web标准。DOM定义了一系列对象、方法和属性，用于访问、操纵和创建文档中的内容、结构、样式以及行为。

在学习规范之前，我要提醒你以下两件事。

- DOM不是JavaScript。
- DOM不是DHTML。

无论以前你听说过、看到过或者认为过什么，DOM与JavaScript都不是一回事。前面说过，DOM是由W3C开发的一组规范。这些规范规定了JavaScript这样的语言为符合标准需要实现的对

www.TopSage.com

象、方法和属性。通过建立并遵守规范，可以保证你编写的JavaScript代码在不同的操作环境（例如不同的浏览器）中具有一致的行为和相同的预期效果。

还记得DHTML这个概念吗？DHTML跟DOM也不是一回事——无论你在因特网中曾经看到过什么介绍。DHTML指的是Dynamic HTML（动态HTML），仅仅是一个由浏览器厂商为浏览器厂商创造的暧昧不明的市场概念。这个概念描述的是添加到早期4.*x*浏览器中的一些在很大程度上与今天的DOM方式类似的操纵文档、样式和行为的新特性。根据W3C的说法，“动态HTML是某些厂商用来描述通过组合HTML、样式表和脚本使文档呈现动画效果的一个术语”（http://www.w3.org/DOM/#why）。

DOM背后的思想中确实结合了DHTML的概念，但DHTML包含了在浏览器间存在差异的非标准对象，例如`document.all`。是的，没错，`document.all`以及其他一些集合实际上都不属于W3C标准或推荐标准的范畴。在使用DOM的情况下，`document.all`集合是通过类似`document.getElementsByTagName('*')`的方式取得的。在本书中，所有代码都会尽可能地避开使用非标准的集合，而严格使用W3C DOM方法，除非另有说明。

### 对象和接口

我们从第2章开始探究的许多JavaScript对象所实现的属性，都是在W3C规范的接口中定义的。每个对象可能实现多个不同的接口，而且对象之间也可能通过对象继承（我们在第2章学习的）相互扩展。任何使用对象解析基于XML标记的语言（不只是JavaScript）都可以实现这些接口。

我们在第2章学习过，JavaScript中一切都是对象，但它是基于原型的面向对象语言，即没有可以直接访问的类或者接口。JavaScript是在对象中自动实现规范中所定义的接口的，因此在本章中，我们会把规范中定义的接口称之为“对象”，因为那是它们在JavaScript中的存在方式。这样称呼只是为了解释方便，如果你看过规范，可不要因为我把规范中的接口称为对象而误解。

## 3.2 DOM的级别

W3C DOM并不是只有一份大而全的规范。它分成不同的级别，每个级别包含不同的子规范和模块。每个级别都在上一个级别基础上实现了一些新的改进特性，但在某些情况下，一个级别可能会与上一个级别不兼容。

### 3.2.1 DOM 0级

没有DOM 0级，因此也没有0级规范。如果有人提到“0级”，那很可能是指一组专有的DHTML方法、对象和集合。而这些专有的特性在成为标准的规范之前，在不同浏览器中的实现是不一致的。

### 3.2.2 DOM 1级

DOM 1级（http://www.w3.org/DOM/DOMTR#dom1）于1998年10月发布，是作为推荐标准发布的第一个DOM标准版本。DOM1级是一个规范，由如下两部分组成。

- DOM Core：为XML文档规定了一般性的树形节点结构的内部运行机制，同时给出了创建、

www.TopSage.com

编辑和操纵这个树形结构的必要属性和方法。

- DOM HTML：为与HTML文档、标签集合以及个别的HTML标签相关的具体元素定义了对象、属性和方法。

DOM 1级规范包含诸如Document、Node、Attr、Element、Text、HTMLDocument、HTMLElement和HTMLCollection等对象的定义，这些对象都将在本章提到。

### 3.2.3 DOM 2级

DOM 2级（http://www.w3.org/DOM/DOMTR#dom2）于2000年11月发布，更新了核心（DOM2核心）并增加了其他一些规范。DOM2 HTML规范于2003年1月发布，添加了针对HTML 4.01和XHTML 1.0的更多对象、属性和方法。DOM2推荐标准分成了以下6个不同的规范。

- DOM2 Core：类似DOM Core，规定了对DOM文档结构的控制机制，添加了更多的特性，比如针对命名空间的方法等（具体的变化可以通过http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/changes.html的Appendix部分了解到）。
- DOM2 HTML：类似DOM HTML，规定了针对HTML的DOM文档的控制机制，还包括了另外一些属性（http://www.w3.org/ TR/2003/REC-DOM-Level-2-HTML-20030109/changes.html的Appendix部分有具体说明）。
- DOM2 Events：规定了对与鼠标相关的事件（包括目标、捕获、冒泡和取消）的控制机制，但不包含与键盘相关事件的处理部分。
- DOM2 Style：或者叫做DOM2 CSS，提供了访问和操纵所有与CSS相关的样式及规则的能力。
- DOM2 Traversal and Range：这两个规范使你能够迭代访问DOM，以便根据需要对文档进行遍历或操作。
- DOM2 Views：提供了访问和更新文档表现的能力。

以上多数规范都为文档中超出实际结构化标记的方面提供了标准化，例如事件和迭代等。而且，其中许多规范之间还具有依存关系，如图3-1所示。

本书没有完整地介绍所有这些DOM2规范，除本章介绍的DOM2核心和DOM2 HTML之外，我们将在第4章介绍DOM2 事件，在第5章讨论DOM2 样式。

### 3.2.4 DOM 3级

DOM 3级（http://www.w3.org/DOM/DOMTR#dom3）包含更新之后的核心（DOM3核心），总共包括以下3个推荐规范。

- DOM3 Core：向原有核心添加了更多的新方法和新属性，同时也修改了已有的一些方法（具体变化详见http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/changes.html的Appendix部分）。
- DOM3 Load and Save：提供将XML文档的内容加载到DOM文档中和将DOM文档序列化为XML文档的能力。
- DOM3 Validation：提供了确保动态生成的文档的有效性（或者符合文档类型声明）的能力。

www.TopSage.com

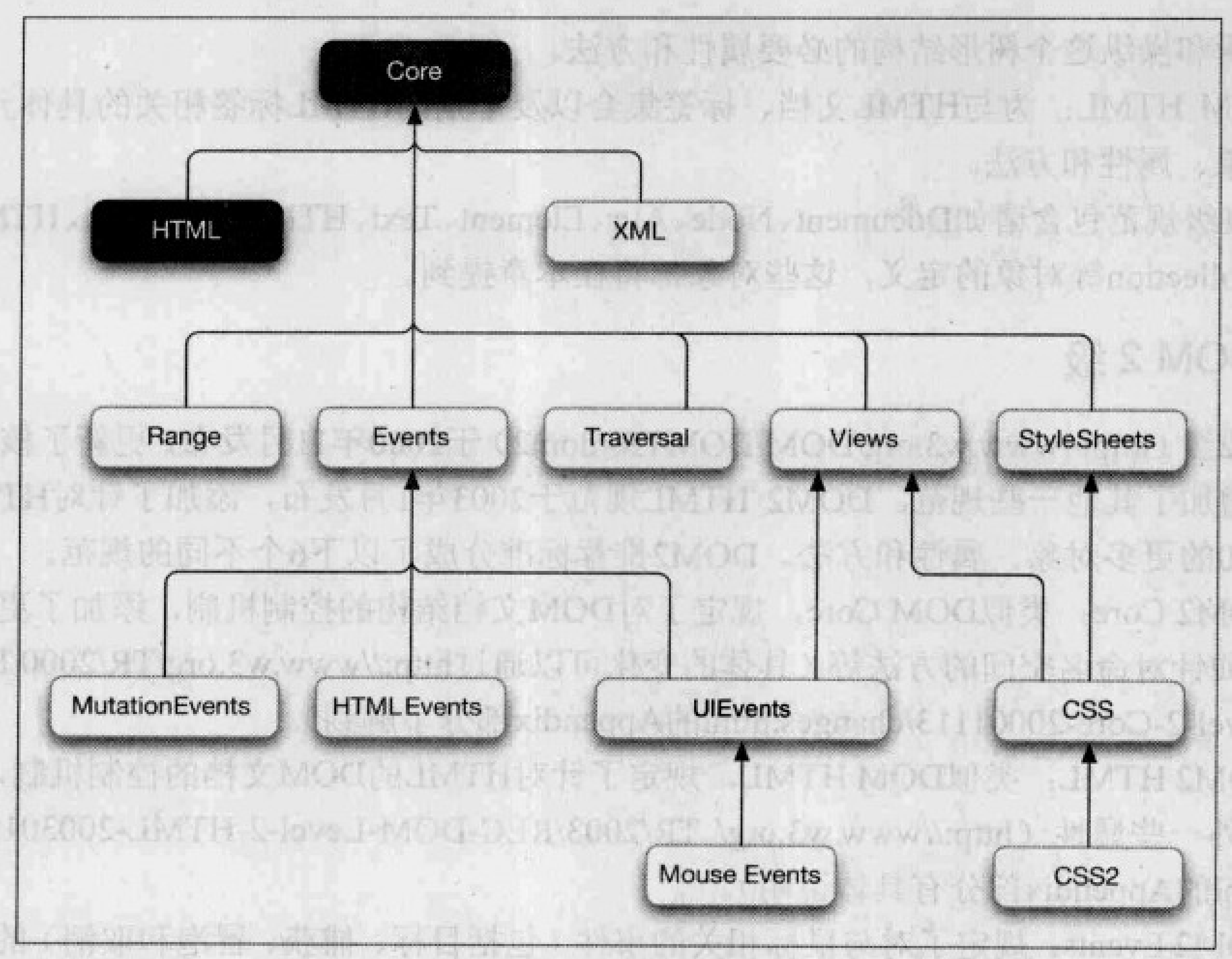



图3-1　DOM2级规范的依存关系

另外，还有其他一些规范还处于早期开发阶段，没有形成最终的推荐标准。

- DOM3 Events：将提供访问键盘和鼠标相关事件的能力。
- DOM3 XPath：将提供使用XPath 1.0查询（http://www.w3.org/TR/xpath）访问DOM文档的树形结构的能力。
- DOM3 Views、Formatting与DOM3 Abstract Schemas：将提供动态访问和更新文档的内容、结构及样式的能力。

令人吃惊的是，有些现代的浏览器都已经支持了很多DOM3规范，如表3-1所示。对你来说，决定采用何种标准主要取决你计划支持哪些浏览器，目标市场愿意接受什么功能以及在多大程度上考虑适应未来的变化。

表3-1　W3C报告的各种浏览器的DOM模块实现级别的兼容性

| 模　块 | Firefox 1.5 | Firefox 2.0 | IE 6 | IE 7 | Opera 9 | Safari 2.0 |
|---|---|---|---|---|---|---|
| Core | 2 | 2 | | | 2 | 2 |
| XML | 2 | 1和2 | | | 1和2 | 1和2 |
| HTML | 2 | 1和2 | 1 | 1 | 1和2 | 1和2 |
| Views | 2 | 2 | | | 2 | 2 |
| StyleSheets | 2 | 2 | | | 2 | 2 |
| CSS | 2 | 2 | | | 2 | 2 |
| CSS2 | 2 | 2 | | | 2 | 2 |

www.TopSage.com

（续）

| 模块 | Firefox 1.5 | Firefox 2.0 | IE 6 | IE 7 | Opera 9 | Safari 2.0 |
|---|---|---|---|---|---|---|
| `Events` | 2 | 2 | | | 2 | 2 |
| `UIEvents` | 2 | 2 | | | 2 | 2 |
| `MouseEvents` | 2 | 2 | | | 2 | 2 |
| `MutationEvents` | | | | | 2 | 2 |
| `HTMLEvents` | 2 | 2 | | | 2 | 2 |
| `Range` | 2 | 2 | | | 2 | 2 |
| `Traversal` | | | | | 2 | 2 |
| `LS` | | | | | 3 | |
| `LS-Async` | | | | | 3 | |
| `Validation` | | | | | | |

### 3.2.5 哪个级别适合你

要知道你选择的浏览器支持何种W3C DOM特性，可以使用`DOMImplementation`对象，该对象在核心规范中规定。在Web浏览器中，`DOMImplementation`对象被实例化为`document.implementation`。如果浏览器的官方声称支持某种特性，那么可以通过`document.implementation.hasFeature()`方法报告的结果进行验证。`document.imlementation.hasFeature()`方法接受两个参数，第1个参数是下列之一。

- Core：DOM 1级和2级的基本方法，以及DOM 2级中的XML命名空间。
- XML：DOM 1级、2级和3级中的XML 1.0。
- HTML：DOM 1级、2级和3级中的HTML 4.0和DOM 2级中对XHTML 1.0的支持。
- Views：DOM 2级，用于CSS和UIEvents模块。
- StyleSheets：DOM 2级，针对关联样式表和文档。
- CSS：DOM 2级，针对层叠样式表进行的扩展。
- CSS2：DOM 2级，针对层叠样式表2级进行的扩展。
- Events：DOM 2级，针对一般事件。
- UIEvents：DOM 2级，针对一般用户界面事件。
- MouseEvents：DOM 2级，针对鼠标事件。
- MutationEvents：DOM 2级，针对DOM树中的事件变化。
- HTMLEvents：DOM 2级，针对HTML 4.01的特定事件。
- Range：DOM 2级，针对DOM树中的范围操作。
- Traversal：DOM 2级，对DOM树的迭代和遍历方法。
- LS：DOM 3级，动态将文档加载到DOM树中。
- LS-Async：DOM 3级，动态异步将文档加载到DOM树中。
- Validation：DOM 3级，对面向模式（schema）修正DOM的支持[①]。

① 此处翻译参考了http://www.w3.org/2003/02/06-dom-support.html。——译者注

www.TopSage.com

第2个参数是DOM级别，即1.0、2.0或3.0。

要测试某个具体的模块，例如2级核心，可以像下面这样：

```
if(document.implementation) {
    if(document.implementation.hasFeature('Core','2.0')) {
        alert('DOM2 Core Supported');
    } else {
        alert('DOM2 Core Not Supported')
    }
} else {
    alert('No DOMImplementation Support');
}
```

如果你的浏览器中不存在document.implementation对象，那么基本上就可确定它根本不支持DOM，不过也有可能是部分支持。例如，微软的IE 6会报告支持HTML而不是Core（核心）。但为了支持HTML，也自然会支持DOM核心的某些部分，因为HTML需要核心方法。

要简略地查看你的浏览器支持哪些模块，可以用它打开W3C提供的页面http://www.w3.org/2003/02/06-dom-support.html试一试。该页面会测试用于查看该页面的浏览器所有的报告特性。表3-1汇总了当前流行的浏览器通过这个页面进行测试的结果。

通过表3-1显示的结果我们可以看到，微软的IE在同符合标准的DOM脚本合作时，好像更不幸一些。但这并不是说IE的能力有限或者没有支持标准，它只是自行其事而已。因此，你必须要考虑到标准方式和IE的方式（注意先后顺序——标准的方式，然后才是专有的方式，而不是反过来）。

你也可以使用http://www.w3.org/DOM/Test中提供的测试套件，对每个模块的个别部分进行更具体地测试。这种测试的时间稍长，但它会检测每个对象的每个方法，因而可以检测出哪些方法符合规范。

DOM核心和HTML模块中包含了在进行DOM脚本编程时可能用到的大多数对象。而表3-1所列出的全部浏览器并不是都能够完全支持DOM2核心和DOM2 HTML。为适应多个浏览器，必须要小心谨慎并通过适当的对象检测来避开不支持的特性。

正如我们在前言部分所提到的，本书中的代码将针对下列浏览器：

- Microsoft IE 6+
- Firefox 1.5+
- Safari 2.0+
- Opera 9+

以及实现了W3C规范的其他浏览器。为此，本书也将把DOM2级推荐标准作为焦点，但在少数情况下也会介绍一些DOM3级元素或其他专有的方法。

## 3.3 创建示例文档

本章剩余的内容将介绍DOM2核心和DOM2 HTML的各个方面。虽然你可能熟悉一些基本的要素，例如document.getElementById()，但毕竟还有许多容易被忽略的方法。

www.TopSage.com

下面将会用到你在第2章最后的部分中创建的myLogger.js文件以及ADS.log对象。如果你现在还没有创建ADS.log对象，可以使用本书附带的完整的源代码。

### 3.3.1 创建DOM文件

为了测试和探索DOM，要创建一个名为domTesting.js的新DOM脚本文件，并将它与ADS.js文件和myLogger.js文件一同像下面那样包含到sample.html文件（源文件位置为chapter3/testing/sample.html）的头部：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>AdvancED DOM Scripting Sample Document</title>
    <!--Include some CSS style sheet to make
        everything look a little nicer -->
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="../chapter.css" />

    <!--Your ADS library with the common JavaScript objects -->
    <script type="text/javascript" src="ADS.js"></script>
    <!--Log object from Chapter 2 -->
    <script type="text/javascript" src="myLogger.js"></script>
    <!--Your testing file -->
    <script type="text/javascript" src="domTesting.js"></script>

</head>
<body>
    <h1>AdvancED DOM Scripting</h1>
    <div id="content">
        <p>Examining the DOM2 Core and DOM2 HTML Recommendations</p>
        <h2>Browsers</h2>
        <p>Typically, you'll be expecting the following browsers:</p>
        <!--Other browsers could be added but we'll keep the list
        short for the example. -->
        <ul id="browserList">
            <li id="firefoxListItem">
                <a href="http://www.getfirefox.com/"
                    title="Get Firefox"
                    id="firefox">Firefox 2.0</a>
        </li>
        <li>
            <a href="http://microsoft.com/windows/ie/downloads/"
                title="Get Microsoft Internet Explorer"
                id="msie">Microsoft Internet Explorer 7</a>
        </li>
        <li>
            <a href="http://www.apple.com/macosx/features/safari/"
                title="Check out Safari"
                id="safari">Safari</a>
        </li>
```

www.TopSage.com

```
                <li>
                    <a href="http://www.opera.com/download/"
                        title="Get Opera"
                        id="opera">Opera 9</a>
                </li>
            </ul>
        </div>
    </body>
    </html>
```

这个示例HTML文件在链接JavaScript文件时，使用相对URL指向了与HTML文件位于同一文件夹中的相应文件。本书所有章的源代码示例都使用一个公共的ADS库。在本例中，`myLogger.js`文件是从Chapter2文件夹中引入的。如果你想使用自己编写的版本，可以重新整理这些文件并修改相应的链接。同样地，这个示例文件也包含了一些公共的CSS文件以便页面看起来更美观。

在正确地完成设置之后，应该看到类似图3-2所示的一个朴素、简单的页面。

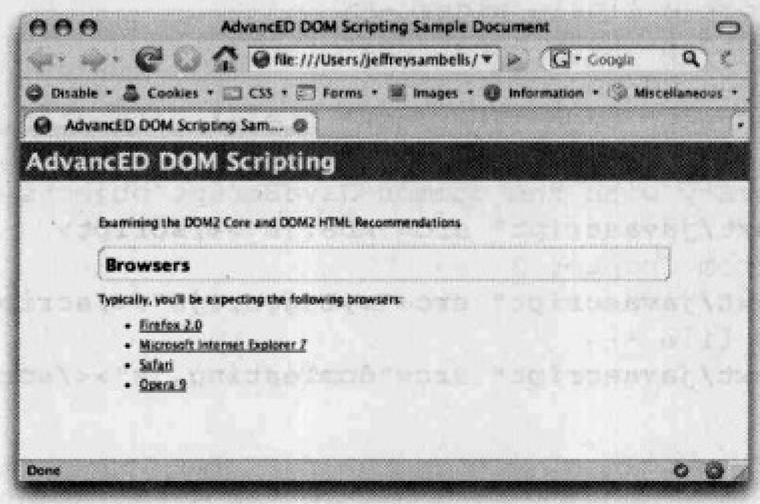



图3-2 在Mac OS X系统的Firefox中显示的`sample.html`文档

## 3.3.2 选择一个浏览器

选择哪个浏览器进行开发取决于个人偏好。就个人而言，我使用Firefox进行开发，因为我至今还没有发现其他任何浏览器具备那么丰富多样的开发者扩展。如果你也使用Firefox，我建议你下载并安装下列扩展：

- **Firebug**: http://getfirebug.com
- **Tamperdata**: http://tamperdata.mozdev.org
- **Web Developer toolbar**: http://chrispederick.com/work/webdeveloper

Firefox，再加上这些扩展，应该说正是一名开发者梦寐以求的，因为只有如此才能无往不胜！

但是，如果你使用的是IE，那么至少可以安装Internet Explorer Developer Toolbar（http://www.microsoft.com/downloads/details.aspx?familyid=e59c3964-672d-4511-bb3e-2d5e1db91038&displaylang=en）。对于Safari用户，可以尝试http://webkit.org上提供的所有针对开发人员的特色工具。

本示例的`sample.html`文档是一个语义化的符合XHTML标准的文档，其中包含一些常规元

www.TopSage.com

素，例如标题、段落、注释、列表和锚链接等。这个文档没有什么特别之处，但对于用作试验目的的测试文档已经足够了。而且，文档标记的树形结构也相当简单，一般来说可以通过图3-3来表示。

图3-3所示的树形结构好像很简单，但是，当浏览器要把示例文件中的标记转换为相应的JavaScript对象时，则需要应对以下挑战：

- 应该怎样表现各种不同类型的标签，以及每个标签的不同属性？
- 彼此相关的这些节点对象之间应该如何维护正确的父/子关系？

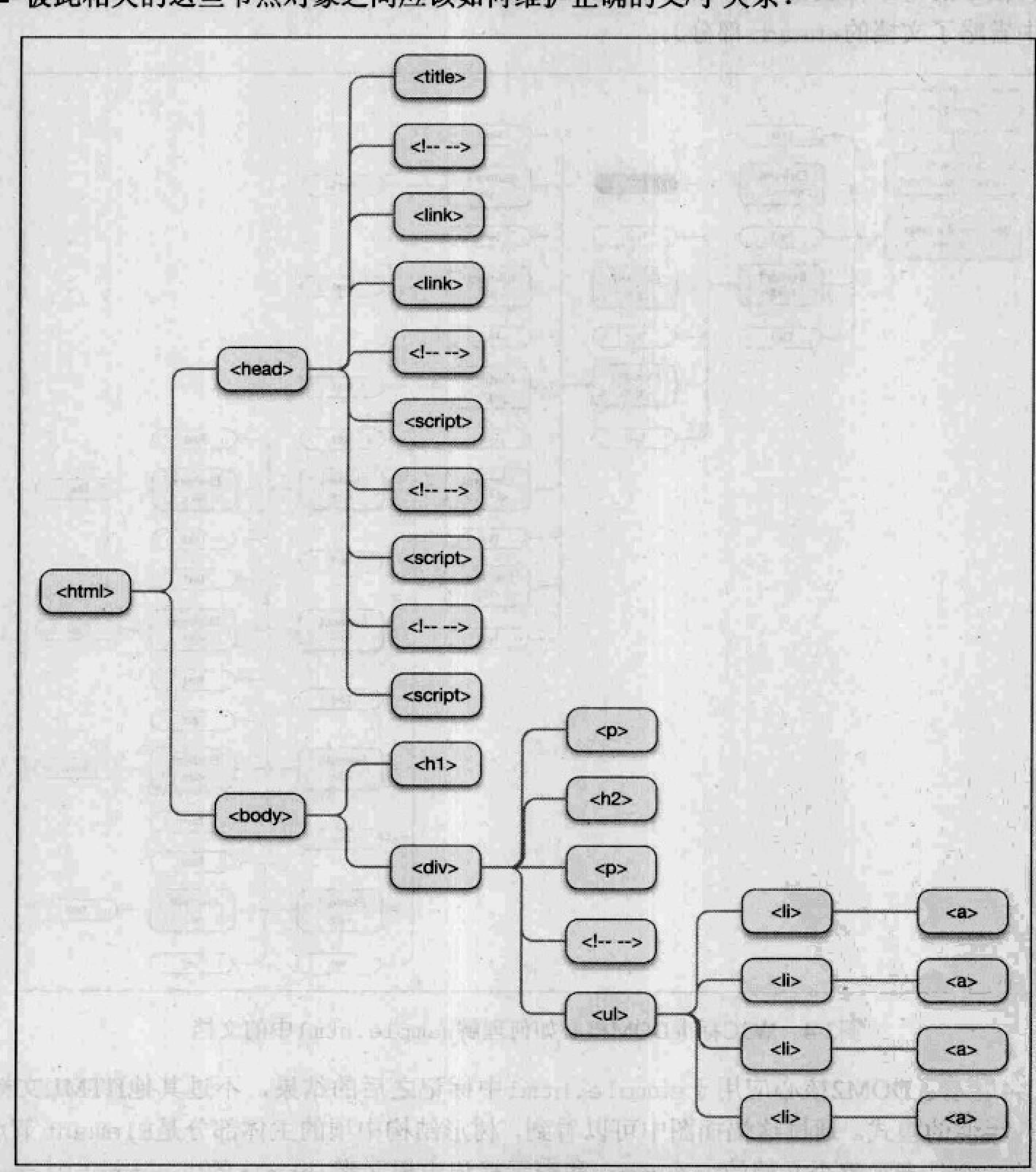



图3-3 sample.html文档结构的树形表示

www.TopSage.com

- 应该怎样处理位于标签之间的那些可爱的空白格式化字符？
- 如果某些文本与空白符相互混合又该如何处理？

这些问题以及更多的问题都可以从下面有关DOM核心与HTML推荐规范的讨论中找到答案。

## 3.4　DOM 核心

当浏览器解析`sample.html`中的标记时，它会根据自身支持的W3C DOM模块把标记转换成对象。文档中的每个标签都可以通过一个核心对象来表示，如图3-4所示（为了使示意图更小一些，图中省略了文档的`<head>`部分）。

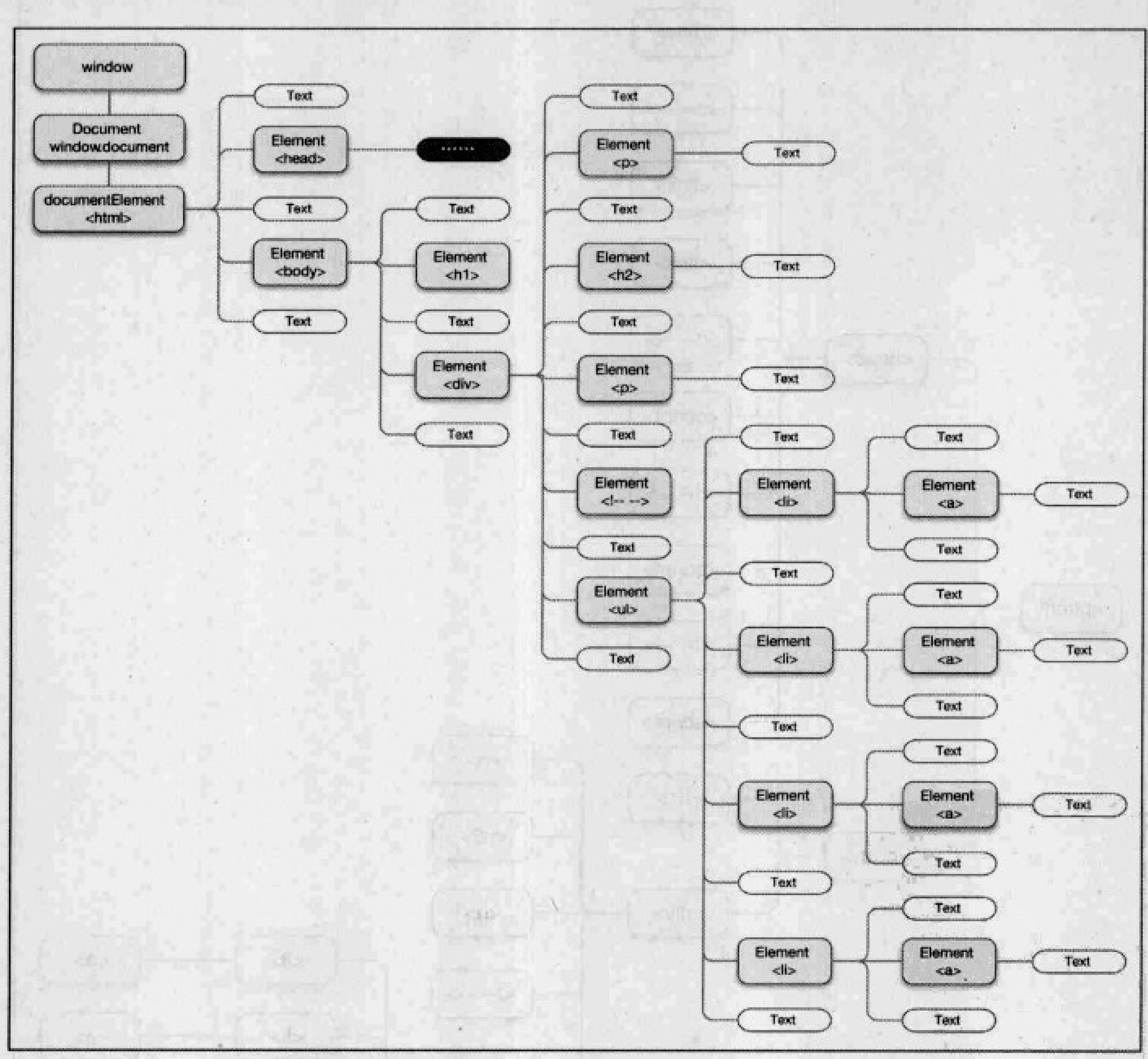



图3-4　W3C标准DOM核心如何理解`sample.html`中的文档

图3-4展示了DOM2核心应用于`sample.html`中标记之后的结果，不过其他HTML文档也将遵循与此类似的模式。通过这幅插图中可以看到，树形结构中项的主体部分是`Element`节点。其中唯一的特例是表示整个文档的`Document`和表示标记中根元素`<html>`的`DocumentElement`。

在图3-4中，我们还注意到标记中每个标签之间的空白符都被转换成了`Text`节点。这也是

www.TopSage.com

DOM规范中规定的对空白符和换行符的处理方式。之所以要通过这种方式来保留空白符，是因为在理想情况下，我们都希望DOM树能够反映与之关联的文档标记中的所有信息。然而，在使用IE的情况下，以上这些文本节点只有当除了空白符外还夹杂有其他的文本字符时才会存在，如图3-5所示。

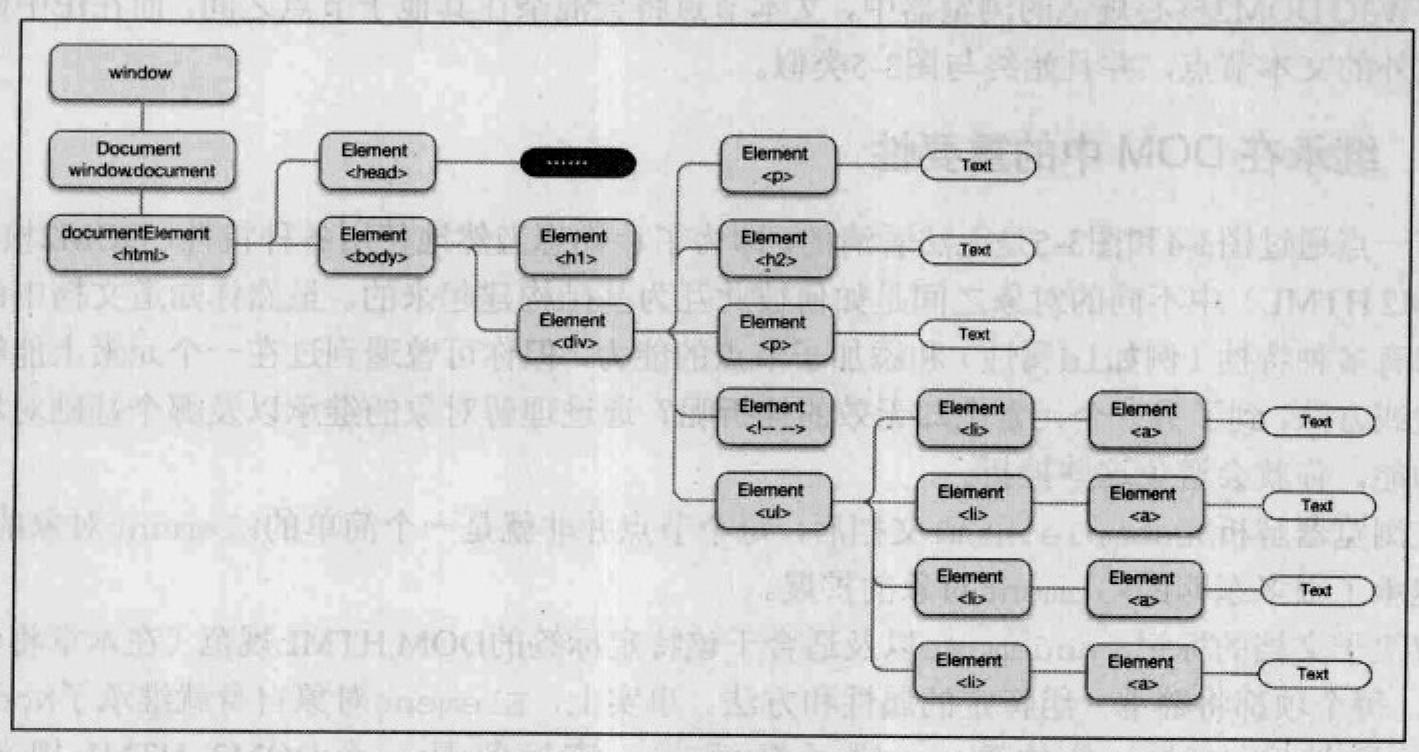



图3-5 DOM核心如何表现没有空白格式化字符的sample.html文档

同样地，如果你像下面标记代码所示的那样，将HTML文件中的所有标签都紧排为一行输出，那么W3C DOM2核心的表现中也不会再出现额外的Text节点，因为它们都是不存在的（见图3-5）：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 ➥
Strict//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> ➥
<html xmlns="http://www.w3.org/1999/xhtml"><head><title>AdvancED ➥
DOM Scripting Sample Document</title><!-- include some CSS style ➥
sheet to make everything look a little nicer --><link ➥
rel="stylesheet" type="text/css" href="../../shared/source.css" ➥
/><link rel="stylesheet" type="text/css" href="../chapter.css" ➥
<!-- ADS Library (full version from source linked here) --><script ➥
type="text/javascript" src="../../ADS-final-verbose.js"></script> ➥
<!-- Log object from Chapter 2 --><script type="text/javascript" ➥
src="../../chapter2/myLogger-final/myLogger.js"></script><!-- Your ➥
testing file --><script type="text/javascript" src="domTesting.js"> ➥
</script></head><body><h1>AdvancED DOM Scripting</h1><div ➥
id="content"><p>Examining the DOM2 Core and DOM2 HTML ➥
Recommendations</p><h2>Browsers</h2><p>Typically, you'll be ➥
expecting the following browsers:</p><!-- Other browsers could ➥
be added but we'll keep the list short for the example. --><ul ➥
id="browserList"><li id="firefoxListItem"><a ➥
href="http://www.getfirefox.com/"title="Get Firefox" ➥
id="firefox">Firefox 2.0</a></li><li><a ➥
href="http://www.microsoft.com/windows/ie/downloads/" title="Get ➥
Microsoft Internet Explorer" id="msie">Microsoft Internet Explorer ➥
```

www.TopSage.com

```
7</a></li> <li><a href="http://www.apple.com/macosx/features/safari/" ➥
title="Check out Safari" id="safari">Safari</a></li><li><a ➥
href="http://www.opera.com/download/" title="Get Opera" id="opera" ➥
>Opera 9</a></li> </ul></div></body></html>
```

当你在迭代每个节点的childNodes时记住这一点是至关重要的。也就是说，在像Firefox这样遵守W3C DOM2核心规范的浏览器中，文本节点将会混杂在其他子节点之间，而在IE中则不会存在额外的文本节点，并且始终与图3-5类似。

### 3.4.1　继承在DOM中的重要性

有一点通过图3-4和图3-5是无法看清的，即为了让你想当然地使用各种特性，DOM2核心（以及DOM2 HTML）中不同的对象之间是如何彼此互为基础构建起来的。虽然你知道文档中的每个元素都有多种特性（例如id属性）和添加子节点的能力，但你可曾遇到过在一个元素上能够使用的属性或方法，到了另一个元素上却无效的挫折呢？通过理解对象的继承以及哪个基础对象提供哪些功能，你就会避免这些挫折。

在浏览器解析完sample.html文档后，每个节点并非就是一个简单的Element对象的实例，而是继承了很多东西的Element对象的扩展。

取决于文档的标记、nodeName以及适合于该特定标签的DOM HTML规范（在本章将学习到这些），每个项都将继承一组特定的属性和方法。事实上，Element对象自身就继承了Node对象的所有属性和方法。具体到<a>锚元素而言，该标签是一个DOM2 HTML规范中的HTMLAnchorElement对象，该对象又扩展自其他一些对象，如图3-6所示。

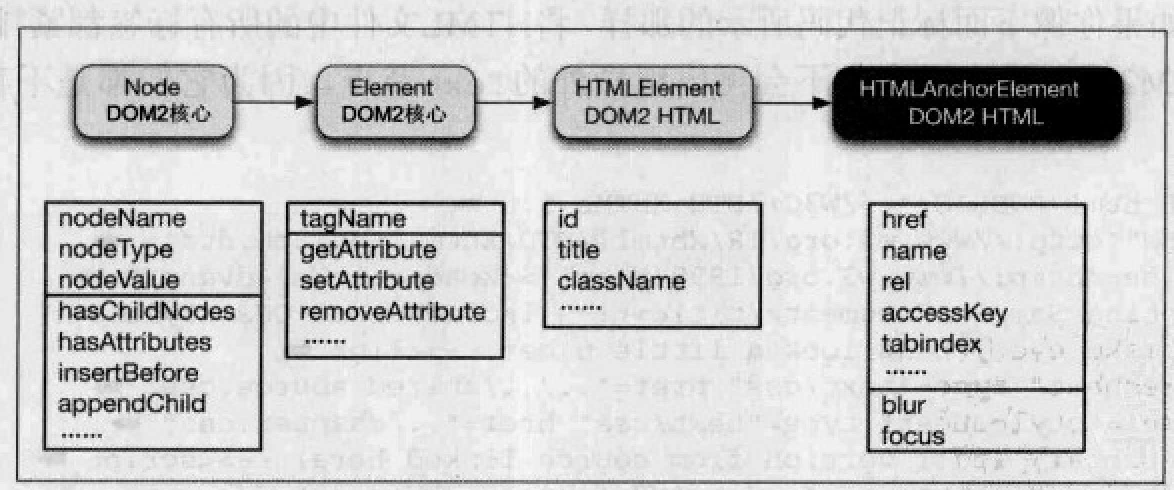



图3-6　从Node到HTMLAnchorElement的对象扩展示意图

通过扩展其他一些对象，页面标记中的锚元素将会具有继承自基础的Node对象的任何对象的全部相关属性和方法，一直到HTMLAnchorElement对象自身。正如我们在第2章中讨论过的，继承是面向对象语言中的一个重要组成部分，通过继承可以在各种对象之间维护公有的功能。同时，继承也解释了为什么段落（<p>）和锚（<a>）元素都是Element对象，但在最终文档的DOM对象中，却存在由DOM2 HTML规范中的HTMLParagraphElement对象和HTMLAnchorElement对象规定的不同属性。

DOM2核心和DOM2 HTML中几乎每一个对象的基础都是Node对象。我们下面就来看一下

www.TopSage.com

Node对象的属性和方法。

## 3.4.2 核心Node对象

图3-5中的每个Element都扩展自Node对象。即使是document和documentElement也是如此，只不过它们也有自己独特的属性和方法。Node对象的属性中包括一些用于识别的特征，例如nodeName、nodeValue、nodeType、parentNode、childNodes、firstChild、lastChild、previousSibling、nextSibling、attributes和ownerDocument。这些属性对于扩展自Node对象的所有DOM对象都是有效的。

### 1. 节点名称、值和类型

对于文档中的Element对象而言，可以使用nodeName属性取得用于区分节点的标签名称。为了保持一致性，nodeName的值会被转换为大写形式。如果将下列载入事件处理程序添加到domTesting.js文件中：

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('testNodeName');
    ADS.log.write('nodeName is: '
        + document.getElementById('firefox').nodeName);
});
```

那么因为sample.html中带有id="firefox"属性的节点是一个锚节点，所以将会在日志窗口中得到下列输出：

```
nodeName is: A
```

对于不基于文档中标签的其他对象（例如Document对象）来说，nodeName的值取决于引用对象的类型。表3-2列出了每种核心对象的节点类型（nodeType）对应的nodeName的返回值。

表3-2 DOM2核心规范中规定的每种nodeType预期的nodeName值

| 对　象 | 返 回 值 |
|---|---|
| Element.nodeName | 元素的名称，大写 |
| Attr.nodeName | 属性的名称，小写 |
| Text.nodeName | #text |
| CDATASection.nodeName | #cdata-section |
| EntityReference.nodeName | 实体引用的名称 |
| Entity.nodeName | 实体的名称 |
| ProcessingInstruction.nodeName | 目标的名称 |
| Comment.nodeName | #comment |
| Document.nodeName | #document |
| DocumentType.nodeName | 文档类型的名称，如HTML |
| DocumentFragment.nodeName | #document fragment |
| Notation.nodeName | 表示法的名称 |

你可能会认为，要取得与Node关联的值，可以使用nodeValue属性。虽然好像这是显而易见的，但nodeValue属性很可能不像你想象的那样。事实上，nodeValue属性只适用于少数DOM

www.TopSage.com

对象，尤其是Attr、ProcessingInstructions、Comments、Text和CDATASection。除此之外，其他所有对象的nodeValue属性都将返回null。例如，下面通过载入事件处理程序获取一个锚的nodeValue值的操作：

```
ADS.addEvent(window,'load',function () {
    ADS.log.header('The node value of the anchor');
    ADS.log.write('nodeValue is: '
        + document.getElementById('firefox').nodeValue);
});
```

将会得到下面所示的结果。因为锚基于Element对象，而Element对象没有与之相关的nodeValue值：

```
nodeValue is: null
```

如果你在domTesting.js文件中同时包含了这个和上一个载入事件处理程序，可能会注意到相应的事件处理程序并没有按照你期望的顺序[①]执行。对这个问题，我们将在第4章中详细讨论。至于现在，因为日志中也包含了标题，所以你可以区别每个事件执行的结果。

为了取得Firefox 2.0这个位于锚标签中的字符串值，必须要在锚的childNodes中查找，因为这个字符串位于锚的一个子Text节点中，例如：

```
var text = document.getElementById('firefox').childNodes[0].nodeValue;
```

此处要注意的是，这行代码假设由getElementById()返回的元素中包含子节点。而如果返回的元素中不包含子节点，那么这行代码就会出错，因为childNodes的第0个索引项不存在。

与nodeName类似，nodeValue属性对于不同的对象类型也具有不同的含义。表3-3列出了每种核心对象类型的nodeValue属性的返回值。

**表3-3　DOM2核心规范中规定的每种nodeType预期的nodeValue值**

| 对　象 | 返　回　值 |
|---|---|
| Element.nodeValue | null |
| Attr.nodeValue | 字符串形式的属性值 |
| Text.nodeValue | 字符串形式的节点内容 |
| CDATASection.nodeValue | 字符串形式的节点内容 |
| EntityReference.nodeValue | null |
| Entity.nodeValue | null |
| ProcessingInstruction.nodeValue | 字符串形式的节点内容 |
| Comment.nodeValue | 字符串形式的注释文本 |
| Document.nodeValue | null |
| DocumentType.nodeValue | null |
| DocumentFragment.nodeValue | null |
| Notation.nodeValue | null |

至于nodeType（你可能已经猜到了它表示的是节点的类型），它会包含与表3-4中的某个命

① 即在载入事件侦听器中注册事件处理程序的先后顺序，或者前面所说的两个事件处理程序在domTesting.js文件中出现的先后顺序。——译者注

www.TopSage.com

名常量对应的一个整数值。表中列出的这些常量表示的是DOM核心对象，因此可以据此确定派生某个节点对象的DOM核心对象的类型。

表3-4 DOM核心对象的nodeType常量

| nodeType值 | 等价命名常量 |
|---|---|
| 1 | Node.ELEMENT_NODE |
| 2 | Node.ATTRIBUTE_NODE |
| 3 | Node.TEXT_NODE |
| 4 | Node.CDATA_SECTION_NODE |
| 5 | Node.ENTITY_REFERENCE_NODE |
| 6 | Node.ENTITY_NODE |
| 7 | Node.PROCESSING_INSTRUCTION_NODE |
| 8 | Node.COMMENT_NODE |
| 9 | Node.DOCUMENT_NODE |
| 10 | Node.DOCUMENT_TYPE_NODE |
| 11 | Node.DOCUMENT_FRAGMENT_NODE |
| 12 | Node.NOTATION_NODE |

如果你再向domTesting.js文件中添加下面的载入事件处理程序：

```
ADS.addEvent(window,'load', function () {
    ADS.log.header('Testing nodeType');
    ADS.log.write('nodeType is: '
        + document.getElementById('firefox').nodeType);
});
```

那么就可以看到文档中某个节点的nodeType了：

```
nodeType is: 1
```

我们在前面曾经提到过，锚标签是HTMLAnchorElement的实例，但后者又扩展自Element对象，所以锚的nodeType值显示为1。

如果你在代码中需要检测nodeType（这经常是必须的），那么要是能够在比较关系中使用DOM常量就好了，比如：

```
if(node.nodeType == Node.COMMENT_NODE) {
    // 针对注释节点的代码
}
```

而不是

```
if(node.nodeType == 8) {
    // 针对……8代表什么意思来着？
}
```

我刚才说“如果能够使用DOM常量就好了”，是因为并不是所有浏览器都支持DOM常量。虽然在Firefox、Safari以及Opera等这些浏览器中都可以使用DOM常量，但在IE中，Node对象则是不存在的。正如我们在本章开头时所提到的，IE不会报告它支持DOM2核心，但是，它却部分地支持该规范。也就是说，对象中也存在一些适当的方法和属性（例如nodeValue），而且会返回

www.TopSage.com

适当的信息。但是，IE却不会直接公开任何核心对象，所以就会导致错误发生。

要消除这种错误，可以定义自己的常量。现在花点时间把下面代码添加到你自己的ADS库中：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

window['ADS']['node'] = {
    ELEMENT_NODE                : 1,
    ATTRIBUTE_NODE              : 2,
    TEXT_NODE                   : 3,
    CDATA_SECTION_NODE          : 4,
    ENTITY_REFERENCE_NODE       : 5,
    ENTITY_NODE                 : 6,
    PROCESSING_INSTRUCTION_NODE : 7,
    COMMENT_NODE                : 8,
    DOCUMENT_NODE               : 9,
    DOCUMENT_TYPE_NODE          : 10,
    DOCUMENT_FRAGMENT_NODE      : 11,
    NOTATION_NODE               : 12
};

……以下是库中已有的内容……

})();
```

> 由于在以上代码的window['ADS']['node']中使用了小写的node，所以如果将来IE的某个版本中加入了对Node对象的直接支持，那么你仍然能够在ADS库中使用Node来访问window的Node对象。

在添加了这个对象之后，就可以在跨浏览器的脚本中使用相同的逻辑了：

```
if(node.nodeType == ADS.node.COMMENT_NODE) {
    // 在任何浏览器中针对注释节点的代码
}
```

使用常量有很多优点。首先，不用死记硬背表3-4中的数值与节点类型的对应关系，只需记住要测试的DOM对象是什么类型即可。其次，你的代码也会因此更容易被人看懂、维护和调试。而且，同样的道理也适用于你打算将自己的代码与别人共享的时候——在比较关系中看到ADS.node.COMMENT_NODE要比看到一个纯粹的数字8，使人能够更多地联想到其中的语义，尤其是在有些人还不知道节点类型能够通过数字表示的情况下。

### 2. 父节点、子节点和同辈节点

DOM2核心中的大多数属性和方法都涉及在树形结构中引用和创建节点。为了方便在树中定位，每个Node对象都有许多预定义的属性，分别引用树中的其他相关节点。这些属性中的每个属性引用的都是一个实际的DOM对象——但有一个例外，即childNodes，它引用的是包含DOM对象的一个类数组的NodeList对象。

在这些属性中，parentNode引用指定节点的直接父节点，即图3-7中箭头所指的DOM元素。

www.TopSage.com

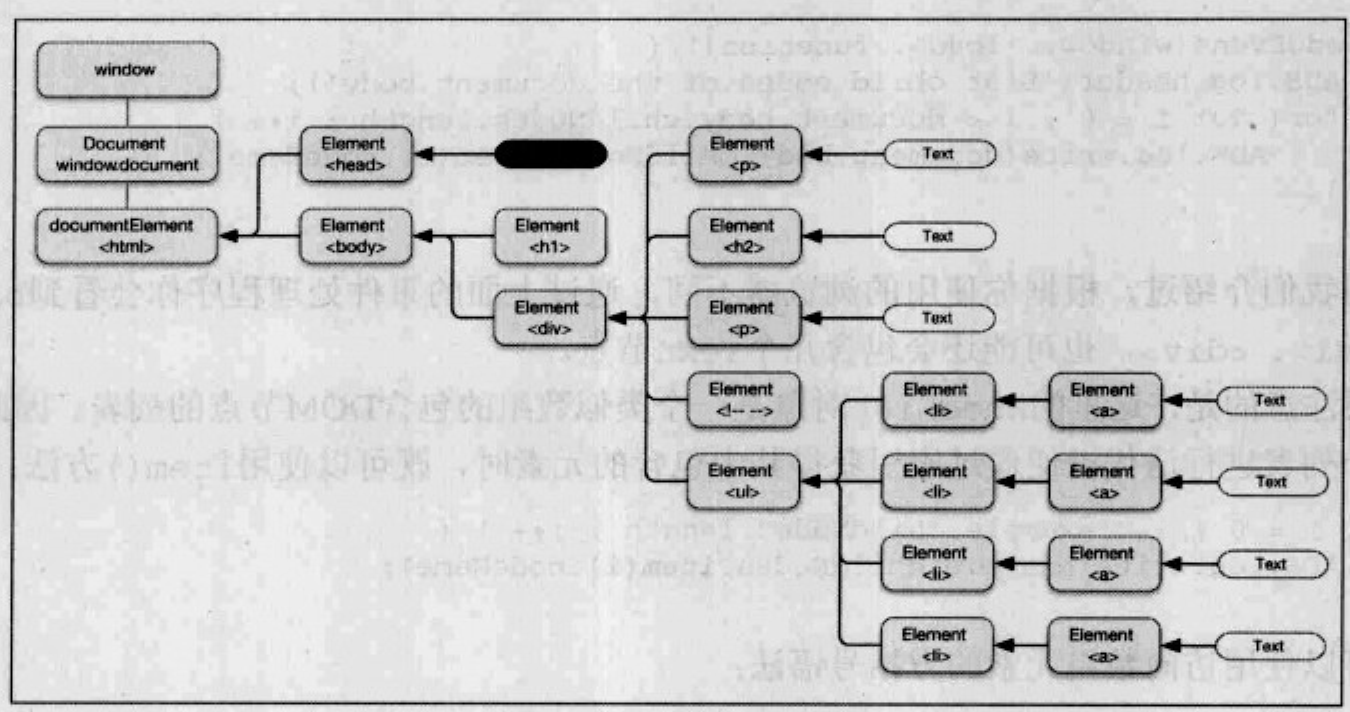



图3-7　DOM核心表现parentNode关系的方式

而childNodes属性引用的则是指定节点的所有子元素，如图3-8中箭头方向所示。

childNodes属性是一个通过数组中以数字索引的元素来表示子节点的NodeList对象。该对象中第一个子元素的索引为0：

```
var first = document.getElementById('browserList').childNodes[0];
```

最后一个子元素的索引为childNodes.length-1：

```
var list = document.getElementById('browserList');
var last = list.childNodes[list.childNodes.length-1];
```

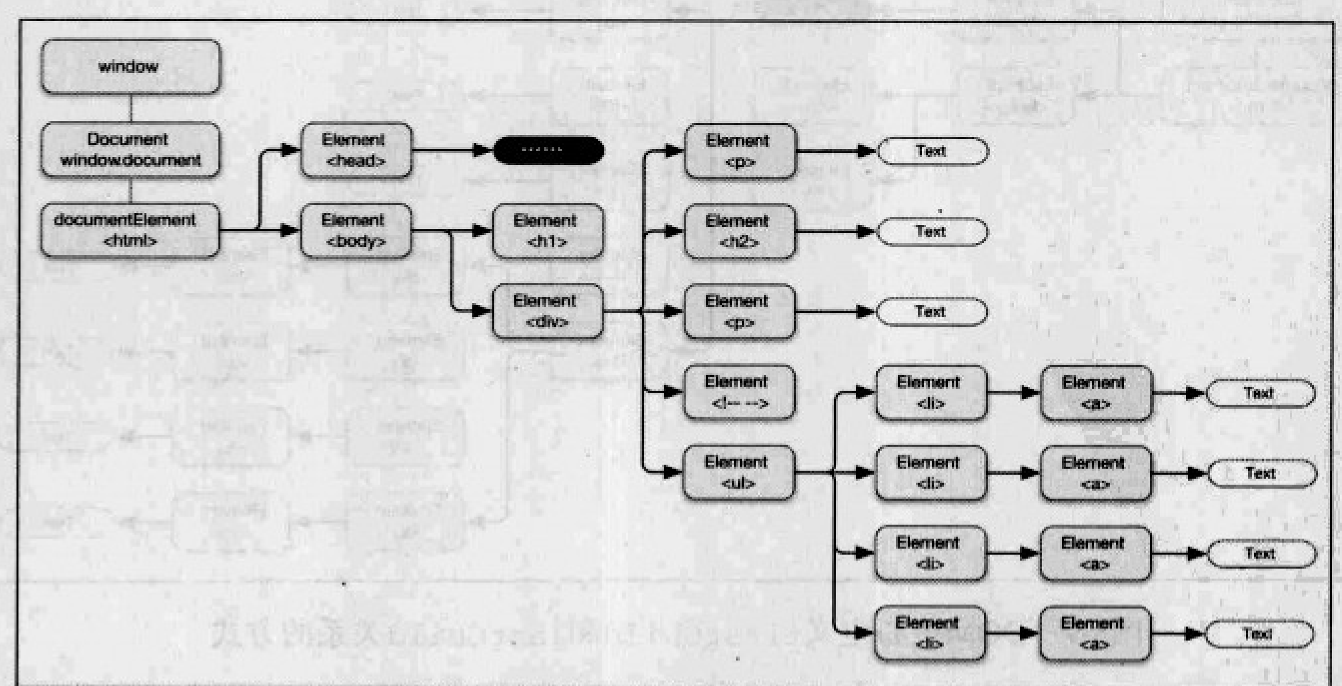



图3-8　DOM核心表现childNodes关系的方式

要列出body元素的所有子节点，可以使用下面的载入事件处理程序：

www.TopSage.com

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('List child nodes of the document body');
    for( var i = 0 ; i < document.body.childNodes.length ; i++ ) {
        ADS.log.write(document.body.childNodes.item(i).nodeName);
    }
});
```

前面我们介绍过，根据你使用的浏览器不同，通过上面的事件处理程序你会看到body元素中包含<h1>、<div>，也可能还会包含几个Text节点。

需要注意的是，这里的NodeList对象是一个类似数组的包含DOM节点的列表。因此，当需要对这个列表进行迭代以便通过索引获得其中包含的元素时，既可以使用item()方法：

```
for( i = 0 ; i < example.childNodes.length ; i++ ) {
    ADS.log.write(example.childNodes.item(i).nodeName);
}
```

也可以使用访问数组元素的方括号语法：

```
for( i = 0 ; i < example.childNodes.length ; i++ ) {
    ADS.log.write(example.childNodes[i].nodeName);
}
```

如图3-9所示，Node对象中也包含分别引用第一个和最后一个子节点的firstChild和lastChild属性。在只有一个子节点的情况下，firstChild和lastChild引用的是同一个节点。同样，不要忘了Text节点。在Firefox中，sample.html文件中的第一个和最后一个子节点几乎总是引用一个包含缩进空白符的文本节点。

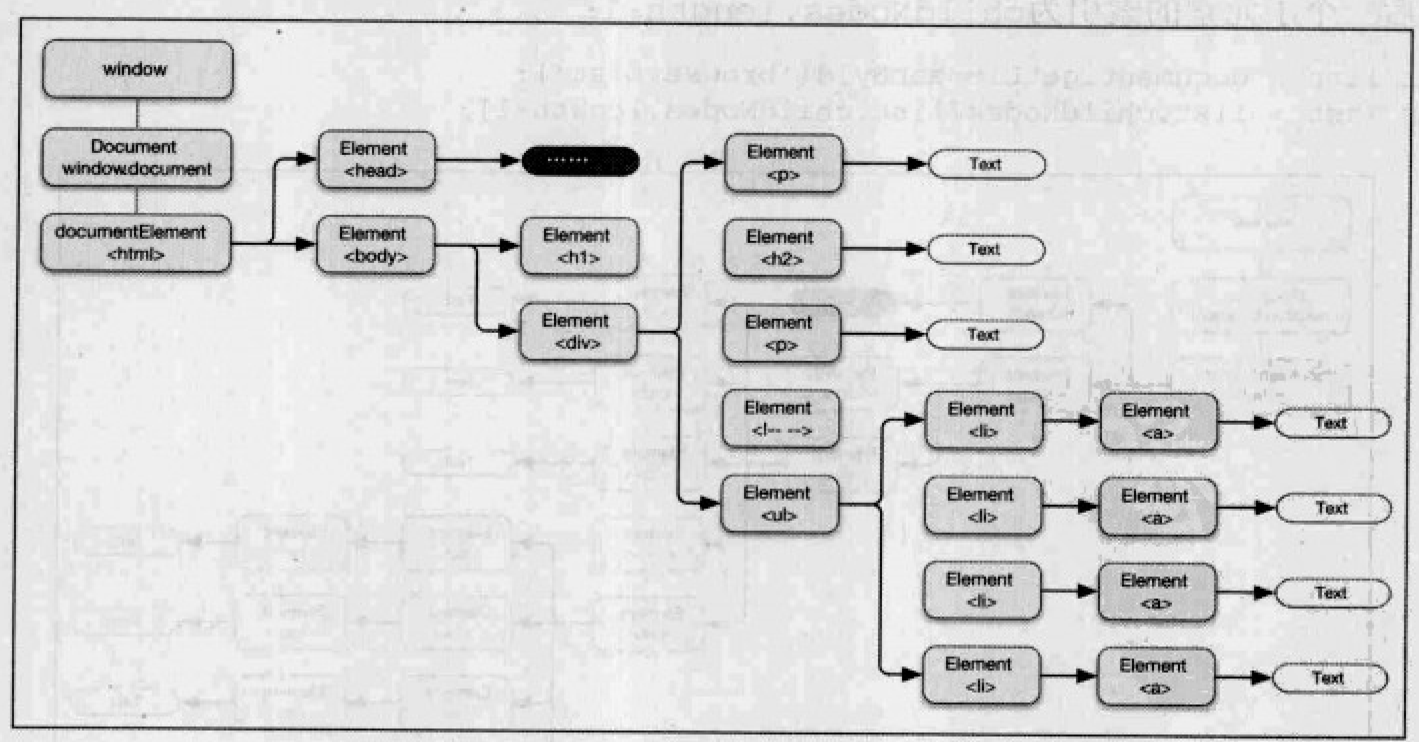



图3-9　DOM核心定义firstChild和lastChild关系的方式

同样地，previousSibling和nextSibling属性分别引用与当前节点前后紧邻的同辈节点，如图3-10所示。如果当前节点是该级别中的第一个节点，那么previousSibling的值为null。同理，如果当前节点是最后一个节点，那么nextSibling则为null。

www.TopSage.com

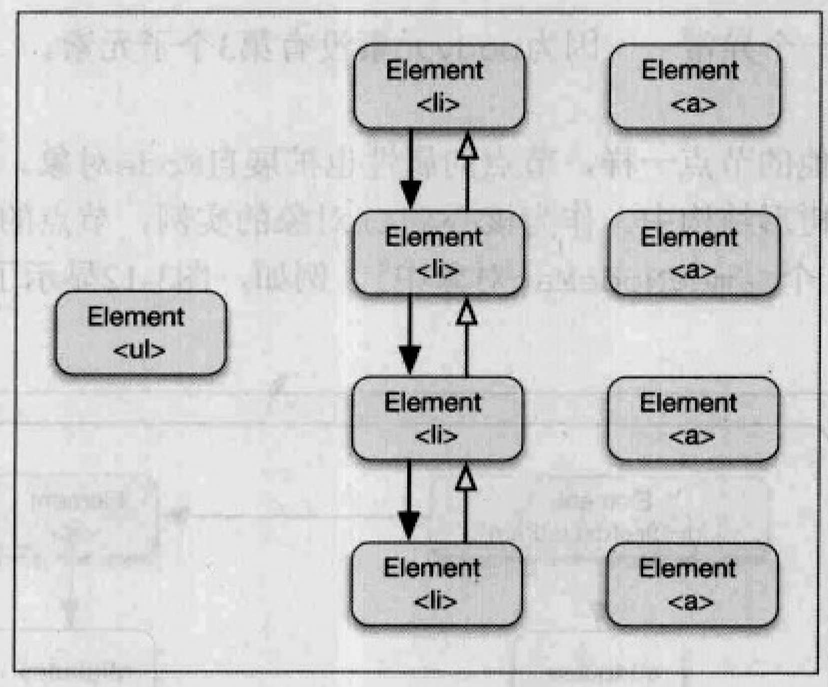



图3-10 DOM核心定义previousSibling和nextSibling关系的方式

你可以随意在domTesting.js文件中试验前面介绍的各种属性，看看都能找到哪些对象。与此同时，你也会进一步体验到在文档的树形结构中遍历和操纵子节点或同辈节点将是一件很费时的事情，所以理解如何访问和引用文档中的每个部分是非常重要的。

可以使用常见的JavaScript连缀语法拼加多个方法。例如，下面这行代码引用的是body元素的第 4 个子元素（其中包括文本节点），如图3-11所示：

```
document.body.firstChild.nextSibling.nextSibling.nextSibling
```

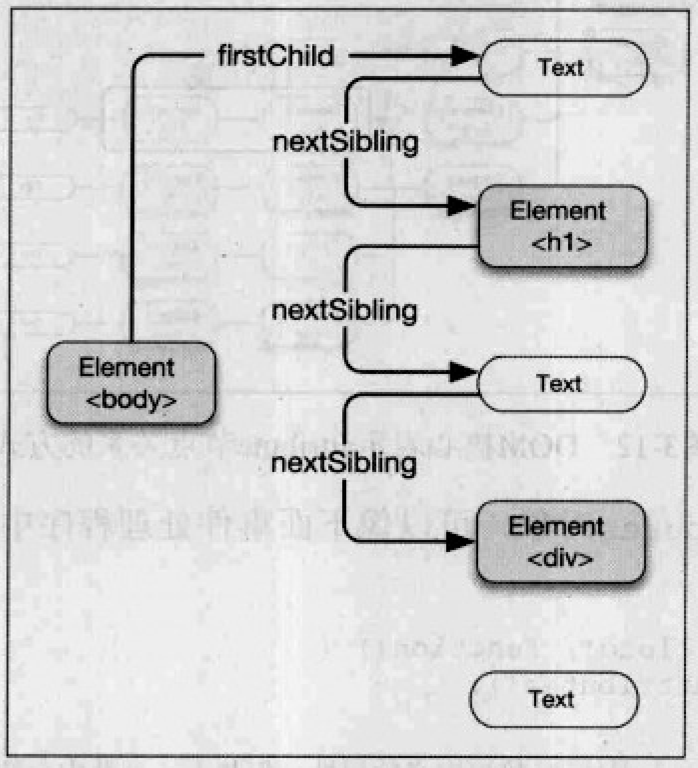



图3-11 通过依次使用多个nextSibling方法来引用节点

如果你在IE中试验这行代码，那么因为不存在Text节点，所以DOM脚本将会在方法链的第2

www.TopSage.com

个nextSibling处抛出一个异常——因为body元素没有第3个子元素。

3. 节点的属性

如同DOM文档中其他的节点一样，节点的属性也扩展自Node对象。但是，它们却不包含在通常的表示父/子关系的树形结构中。作为核心Attr对象的实例，节点的属性被包含在相应节点的attributes成员的一个NamedNodeMap对象中[①]。例如，图3-12显示了Firefox锚标签（<a>）的attributes。

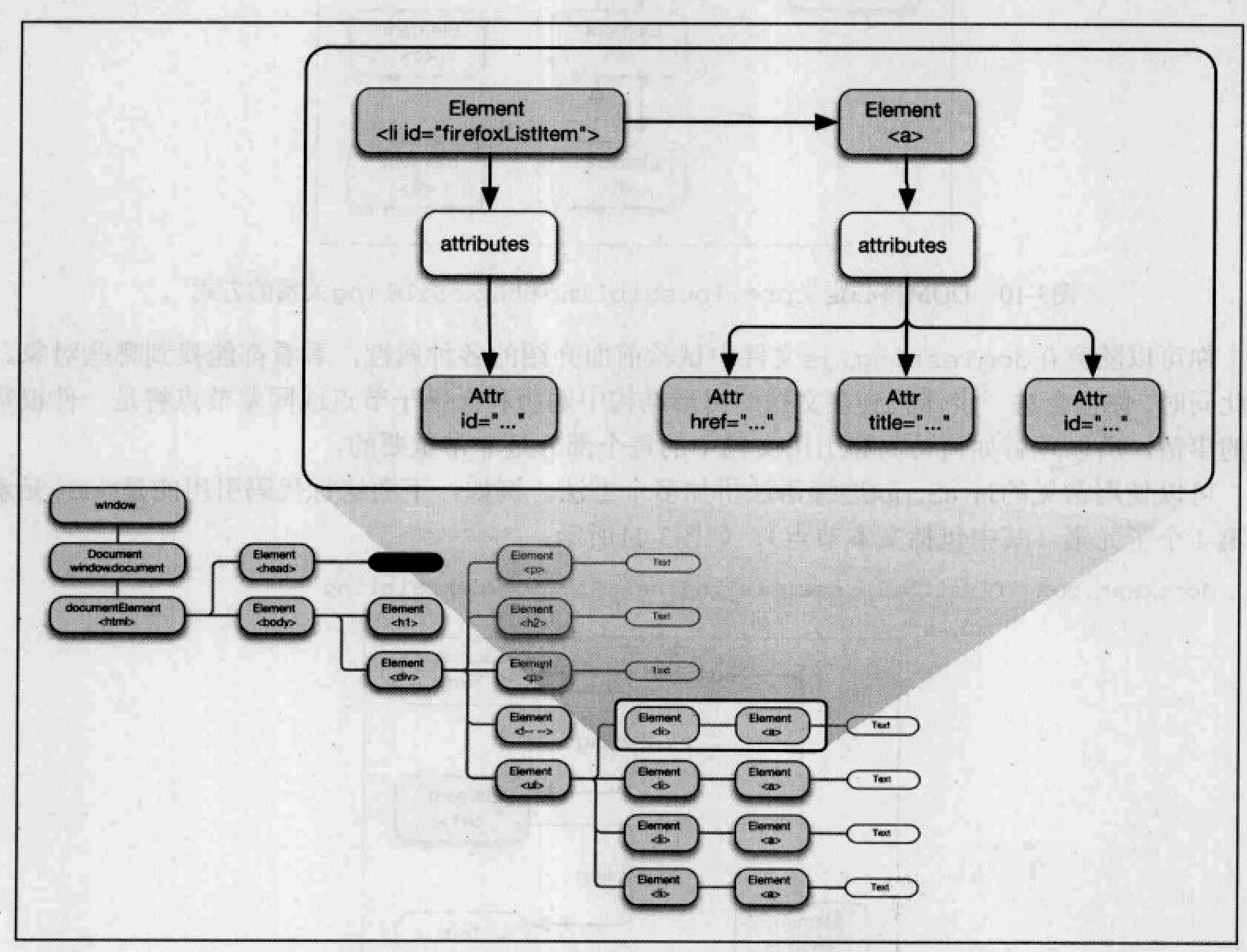



图3-12　DOM核心表示attribute节点关系的方式

图3-12中所示的attributes对象，可以像下面事件处理程序中一样通过锚的attributes成员来访问：

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('Attributes');
```

① 在DOM2核心中，attributes是Node接口定义的属性。但与上一小节中介绍的parentNode、firstChild、previousSibling等Node接口定义的属性不同，attributes还实现了Attr和NamedNodeMap接口。顾名思义，Node接口定义的这个attributes属性，表示的就是节点（特别是元素节点、如XML和HTML中元素）的属性——事实上，attributes中包含的是一个节点的所有属性的集合。

www.TopSage.com

```
    var firefoxAnchor = document.getElementById('firefox');
    for(var i=0 ; i < firefoxAnchor.attributes.length ; i++) {
        ADS.log.write(
            firefoxAnchor.attributes.item(i).nodeName
                +'='
                + firefoxAnchor.attributes.item(i).nodeValue
        );
    }
});
```

根据你使用的浏览器不同，日志窗口显示的结果中除包含Firefox锚中明确设置的以下3个属性外，还可能包含很多其他属性：

- id = firefox
- title = Get Firefox
- href = http://www.getfirefox.com/

在很多浏览器中，attributes成员也将包含许多专有的或DHTML属性。

与Comment节点类似，Attr节点的值也可以使用nodeValue属性来取得（如前面例子所示）。此外，Attr节点中还包含一个Text子节点，这个Text节点中包含着与nodeValue中相同的值。

与NodeList对象类似，NamedNodeMap对象中的项也可以使用方括号语法来访问，因此下面这行代码与前面事件处理程序中的代码会得到相同的结果：

```
firefoxAnchor.attributes[i].nodeName
```

但是，与NodeList对象不同的是，NamedNodeMap对象还具有其他一些便捷的方法。例如，如果要从包含节点属性（集合）的NamedNodeMap对象中取得一个特殊的属性，可以通过getNamed-Item()方法取得指向具体属性节点的引用：

```
var link = firefoxAnchor.attributes.getNamedItem('href').nodeValue;
```

这个方法与Element对象的getAttribute()方法类似，你可能对后者更熟悉一些，不过attributes.getNamedItem()方法在任何节点上都是有效的，也包括那些不是Element对象实例的节点。

#### 4. 节点的ownerDocument属性

一个节点的ownerDocument属性类似于指向节点所属根文档的引用。大多数情况下，都可以通过它在作用域链中引用document，或者window.document，因为浏览器中只会有一个document的实例。但是，假如你像下面例子中那样在某个自定义的对象内部覆盖了document对象，并使它引用其他值：

```
function example(node) {

    // 在作用域链中覆盖document
    // 以使它引用其他值
    var document = 'something else';

    // 使用ownerDocument引用原始的DOM文档
    var anotherNode = node.ownerDocument.getElementById('id');
```

www.TopSage.com

```
    // 下面这行代码将会出错，因为document现在是一个字符串，而非DOM文档
    // 因此getElementById方法在当前的document中是不存在的
    var anotherNode = document.getElementById('id');

}
```

那么，通过使用传递到对象内部的DOM Node对象的ownerDocument属性，仍然可以访问到原始的document。

#### 5. 检测子节点和属性

如果你需要简单地检测一下某个节点是否具有子节点或属性，那么可以使用hasChildNodes()和hasAttributes()方法，如下面的载入事件处理程序中所示：

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('Attributes And ChildNodes');

    var h2 = document.getElementsByTagName('H2')[0];

    ADS.log.write(h2.nodeName);
    ADS.log.write(h2.nodeName + ' hasChildNodes: '
        + h2.hasChildNodes());
    ADS.log.write(h2.nodeName + ' childNodes: ' + h2.childNodes);
    ADS.log.write(h2.nodeName + ' number of childNodes: '
        + h2.childNodes.length);

    ADS.log.write(h2.nodeName + ' attributes: ' + h2.attributes);
    ADS.log.write(h2.nodeName + ' number of attributes: '
        + h2.attributes.length);

    // 下面这行在MSIE中会出错
    ADS.log.write(h2.nodeName + ' hasAttributes: '
        + h2.hasAttributes());
});
```

运行以上代码后，你的日志窗口中将显示下列结果。

- H2
- H2 hasChildNodes: true
- H2 childNodes: [object NodeList]
- H2 number of childNodes: 1
- H2 attributes: [object NamedNodeMap]
- H2 number of attributes: 0
- H2 hasAttributes: false

你会注意到，在Firefox、Opera以及Safari中，即使节点中没有定义的属性，其attributes属性仍然是有效的而且长度值为0。同样，对于这些浏览器中节点的childNodes属性也是如此。当你只想知道某个节点是否包含什么时，使用hasChildNodes()和hasAttributes()方法是非常便利的。但是，如果你还想知道节点中包含多少个子节点或属性（包括零个），那就仍然要使用childNodes和attributes属性。不过，由于IE中不存在hasAttributes()方法，所以当前面的例子在IE中运行时会在最后一个日志项上出现错误：

www.TopSage.com

对象不支持此属性或方法

此外，正如前面所讨论的，attributes会被报告为Object。而且，与这个特定的节点相关的属性大概会有83个，而其中大多数都是IE专有的属性。

#### 6. 操纵DOM节点树

你的大多数DOM脚本的主要任务就是在DOM文档中插入、删除和移动节点。而用于完成这些操作的成员，例如appendChild()和insertBefore()[①]，都是本章到目前为止一直在讨论的Node对象的成员。向Node对象中添加新的子节点相对简单，而且我们在前面的例子中也多次用到过appendChild(newChild)方法：

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('Append Child');
    var newChild = document.createElement('LI')
    newChild.appendChild(document.createTextNode('A new list item!'));
    var list = document.getElementById('browserList');
    list.appendChild(newChild);
});
```

按照定义，添加操作总是把新节点添加到列表的末尾，但如果想要添加到开头，甚至中间该怎么办呢？这就需要使用insertBefore(newChild,refChild)方法了，该方法会在引用的子节点之前的位置插入一个新节点。要在sample.html的列表中倒数第2个元素的位置上插入一个新节点，可以使用以下事件处理程序：

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('Insert Before');
    var newChild = document.createElement('LI')
    newChild.appendChild(document.createTextNode('A new list item!'));
    var list = document.getElementById('browserList');
    list.insertBefore(newChild,list.lastChild);
});
```

这样，就会得到如图3-13所示的新列表项。

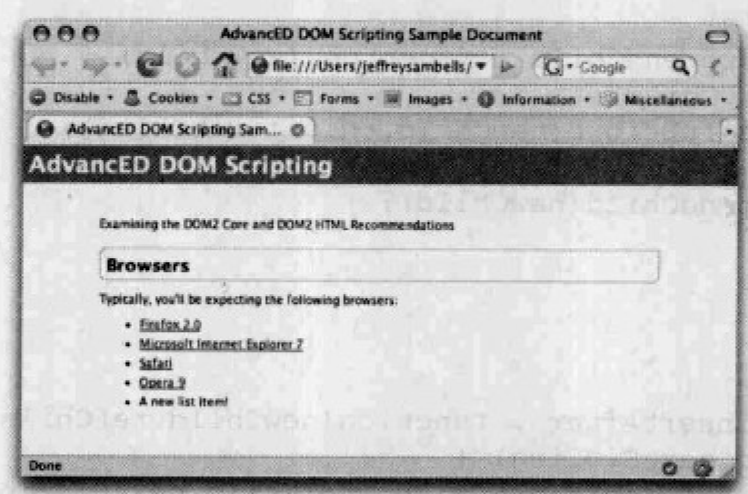



图3-13 Firefox中显示的新插入的节点

且慢！如果你仔细看一看图3-13，就会发现大概是什么地方出错了。我们前面说的是“在倒

① 原文childNodes在上一节刚刚介绍过，且不是操纵节点的方法，疑有误，故将其替换为相关方法。——译者注

www.TopSage.com

数第2个元素的位置上插入一个新节点”。如果你是在Microsoft IE中运行上面的脚本，那么就会看到如图 3-14所示的正确结果，即新节点出现在了倒数第2个元素的位置上。

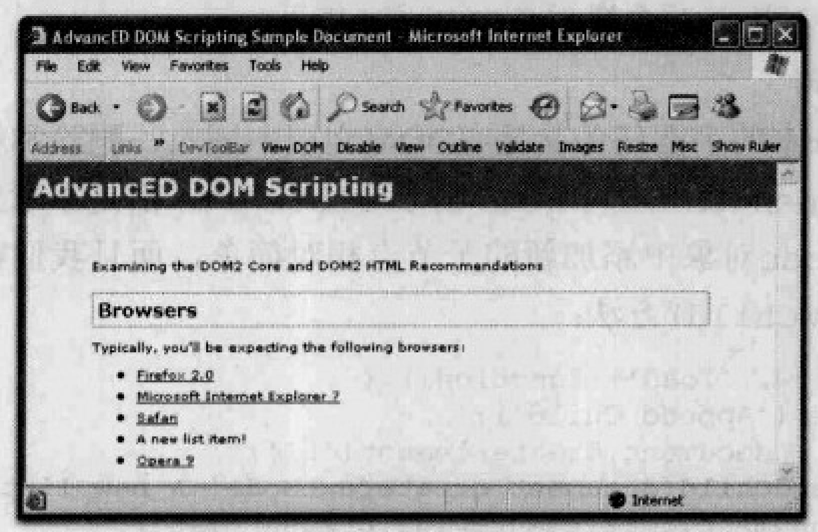



图3-14　Microsoft IE中显示的新插入的节点

事实上，Firefox、Opera、Safari以及图3-13都是完全正确的，而且新节点也的确是被插入在了倒数第2个元素的位置上，问题的原因是前面的脚本没有考虑到由空白字符组成的`Text`节点。我们在前面提到过，IE会忽略空白符，所以`lastChild`引用的是最后一个列表元素`<li>`；而在其他浏览器中，`lastChild`引用的都是最后一个`Text`节点。所以，在编写操纵DOM的脚本时，一定要在必要时将空白符考虑在内。

出于某种原因，W3C决定在核心规范中不包含`prependChild()`和`insertAfter()`方法。不过，正如你在第1章中向ADS库中添加这两个方法时所看到的，可以使用现有的方法轻松地完成相同的任务。

你可能还会想到另外一种方式，即像第2章中操纵对象那样，将这两个方法添加到`Node`对象的`prototype`属性中，大致的代码如下：

```
if(!Node.prependChild)
    Node.prototype.prependChild = function (newChild) {
        if(this.firstChild) {
            this.insertBefore(newChild,this.firstChild);
        } else {
            this.appendChild(newChild);
        }
        return this;
    }
}
if(!Node.insertAfter) {
    Node.prototype.insertAfter = function(newChild,refChild) {
        if (refChild.nextSibling) {
            this.insertBefore(newChild, refChild.nextSibling);
        } else {
            this.appendChild(newChild);
        }
        return this;
    }
}
```

这似乎是一个不错的主意，但是，其中存在几个问题。

- 前面我们曾解释过，IE没有本地的Node对象，因此对IE而言这些方法是无效的。你可能会想到针对IE来修改Object.prototype，但那样做会使所有对象都继承这两个方法，而这两个方法并不适用于所有对象。
- 在Safari中，确实存在本地的Node对象，但Safari却不允许你修改这个本地对象的原型——因此，这种做法是行不通的。
- 而且，即使你能够修改原型对象，也意味着会覆盖将来加入到DOM规范中的同名方法。所以，最佳方式还是像我们在第1章和第2章中介绍的那样，在你自己对象的命名空间内部将这两个方法作为私有方法来创建。

对于替换和删除一个节点的操作，通过使用replaceChild(newChild,oldChild)和removeChild(oldChild)方法则相当容易一些。例如，通过下面的载入事件处理程序将sample.html中的Firefox列表项替换成一个新节点的操作，就和创建一个节点并将其添加到列表中一样简单：

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('Replace a node');
    var newChild = document.createElement('LI')
    newChild.appendChild(document.createTextNode('A new list item!'));
    var firefoxLi = document.getElementById('firefoxListItem');
    firefoxLi.parentNode.replaceChild(newChild,firefoxLi);
});
```

而要删除Firefox列表项，只需在相应节点的父节点上调用removeChild()方法即可：

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('Remove a node');
    var firefoxLi = document.getElementById('firefoxListItem');
    firefoxLi.parentNode.removeChild(firefoxLi);
});
```

这里需要注意的一个关键问题是，replaceChild()和removeChild()都是用来操纵某个节点的子节点的方法，因此不能在你想要替换或删除的节点上直接调用它们。

**7. 复制和移动节点**

DOM脚本新手常犯的另一个错误的根源，是没有理解像document.getElementById()这样的方法返回的是对Node对象的引用，而不是相应对象的副本。如果你在domTesting.js文件中添加了下面的事件处理程序，那么运行该脚本之后会有几个Firefox列表项呢？

```
ADS.addEvent(window, 'load', function() {
    ADS.log.header('Clone and Move a Node');
    var firefoxLi = document.getElementById('firefoxListItem');
    var firefoxLiClone = firefoxLi.cloneNode(true);
    var unorderedList = firefoxLi.parentNode;

    // 添加到列表中
    unorderedList.appendChild(firefoxLi);
    // 添加到列表中
    unorderedList.appendChild(firefoxLiClone);
});
```

www.TopSage.com

你可能认为会有3个：原来的1个和新添加的2个。正确的答案是2个。当载入完成后，页面将如图3-15所示。其中，列表底部有两个Firefox条目，而上面则没有了。

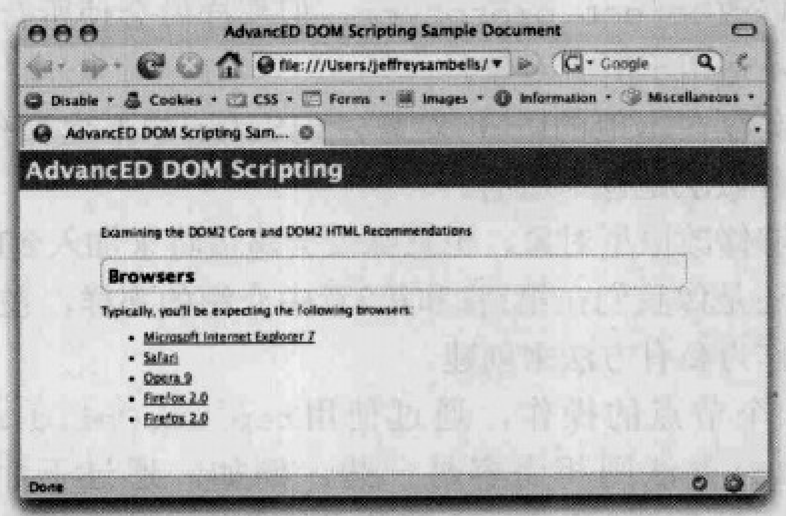



图3-15　当把复制的Firefox列表项也添加到不同的位置后sample.html在浏览器中的视图

当像下面这样通过document.getElementById()方法取得并把节点赋值给firefoxLi变量后，你取得的是一个指向该节点的引用，而非节点的副本：

```
var firefoxLi = document.getElementById('firefoxListItem');
```

随后，当把这个引用添加到列表中时，appendChild()方法会取得引用所指向的firefoxLi节点并将它添加到无序列表的末尾：

```
unorderedList.appendChild(firefoxLi);
```

也就是说，你在这里是让该方法向列表中添加已经在文档中存在的一个节点的引用，所以节点被移动（而不是被复制）到了新位置。

## 3.4.3　核心Element对象

介绍完Node对象，下面就该介绍Element对象了。Element对象及其方法将是你在日常DOM脚本编程中最常用的。如图3-4所展示的，DOM文档树形结构的主体部分都是由Element节点构成的（但其中特殊的DOM2 HTML对象还会进一步扩展Element对象）。所有Element对象都拥有Node对象的属性和方法，同时还有其他一些便于操纵节点属性和查找子Element对象的方法，以下几节将分别介绍这些方法。

### 1. 操纵Element对象的属性

为了简化对attributes的处理，Element对象中包含了很多种用来操纵基础的Node对象的attributes属性的方法，如下。

- getAttribute(name)方法基于一个字符串形式的属性名称取得相应属性的值，例如：

```
ADS.addEvent(window,'load',function() {
    ADS.log.header('Get Safari href attribute');
    var safariAnchor = document.getElementById('safari');
    var href = safariAnchor.getAttribute('href');
    ADS.log.write(href);
```

www.TopSage.com

```
});
```

- setAttribute(name,value)方法基于一个字符串形式的属性名称设置相应属性的值，例如：

```
ADS.addEvent(window,'load',function() {
    ADS.log.header('Set Safari title attribute');
    var safariAnchor = document.getElementById('safari');
    safariAnchor.setAttribute('title','Safari is for Mac OS X');
});
```

- removeAttribute(name)方法基于一个字符串形式的属性名称删除相应属性的值，例如：

```
ADS.addEvent(window,'load',function() {
    ADS.log.header('Remove Firefox title attribute');
    var firefox = document.getElementById('firefoxListItem');
    firefox.removeAttribute('title');
});
```

类似地，也有一些基于实际的DOM Attr节点对象，而不是基于字符串形式的名称和值来操纵属性的方法：

- getAttributeNode(name)方法取得指定属性的Attr节点。
- setAttributeNode(newAttr)方法基于新的Attr对象的实例设置属性。
- removeAttributeNode(oldAttr)方法删除指定的属性节点，与使用removeChild()方法删除一个子节点的方式相同。

> 与Node对象相似，Element对象也有一些针对命名空间的方法。由于这些方法并没有在所有现代浏览器中实现，所以这里就不作介绍了。

### 2. 在Element对象中查找Element对象

在Element对象的范围内，可以用来查找其他节点的唯一有效方法就是getElementsByTagName()。getElementsByTagName()方法返回的是一个NodeList对象，其中包含与给定标签名称匹配的所有祖先元素的引用。在最终的NodeList对象列表中，所有元素的都是按照它们在DOM文档中出现的先后顺序排列的（就像你从左到右，自上而下地阅读本书的顺序一样）。在getElementsByTagName()方法中，也可以使用星号（*）作为通配符来匹配所有元素的标签名称。可以试试下面的载入事件处理程序：

```
ADS.addEvent(window,'load',function() {
    ADS.log.header('Get all browserList elements by tag name');
    var list = document.getElementById('browserList');
    var ancestors = list.getElementsByTagName('*');
    for(i = 0 ; i < ancestors.length ; i++ ) {
        ADS.log.write(ancestors.item(i).nodeName);
    }
});
```

假如示例文档sample.html没有被任何载入事件处理程序修改过，那么你会看到下面所示的4个列表项及其子元素的列表，排列顺序与它们在文档中出现的顺序一致：

- LI
- A
- LI

www.TopSage.com

- A
- LI
- A
- LI
- A

在观察这一结果的同时，你也会注意到*通配符不会匹配位于锚标签内部的文本节点。事实上，正如getElementsByTagName()方法名称中的“Elements”所指出的，该方法返回的节点中只包含Element节点，不包含其他类型的节点。

Element对象中没有用于创建新Element对象的方法。创建新DOM元素的操作完全由Document对象负责处理，下面我们继续介绍Document对象。

### 3.4.4　核心Document对象

我们在第2章中介绍过，JavaScript的全局对象是window对象。而在讲解DOM时，我们要讨论的则是window对象的document属性，或者说window.document。DOM核心规范中的Document对象本身也扩展自Node对象，因此我们在前面提到的Node对象的所有属性和方法，也都适用于Document对象。

在W3C DOM规范诞生之前，window.document就已经在DHTML中存在了。因此，算上W3C DOM的方法在内，window.document对象中也可能包含其他浏览器专有的成员，例如document.all。不过只要可能，就应该避免使用专有的、非标准的成员，而采用基于标准的方法。

如果浏览器不支持标准的方法，但它将来可能会支持，那么你可以编写代码来模仿标准的方法。例如，假设你的网站需要支持像IE 4这样古老的浏览器，但该浏览器只支持非标准的document.all，不支持标准的document.getElementById()方法。那么，只要使用一些简单的对象检测，在查询all方法的脚本开始处添加一个自定义的document.getElementById()方法即可：

```
if(document.all && !document.getElementById) {
    document.getElementById = function(id) {
        return document.all[id];
    }
}
```

在添加以上代码后，如果存在document.all且不存在document.getElementById()，那么就创建一个与标准方法执行相同任务的getElementById()方法。然后，你就可以放心地使用标准方法来进行编码了，而且即使是罕见的IE 4出现了也完全没有问题。

> 由于必须对需要支持的每个方法都进行如此的包装，所以你必须事先决定要完整地支持哪个浏览器，而对哪个浏览器则只能给出自己的平稳退化版。

#### 1. document.documentElement属性

document.documentElement属性是访问文档根元素的快捷方式。对于在浏览器中呈现的

www.TopSage.com

HTML文档而言，所谓的根元素就是<html>标签。

2. 使用Document对象的方法创建节点

Document对象中包含很多可以用来创建DOM核心中各种节点类型的新实例的方法。这些节点类型中既包含我们已经讨论过的Element，也包含其他一些节点类型，如下。

- createAttribute(name)：创建节点类型为Node.ATTRIBUTE_NODE的Attr节点。
- createCDATASection(data)：创建节点类型为Node.CDATA_SECTION_NODE的CDATASection节点。
- createComment(data)：创建节点类型为Node.COMMENT_NODE的Comment节点。
- createDocumentFragment()：创建节点类型为Node.DOCUMENT_FRAGMENT_NODE的DocumentFragment节点。
- createElement(tagName)：创建节点类型为Node.ELEMENT_NODE的Element节点。
- createEntityReference(name)：创建节点类型为Node.ENTITY_REFERENCE_NODE的EntityReference节点。
- createProcessingInstruction(target,data)：创建节点类型为Node.PROCESSING_INSTRUCTION_NODE的ProcessingInstruction节点。
- createTextNode(data)：创建节点类型为Node.TEXT_NODE的Text节点。

还有其他一些涉及命名空间的节点类型，但它们无法与所有浏览器兼容，所以这里也不作介绍。

无论何时，当需要在文档树中创建一个新的分支或节点时，都需要在代码中用到这些create*方法。但大数情况下，你都会使用createElement()和createAttribute()。不过，其他方法也都会派上用场。

3. 使用Document对象的方法查找Element对象

核心Document对象中另外两个重要的方法是getElementsByTagName()和getElementById()。

对于getElementById()方法，大概你早已知道它返回的是与你要求的ID对应的元素。即调用下面的方法:

```
document.getElementById('outer-wrapper');
```

将会返回带有id="outer-wrapper"属性的Element对象。这里有必要指出的是，getElementById()方法是单一用途的方法，它只返回一个元素。虽然浏览器通常会默许，但在有效的HTML文档中则不应该包含相同ID的多个的实例。也就是说，为了避免在使用getElementById()方法时造成混乱，任何ID在整个文档中都必须保持唯一性。

虽然Document对象的getElementsByTagName()方法与Element对象的同名方法功能相同，不过从技术上讲，它们并不是同一个函数。Document对象虽然不是继承自Element对象的，但它却包含了功能相同的getElementsByTagName()方法，因而可以用这个方法来查询整个文档。

此时此刻，你可能会奇怪为什么没有原生的getElementsByClassName()方法。是这样，这个方法之所以没有包含在DOM核心中，是因为类属性并不适用于所有类型的XML文档。因而，这个差别也是为什么理解规范有助于理解问题实质的原因之一。getElementsByClassName()方法的确只适用于HTML文档，而不适用于XML文档。原因是虽然XML文档中也可以带有class属性，但那

www.TopSage.com

却不是XML规范所规定的。从另一方面来看，HTML文档中的HTMLElement对象中却有一个规范规定的className属性。因此getElementsByClassName()方法在DOM2 HTML规范中是有意义的。但是，W3C决定不包含这个方法，所以你只能像在第1章中所做的那样被迫创建自己的版本。

### 3.4.5 遍历和迭代DOM树

在你的许多脚本中，都需要以某种方式来检查DOM元素。而其中最频繁发生的，则是在文档树中递归地检查每个节点及其子节点。在理想的世界里，你应该能够使用DOM2遍历和范围规范中的对象来完成对文档树的迭代操作，因为这样能够提供一种跨浏览器并且与标准兼容的方法，同时确保脚本总能正确地运行（别忘了考虑到IE中缺少的文本节点）。但是，遍历和范围规范中的对象在各种浏览器中只得到了零散地实现，因此如果你想让自己的脚本能够到处运行，那么就必须拿出自己的解决方案。

要创建你自己的递归"爬树（tree-walking）"方法的最简单方式，就是像我们在第2章中曾经讨论过的那样创建一个普通的函数，让它来完成遍历并在每个节点上执行对一个匿名函数的调用。为了在学习本书其他内容的时候方便地使用这些方法，现在最好就把它们添加到你的ADS库中。

如果你不关心节点在DOM树中的深度，那么可以使用document.getElementsByTagName('*')方法取得指定节点中的所有Element节点，并循环遍历这些节点：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

    ……以上是库中已有的内容……

    function walkElementsLinear(func,node) {
        var root = node || window.document;
        var nodes = root.getElementsByTagName("*");
        for(var i = 0 ; i < nodes.length ; i++) {
            func.call(nodes[i]);
        }
    }

    ……以下是库中已有的内容……

})();
```

之所以把上面的方法命名为walkElementsLinear()，是因为它的查找方式不是递归地。换句话说，它只是通过getElementsByTagName()方法取得了一个包含Element对象的列表。

而如果你希望跟踪节点的深度，或者构建一个路径，那么可以通过递归的方式来遍历DOM树：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

    ……以上是库中已有的内容……

    function walkTheDOMRecursive(func,node,depth,returnedFromParent) {
        var root = node || window.document;
        var returnedFromParent = func.call(root,depth++,returnedFromParent);
        var node = root.firstChild;
```

www.TopSage.com

```
        while(node) {
            walkTheDOMRecursive(func,node①,depth,returnedFromParent);
            node = node.nextSibling;
        }
    };
    ……以下是库中已有的内容……

})();
```

你可能还希望一道查找每个节点的属性：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

    ……以上是库中已有的内容……

    function walkTheDOMWithAttributes(node,func,depth,returnedFromParent) {
        var root = node || window.document;
        returnedFromParent = func(root,depth++,returnedFromParent);
        if (root.attributes) {
            for(var i=0; i < root.attributes.length; i++) {
                walkTheDOMWithAttributes(root.attributes[i],
                        func,depth-1,returnedFromParent);
            }
        }
        if(root.nodeType != ADS.node.ATTRIBUTE_NODE) {
            node = root.firstChild;
            while(node) {
                walkTheDOMWithAttributes(node,func,depth,returnedFromParent);
                node = node.nextSibling;
            }
        }
    };
    ……以下是库中已有的内容……
})();
```

这些方法都不是唯一可用的方法，因为实现同一任务的可能方式是多种多样的。比如，也可以采用Douglas Crockford（http://javascript.crockford.com）编写的walkTheDom()函数：

```
function walkTheDOM(node, func) {
    func(node);
    node = node.firstChild;
    while (node) {
        walkTheDOM(node, func);
        node = node.nextSibling;
    }
}
```

不过，我个人更喜欢的还是前面的版本（称其为walkTheDOMRecursive()可以防止发生冲突），因为它将文档的根元素设为默认值，并在节点的子节点中调用函数。因此，你可以使用this来引用节点而不是一个传递给函数的参数。而且，使用call()也能够防止对递归函数的作用域进行意外操作。

① 原文root.childNodes[i]有误。——译者注

www.TopSage.com

## 3.5　DOM HTML

在本章所有小节中，如果要介绍完整的规范，那么DOM2 HTML这一节应该是最长的。DOM2 HTML规范确实非常长，因为它包含针对所有HTML元素的特定对象。本节会简单地讨论几个常用的对象以及它们如何与DOM核心协同运作，但不会介绍所有这些对象。因为这些对象基本上都是可以顾名思义的——假如逐一介绍所有对象你很可能会因为无聊而跳过这一章。

在本章开始时的图3-4中，我们看到了DOM2核心表现`sample.html`中文档的方式。图3-16则展示了根据DOM2 HTML规范表现同一文档的方式。

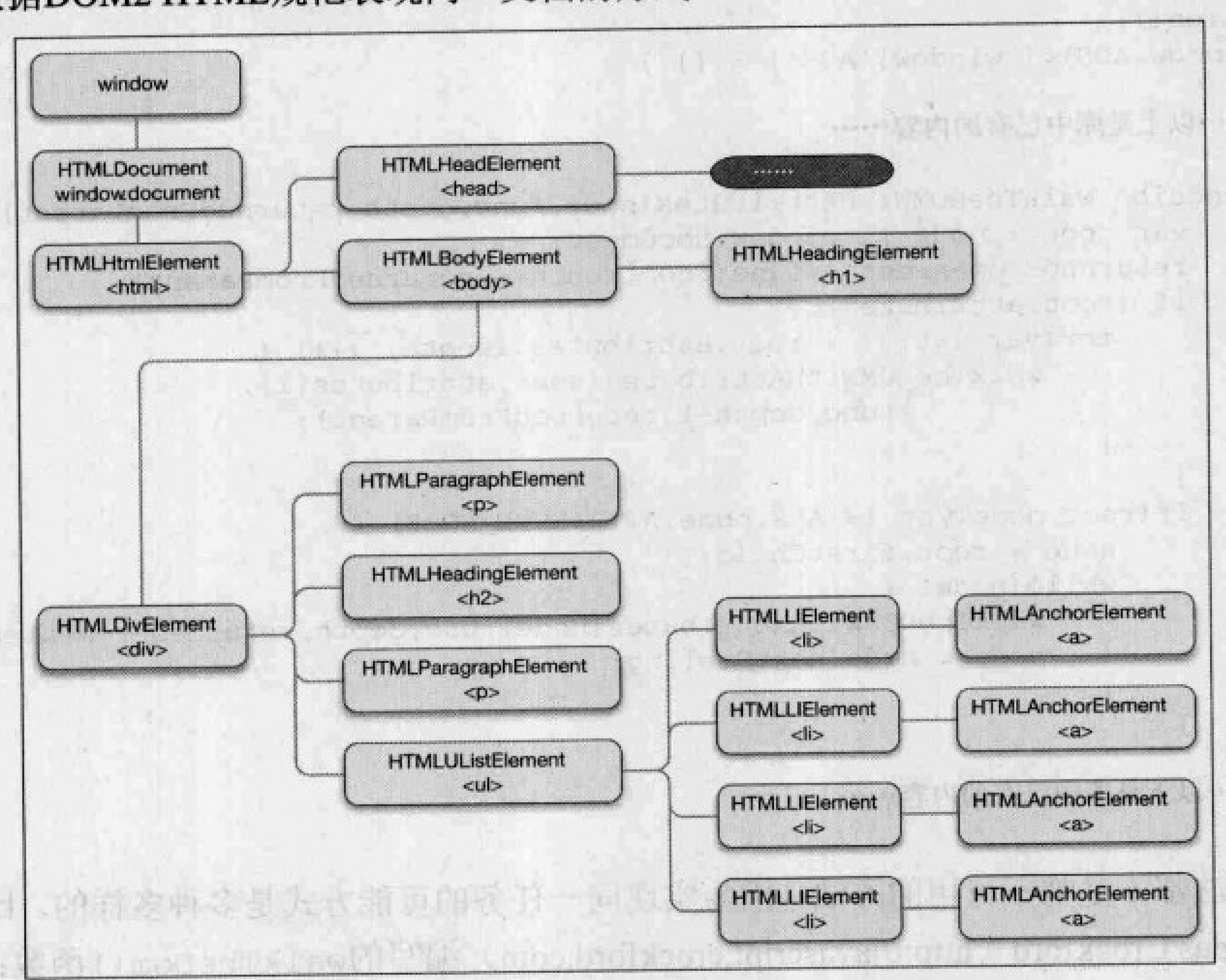



图3-16　DOM2 HTML规范表现`sample.html`中文档的方式

在这里要记住的一点是，此前我们介绍的DOM2核心对象仍然适用，因为每个DOM2 HTML对象都扩展自核心对象。

### 3.5.1　DOM2 HTML的`HTMLDocument`对象

当HTML文档呈现在浏览器中时，`window.document`中的DOM文档对象实际上是`HTMLDocument`对象的一个实例，如图3-16所示。`HTMLDocument`对象从核心的`Document`对象中继承了所有成员，而且还添加了一些你大概已经很熟悉的属性和方法。其中增加的属性（有一些也称为集合）如下。

- `title`：包含位于`<title>`标签中的字符串，对于`sample.html`而言就是AdvancED DOM Scripting。

www.TopSage.com

- referrer：包含链接到当前页面的前一个页面的URL。
- domain：包含当前站点的域名。
- URL：包含浏览器在查看当前页面时地址栏中的URL。
- body：引用从<body>节点开始的DOM树。
- images：是一个包含当前文档中所有<img>标签的数组（集合）。
- applets：是一个包含与当前文档中所有<applet>标签对应的DOM节点的数组（集合）。
- links：是一个包含与当前文档中所有<link>标签对应的DOM节点的数组（集合）。
- forms：是一个包含与当前文档中所有<form>标签对应的DOM节点的数组（集合）。
- anchors：是一个包含与当前文档中所有<a>标签对应的DOM节点的数组（集合）。
- cookie：是一个包含当前页面中所有cookie信息的字符串。

如果你仔细观察一下上面列出的属性，可能会发现缺少一些东西，例如frames、plugins、scripts、stylesheets等。虽然这些缺少的集合在不少浏览器中都可用，但它们却不是官方DOM2 HTML规范的内容。因此，应该像我们在前面讨论document.all时那样，使用其他基于标准的手段取得同样的信息。

HTMLDocument对象也包含如下一些方法。

- open()：打开一个文档以便接受write()或writeln()方法的输出。
- close()：关闭当前的文档。
- write(data)：将输入写入到文档中。
- writeln(data)：将输入写入文档的同时写入一个换行符。
- getElementsByName(elementName)：除了不使用标签名而使用name="example"属性外，与getElementsByTagName()的工作方式相同。

由此可见，document.write()方法实际上也是DOM2 HTML规范的内容，因此使用这个方法不违反标准。也就是说，可以使用下列代码创建一个文档：

```
var newDocument = document.open("text/html");
var markup = '<html>'
    + '<head><title>Write Example</title></head>'
    + '<body>This isn't really AdvancED!</body></html>';
newDocument.write(markup);
newDocument.close();
```

但这种方式仍然会替换当前文档中已有的内容，并导致其他浏览器的怪异行为（quirk）——这里不作讨论。此外，我们在第1章中也曾介绍过，document.write()方法还可能会导致其他坏习惯，所以本书将避免使用该方法。

### 3.5.2 DOM2 HTML 的 HTMLElement 对象

继承自DOM2核心中Element的HTMLElement对象，同样也添加了如下一些你可能比较熟悉的属性。

- id：包含可以供document.getElementById('inner-wrapper')方法使用的id="inner-wrapper"属性。

- title：用于进一步对元素进行语义化描述和悬停工具条。
- lang：是在RFC 1766[①]中为节点语言定义的语言代码。
- dir：表示节点中文本的方向（默认是表示“从左向右”的ltr）。
- className：包含用作CSS连接点（hook）的所有class="button"属性。

文档中的每个特定的标签都会通过一个针对标签的DOM HTML对象，以额外的属性和方法来加以扩展。

## 3.6　实例：将手工HTML代码转换为DOM代码

作为一名Web开发者，我最讨厌的一件事就是重复性的任务。难道我们中还有人愿意一而再，再而三地不断重复一件事吗？摆脱乏味的日常重复性事务的一种方法，是借助可重用的对象或者说与你现在建立的ADS库类似的库，以及本书第3部分中介绍的那些可重用对象。但这样仍然会把手工编写DOM文档片段的任务留给你，所以下面我们就来探索一种简化手工编写代码的方式。

另一种能够让事情变得有意思并且能够加速开发进程的方式，是编写能够创建代码的代码。为了应用你在本章所学到的知识，我们要创建一个简单的工具，通过它根据一个已有的文档片段生成必要的DOM代码，这个工具的界面如图3-17所示。

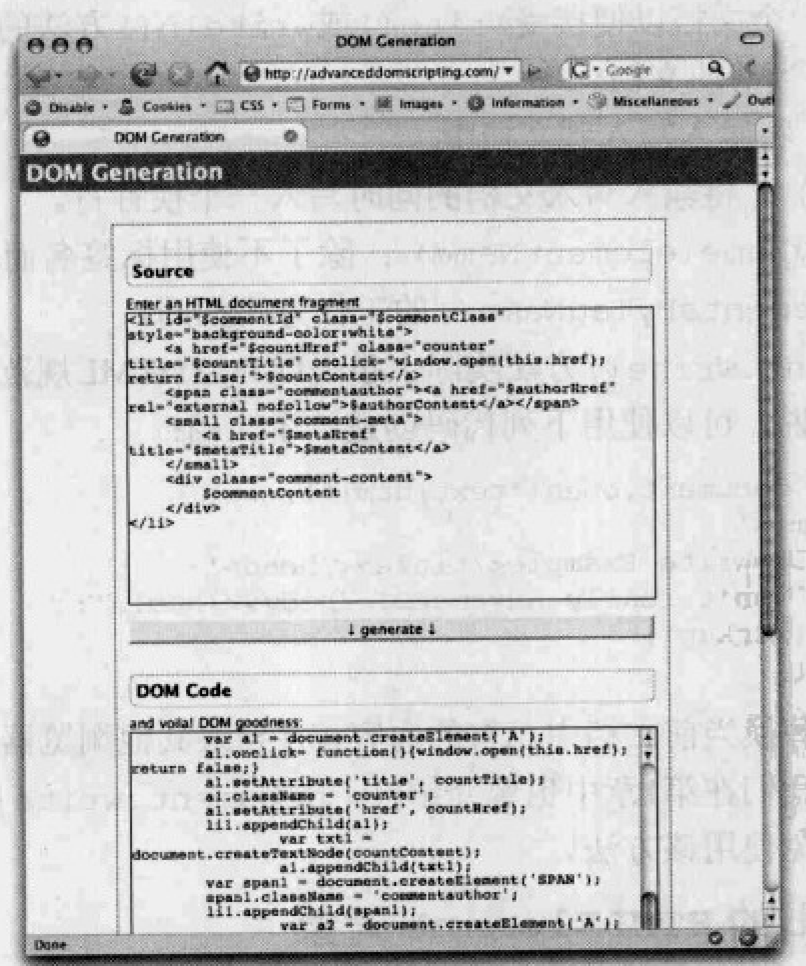



图3-17　HTML到DOM代码的转换工具

① 是IETF（Internet Engineering Task Force，因特网工程工作组）发布的通过标签标识人类语言的一份正式文档，即Tags for the Identification of Languages（语言识别标签）。——译者注

www.TopSage.com

这个工具的用武之地，就是它可以在快速生成必要的DOM代码时用来取代使用innerHTML字符串。

### 3.6.1 DOM 生成工具的 HTML 文件

你的DOM生成工具需要一个HTML表单，以便向其中粘贴HTML片段代码并根据需要转换这些代码。无论你怎样对页面进行布局都没问题，但关键是要有两个<textarea>元素和一个<button>元素，如chapter3/generateDOM/generate.html文件中所包含的：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Generate DOM</title>
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="style.css" />
    <!-- ADS Library (full version from source linked here) -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>
    <script type="text/javascript" src="generateDom.js"></script>
    <script type="text/javascript" src="load.js"></script>
</head>
<body>
<h1>DOM Generation</h1>
<div id="content">
<form id="generator" action="">
    <fieldset>
        <h2>Source</h2>
        <label for="source">Enter an HTML document fragment</label>
        <textarea id="source" cols="30" rows="15"></textarea>
        <input id="generate" type="button" value="&#8595;
            generate &#8595;" />
        <h2>DOM Code</h2>
        <label for="result">and voila! DOM goodness:</label>
        <textarea id="result" cols="30" rows="15"></textarea>
    </fieldset>
</form>
</div>
</body>
</html>
```

这个页面将完全依赖于JavaScript，并且也没有可替代的方法。其中包含一个generateDOM.js文件和一个调用generateDOM()方法转换HTML代码的非常简单的load.js脚本：

```
// 向页面中添加载入事件，注册事件侦听器
ADS.addEvent(window, 'load', function() {

    // 在按钮上注册一个单击事件侦听器
    ADS.addEvent('generate','click', function(W3CEvent) {

        // 取得HTML源代码
        var source = ADS.$('source').value;
```

www.TopSage.com

```
        // 将HTML转换为DOM并放到#result文本区
        ADS.$('result').value = generateDOM(source);

    });
});
```

因此，在generateDOM.js文件中建立generateDOM()方法将是本章剩余内容的焦点。

### 3.6.2 使用示例HTML片段进行测试

为了测试这个工具，可以使用下面这个典型的HTML文档片段。这个片段实际上是取自K2 WordPress主题（http://binarybonsai.com/wordpress/k2/）中的评论标记：

```
<li id="comment-1" class="comment c1 c-y2007 c-m01 c-d01 c-h05 alt">
    <a href="#comment-1" class="counter"
        title="Permanent Link to this Comment">1</a>
    <span class="commentauthor">
        <a href="http://wordpress.org/" rel="external
            nofollow">Mr WordPress</a>
    </span>
    <small class="comment-meta">
        <a href="#comment-1" title="Permanent Link to this ➥
Comment">Aug 22nd, 2006 at 5:09 pm</a>
    </small>
    <div class="comment-content">
        <p>Hi, this is a comment.<br>To delete a comment, just log ➥
in, and view the posts' comments, there you will have the option ➥
to edit or delete them.</p>
    </div>
</li>
```

如果你希望为应用K2主题的WordPress网站（或其他任何网站）编写自己的基于Ajax的可退化的评论系统，那么你的JavaScript API需要在评论提交之后向当前列表中添加一条新评论。我们将在第7章讨论Ajax以及适当退化的内容，所以在这里，我们只关注如何简化快速创建必需的DOM代码的过程。虽然这里使用的是默认的K2评论代码块，但你也可以使用任意HTML代码块进行测试。

实际上，这个工具的总目标就是取得像下面这个锚一样的HTML片段：

```
<a href="http://wordpress.org/" rel="external ➥
nofollow">Mr WordPress</a>
```

并将其转换为如下等价的DOM代码：

```
var a = document.createElement('A');
a.setAttribute('href','http://wordpress.org');
a.setAttribute('rel','external nofollow');
```

如果你总是希望锚链接指向http://wordpress.org，这当然再好不过了；但关键是，多数情况下你都希望其中的href属性使用DOM脚本中的一个变量。为了识别这些变量，需要修改HTML代码，并使用美元符号（$）作为每个属性值的前缀。例如，$var中的var就是你希望指定给属性的变量名：

www.TopSage.com

```
<li id="$commentId" class="$commentClass" style="background-color:white">
    <a href="$countHref" class="counter" title="$countTitle" ➥
onclick="window.open(this.href); return false;">$countContent</a>
    <span class="commentauthor"><a href="$authorHref" rel="external ➥
nofollow">$authorContent</a></span>
    <small class="comment-meta">
        <a href="$metaHref" title="$metaTitle">$metaContent</a>
    </small>
    <div class="comment-content">
        $commentContent
    </div>
</li>
```

这样，当在generateDOM()方法中检查节点时，就可以将任何像下面这样的$varName的HTML实例：

```
<a href="$metaHref" rel="external nofollow">Mr WordPress</a>
```

转换为也包含相同变量的DOM代码：

```
var a = document.createElement('A');
a.setAttribute('href',metaHref);
a.setAttribute('rel','external nofollow');
// 其他代码
```

这样一来，你就可以在脚本中使用metaHref变量来动态地定义锚的href属性的内容了。此外，我们还注意到，前面HTML代码中的一个链接带有onclick事件属性，而<li>元素则带有一个嵌入的样式属性。通过这些有意添加的属性你可以更全面地测试generateDOM对象的所有功能。

> 你可能会奇怪为什么我选择以美元符号作为前缀的变量格式（$var）。当然，你也可以使用自己认为合适的格式，不过$是在PHP这样的服务器端脚本语言中定义变量的常用符号。因此这里使用$可以让你方便地将服务器端的HTML模板直接复制粘贴到这个工具中，而且，这样还能够在你的DOM脚本中维护与服务器端脚本相同的变量名，从而保持语义上的一致性。

### 3.6.3 扩充 ADS 库

在构建generateDOM对象的框架之前，你还需要向ADS.js文件中再添加几个方法。

首先，在你现有的ADS命名空间外部，为JavaScript String对象的原型添加两个新方法，分别用于生成重复的字符串和清除字符串两端的空白符：

```
// 重复一个字符串
if (!String.repeat) {
    String.prototype.repeat = function(l){
        return new Array(l+1).join(this);
    }
}

// 清除结尾和开头处的空白符
if (!String.trim) {
    String.prototype.trim = function() {
        return this.replace(/^\s+|\s+$/g,'');
    }
```

www.TopSage.com

```
}
```

其中，repeat()方法可以按照给定的次数重复字符串，该方法在创建空白缩进和其他重复性字符串时有用：

```
var example = 'a'.repeat(5);
// example现在是aaaaa
```

对于HTML片段中的所有文本节点而言，需要使用trim()方法来帮助发现其中是否包含除了空白符之外的字符。在很多别的语言中，trim函数都是一个用来清除字符串两端空白符的常用函数，如果字符串中不包含任何非空白符，那么这个字符串将被缩减为空字符串①。

> 把对原型的修改放在ADS命名空间之外，是为了提醒你对内部String对象的prototype的修改，会影响到整个脚本中的每一个字符串，而不仅仅是只在ADS.generateDOM对象内部有影响。与你的自定义对象的prototype一样，你也可以修改已有JavaScript对象（例如String）的prototype，可以向其中添加特性或者覆盖已有的特性。同样地，新添加的方法只有当原先不存在该方法时才会有效，也就是说如果存在原生方法，那么还会使用原生方法。

下面，是要添加到ADS命名空间中的camelize方法（在不使用我提供的例子代码时有用）：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

    ……以上是库中已有的内容……

    /* 把word-word转换为wordWord */
    function camelize(s) {
        return s.replace(/-(\w)/g, function (strMatch, p1){
            return p1.toUpperCase();
        });
    }
    window['ADS']['camelize'] = camelize;

    ……以下是库中已有的内容……

})();
```

以上ADS.camelize()方法主要用于处理嵌入的样式属性。虽然本例由于正确地使用了类和ID，没有包含嵌入的样式，但仍然说不定会在什么时候遇到带有嵌入样式的HTML。而一旦遇到了这种情况，你的generateDOM对象至少应该知道如何适当地处理样式属性。因为CSS和ECMAScript规范之间的不一致，CSS属性中使用了连字符；但连字符却不适合作为JavaScript中的成员名称，原因是连字符不能用作标识符②。所以，对于font-size这样的CSS样式属性，在JavaScript中必须要变成驼峰形大小写形式（camel cased），即font-size要变成fontSize，而这一转换就是通过camelize()方法完成的。

① 这里的空字符串指什么也不包含，如''。——译者注

② 事实上，连字符在ECMAScript规范中用作减号操作符。——译者注

www.TopSage.com

### 3.6.4 generateDOM对象的框架

现在剩下的唯一一件事就是在generateDOM.js文件中创建generateDOM对象了。框架以创建一个新的命名空间开始，然后包含了一些辅助方法和属性，最后是为window方法赋值的代码：

```
/* generateDOM对象的新命名空间 */
(function(){

    function encode(str) { }

    function checkForVariable(v) { }

    var domCode = '';
    var nodeNameCounters = [];
    var requiredVariables = '';
    var newVariables = '';

    function generate(strHTML,strRoot) { }

    function processAttribute(tabCount,refParent) { }

    function processNode(tabCount,refParent) { }

    window['generateDOM'] = generate;

})();
```

#### 1. encode()方法

generateDOM.js文件中的第一个方法是encode()。虽然JavaScript中已经有了一个内置的escape()方法，但escape()方法会将所有非ASCII字符（包括空白符和标点符号），替换为十六进制编码格式，如%xx（其中xx是相应字符的ASCII-safe的十六进制数字表示法）。当使用escape(' ')转义一个空格符时，空格符会被转换成%20。

对于我们这里生成DOM的方法来说，encode()方法将用于保证字符串是一个安全的JavaScript字符串。因为转换工具要生成的字符串会被包含在单引号中，所以只需要转义反斜杠、单引号和换行符即可：

```
var example = 'a string';
```

以上字符串'a string'中不能包含纯粹的反斜杠、单引号或者换行符，因此可以通过自定义的encode()方法只对这些字符以反斜杠进行适当地转义：

```
function encode(str) {
    if (!str) return null;
    str = str.replace(/\\/g,'\\\\');
    str = str.replace(/';/g, "\\'");
    str = str.replace(/\s+^/mg, "\\n");
    return str;
}
```

#### 2. checkForVariable()方法

generateDOM.js文件中的第二个方法是checkForVariable()。你会用这个方法来查找所

有节点值中那些特殊的$var字符串，并对它们进行相应处理。该方法只是简单地检查字符串中是否包含一个美元符号，如果是，则返回一个带引号的字符串或者一个变量名称。而且，还会把变量声明添加到requiredVariables字符串中，以便在结果中显示它们：

```
function checkForVariable(v) {
    if(v.indexOf('$') == -1) {
        v='\'' + v + '\'';
    } else {
        // 因MSIE会添加锚的完整路径，故需要
        // 取得该字符串从$到结尾处的子字符串
        v = v.substring(v.indexOf('$')+1)
        requiredVariables += 'var ' + v + ';\n';
    }
    return v;
}
```

### 3. generate()方法

随着以上方法的就绪，现在就可以接触对象的核心方法——generate()了。你可以先按照下面的代码完成generate()方法，并借以熟悉它的内部工作过程：

```
function generate(strHTML,strRoot) {

    // 将HTML代码添加到页面主体中，以便能够遍历相应的DOM树
    var domRoot = document.createElement('DIV');
    domRoot.innerHTML = strHTML;

    // 重置变量
    domCode = '';
    nodeNameCounters = [];
    requiredVariables = '';
    newVariables = '';

    // 使用processNode()处理domRoot中的所有子节点
    var node = domRoot.firstChild;
    while(node) {
        ADS.walkTheDOMRecursive(processNode,node,0,strRoot);
        node = node.nextSibling;
    }

    // 输出生成的代码
    domCode =
        '/* requiredVariables in this code\n' + requiredVariables + '*/\n\n'
        + domCode + '\n\n'
        + '/* new objects in this code\n' + newVariables + '*/\n\n';

    return domCode;
}
```

> walkTheDOMRecursive()方法不会处理与HTML节点关联的属性节点。我们将在generate()方法中单独地处理属性节点，因为这样会使处理class和style属性的特殊情况简单一些。

生成必需的DOM代码的最简单方式，就是遍历一个已有的DOM树并检测其中所有的节点，

www.TopSage.com

然后按照需要再重新创建它们。为此，我们应用了一点小技巧，即使用innerHTML根据HTML字符串创建了初始的DOM树：

```
// 将HTML代码添加到页面主体中，以便能够遍历相应的DOM树
var domRoot = document.createElement('DIV');
domRoot.innerHTML = HTML;
```

这个小技巧将会强迫浏览器将HTML字符串解析为domRoot中的DOM结构。由于这个对象的设计用途是作为一个开发工具，因而你可以乐得使用innerHTML属性，原因是你可以自己控制在哪个浏览器或者在什么环境下运行这个工具。

在generate()方法的最后，循环遍历domRoot的子节点并调用了walkTheDOMRecursive()方法。其中使用了processNode()方法来处理每个节点：

```
var node = domRoot.firstChild;
while(node) {
    walkTheDOMRecursive(processNode,node,0,strRoot);
    node = node.nextSibling;
}
// 输出生成的代码
domCode =
    '/* requiredVariables in this code\n' + requiredVariables + '*/\n\n'
    + domCode + '\n\n'
    + '/* new objects in this code\n' + newVariables + '*/\n\n';

return domCode;
```

#### 4. processNode()和processAttribute()方法

当循环遍历domRoot的子节点时，将使用processNode()方法分析树中的每个节点，确定节点的类型、值和属性，以便重新创建适当的DOM代码。先按照下面的代码完成processNode()方法，然后我们再讨论它的工作过程：

```
function processNode(tabCount,refParent) {
    // 根据树的深度级别重复制表符
    // 以便对每一行进行适当的缩进
    var tabs = (tabCount ? '\t'.repeat(parseInt(tabCount)) : '');

    // 确定节点类型并处理元素和文本节点
    switch(this.nodeType) {
        case ADS.node.ELEMENT_NODE:
            // 计数器加1并创建一个使用标签和计数器的值
            // 表示的新变量，例如：a1、a2、a3
            if(nodeNameCounters[this.nodeName]) {
                ++nodeNameCounters[this.nodeName];
            } else {
                nodeNameCounters[this.nodeName] = 1;
            }

            var ref = this.nodeName.toLowerCase()
                + nodeNameCounters[this.nodeName];

            // 添加创建这个元素的DOM代码行
            domCode += tabs
                + 'var '
```

```
            + ref
            + ' = document.createElement(\''
            + this.nodeName +'\');\n';

        // 将新变量添加到列表中
        // 以便在结果中报告它们
        newVariables += '' + ref + ';\n';

        // 检测是否存在属性，如果是则循环遍历这些属性
        // 并使用processAtrribute()方法遍历它们的DOM树
        if (this.attributes) {
            for(var i=0; i < this.attributes.length; i++) {
                ADS.walkTheDOMRecursive(
                    processAttribute,
                    this.attributes[i],
                    tabCount,
                    ref
                );
            }
        }

        break;
    case ADS.node.TEXT_NODE:

        // 检测文本节点中除了
        // 空白符之外的值
        var value = (this.nodeValue ? encode(➡
    this.nodeValue.trim()) : '' );
        if(value) {

            // 计数器加1并创建一个使用txt和计数器的值
            // 表示的新变量，例如txt1、txt2、txt3
            if(nodeNameCounters['txt']) {
                ++nodeNameCounters['txt'];
            } else {
                nodeNameCounters['txt'] = 1;
            }
            var ref = 'txt' + nodeNameCounters['txt'];

            // 检查是不是$var格式的值
            value = checkForVariable(value);

            // 添加创建这个元素的DOM代码
            domCode += tabs
                + 'var '
                + ref
                + ' = document.createTextNode('+ value +');\n';
            // 将新变量添加到列表中
            // 以便在结果中报告它们
            newVariables += '' + ref + ';\n';

        } else {
            // 如果不存在值（或者只有空白符）则返回
            // 即这个节点将不会被添加到父节点中
            return;
        }
        break;
```

www.TopSage.com

```
        default:
            // 忽略其他情况
            break;
    }

    // 添加将这个节点添加到其父节点的代码
    if(refParent) {
        domCode += tabs + refParent + '.appendChild('+ ref + ');\n';
    }
    return ref;

}
```

读一遍以上代码，你会发现该方法做了以下几件事。

首先，processNode()方法会基于递归的深度来确定DOM代码缩进的级别，并按照需要重复制表符：

```
var tabs = (tabCount ? '\t'.repeat(parseInt(tabCount)) : '');
```

这种缩进并不是真正必需的，但它却有助于生成的代码更清晰，也更容易理解。

其次，该方法会根据要处理节点的类型来完成一些特殊的任务。对于这个例子而言，需要处理的是两种节点类型：ADS.node.ELEMENT_NODE（类型值为1）和ADS.node.TEXT_NODE（类型值为3）。此外，还将使用processAttribute()方法来处理属性节点。除此之外，任何其他的节点类型都会被忽略不计。假如还需要包含处理其他节点的代码，那么可以通过向switch语句中添加更多的case子句来扩展这个例子：

```
switch(node.nodeType) {
    case ADS.node.ELEMENT_NODE:
        // 处理元素节点
        break;
    case ADS.node.TEXT_NODE:
        // 处理文本节点
        break;
    default:
        // 忽略其他情况
        break;
}
```

你会注意到switch语句的比较关系中使用的是ADS.node常量，而非数字值。原因也是一样，即这样可以令代码更容易理解（不言自明）。

如果HTML代码片段中的节点是一个ELEMENT_NODE，processNode()方法会通过连接nodeName和计数器的数字值来创建一个新的变量名称，以便能够引用具有相同nodeType的多个节点的实例。此外，该方法还将新创建的变量名称添加到了newVariables字符串中，以便清晰地输出包含所有新创建的节点的列表：

```
// 计数器加1并创建一个使用标签和计数器的值
// 表示的新变量，例如：a1、a2、a3
if(nodeNameCounters[this.nodeName]) {
    ++nodeNameCounters[this.nodeName];
} else {
```

www.TopSage.com

```
        nodeNameCounters[this.nodeName] = 1;
    }

    var ref = this.nodeName.toLowerCase()
        + nodeNameCounters[this.nodeName];

    // 添加创建这个元素的DOM代码行
    domCode += tabs
        + 'var '
        + ref
        + ' = document.createElement(\'' + this.nodeName +'\');\n';
    // 将新变量添加到列表中
    // 以便在结果中报告它们
    newVariables += '' + ref + ';\n';
```

所有ELEMENT_NODE节点都可能带有属性。前面我们曾经讨论过，所有属性自身也都是节点，但却没有被包含在childNodes中，也无法通过同辈定位的方法进行迭代。属性节点都包含在node.attributes数组中，因此必须要单独地对它们进行遍历：

```
if (node.attributes) {
    for(var i=0; i < node.attributes.length; i++) {
        myWalkTheDOM(processAttribute,
            node.attributes[i], tabCount, ref);
    }
}
```

这个过程与遍历domRoot中的子节点的过程相同。但在这里，使用的是processAttribute()方法而非processNode()方法（我们会在分析完processNode()方法后创建processAttribute()方法）。

processNode()方法需要处理的另一种元素是TEXT_NODE节点。我们前面也曾讨论过，文本节点就是包含所有空白符和位于标签外部文本的节点。当使用DOM方法创建HTML文档时，无需考虑如何使用制表符和换行符来控制硬编码的HTML标记，以使源代码看起来显得更规范，而只需关心如何捕获标记中的文本即可。不过，要是你在DOM代码中随意地生成换行符节点，那也是够令人讨厌的：

```
document.createTextNode('\n');
```

为了保证代码更清晰，processNode()方法按照我们前面讨论过的做法对每个值都调用了trim()方法，以便清除节点内容开头和结尾处的空白符。如果节点内容是空的，则只需返回null并移动到下一个节点。同样地，还需要调用encode()方法将JavaScript字符串不能接受的特殊字符转换为转义后的格式：

```
var value = (this.nodeValue ? encode(this.nodeValue.trim()):'' );
```

与处理ELEMENT_NODE节点时的做法类似，也需要为新的文本节点创建一个新变量，并将其添加到newVariables列表中。然后，代码还调用了受保护的checkForVariable()方法来检查节点值中是否包含特殊格式的$var：

```
// 检查是不是$var格式的值
value = checkForVariable(value);
```

www.TopSage.com

```
// 添加创建这个元素的DOM代码
domCode += tabs
    + 'var '
    + ref
    + ' = document.createTextNode('+ value +');\n';
// 将新变量添加到列表中
// 以便在结果中报告它们
newVariables += '' + ref + ';\n';
```

在处理完节点之后，唯一要做的就是将其添加到父节点中：

```
// 添加将这个节点添加到其父节点的代码
domCode += tabs + refParent + '.appendChild('+ ref + ');\n';
```

如果你在创建processAttribute()方法之前，使用示例的HTML片段来试验这个工具，将会得到对这个HTML片段进行细致格式化之后的DOM代码的表示：

```
/* requiredVariables in this code
var countContent;
var authorContent;
var metaContent;
var commentContent;
*/

var li1 = document.createElement('li');
document.body.appendChild(li1);
    var a1 = document.createElement('a');
    li1.appendChild(a1);
        var txt1 = document.createTextNode(countContent);
        a1.appendChild(txt1);
    var span1 = document.createElement('span');
    li1.appendChild(span1);
        var a2 = document.createElement('a');
        span1.appendChild(a2);
            var txt2 = document.createTextNode(authorContent);
            a2.appendChild(txt2);
    var small1 = document.createElement('small');
    li1.appendChild(small1);
        var a3 = document.createElement('a');
        small1.appendChild(a3);
            var txt3 = document.createTextNode(metaContent);
            a3.appendChild(txt3);
    var div1 = document.createElement('div');
    li1.appendChild(div1);
        var txt4 = document.createTextNode(commentContent);
        div1.appendChild(txt4);

/* new objects in this code
li1;
a1;
txt1;
span1;
a2;
txt2;
small1;
a3;
```

www.TopSage.com

```
txt3;
div1;
txt4;
*/
```

结果中包含所有ELEMENT_NODE和TEXT_NODE节点以及完整的变量名称。但是，其中还缺少与每个节点关联的所有属性，而这正是需要processAttribute()方法来解决的问题。现在就将下列代码添加到processAttribute()方法中：

```
function processAttribute(tabCount,refParent) {

    // 跳过文本节点
    if(this.nodeType != ADS.node.ATTRIBUTE_NODE) return;
    // 取得属性值
    var attrValue = (this.nodeValue ? encode(➥
this.nodeValue.trim()) : '');
    if(this.nodeName == 'cssText') alert('true');
    // 如果没有值则返回
    if(!attrValue) return;
    // 确定缩进的级别
    var tabs = (tabCount ? '\t'.repeat(parseInt(tabCount)) : '');
    // 根据nodeName进行判断。除了class和style需要
    // 特殊注意以外，所有类型都可以按常规来处理
    switch(this.nodeName){
        default:
            if (this.nodeName.substring(0,2) == 'on') {
                // 如果属性名称以'on'开头，说明是
                // 一个嵌入的事件属性，也就需要
                // 重新创建一个给该属性赋值的函数
                domCode += tabs
                    + refParent
                    + '.'
                    + this.nodeName
                    + '= function(){' + attrValue +'}\n';
            } else{

                // 对于其他情况则使用setAttribute
                domCode += tabs
                    + refParent
                    + '.setAttribute(\''
                    + this.nodeName
                    + '\', '
                    + checkForVariable(attrValue)
                    +');\n';
            }
        break;
        case 'class':
            // 使用className属性为class赋值
            domCode += tabs
                + refParent
                + '.className = '
                + checkForVariable(attrValue)
                + ';\n';
            break;
```

www.TopSage.com

```
            case 'style':
                // 使用正则表达式基于;和邻近的
                // 空格符来分割样式属性的值
                var style = attrValue.split(/\s*;\s*/);

                if(style){
                    for(pair in style){

                        if(!style[pair]) continue;

                        // 使用正则表达式基于:和邻近的
                        // 空格符来分割每对样式属性
                        var prop = style[pair].split(/\s*:\s*/);
                        if(!prop[1]) continue;

                        // 将css-property格式的CSS属性
                        // 转换为cssProperty 格式
                        prop[0] = ADS.camelize(prop[0]);

                        var propValue = checkForVariable(prop[1]);
                        if (prop[0] == 'float') {
                            // float是保留字，因此属特殊情况
                            // cssFloat是标准的属性
                            // styleFloat是IE使用的属性
                            domCode += tabs
                                + refParent
                                + '.style.cssFloat = '
                                + propValue
                                + ';\n';
                            domCode += tabs
                                + refParent
                                + '.style.styleFloat = '
                                + propValue
                                + ';\n';
                        } else {
                            domCode += tabs
                                + refParent
                                + '.style.'
                                + prop[0]
                                +'='
                                + propValue + ';\n';
                        }
                    }
                }
            break;
        }
    }
```

processAttribute()方法大致遵循了与processNode()方法相同的处理模式，但处理属性还是有些不同。处理属性时的第一个差别是访问节点值的方式。前面我们看到过，属性节点有一个nodeValue属性，因此可以通过该属性来取得节点的值。然而，问题是你在前面创建的遍历DOM的函数，仍然会迭代ATTRIBUTE_NODE中的TEXT_NODE，所以如果要使用nodeValue属性，就必须跳过非ATTRIBUTE_NODE节点：

```
    // 跳过文本节点
```

www.TopSage.com

```
    if(this.nodeType != ADS.node.ATTRIBUTE_NODE) return;

    // 取得属性的值
    var attrValue = (this.nodeValue ? encode(this.nodeValue.trim()) : '');
    if(this.nodeName == 'cssText') alert('true');
    // 如果没有值则返回
    if(!attrValue) return;
```

接着，要检测每一个属性并针对class和style属性做一些特殊的处理。如果不存在class或style属性，则使用setAttribute()方法在父节点的引用上面创建相应的代码，但仍然需要检查嵌入的事件属性并创建必要的函数：

```
    default:
        if (this.nodeName.substring(0,2) == 'on') {
            // 如果属性名称以'on'开头，说明是
            // 一个嵌入的事件属性，也就需要
            // 重新创建一个赋值给该属性的函数
            domCode += tabs
                + refParent
                + '.'
                + this.nodeName
                + '= function(){' + attrValue +'}\n';
        } else{

            // 对于其他情况则使用setAttribute
            domCode += tabs
                + refParent
                + '.setAttribute(\''
                + this.nodeName
                + '\', '
                + checkForVariable(attrValue)
                +');\n';
        }
    break;
```

正如我们在第1章曾经提到的，嵌入的事件是最大的一个禁忌。不过，如果你的DOM生成程序可以检查出它们来，就意味着它可以在升级旧代码时派上用场。如果属性是class，就需要将class节点的值赋给节点的className属性：

```
    case 'class':
    // 使用className属性为class赋值
    domCode += tabs
        + refParent
        + '.className = '
        + checkForVariable(attrValue)
        + ';\n';
    break;
```

可是，如果该节点是一个嵌入的style属性（同样也是一个禁忌），则需要更多的处理工作，包括分割不同的选择符以及属性：

```
    case 'style':
        // 使用正则表达式基于;和邻近的
        // 空格符来分割样式属性的值
        var style = attrValue.split(/\s*;\s*/);
```

www.TopSage.com

```
    if(style){
    for(pair in style){

        if(!style[pair]) continue;

        // 使用正则表达式基于:和邻近的
        // 空格符来分割每对样式属性
        var prop = style[pair].split(/\s*:\s*/);
        if(!prop[1]) continue;

        // 将css-property格式的CSS属性
        // 转换为cssProperty 格式
        prop[0] = ADS.camelize(prop[0]);

        var propValue = checkForVariable(prop[1]);
        if (prop[0] == 'float') {
            // float是保留字，因此属特殊情况
            // cssFloat是标准的属性
            // styleFloat是IE使用的属性
            domCode += tabs
                + refParent
                + '.style.cssFloat = '
                + propValue
                + ';\n';
            domCode += tabs
                + refParent
                + '.style.styleFloat = '
                + propValue
                + ';\n';
        } else {
            domCode += tabs
                + refParent
                + '.style.'
                + prop[0]
                +'='
                + propValue + ';\n';
        }
    }
}
break;
```

如果此时再运行测试页面，就会看到可以生成示例HTML评论代码块的完整功能的DOM代码了：

```
/* requiredVariables in this code
var commentClass;
var commentId;
var countTitle;
var countHref;
var countContent;
var authorHref;
var authorContent;
var metaTitle;
var metaHref;
var metaContent;
```

www.TopSage.com

```
var commentContent;
*/

var li1 = document.createElement('li');
li1.style.backgroundColor = 'white';
li1.className = commentClass;
li1.setAttribute('id', commentId);
document.body.appendChild(li1);
    var a1 = document.createElement('a');
    a1.onclick= function(){window.open(this.href); return false;}
    a1.setAttribute('title', countTitle);
    a1.className = 'counter';
    a1.setAttribute('href', countHref);
    li1.appendChild(a1);
        var txt1 = document.createTextNode(countContent);
        a1.appendChild(txt1);
    var span1 = document.createElement('span');
    span1.className = 'commentauthor';
    li1.appendChild(span1);
        var a2 = document.createElement('a');
        a2.setAttribute('rel', 'external nofollow');
        a2.setAttribute('href', authorHref);
        span1.appendChild(a2);
            var txt2 = document.createTextNode(authorContent);
            a2.appendChild(txt2);
    var small1 = document.createElement('small');
    small1.className = 'comment-meta';
    li1.appendChild(small1);
        var a3 = document.createElement('a');
        a3.setAttribute('title', metaTitle);
        a3.setAttribute('href', metaHref);
        small1.appendChild(a3);
            var txt3 = document.createTextNode(metaContent);
            a3.appendChild(txt3);
    var div1 = document.createElement('div');
    div1.className = 'comment-content';
    li1.appendChild(div1);
        var txt4 = document.createTextNode(commentContent);
        div1.appendChild(txt4);

/* new objects in this code
li1;
a1;
txt1;
span1;
a2;
txt2;
small1;
a3;
txt3;
div1;
txt4;
*/
```

现在，当需要创建交互式的评论对象时，可以简单地将这个工具生成的结果复制并粘贴到代码编辑器中，然后继续做其他事，从而免去了许多无谓的DOM脚本编程的工作量。

www.TopSage.com

## 3.7 小结

学会所有正确的DOM方法和属性是进行DOM脚本编程的基础，但更重要的是，只有理解所有这些方法和属性的来源，以及每个规范与其他规范之间协同的方式，才能对自己编程的上下限有更好地把握。本章主要介绍了DOM2核心和DOM2 HTML规范的基本知识，虽然这只是众多规范中的两个，但你已经理解了像Element这样的核心对象构建于核心Node对象之上的过程，以及Element对象又在HTML规范中被进一步扩展为HTMLElement对象和针对每个标签的对象这一事实。对于每个属性和方法的来源以及对象之间互为基础创建过程的理解，将有助于你理解后续章节中更多的例子。

本章同时也提示了另外一个事实，即浏览器并非完美的开发环境，而且它们之间的差别经常会令开发者们头痛不已。比如说你吧，本来想遵循标准，但却不得已还要照顾到那些非标准的浏览器。解决这个问题的最佳方案就是，将按照标准开发放在首位，然后通过适当的能力检测来启用非标准的特性。通过将执行标准放在首位，所有浏览器最终都将按照预期执行你的代码，而你则可以逐步放弃对非标准特性的支持，这样显然要胜过将来再为了实现标准化而重新编码的结局。

在本章中，你也进一步丰富了自己的ADS库，并且拥有了两个在其余章节的开发中能够派上用场的得力工具。为在目前所学知识的基础上继续扩展你的ADS库，我们还需要在第4章中详细地剖析事件及如何响应用户操作的主题。

www.TopSage.com

第4章

# 响应用户操作和事件

事件在Web应用程序中是有魔力的要素。在创建高级用户界面时，简单的老式HTML和表单已经力不从心了。用户填写完表单，单击了保存按钮，然后就在那里等着整个文档重新载入（以及所有辅助过程的完成），而等待的结果只是隐藏在整个网页中某个地方的微小变化，这种工作方式早已作古。你所希望的Web应用程序应该流畅并且不唐突地响应用户操作，就像是桌面应用程序一样。能够让Web应用程序更具有桌面应用程序的感觉虽然算不上具有革命性，但确实要求改变看待问题的方式。事实上，只要多一些独创性和预见性，你就能做到以最小的努力、最低的成本和最少的时间投入，开发出具有桌面应用程序体验的Web应用。不过，前提是你必须理解浏览器如何与人交互。

没有事件的网页可以比喻成模拟电视信号。这些网页虽然看上去不错，而且也能够提供有用的信息，但是用户却无法与之交互。当我提到浏览器中或者DOM脚本中的事件这个概念时，指的是当浏览器检测到某种操作发生时所采取的行动。所谓的操作可能是浏览器载入页面，或者用户单击并滚动鼠标。

事件就是操作检测与脚本执行的组合，或者基于检测到的操作类型在某个对象上调用事件侦听器。在本书中，你曾经接触过的事件有window对象的load事件和锚元素的click事件。如果本书具有智能并且知道你正在看这一段文字，那么它可能会在你刚一开始读到本段内容时，就在这个段落上调用beginReading事件。而且，它还会检测到你仍然在阅读这一段，因此又会在你每开始看一个新字时连续地在这个段落上调用reading事件。当你读完本句时，本段也就读完了，它又会检测到你已经读完并在这个段落上调用finishedReading事件。

JavaScript还没有聪明到能够知悉你在阅读一段文字（至少现在还不能），因而也没有什么阅读段落的事件，但确实有一些基于浏览器和用户操作的事件。你可以使用这些事件来创建一个事件侦听器，即当事件被调用时你希望执行的某些代码。而如何指定事件侦听器则取决于事件的类型。不幸的是，还要取决于浏览器的类型。

在本章中，我们将探索W3C DOM2级事件规范及其在各种浏览器中的应用。还会介绍不同浏览器在注册事件、调用事件、访问事件方面的差异，以及如何通过事件执行几乎任何操作。在学习本章的同时，你还会在自己的ADS库中添加用于跨浏览器事件处理的更多方法。

在开始之前，需要澄清几个术语，以便我们能站在同一起点上。如果你以前使用过浏览器事

www.TopSage.com

件，或者在某些资料中看到过有关事件的介绍，那么一定接触过用于描述事件各个方面的术语。为了清晰起见，我会在本章统一使用下列术语。

- **事件**：这里需要澄清的是事件并不以“on”开头。例如，onclick不是事件，而click才是事件。在本章后面你会看到，onclick引用的是一个对象的属性，通过它可以为DOM元素指定一个click事件。知道这个差别非常重要，因为如果在DOM2级事件规范的方法中使用了on前缀，那你的脚本将不会运行。
- **事件侦听器**：即当指定的事件发生时会执行的JavaScript函数或方法。侦听器有时也被称为“事件处理程序”，但在本章中我们会遵循DOM2级事件规范中的术语称其为“事件侦听器”。
- **事件注册**：这是为一个DOM元素的具体事件指定事件侦听器的过程。注册可以通过几种不同的方式完成，这些本章都将讨论到。有时候，事件注册也被称为事件绑定。
- **调用**：本章会使用动词“调用（invoke）”来描述浏览器在检测到某种操作之后执行相应事件侦听器的情形。换句话说，当用户在文档中单击鼠标时，浏览器会调用与鼠标单击事件关联的事件侦听器。类似的术语可能还有叫用（called）、激发（fired）、执行（executed）或触发（triggered）等[①]。

在统一了这些术语的称谓之后，我们先来看一下W3C DOM2级事件规范与多数浏览器不兼容的问题。

## 4.1 DOM2 级事件

DOM2级事件规范（DOM2 Event）以（第3章所讲到的）DOM2级HTML和DOM2级核心规范定义文档结构的相同方式，为一个通用的事件系统定义了相应的方法和属性。遗憾的是，DOM2级事件规范至今仍然没有被浏览器普遍采用，因此在很多情况下（其中最多的是涉及到IE的情况），你都必须使用不止一种方式来完成相同的目标。在学习本章的过程中，你会向自己的ADS库中不断增加很多通用的事件函数，以便实现跨浏览器的事件注册和相应的操作。

可以通过下面的链接简单地看一看不同浏览器的事件处理方式。

- **Firefox**：http://advanceddomscripting.com/links/events/firefox
- **Opera**：http://advanceddomscripting.com/links/events/opera
- **Microsoft IE**：http://advanceddomscripting.com/links/events/msie
- **Safari**：http://advanceddomscripting.com/links/events/safari

你会注意到以上每种浏览器都会为多种不同的对象提供各式各样的事件，而且其中很多事件在浏览器之间都存在着差异。

W3C在规范中整合了许多标准化之前已经存在的事件，同时也增加了一些针对DOM文档结构的变化和交互定义的DOM事件。然而遗憾的是，主流市场使用的浏览器没有正式支持DOM事件模型，因此我们只能聚焦于常用的事件，并简单地介绍你最终能在日常开发中用到的针对DOM

① 知道了这些术语的含义其实都相同以后，读者在阅读其他英文技术书时就不用再为它们而感到费解了。——译者注

www.TopSage.com

的事件。

在下面开始探索事件时，我们会简单地回顾一下各种可用的事件类型，然后再深入到事件注册、调用以及控制的各种细节中。而且，当你学习到第6章的案例时，还将看到在构建基于Web的交互式图像缩放和裁剪工具时，如何完美地将本章所学的知识付诸实践。

## 4.2　事件的类型

事件可以分成几种类型：对象事件、鼠标事件、键盘事件、表单事件、W3C事件以及针对浏览器的事件。对最后一类事件，即针对浏览器的事件，本章将不会涉及——除非是作为DOM方案的必要替代的情况。

### 4.2.1　对象事件

对象事件既适用于JavaScript对象（例如window对象），也适用于DOM对象（例如HTMLImageElement对象）。

**1. load和unload事件**

在本书各个章节中，我们都在使用window对象的load事件，以便在页面载入时调用行为增强的脚本代码：

```
function yourLoadEvent() {
    // 对文档进行某些操作
}
ADS.addEvent(window,'load', yourLoadEvent);
```

在应用到window对象时，load和unload事件是针对浏览器的，而且是在DOM2事件规范的领域之外执行的。浏览器会在完成对页面的载入时调用window的load事件。通过借助于load事件，可以使不唐突的行为增强只有在JavaScript有效时才会应用到页面上。

> window对象的load的事件存在一个缺点，即在页面标记中嵌入的所有图像没有载入完成之前不会调用该事件。如果你是使用适当的CSS样式表来为标记添加样式，则不会存在这个问题。但是，如果你在标记中加入了很多嵌入的图像，那么在这个页面上注册的载入事件侦听器不到所有图像下载完成就不会运行。也就是说，在执行脚本代码之前会出现明显过长的停顿。在本章后面，我们还会重新讨论ADS.addEvent()方法，并创建一个ADS.addLoadEvent()方法来解决上面所说的载入图像造成的问题。

同样地，当用户通过单击链接或者关闭窗口而即将离开当前页面时，会调用window对象的unload事件。因此，可以通过unload事件侦听器在页面被关闭之前捕获最后一瞬间的信息。

> 要阻止窗口卸载是不可能的。无论你怎样想方设法，浏览器都会离开当前页面并载入下一个页面。

此外，load事件也适用于其他载入外部内容的DOM对象，例如框架和图像。

www.TopSage.com

### 2. abort和error事件

与load和unload事件类似，error事件既适用于window对象，也适用于图像对象。error事件在动态载入并向文档中添加图像时，可以用来识别图像载入错误。如果你创建一个新的img DOM元素，那么就可以使用load事件侦听器在且只在图像载入成功的情况下再将这个DOM元素添加到文档中。同样地，对于载入图像时出现错误的情况，可以使用error事件侦听器来进行说明，并采取适当的行动：

```
ADS.addEvent(window,'load',function() {

    // 创建一个图像元素
    var image = document.createElement('IMG');

    // 当图像载入后将其添加到文档主体
    ADS.addEvent(image, 'load', function() {
        document.body.appendChild(image);
    });

    // 如果载入时出错则添加相应信息
    ADS.addEvent(image, 'error' ,function() {
        var message = document.createTextNode( ➥
'The image failed to load');
        document.body.appendChild(message);
    });
    // 设置图像的src属性以便浏览器取得图像
    image.setAttribute(
        'src',
        'http://advanceddomscripting.com/images/working.jpg'
    );

    // 除了下面这幅图像不存在而且会发生载入错误外，与上面都相同
    var imageMissing = document.createElement('img');
    ADS.addEvent(imageMissing, 'load', function() {
        document.body.appendChild(imageMissing);
    });
    ADS.addEvent(imageMissing, 'error' ,function() {
        var message = document.createTextNode( ➥
'imageMissing failed to load');
        document.body.appendChild(message);
    });
    imageMissing.setAttribute(
        'src',
        'http://advanceddomscripting.com/images/missing.jpg'
    );

});
```

在图像载入成功的情况下，它会被添加到文档主体中。但如果载入失败，则会看到出错信息。

至于abort事件，它的作用很小，你很可能永远都不会需要它。只有在图像完全载入之前，因浏览器停止了载入页面而导致图像载入失败情况下，才会调用这个事件。而这种情况通常是在单击浏览器中的“停止”按钮时才会发生。

### 3. resize事件

当调整浏览器窗口的大小并导致文档的视图发生改变时会发生resize事件。在窗口完成大

www.TopSage.com

小调整时resize事件总会被调用，但在有些情况下，该事件也可能在调整操作期间被多次调用。

**4. scroll事件**

scroll事件适用于具有overflow:auto样式的元素，并且会在元素滚动期间连续地被调用。可能会引发滚动事件的操作包括拖动滚动条、滚动鼠标滚轮、按下键盘中的方向键或其他滚动操作。

### 4.2.2 鼠标移动事件

通过侦听鼠标事件来捕获并响应用户的操作是最常见的情况。当用户在页面中移动鼠标指针时，即使不单击也会引发一些事件：

- 当鼠标处于移动过程中时，鼠标指针下方的对象就会连续地调用mousemove事件。
- 当指针移动到一个新对象上面时则会触发mouseover事件。
- 当指针移出对象时会触发mouseout事件。

> 触发mousemove事件时没有指定的周期或距离。只要指针处于运动状态中，该事件就会被重复触发；而当指针未处于运动状态时，该事件就不会被触发。

如果你在浏览器中打开本书源文件chapter4/move/move.html，会看到鼠标移动事件不断出现于你在第2章创建的ADS.log对象生成的日志窗口中。这个测试文件由非常简单的HTML文档和相应的CSS文件组成：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Mouse Move Events</title>
    <!--include some CSS style sheet to make
        everything look a little nicer -->
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="../chapter.css" />
    <link rel="stylesheet" type="text/css" href="style.css" />

    <!--ADS Library (full version from source linked here) -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>

    <!--Log object from Chapter 2
        (full version from source linked here) -->
    <script type="text/javascript"
        src="../../chapter2/myLogger-final/myLogger.js"></script>

    <!--A quick testing file -->
    <script type="text/javascript" src="move.js"></script>
</head>
<body>
<h1>Mouse Move Events</h1>
<div id="content">
    <div id="box"> Test Box </div>
```

www.TopSage.com

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

位于chapter4/move/move.js文件中的`load`事件侦听器在`document`和box元素上注册了所有鼠标移动事件：

```
ADS.addEvent(window,'load',function(W3CEvent) {

    // 一个用于将事件类型和对象
    // 记录到日志窗口中的方法
    function logit(W3CEvent) {
        switch(this.nodeType) {
            case ADS.node.DOCUMENT_NODE:
                ADS.log.write(W3CEvent.type + ' on document');
            break;
            case ADS.node.ELEMENT_NODE:
                ADS.log.write(W3CEvent.type + ' on box');
            break;
        }
    }

    // 添加鼠标移动事件
    ADS.addEvent(document,'mousemove',logit);
    ADS.addEvent(document,'mouseover',logit);
    ADS.addEvent(document,'mouseout',logit);

    var box = document.getElementById('box');
    ADS.addEvent(box,'mousemove',logit);
    ADS.addEvent(box,'mouseover',logit);
    ADS.addEvent(box,'mouseout',logit);

});
```

图4-1展示了在打开chapter4/move/move.html文件的浏览器窗口中，从左边向右移动鼠标指针时所发生的事件。

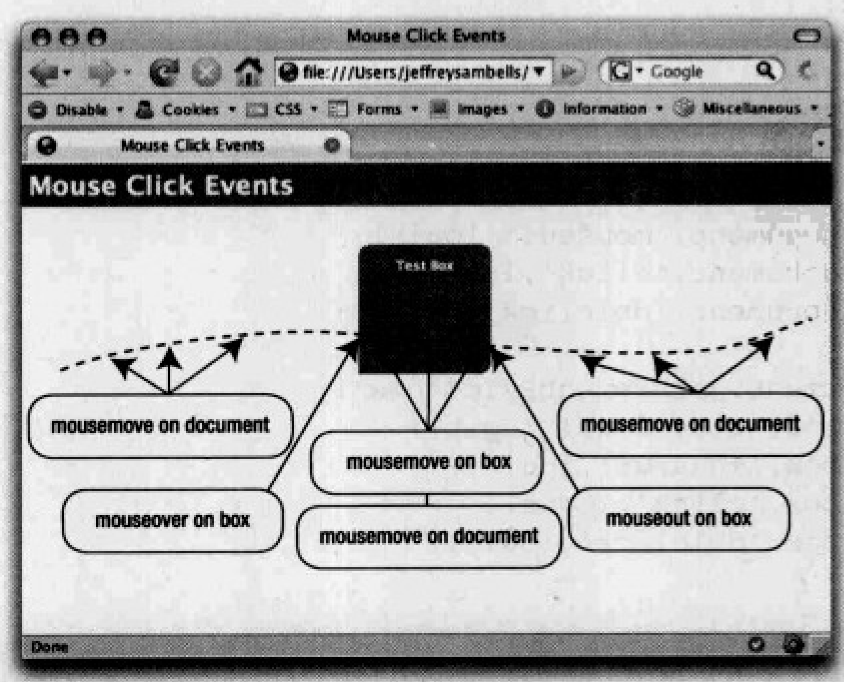



图4-1 通过在示例的文档中从左向右移动鼠标指针来调用事件

www.TopSage.com

而且，浏览器会在你按下鼠标键时调用mousemove事件——同时也会调用与单击相关的事件——这在创建可拖放的对象时非常有用，而且将在第6章用到。

### 4.2.3 鼠标单击事件

当用户在页面上单击鼠标时，在发生鼠标移动事件的同时还会启动另一条事件链。如果用户单击并释放鼠标左键后，保持鼠标指针不动，浏览器会调用下列事件。

- 当鼠标按键在对象上方被按下时发生mousedown事件。
- 当鼠标按键被释放时发生mouseup。
- 只有在鼠标保持不动时才会发生click事件。
- 如果快速按两次按键，则会在click事件后发生dblclick事件。

需要注意的是，只有当鼠标指针保持不动的情况下，以上事件发生的顺序才是正确的。而且，这些事件发生的先后顺序还将受到事件流的影响，这一点我们稍后会介绍到。

如果用户按下鼠标键并在释放之前移动鼠标，那么事情就会变得更有趣起来。同样地，你可以使用chapter4/click/click.html文件来试验一下会出现什么情况，click.html与前面的move.html相比，除了位于chapter4/click/click.js中的载入事件侦听器部分之外几乎完全相同：

```
ADS.addEvent(window,'load',function(W3CEvent) {

    // 一个用于将事件类型和对象
    // 记录到日志窗口中的方法
    function logit(W3CEvent) {
        switch(this.nodeType) {
            case ADS.node.DOCUMENT_NODE:
                ADS.log.write(W3CEvent.type + ' on the document');
            break;
            case ADS.node.ELEMENT_NODE:
                ADS.log.write(W3CEvent.type + ' on the box');
            break;
        }
    }

    // 添加鼠标单击事件
    ADS.addEvent(document,'mousedown',logit);
    ADS.addEvent(document,'mouseup',logit);
    ADS.addEvent(document,'click',logit);
    ADS.addEvent(document,'dblclick',logit);

    var box = document.getElementById('box');
    ADS.addEvent(box,'mousedown',logit);
    ADS.addEvent(box,'mouseup',logit);
    ADS.addEvent(box,'click',logit);
    ADS.addEvent(box,'dblclick',logit);

});
```

测试页面chapter4/click/click.html的外观与图4-1中的页面看起来一样，但其中为document及box元素指定的是click、dblclick、mousedown和mouseup事件。根据你单击及释放鼠标的

www.TopSage.com

位置不同，你可能会看到几种不同的事件相继发生。首先尝试一下图4-2所示的位置。

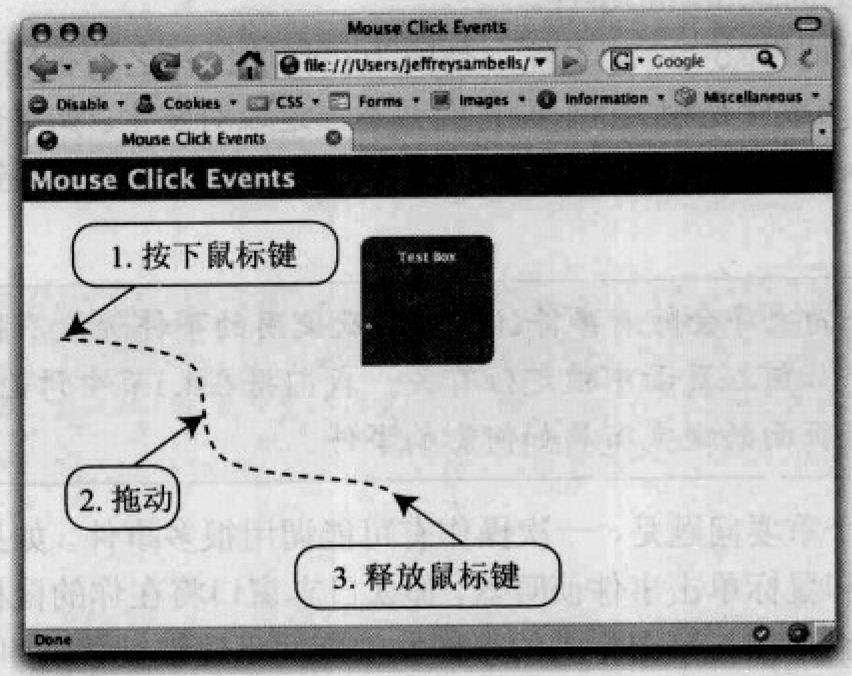



图4-2 在示例文件的页面上单击并释放鼠标指针不会发生click事件的路径示意图

图4-2中的路径会在ADS.log对象创建的日志窗口中揭示出两个事件：

(1) document对象上的mousedown事件。

(2) document对象上的mouseup事件。

在单击和释放鼠标之间，由于拖动了鼠标指针，所以不会发生click事件。

下面，再试一试图4-3所示的路径。

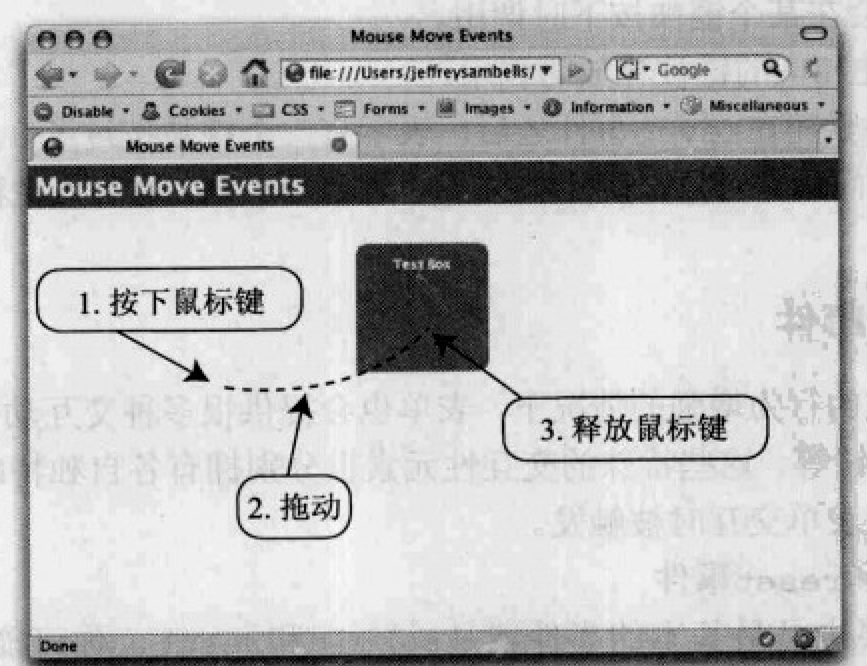



图4-3 在示例文件的页面上单击并在box元素上面释放鼠标指针的路径示意图

图4-3将会揭示下列与document和box对象相关的事件：

(1) document对象上的mousedown事件。

www.TopSage.com

(2) box对象上的mouseup事件。

(3) document对象上的mouseup事件。

之所以会在box上面调用mouseup事件，是因为在释放鼠标之前把它移动到了box上面。虽然mouseup和mousedown事件是相关的，但它们并不一定成对发生。如果你试着按相反的路径进行操作，即在灰色的box上面按住鼠标键并拖动到其外部再释放，将会得到两个mousedown事件和一个mouseup事件。

> 其中mouseup事件的次序会随着事件注册方法及使用的事件阶段不同而改变。此外，还与文档标记的方式及对象如何在页面中被定位有关。我们将在4.3节中讨论事件的阶段（包括捕获阶段和冒泡阶段）及页面的视觉布局如何影响事件。

这里需要记住的一个重要问题是，一次操作有可能调用很多事件。如果在前面试验的例子文件中同时注册鼠标移动和鼠标单击事件侦听器，那么日志窗口将在你的鼠标从页面中一点移动到另一点的过程中，同时也是在与单击相关的事件发生期间迅速地被移动事件填满。

此外，可以通过访问事件对象来确定是哪个鼠标按键调用的事件，这一点我们在本章后面再讨论。

### 4.2.4 键盘事件

与单击事件类似，按下键盘中的一个按键也会导致浏览器调用与按键相关的事件，但键盘事件只适用于document对象。当某个键被按下时，浏览器调用事件的顺序与click事件相同，只不过都是与按键相关的事件：

(1) keydown事件会在某个键被按下时调用。

(2) keyup事件会在相应的键被释放时调用。

(3) keypress事件紧随在keyup事件之后，表示有一个键被按过了。

同样，可以通过访问事件对象来确定被按下的是哪一个键，这一点在稍后我们讨论如何访问事件对象时会介绍。

### 4.2.5 表单相关的事件

在不提供任何形式的行为增强的情况下，表单也会提供很多种交互功能，例如下拉菜单、文本框、复选框、单选按钮等。这些特殊的交互性元素也分别拥有各自独特的事件，这些事件会在用户与表单中的元素及表单交互时被触发。

#### 1. 表单的submit和reset事件

与表单相关的首要也是最基本的事件就是submit和reset。你可能已经猜到了，表单的submit事件会在用户单击提交按钮，或者按下键盘中的某个键将表单提交到服务器时被调用。而reset事件则会以类似的方式在表单被重置时被调用。

可以将submit和reset事件用于许多场合，包括客户端表单验证或者为Ajax操作序列化表单内容。现在来看一下chapter4/address/address.html的源代码，其中包含了许多网站中都会有的简

www.TopSage.com

单的地址表单：

```
<form action="/path/to/server/script" ➥
method="post" id="canadianAddress">
    <div>
        <label for="name">Name</label>
        <input type="text" id="name" name="name"/>
    </div>
    <div>
        <label for="postalCode">Postal Code (required)</label>
        <input type="text" id="postalCode"➥
name="postalCode" class="required"/>
        <p>In the format A#A #A#</p>
    </div>
    <div>
        <label for="street">Street</label>
        <input type="text" id="street" name="street"/>
    </div>
    <div>
        <label for="city">City</label>
        <input type="text" id="city" name="city"/>
    </div>
    <div>
        <label for="province">Province</label>
        <input type="text" id="province" name="province"/>
    </div>
    <div class="buttons">
        <input type="submit" value="submit" />
    </div>
</form>
```

为了验证表单中信息的格式是否正确，本例中包含了一个submit事件侦听器。例子中的表单适用于加拿大的地址，而加拿大的邮政编码（postal code）格式为A#A#A#，其中#代表一个数字，A代表一个字母。在chapter4/address/address.js文件的载入事件侦听器中包含了一个函数，该函数用于在提交表单之前验证邮政编码的格式：

```
function isPostalCode(s) {
    return s.toUpperCase().match( ➥
/[A-Z][0-9][A-Z]\s*[0-9][A-Z][0-9]/i);
}

ADS.addEvent(window,'load',function() {

    ADS.addEvent(
        document.getElementById('canadianAddress'),
        'submit',
        function(W3CEvent) {

            var postalCode = document.getElementById( ➥
'postalCode').value;

            // 使用正则表达式来检查格式是否有效
            if (!isPostalCode(postalCode)) {
                alert('That\'s not a valid Canadian postal code!');

                // 接着使用本章后面即将讨论的
                // ADS.preventDefault()方法取消
```

www.TopSage.com

```
                // 提交表单的操作
                ADS.preventDefault(W3CEvent);
            }
        }
    );
});
```

当你尝试以无效的信息提交表单时，submit事件会提示一个错误信息，如图4-4所示。

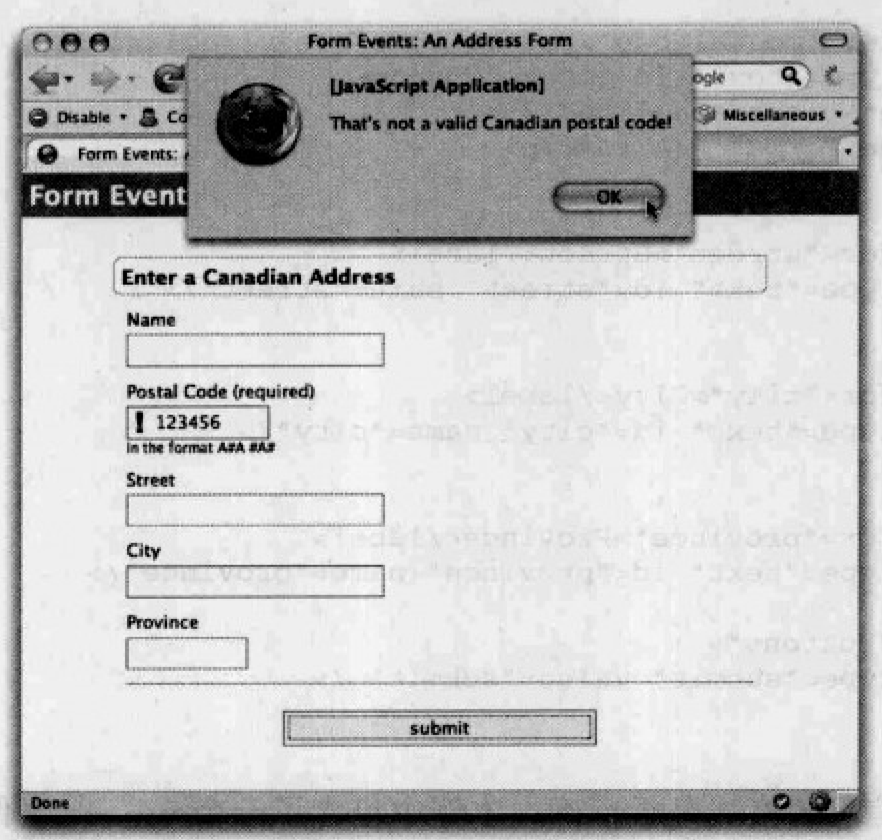



图4-4　当用户尝试使用无效的加拿大邮政编码来提交表单时会显示一个错误信息

这种渐进增强可以让表单在没有JavaScript的情况下正常使用，但当JavaScript有效时则可以得到更及时的反馈。在错误的事件中，阻止表单的提交操作依赖于你使用的事件注册模型，这一点我们很快会讨论到。

> 在实现客户端验证时，要保证不会忽略服务器端。表单提交的信息仍然需要在服务器端进行验证，是因为没有什么能够阻止一个懂得Web知识的用户绕过JavaScript的限制（或者只是禁用它）提交无效数据。

### 2. blur和focus事件

blur和focus事件适用于<label>、<input>、<select>、<textarea>和<button>等表单元素。focus事件会在用户单击一个元素或者通过按Tab键切换到一个元素时被调用。而单击元素之外的其他地方或者通过按Tab键离开该元素，则会在原先调用focus事件的元素上调用blur事件。

DOM脚本可以使用这些事件来基于某个元素获得了焦点对文档进行修改。再来看一下chapter4/address/address.html中邮政编码的例子，你会注意到邮政编码字段的背景使用了不同的图

www.TopSage.com

案从视觉上表示该字段是必填的，如图4-5所示。

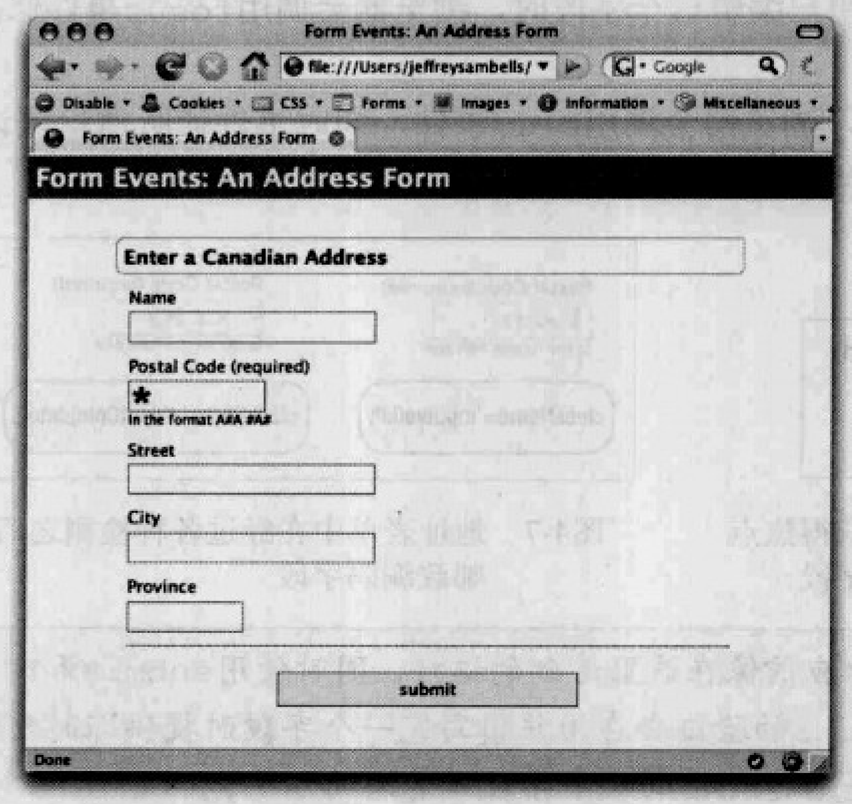



图4-5　在Firefox中查看地址表单时，邮政编码字段通过图案表示该字段是必填的

chapter4/address/address.js中也包含了为邮政编码字段添加blur和focus侦听器的载入事件侦听器：

```
ADS.addEvent(window,'load',function() {
    // 添加初始样式
    var postalCode = document.getElementById('postalCode');
    postalCode.className = 'inputMissing';

    // 当获得焦点时将类修改为编辑
    ADS.addEvent(postalCode,'focus',function(W3CEvent) {
        // 通过修改类来表示用户正在编辑这个字段
        this.className = 'inputEditing';
    });

    // 当失去焦点时对字段中的信息进行验证
    //并根据输入的值是否有效重新修改样式
    ADS.addEvent(postalCode,'blur',function(W3CEvent) {
        if(this.value == '') {
            // 修改类以表示缺少内容
            this.className = 'inputMissing';
        } else if(!isPostalCode(this.value)) {
            // 修改类以表示内容无效
            this.className = 'inputInvalid';
        } else {
            // 修改类以表示内容完成
            this.className = 'inputComplete';
        }
    });
});
```

www.TopSage.com

这些事件侦听器通过修改文本输入框的className属性，实现了文本输入框的样式随着获得和失去焦点而改变。当用户编辑这个字段时，浏览器会调用focus事件修改类以及关联的样式，如图4-6所示。

当退出这个字段时，浏览器会调用blur事件，同时事件侦听器检测该字段中是否包含有效的信息，并重新应用适当的类，如图4-7所示。

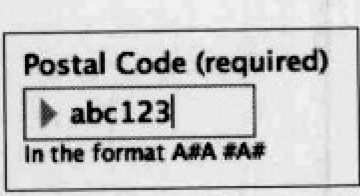



图4-6　地址表单中处于获得焦点状态的邮政编码字段

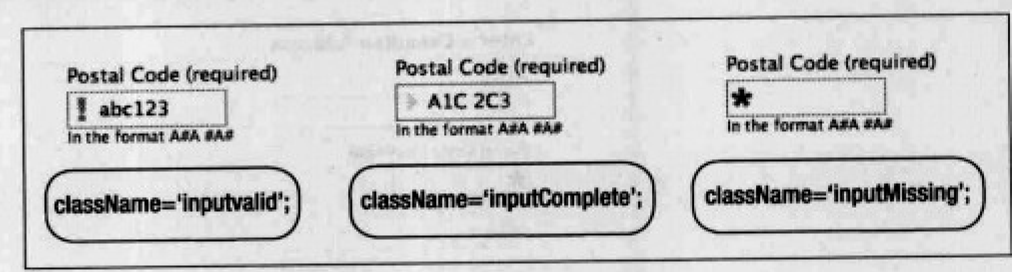



图4-7　地址表单中在经过各种检测之后处于失去焦点状态的邮政编码字段

> 理想的情况下，你应该像在这里看到的这样，同时使用submit和blur事件来验证用户在表单中的输入。基于blur的验证会在用户填写完一个字段时提供即时反馈，而基于submit的验证则用来检查用户是否忘记填写某个字段或者忽略了早先的警告。

### 3. change事件

change事件适用于<input>、<select>和<textarea>表单元素，该事件会在focus[①]事件发生后，当用户在focus和blur事件之间修改元素的值时被调用。

当需要根据某个字段值的变化修改文档时，你会发现change事件非常有用。同样，还以chapter4/address/address.html中的邮政编码字段为例，当你填写完邮政编码后，change事件会向服务器端脚本发送一个XMLHttpRequest请求。该请求不仅可以利用服务器端的服务来验证邮政编码的格式，而且还能验证邮政编码的真实性（即现实中的确存在该邮政编码）。与此同时，服务器端脚本可以查询街道、城市以及加拿大省的信息，并自动预先生成表单中其他字段的内容：

```
ADS.addEvent(window,'load',function() {
    var postalCode = ADS.$('postalCode');

    ADS.addEvent(postalCode,'change',function(W3CEvent) {

        var newPostalCode = this.value

        if(!isPostalCode(newPostalCode)) return;

        var req = new XMLHttpRequest();
        req.open('POST','server.js?postalCode=' + newPostalCode,true);
        req.onreadystatechange = function() {
            if (req.readyState == 4) {
                eval(req.responseText);
```

① 原文blur有误。——译者注

www.TopSage.com

```
                    if(ADS.$('street').value == '') {
                        ADS.$('street').value = street;
                    }
                    if(ADS.$('city').value == '') {
                        ADS.$('city').value = city;
                    }
                    if(ADS.$('province').value == '') {
                        ADS.$('province').value = province;
                    }
                }
            }
            req.send();
        });
    });
```

这个例子中直接使用了XMLHttpRequest对象，因此将会妨碍它在IE 6中运行。如果需要一个在IE中也有效的版本，请参考第7章。同样，这里引用的服务器端脚本server.js，也只是一个包含街道、城市以及省等测试信息的JavaScript文件：

```
var street = '123 Somewhere';
var city = 'Ottawa';
var province = 'Ontario';
```

> 在第7章中，你将看到许多为真实的Ajax请求准备的不同的服务器端选项。

最终的结果是得到一个既验证了邮政编码真实性，同时又获得了街道、城市和省信息的表单，如图4-8所示。

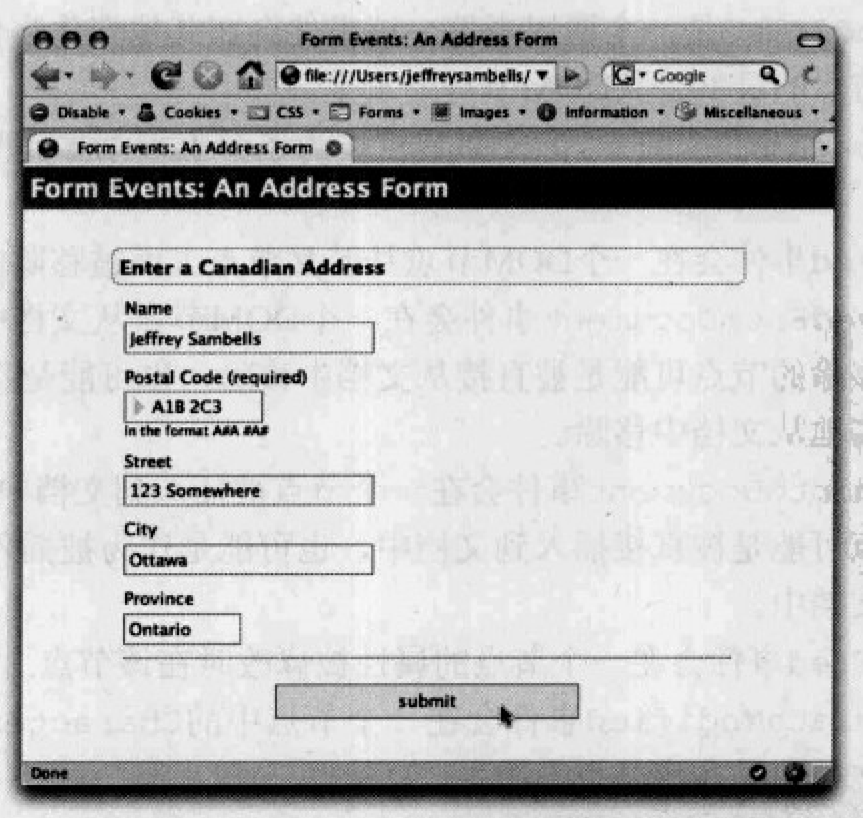



图4-8 当填写了有效的邮政编码后，地址表单会自动生成相应的地址信息（虚构的）

也可以使用相同的方法来查找其他信息，例如电话号码等。像这样细致的界面增强会给用户

www.TopSage.com

留下深刻印象，而且也会为用户节省很多填写表单的时间。

> 像本例中这样使用Ajax也存在一些可能的问题。我们会在第7章中深入研究Ajax的内容，所以如果你对Ajax还不熟悉或者现在还没有发现其中可能存在的问题，也不必担心。

### 4.2.6 针对W3C DOM的事件

在对W3C DOM2事件规范支持较多的浏览器，例如Firefox 2.0、Opera 9和Safari 2.0中，也存在一些与DOM相关的事件。目前，这些与DOM相关的事件还没有得到IE的支持，而且也不存在内置的等效功能。因此，本节将对这些事件作一简单介绍，以便当某一天它们被支持的时候，你可以参考这里的内容并进一步了解它们的细节。

在DOM2事件规范中，有三个用户界面事件：

- `DOMFocusIn`和`DomFocusOut`事件原理上与`focus`和`blur`事件相同。只不过`DOMFocusIn`和`DomFocusOut`适用于任何DOM元素而不仅适用于表单元素。浏览器会在鼠标指针移入和移出一个元素，或者使用键盘上的Tab键切换元素焦点时调用这两个事件。
- `DOMActivate`事件会在DOM元素被鼠标指针单击或者按下键盘上的某个键而激活时被调用。除了事件对象之外，事件侦听器还会取得一个表示激活类型的数字参数：
  - 1表示通过鼠标单击或按回车键引发的简单激活。
  - 2表示超级激活（hyperactivation），例如通过双击或按Shift+回车键引发的激活。

此外，当修改DOM文档的结构时，还会调用7种变化（Mutation）事件[①]：

- `DOMSubtreeModified`是一个通用事件，该事件会在其他变化事件发生之后，在发生其他变化事件的节点的最低公共DOM节点上发生。
- `DOMNodeInserted`事件会在一个DOM节点作为子节点被添加到另一个节点上面时在该节点上发生。
- `DOMNodeRemoved`事件会在一个DOM节点从其父节点上面被移除时在该节点上发生。
- `DOMNodeRemovedFromDocument`事件会在一个DOM节点从文档中被移除时在该节点上发生，这个被移除的节点可能是被直接从文档中移除，也可能是作为被移除的子DOM树的一部分被间接地从文档中移除。
- `NodeInsertedIntoDocument`事件会在一个节点被插入到文档中时在该节点上发生，这个被插入的节点可能是被直接插入到文档中，也可能是作为被插入的子DOM树的一部分被间接插入到文档中。
- `DOMAttrModified`事件会在一个节点的属性被修改时在该节点上发生。
- `DOMCharacterDataModified`事件会在一个节点中的`CharacterDate`被修改而这个节点仍然保留在文档中时在该节点上发生。

① 要了解这些事件的更多内容，读者可以参阅http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/events.html#Events-eventgroupings-mutationevents。——译者注

www.TopSage.com

以上只是对你最终会用到的几个针对DOM的事件[①]的简单概述。要了解更多的相关信息和这些内容的更新以及针对DOM的其他特性，读者可以参考本书网站http://advanceddomscripting.com。

### 4.2.7 自定义事件

有时候你可能会想到为自己的Web应用程序中的对象提供自己的自定义事件。例如，如果你的Web应用程序有保存功能，你可能会希望为其他开发者提供一个`save`事件，以便他们能够像你在前面使用`submit`事件那样，以相同的方式来调用更多的事件。

然而，为创建自定义事件提供一个跨浏览器的方案是一项艰难的工作，而且也超出了本书的范围。不过，在第9章中我们会介绍一些具备了高级事件系统的久经考验的JavaScript库，其中有几个就包含自定义事件。

## 4.3 控制事件流和注册事件侦听器

这里必须要提前声明一点，对于高级的跨浏览器事件处理而言，还没有出现过适当的解决方案。在排序事件、注册事件侦听器以及访问事件属性方面，每个浏览器都不相同。在本节中，我们将讨论事件流的基本内容，同时你还将为自己的ADS库创建几个方法，以便能够更好地完成符合W3C规范的和跨浏览器的事件处理。

### 4.3.1 事件流

在添加并调用事件侦听器之前，需要先熟悉有关事件流的一些基本概念。你很有可能到现在一直都忽略了，甚至都还不知道事件的冒泡或捕获阶段这样的概念。而这些概念对于理解何时需要处理可能互相嵌套的多个对象上的多重事件至关重要。同样，这些概念对于理解IE与W3C的事件流模型之间存在的彻底差异也非常重要。

为了更好地解释这些概念，我们来看一看本书源文件中的另一个文档。当在浏览器中打开chapter4/flow/flow.html时，这个文档应该如图4-9所示。

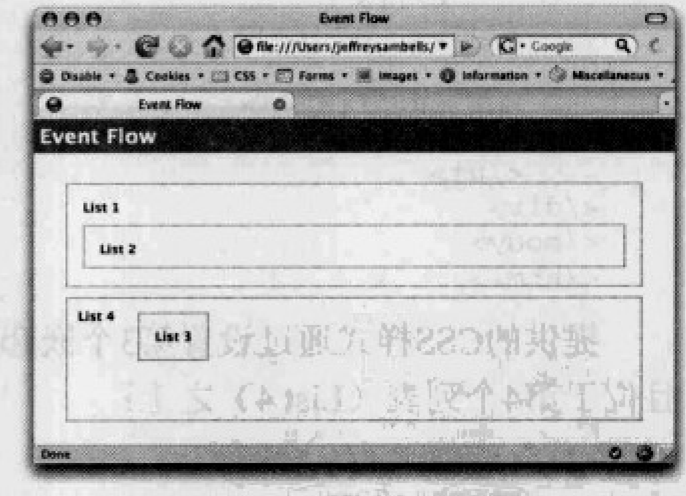



图4-9 Firefox中呈现的事件流示例页面

示范事件流的`flow.html`[②]文件标记中包含一些嵌套的列表和一些引入适当JavaScript和CSS文件的链接：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Event Flow</title>
    <!-- include some CSS style sheet to make
        everything look a little nicer -->
```

① 原文methods有误。——译者注

② 原文`index.html`有误。——译者注

www.TopSage.com

```
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="../chapter.css" />
    <link rel="stylesheet" type="text/css" href="style.css" />

    <!--ADS Library (full version from source linked here) -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>
    <!--Log object from Chapter 2
        (full version from source linked here) -->
    <script type="text/javascript"
        src="../../chapter2/myLogger-final/myLogger.js"></script>
    <!--A quick testing file -->
    <script type="text/javascript" src="flow.js"></script>
</head>
<body>
<h1>Event Flow</h1>
<div id="content">
    <ul id="list1">
        <li>
            <p>List 1 </p>
            <ul id="list2">
                <li>
                    <p>List 2 </p>
                    <ul id="list3">
                        <li>
                            <p>List 3 </p>
                        </li>
                    </ul>
                </li>
            </ul>
        </li>
    </ul>

    <ul id="list4">
        <p>List 4 </p>
    </ul>
</div>
</body>
</html>
```

提供的CSS样式通过设置第3个嵌套列表（List 3）的位置，使其在视觉上处于祖先元素之外、且位于第4个列表（List 4）之上：

```
#list1 {
    height:80px;
}
#list2 {
    height:20px;
}

#list3 {
    position:absolute;
    top:190px;
    left:100px;
}

#list4 {
    margin-top:10px;
```

www.TopSage.com

```
    height:100px;
}
```

这个页面的视觉效果与实际的标记嵌套结构是要关注的重点，因为即使List 3浮动到了List 4的上面，它仍然被嵌套在其祖先列表中。

下面，再看一看chapter4/flow/flow.js文件，其中为每个无序列表都附加了一个click事件侦听器。这些侦听器的主要任务就是修改被单击元素的类属性：

```
ADS.addEvent(window,'load',function() {

    // 为了演示问题所在，使用一个修改后的
    // addEvent方法。具体解释见本书正文
    function modifiedAddEvent( obj, type, fn ) {
        if(obj.addEventListener) {
            // W3C方式

            // 这个方法是在第1章的addEvent()方法
            // 基础上进行修改的，修改以后启用了
            // 捕获阶段而取消了冒泡阶段
            obj.addEventListener( type, fn, true );

        } else if ( obj.attachEvent ) {
            // Microsoft方式
            obj['e'+type+fn] = fn;
            obj[type+fn] = function(){obj['e'+type+fn](window.event);}
            obj.attachEvent( 'on'+type, obj[type+fn] );
        } else {
            return false;
        }
    }

    var counter = 0;

    // 取得无序列表
    var lists = document.getElementsByTagName('ul');
    for(var i = 0 ; i < lists.length ; i++ ) {

        // 注册单击事件侦听器
        modifiedAddEvent(lists[i],'click',function() {
            // 向段落中添加表示捕获单击事件先后顺序的数字
            var append = document.createTextNode(':' + counter++);
            this.getElementsByTagName('p')[0].appendChild(append) ;

            // 修改类名以突出显示被单击的元素
            this.className = 'clicked';
        });
    }
});
```

如果你在浏览器中单击List 3，可能会惊讶地发现List 1和List 2也改变了颜色（如图4-10所示），而这与你单击的List 3的位置无关。

> 如果你使用的是IE，则会发现位于列表名称后面的数字顺序恰好是反向的。其原因就在于事件流模型的差异，这一点稍后会解释。

www.TopSage.com

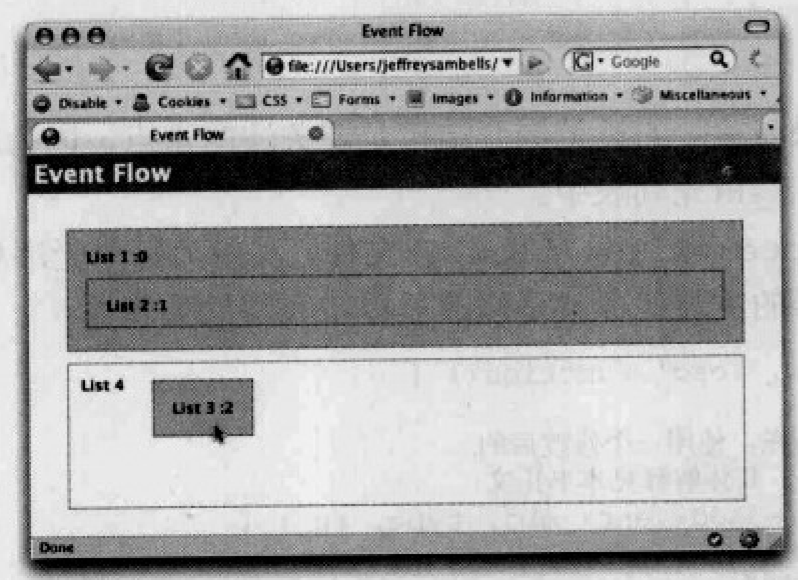



图4-10　当在Firefox中单击List 3之后的flow.html[①]页面

在这些click事件侦听器中，我们并没有采取防止或阻止事件流连续地穿过祖先树结构的操作。如果你忽略视觉上的页面布局而只关注实际嵌套的标记（其中每个子列表都被嵌套在父列表中），则会更容易理解这一过程。实际上，当你单击List 3时（如图4-11所示），也单击了其祖先列表。

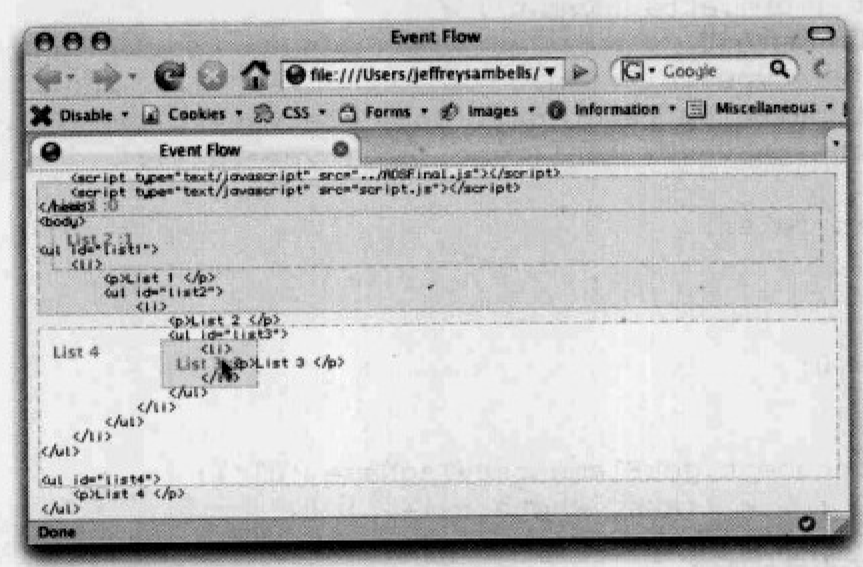



图4-11　单击List 3时就如同单击flow.html页面中List3标记的示意图

你可能还会对单击List 3时为什么List 4的颜色不变而感到奇怪。页面中元素的视觉布局和对象的层次的确对事件有影响，即位于最顶层的对象最早接收事件。虽然事件会在其目标元素及相应的祖先标记上被调用，但事件流却不会通过视觉样式层进行传递。同样地，对于图4-11所示的标记结构而言，是不可能同时单击两个不同元素的——除非它们之间存在嵌套关系，因此不可能同时在两个无关的元素（例如List 3和List 4）上面调用click事件。你可能会认为这样说不对，因为发生在List 2上面的click事件也会影响到List 1。事实上，还是同样的道理，那仍然是因为List 1和List 2在标记中存在着嵌套关系，而不是因为它们在页面视图上的位置。

### 1. 事件的顺序

如果仔细观察一下图4-10，就会注意到在事件侦听器被调用之后，出现在列表名称后面的数

① 原文eventFlow index page有误。——译者注

www.TopSage.com

字。这些数字表示的是事件被处理的顺序，0表示首先被处理。假如你在自己的浏览器中看到的结果与图4-10中的结果不一致，而且数字顺序相反，则说明你使用的不是与DOM2事件规范兼容的浏览器。但这不是浏览器中存在缺陷，而是因为我有意修改了addEvent函数，使得在兼容W3C标准的浏览器中以不同的方式注册了click事件侦听器：

```
function modifiedAddEvent( obj, type, fn ) {
    if(obj.addEventListener) {
        // W3C方式

        // 这个方法是在第1章的addEvent()方法
        // 基础上进行修改的，修改以后启用了
        // 捕获阶段而取消了冒泡阶段
        obj.addEventListener( type, fn, true );
    } else if ( obj.attachEvent ) {
        // Microsoft方式
        obj['e'+type+fn] = fn;
        obj[type+fn] = function(){obj['e'+type+fn](window.event);}
        obj.attachEvent( 'on'+type, obj[type+fn] );
    } else {
        return false;
    }
}
```

以上修改后的函数将DOM2事件规范中的addEventListener()方法的最后一个参数换成了true，即启用了事件流的捕获阶段，而禁用了冒泡阶段。这一修改只会影响像Firefox这样与W3C DOM2事件规范兼容的浏览器。当事件被注册在捕获阶段后，浏览器会先从最外层的祖先元素开始调用click事件，然后向内部依次调用到目标元素。然而，IE不支持捕获，其attachEvent()方法只能按照从目标元素向外依次通过祖先元素的顺序来冒泡事件。有关这一点我将在下面作出更详细一些的解释，不过现在我们先来看一看图4-12（Firefox中的截图）与图4-13（Microsoft IE的截图）之间的差别。

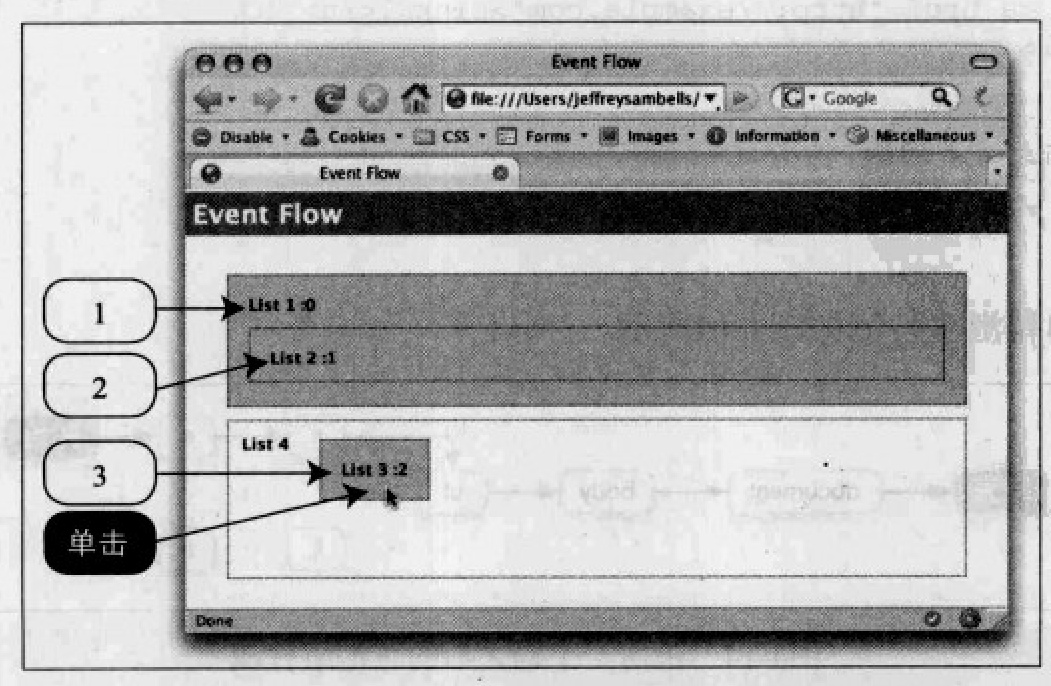



图4-12 在Firefox中看到的事件流

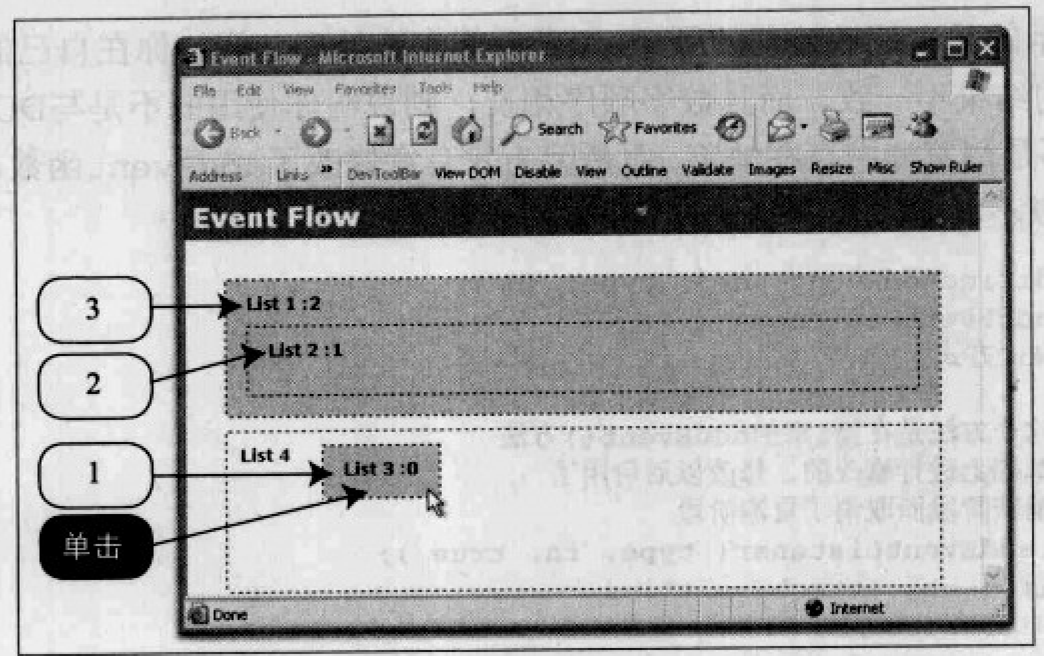



图4-13　在Microsoft IE 6中看到的事件流

**2. 两个阶段和三个模型**

事件冒泡和事件捕获恰好是相反的过程，它们最初都是浏览器厂商在市场竞争过程中提出来的。由Microsoft IE提出的事件冒泡，指的是目标元素的事件方法优先并且会被首先执行。然后，事件会向外传播到每个祖先元素，直至document对象。考虑下面这个非常简单的文档：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Sample Markup</title>
</head>
<body>
    <ul>
        <li>
            <a href="http://example.com">Link 1</a>
        </li>
        <li>
            <a href="http://example.com">Link 2</a>
        </li>
    </ul>
</body>
</html>
```

单击Link 1的冒泡事件流如图4-14所示。

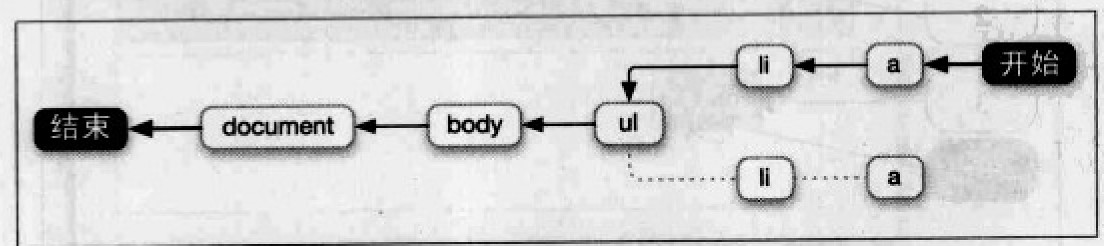



图4-14　单击一个锚之后的冒泡事件流

相反，由Netscape提出的事件捕获，则把优先权赋予了最外层的祖先元素，事件相应地被由

www.TopSage.com

外向内传播，直至抵达目标元素，如图4-15所示。

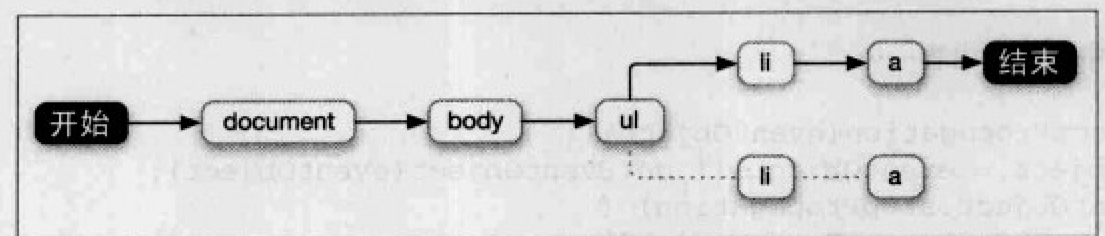



图4-15 单击一个锚之后的捕获事件流

在所有支持W3C DOM的现代浏览器中，冒泡事件流都是针对传统事件注册的默认事件流。因此，尽管可能没有意识到，但你最熟悉的可能就是冒泡方法。

> 在Microsoft IE 7及更低版本中，事件冒泡是唯一的选择。但愿IE 8会包含更多兼容DOM标准的特性，不过Microsoft并没有对此作出任何承诺。

事实上，W3C事件规范中并没有提出不同的事件流，但却通过同时包含捕获和冒泡阶段较好地解决了这两个极端的问题。这种方案可以让开发者自己为每个事件侦听器选择适用的事件流。在前面的例子中，我们为W3C方法选择了在捕获阶段注册事件侦听器，进而示范了它与IE之间的不同。

在W3C DOM2事件模型中，当目标元素被单击时，所有在捕获阶段注册的事件侦听器会依次被调用，直至目标元素。然后，事件流反向传播，所有在冒泡阶段注册的事件侦听器又会被依次调用，这一过程可以通过图4-16来表示。

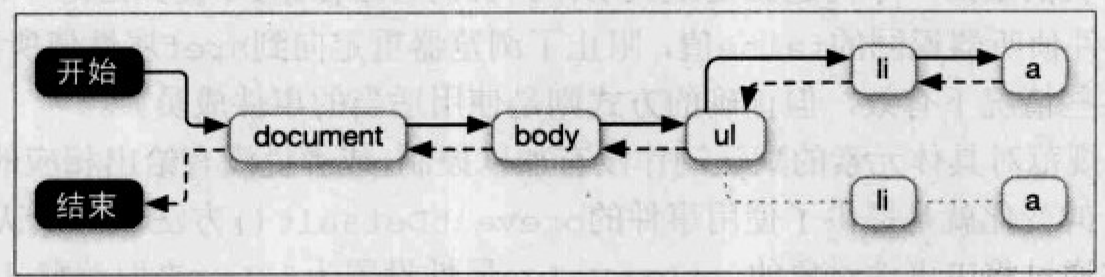



图4-16 单击一个锚之后的W3C捕获及冒泡事件流的示意图

通过将这两个阶段组合到一个模型中，DOM2事件方法提供了最佳解决方案。

### 3. 阻止冒泡

存在事件冒泡并不意味着不能阻止冒泡。在多数情况下，你可能都会希望事件只在单击的元素上发生，而不希望它传播到周围的元素上去。此时，你可以在“泡泡”离开对象之前刺破它，进而取消可能会沿着路径传播到更远的元素的任何冒泡事件。

要以W3C DOM2事件中的方法取消冒泡阶段，可以简单地在事件对象上调用`stopPropagation()`方法。同样地，对于Microsoft IE，你需要将事件的`cancelBubble`属性设置为`true`。

花点时间把下面的`ADS.stopPropagation()`方法添加到你的ADS库中。借助一点能力检测，就可以创建一个阻止传递到其中的事件进一步传播的跨浏览器的有用函数：

www.TopSage.com

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

function stopPropagation(eventObject) {
    eventObject = eventObject || getEventObject(eventObject);
    if(eventObject.stopPropagation) {
        eventObject.stopPropagation();
    } else {
        eventObject.cancelBubble = true;
    }
}
window['ADS']['stopPropagation'] = stopPropagation;

……以下是库中已有的内容……

})();
```

以上ADS.stopPropagation()方法依赖于作为参数传递到其中的事件对象。由于我们还没有讨论到如何访问事件对象，所以此时该方法中的getEventObject()方法还不存在，不过你会在本章后面把它添加到库中。

> DOM标准中相同的stopPropagation()方法也适用于捕获阶段。不过，要记住IE 7之前的版本都不支持捕获，因此不存在针对捕获阶段的等效的Microsoft方法。

**4. 取消默认动作**

与事件流有关的最后一个问题就是默认动作。我们曾经在第1章提到过这个问题，当时我们使用由click事件侦听器返回的false值，阻止了浏览器重定向到href属性值所指向的页面。虽然这种方式在某些情况下有效，但正确的方式则是使用适当的事件成员。

DOM2事件规范对具体元素的默认动作没有加以控制，或者说没有给出相应的规定。而DOM2事件规范中唯一的变化就是提供了使用事件的preventDefault()方法取消默认动作的方式。类似地，IE也允许通过将IE事件对象的returnValue属性设置为false来取消默认事件。

同样，再花点时间把下面这个跨浏览器的ADS.eventPreventDefault()函数添加到你的ADS库中，这个函数也要依赖在参数中传递进来的事件对象：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

function preventDefault(eventObject) {
    eventObject = eventObject || getEventObject(eventObject);
    if(eventObject.preventDefault) {
        eventObject.preventDefault();
    } else {
        eventObject.returnValue = false;
    }
}
```

www.TopSage.com

```
window['ADS']['preventDefault'] = preventDefault;
```

**……以下是库中已有的内容……**

```
})();
```

不过，在这里还需要提醒一点的是，有一些与默认动作关联的事件不能被取消。表4-1列出了各种DOM事件及它们的可取消（cancelable）属性。

**表4-1　DOM事件流中的哪个事件可以被取消**

| 事　件 | 可取消 |
|---|---|
| click | 是 |
| mousedown | 是 |
| mouseup | 是 |
| mouseover | 是 |
| mousemove | 否 |
| mouseout | 是 |
| load | 否 |
| unload | 否 |
| abort | 否 |
| error | 否 |
| select | 否 |
| change | 否 |
| submit | 是 |
| reset | 否 |
| focus | 否 |
| blur | 否 |
| resize | 否 |
| scroll | 否 |
| DOMFocusIn | 否 |
| DOMFocusOut | 否 |
| DOMActivate | 是 |
| DOMSubtreeModified | 否 |
| DOMNodeInserted | 否 |
| DOMNodeRemoved | 否 |
| DOMNodeRemovedFromDocument | 否 |
| DOMNodeInsertedIntoDocument | 否 |
| DOMAttrModified | 否 |
| DOMCharacterDataModified | 否 |

## 4.3.2　注册事件

在知道了事件侦听器需要遵循的顺序之后，就该在对象上注册事件侦听器了。与到目前为止

www.TopSage.com

我们看到的情况没什么两样，要适应不同的浏览器的确是一场恶梦。这些浏览器不仅具有不同的事件和不同的事件流处理方式，而且它们注册事件侦听器的方式也不一样。

**1. 嵌入式注册模型**

在第1章，我们曾经探讨了在文档中包含脚本的各种方式：在文档主体中使用嵌入的`<script>`元素、在文档头部嵌入`<script>`标签和使用外部文件。你也看到了像下面这样的嵌入式的事件侦听器：

```
<a href="http://example.com" ➥
    onclick="window.open(this.href); return false;" ➥
>http://example.com</a>
```

其实可以通过更好的方式来实现，比如使用渐进增强和不唐突的DOM脚本技术。在事件的概念刚刚出现时，嵌入式的事件注册是唯一可用的方法，这种方式需要在标记中把每个事件作为HTML属性进行硬编码。作为唯一的方法，它强迫开发人员为每个元素都重复编写相同的代码，这不仅导致标记混乱不堪，而且还会使文件明显变大。直到W3C提出友好的事件注册方式之前，浏览器开发商才意识到嵌入式注册的问题并拿出了解决方案，其中一些方案已经被整合到你在前面创建的`ADS.addEvent()`方法中了。

**2. 深入理解`ADS.addEvent()`方法**

到目前为止，你看到的所有例子都在使用第1章创建的自定义的`ADS.addEvent()`方法，将事件侦听器添加到DOM元素中。这个最早由Scott Andrew LePera（http://www.scottandrew.com/weblog/articles/cbs-events）开发并推而广之的`addEvent()`方法，是一个将W3C和Microsoft不同的事件模型组合到一个函数中的简单的包装方法：

```
function addEvent(obj, evType, fn, useCapture){
    if (obj.addEventListener){
        obj.addEventListener(evType, fn, useCapture);
        return true;
    } else if (obj.attachEvent){
        var r = obj.attachEvent("on"+evType, fn);
        return r;
    } else {
        alert("Handler could not be attached");
    }
}
```

显然，Microsoft和W3C的方法完全不同，而且根本就不应该对它们一视同仁，许多人都认为使用像`addEven()`这样的方法实际上弊大于利。对原始的`addEvent()`函数而言，我倾向于同意这种看法。由于Microsoft的事件模型不允许捕获，因此其中的`useCapture`参数多少有点暧昧不清。而且，正如你将在下面看到的，侦听器环境下的`this`关键字在IE中和在W3C中也是不相同的。

我们在第1章创建的`addEvent()`函数的新版本，是在John Resig（http://ejohn.org/projects/flexible-javascript-events/）编写的替代原始的`addEvent()`函数的版本基础上稍加修改完成的：

```
function addEvent( obj, type, fn ) {
    if ( obj.attachEvent ) {
```

www.TopSage.com

```
        obj['e'+type+fn] = fn;
        obj[type+fn] = function(){obj['e'+type+fn]( window.event );}
        obj.attachEvent( 'on'+type, obj[type+fn] );
    } else
    obj.addEventListener( type, fn, false );
}
```

这个新版本去掉了useCapture参数，将DOM的默认方法设置为只接收事件冒泡（与IE相似），而且通过使用匿名函数使this关键字在Microsoft和W3C的环境中保持一致，其中this引用的是将侦听器指派给的对象。

我觉得这个addEvent()方法（以及我们在第1章中轻微修改的版本）是以相同的方式来对待两种事件模型的，因此你可以安全地使用它而不必犹豫。

在理解了不同的模型以及如何解决这些问题之后，我们再看一下每种注册方法以及它们各自的优缺点。

### 3. 传统的事件模型

使用传统方法在任何浏览器中为特定的对象注册事件侦听器时，你要做的就是定义当事件发生时你希望执行的JavaScript方法，然后把这个方法指定给对象的事件侦听器属性。这种传统方法你曾经在第1章使用锚标签的onclick属性时见到过，类似下面这样：

```
// 传统的事件注册
window.onload = function() {
    var anchor = document.getElementById('example');
    anchor.onclick = function(){
        // 单击事件发生时执行的代码
    }
}
```

传统方法中的基本概念就是将事件侦听器视为一个指定给与事件相关的方法的JavaScript函数。而对象中与事件相关的方法就是事件名称再加上on前缀，例如与click事件相关的onclick方法。

如果想移除锚上的事件侦听器，可以简单地将该方法设置为null，侦听器的定义就被取消了：

```
anchor.onclick = null;
```

在传统的方法中，每个事件侦听器都类似一个常规的JavaScript方法，因此可以通过直接执行该方法来手工调用事件：

```
anchor.onclick();
```

注意，当手工调用事件时在方法末尾要添加圆括号。这一点是非常重要的，因为圆括号会告诉JavaScript执行相应的函数。如果没有圆括号，JavaScript将把该方法简单地看成对函数实例的引用，就如同将eventListener方法指定为onclick的侦听器一样。如果想更完美一些，还应该组合使用能力检测以首先保证确实存在onclick方法，因此以上代码可以修改为：

```
if(anchor.onclick ) {
    anchor.onclick();
}
```

通过先行检查，可以保证JavaScript不会因执行不存在的方法而发生意外错误。而且，手工调

www.TopSage.com

用事件也不能访问事件对象，因为不存在。

传统方法是最直观的也是不针对特定浏览器的事件注册方法，但这种方法会创建很多令人费解的环境，而且并不完美。

首先，如果你还记得第2章中有关this关键字的讨论，那么就会注意到传统事件注册方法中的this关键字引用的是目标对象。虽然this引用附加了事件侦听器的对象可能也正是你所希望的，但也是你必须要明确知道的。

其次，事件侦听器只能是一个单独的函数。如果你想在同一个对象的某个特定事件发生时调用多个侦听器，就必须将多个侦听器包装在一个函数体内。当然这也很简单：

```
link.onclick = function() {
    eventListenerA();
    eventListenerB();
}
```

不过，要阻止其中一个侦听器将涉及使用更多的逻辑代码按照需要组合事件侦听器，而这正是令人讨厌的地方。同时，要想移除个别的事件侦听器也并非易事。

最后，传统的方法从属于浏览器默认的事件流。没有办法指定是在捕获阶段还是在冒泡阶段调用事件。

#### 4. Microsoft特有的事件模型

Microsoft在传统方法的基础上向前迈进了一步，它定义了自己的与事件相关的方法，通过这些方法可以注册事件侦听器——但仅对IE适用。在Microsoft的解决方案中，分别使用attachEvent()和detachEvent()方法注册和移除事件侦听器。

要在具体的对象上注册一个事件侦听器，必须首先定义一个JavaScript函数，然后使用该对象的attachEvent(event[1],listener)方法，将作为第2个参数的侦听器指定给作为第1个参数的特定事件。而事件的名称中仍然要使用与传统方法相同的on前缀：

```
// Microsoft事件注册
function eventListener() {
    // 响应单击事件的代码
}
window.attachEvent('onload', function() {
    var link = document.getElementById('example');
    link.attachEvent('onclick',eventListener);
});
```

同样，可以在此后使用带有相同参数的detachEvent()方法来移除事件侦听器。

```
link.detachEvent('onclick',eventListener);
```

使用Microsoft的方法，也可以为同一个对象指定多个事件侦听器：

```
link.attachEvent('onclick', eventListenerA);
link.attachEvent('onclick', eventListenerB);
link.attachEvent('onclick', eventListenerC);
```

而且，还可以使用fireEvent()方法来手工调用事件：

---

① 原文object有误。——译者注

www.TopSage.com

```
link.fireEvent("onclick");
```

虽然Microsoft的方法比传统方法要清晰一些，但仍然存在几个问题。

首先，这种模型只对IE有效，对其他浏览器毫无用处。

其次，与传统的模型不同，Microsoft的模型只是引用而非复制事件侦听器，因此在使用this关键字时，this引用的将是原始的JavaScript函数（通常都位于window对象中），而不是附加事件侦听器的那个对象。

最后，IE不支持事件捕获，因此无法指定事件流的捕获阶段，而且所有事件始终会冒泡——除非使用前面介绍的cancelBubble属性[①]来阻止冒泡。在IE同时实现捕获阶段之前没有更好的解决方案。

#### 5. W3C DOM2事件模型

为了突破传统模型的限制，同时进一步改进Microsoft的模型，W3C拿出了折衷的方案，即同时包含了事件流的两个阶段以及类似的方法。

DOM2事件规范中包含addEventListener()和removeEventListener()方法，这两个方法都接受事件和事件侦听器参数（这与Microsoft方法相似），同时还允许通过第3个参数指定事件阶段。此外，W3C采取了去掉on前缀的方案，因此所有事件都必须使用事件名称而非传统的方法名称来标识：

```
// W3C事件注册
function eventListener() {
    // 响应单击事件的代码
}
window.addEventListener('load', function(W3CEvent) {
    var link = document.getElementById('example');
    link.addEventListener('click',eventListener,false);
}, false);
```

如果第3个参数是true，事件侦听器将在捕获阶段内执行；否则，如果是false，则会在冒泡阶段发生（正如你在本章前面所看到的那样）。

可以使用removeEventLinstener()方法移除事件侦听器：

```
link.removeEventListener('click',eventListener,false);
```

而且，也可以为同一个对象添加任意多个事件侦听器：

```
link.addEventListener('click',eventListenerA,false);
link.addEventListener('click',eventListenerB,false);
link.addEventListener('click',eventListenerC,false);
```

> 不过，对于在一个事件的相同阶段上为对象注册多个事件侦听器的情况，W3C DOM2事件规范没有规定事件发生的先后顺序。仅凭首先添加的是eventListenerA，并不能确定在事件发生时该侦听器会第一个或者最后一个被执行。保证事件侦听器按照特定顺序发生的唯一方法，就是只在对象上注册一个事件侦听器，然后在这个事件侦听器内部按期望的顺序调用多个函数。

① 原文returnValue有误。——译者注

www.TopSage.com

在W3C模型中，也可以通过组合document.createEvent()方法和对象的dispatch-Event()方法来手工调用事件。要模仿一次单击事件，需要创建一个MouseEvent事件，并在将该事件分派给由anchor变量引用的DOM元素之前初始化它的属性：

```
// 使用W3C的方法手工调用事件
var event = document.createEvent("MouseEvents");
event.initMouseEvent(
'click', // 事件类型
true, // 可以冒泡
true, // 可以取消
window, // 视图类型
0, // 鼠标单击数
0, // 屏幕的x轴坐标
0, // 屏幕的y轴坐标
0, // 客户端的x轴坐标
0, // 客户端的y轴坐标
false, // 是否按住了Ctrl键
false, // 是否按住了Alt键
false, // 是否按住了Shift键
false, // 是否按住了Meta键
0, // 表示按下的鼠标按钮的次数
null // 相关的目标对象
);
anchor.dispatchEvent(evt);
```

我们将在下一节更详细地讨论W3C DOM2事件规范中的MouseEvent事件及其属性，届时你会看到如何在事件侦听器内部访问事件对象。

**6. load事件的问题**

不论使用哪种事件注册方法，通过window对象的load事件来初始化DOM脚本都会存在一个固有的问题。当页面中包含许多大文件（比如在标记中嵌入的<img>元素）时，load事件会一直等到所有图像全部载入完成后才会被调用。如果你从来没在网页中使用过嵌入的图像，那就无所谓了。但是，如果你创建的应用程序需要使用嵌入的图像（例如一个照片管理工具），那么你可能会希望load事件在嵌入的图像载入完成之前运行。要解决这个问题，你需要在自己的ADS库中再添加下面这个ADS.addLoadEvent()方法：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

function addLoadEvent(loadEvent,waitForImages) {
    if(!isCompatible()) return false;

    // 如果等待标记是true则使用常规的添加事件的方法
    if(waitForImages) {
        return addEvent(window, 'load', loadEvent);
    }

    // 否则使用一些不同的方式包装loadEvent()方法

    // 以便为this关键字指定正确的内容，同时确保
    // 事件不会被执行两次
```

www.TopSage.com

```
    var init = function() {

        // 如果这个函数已经被调用过了则返回
        if (arguments.callee.done) return;

        // 标记这个函数以便检验它是否运行过
        arguments.callee.done = true;

        // 在document的环境中运行载入事件
        loadEvent.apply(document,arguments);
    };

    // 为DOMContentLoaded事件注册事件侦听器
    if (document.addEventListener) {
        document.addEventListener("DOMContentLoaded", init, false);
    }

    // 对于Safari，使用setInterval()函数检测
    // document是否载入完成
    if (/WebKit/i.test(navigator.userAgent)) {
        var _timer = setInterval(function() {
            if (/loaded|complete/.test(document.readyState)) {
                clearInterval(_timer);
                init();
            }
        },10);
    }

    // 对于IE（使用条件注释）
    // 附加一个在载入过程最后执行的脚本，
    //并检测该脚本是否载入完成
    /*@cc_on @*/
    /*@if (@_win32)
    document.write("<script id=__ie_onload defer ➡
src=javascript:void(0)><\/script>");
    var script = document.getElementById("__ie_onload");
    script.onreadystatechange = function() {
        if (this.readyState == "complete") {
            init();
        }
    };
    /*@end @*/
    return true;
}
window['ADS']['addLoadEvent'] = addLoadEvent;
```

……以上是库中已有的内容……

```
})();
```

以上改进载入事件的方法是以Dean Edwards（http://dean.edwards.name/weblog/2005/09/busted）论述的解决方案为基础的，其中使用了不同的方法在图像完成载入之前调用DOM脚本：

- 如果浏览器中存在`addEventListener()`方法，则使用`DOMContentLoaded`事件，该事件会在文档标记载入完成时被调用。
- 对于Safari，则使用`setInterval()`函数周期性地检查`document`的`readyState`属性，随时监控文档是否载入完成。
- 对于IE，则运用了一点小技巧。即向文档中写入一个新的`script`标签，但该标签会延迟

www.TopSage.com

到文件最后载入。然后，使用script对象的onreadystatechange方法在进行类似的readyState检查后及时调用载入事件。

将ADS.addLoadEvent()方法的第2个参数设置为true，可以调用包含在其中的原始的ADS.addEvent(window,'load'...)方法。这样，就能够在这两个方法之间方便地进行切换了。

可以打开chapter4/load/load.html来实际地观察一下改进后的载入事件的应用。这个例子中包含了一个大小为580 KB的JPG图像（The Blue Marble，该图像的链接位于NASA的Visible Earth网站http://visibleearth.nasa.gov/上面）以及位于chapter4/load/load.js文件中的3个载入事件：

```
// 使用常规的addEvent()方法为window对象注册载入事件
ADS.addEvent(window,'load',function(W3CEvent) {
    ADS.log.write('ADS.addEvent(window,load,...) invoked');
});

// 使用改进后的addLoadMethod()方法
ADS.addLoadEvent(function(W3CEvent) {
    ADS.log.write('ADS.addLoadEvent(...) invoked');
});

// 通过改进后的addLoadMethod()方法调用原始的addEvent()方法
ADS.addLoadEvent(function(W3CEvent) {
    ADS.log.write('ADS.addLoadEvent(...,true) invoked');
},true);
```

当页面载入后，中间的ADS.addLoadEvent()方法添加的载入事件将会先于图像载入完成被调用。而第一个和最后一个载入事件则需要一直等到最后才会被调用，图4-17显示了正在载入图像的情景。

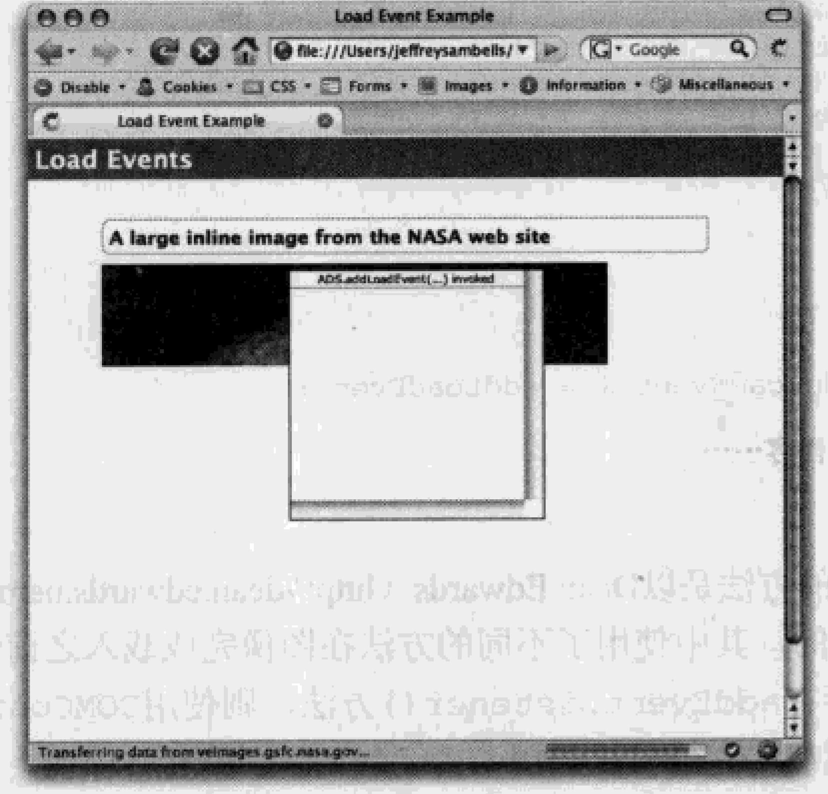



图4-17　在图像载入完成之前显示的addLoadEvent()事件已经发生的日志条目

如果重新载入（刷新）页面并且已经缓存了图像，则会注意到第二组载入事件几乎会在图像

www.TopSage.com

迅速地从缓存中被载入后立即发生。

使用这个改进方法的唯一注意事项，就是如果你的DOM脚本需要访问针对图像的属性（例如宽度和高度），那么当脚本运行时图像有可能还没有载入完成——这就是提前执行载入事件的全部问题。

### 4.3.3 在事件侦听器中访问事件对象

到现在为止，你已经熟悉了事件流及为对象注册事件侦听器的内容，但我们还没有讨论事件侦听器本身。所谓事件侦听器，其实就是一个常规的JavaScript函数，不过这个函数也要符合几个特殊的要求。

你会注意到在下面这个函数中没有定义任何参数：

```
function eventListener() {
    // 你的代码
}
```

在前面的例子中，我们曾经使用this（或作用域中的其他变量）来操纵和影响文档。由于无法让浏览器知道哪个参数是为哪个侦听器定义的，即浏览器会认为所有事件侦听器都相同，所以不可能在事件侦听器中指定自定义的参数。然而，这种说法并不总是正确——不过同样，正确与否还是取决于浏览器。

在W3C的模型中，事件侦听器会取得一个表示事件自身的参数：

```
function eventListener(W3CEvent){
    // 你的代码
}
```

但在IE中，事件侦听器则不会取得任何参数，相应的事件对象被保存在window.event中，所以需要通过简单的检测来在这两种可能之间进行转换：

```
function eventListener(W3CEvent){
    var eventObject = W3CEvent || window.event;
    // 你的代码
}
```

**1. 简捷的语法**

由于事件在获得跨浏览器支持的过程中存在许多问题，因此需要很多关键的检测才能最终确定选择哪种方法。与其编写长长的if/else语句，我们将使用更节省版面和键盘输入量的一些简捷的语法形式，因此你会看到下面这样的代码：

```
var example = (a != 2) ? 'no' : 'yes';
```

这是一条运用三元操作符的简捷语句。它与下面这种写法是等价的：

```
if (a!= 2){
    var example = 'no';
} else {
    var example = 'yes';
}
```

说不定什么时候你就会碰到这种使用三元操作符的语法，所以最好现在把它搞明白。

www.TopSage.com

此外，你可能还会看到下面这样的代码：

```
var example = a || b;
```

在这个例子中，example应该取得a或b的值，即只要a的值不是null、undefined或false，a的值就会被赋给example；否则，将会使用b的值。这种语法对于设置可选变量的默认值非常有用。对这个例子而言，b应该包含默认值，而如果指定了a的值，那么a的值则会覆盖b中的默认值。

**2. ADS.getEventObject()方法**

记住要取得适当的事件对象是非常令人讨厌的，因此最好还是花点时间把下面的ADS.getEventObject()函数添加到你的ADS库中：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

function getEventObject(W3CEvent) {
    return W3CEvent || window.event;
}
window['ADS']['getEventObject'] = getEventObject;

……以下是库中已有的内容……

})();
```

现在你就可以使用ADS.getEventObject()函数方便地访问事件对象，而不必每次都手工编写检测代码了。

```
function eventListener(W3CEvent) {
    var eventObject = ADS.getEventObject(W3CEvent);
    // 涉及this、作用域键或eventObject的代码
}
```

这个辅助函数为你的代码适应未来提供了另一层机制，即可以在其他浏览器需要以不同的方法取得事件对象时，对其进行适当的修改而不会影响调用该函数的原有代码。而且，前面添加的ADS.preventDefault()和ADS.stopPropagation()方法也整合了这个方法，所以一定不要忘记在你的库中添加这个方法。

> 如果你在ADS库的外部定义事件侦听器，则需要添加ADS对象前缀，因为侦听器的作用域链位于库的外部。如果你在库的内部定义侦听器，则不需要添加任何前缀，因为相应的作用域链会包含ADS命名空间。要了解更多有关作用域链的内容，请参考第1章和第2章。

同以前一样，W3C和Microsoft在定义事件对象时也存在一些差别，因此对于彻底的跨浏览器解决方案来说还必须作进一步检查。

### 4.3.4　跨浏览器的事件属性和方法

当你在阅读本节内容时，最好把手从头上拿开，以免在编写跨浏览器的事件侦听器时拽掉自

www.TopSage.com

己的头发。现在我们就从W3C DOM2事件规范谈起，然后再看看不同浏览器间的不一致性。

在第3章中，你已经知道了DOM文档中的每个节点实际上都扩展自其他一些DOM对象。同样的逻辑也适用于事件对象。在W3C模型中，传递到事件侦听器中的事件对象并不总是相同的，不过这些事件对象却拥有相同的属性。当浏览器调用load事件时，传递到事件侦听器中的对象是DOM2事件规范中定义的Event对象，而当调用click事件时，传递到事件侦听器中的对象则是一个MouseEvent对象。与DOM document对象类似，每个事件对象都扩展并维护了其他事件对象的方法和属性，如图4-18所示。

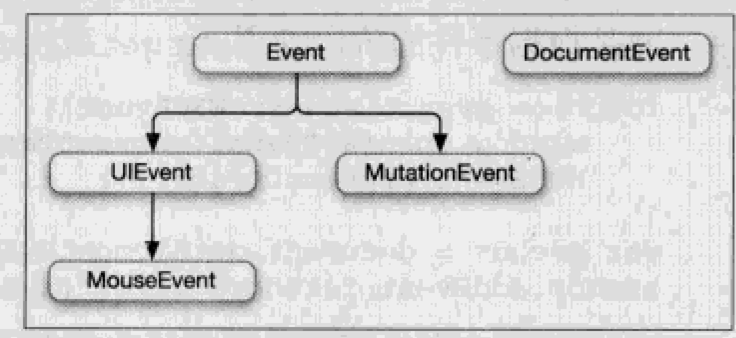



图4-18 DOM2事件对象的继承关系示意图

下面我们主要看一看Event和MouseEvent对象的属性，并介绍通过这些属性可以获得哪些信息。

### 1. DOM2事件规范中的Event对象

事件对象中包含着用于控制事件流和目标对象的方法和属性。W3C DOM2事件模型为Event对象定义了下列属性。

- bubbles，是一个布尔值，表示事件是否是冒泡阶段的事件。如果是冒泡阶段的事件，bubbles值为true；否则，值为false。
- cancelable，是一个布尔值，表示事件是否具有可以被取消的默认动作。如果事件具有的默认动作可以被取消，cancelable值为true；否则值为false。
- currentTarget，是当前正在处理的事件侦听器所在的事件流中的DOM元素。currentTarget可能会不同于target属性，因为在事件流的捕获或冒泡阶段中，事件侦听器可能不是注册到target（目标对象）上，而是注册到目标的祖先元素上。
- target，是DOM文档中最早调用事件序列的目标对象（EventTarget对象的实例）。
- timestamp，是一个DOMTimeStamp对象，可以用来确定自创建事件的纪元时间算起经过的毫秒数，但不一定在所有系统中都有效。这里纪元的概念可以指系统创建的时间，也可以指UTC时间1970年1月1日0时0分0秒。
- type，是一个包含事件名称（例如click）的字符串值。
- eventPhase，表示当前事件侦听器处于事件流的哪个阶段。使用整数1～3表示，不过也可以在比较表达式中使用Event常量CAPTURING_PHASE,、AT_TARGET和BUBBLING_PHASE，如下所示：

```
function eventListener(W3CEvent){
    switch(W3CEvent.eventPhase){
        case Event.CAPTURING_PHASE:
            // 如果处于捕获阶段要运行的代码
            break;
        case Event.AT_TARGET:
            // 如果当前是目标对象要运行的代码
            break;
        case Event.BUBBLING_PHASE:
```

www.TopSage.com

```
        // 如果处于冒泡阶段要运行的代码
        break;
}
```

这些事件属性可以确保以正确的方式来访问正确的对象。例如，可以不必依赖于this关键字，而是根据事件的阶段使用target或currentTarget属性：

```
function eventListener(W3CEvent){
    window.open(W3CEvent.currentTarget.href);
    W3CEvent.preventDefault();
}
var anchor = document.getElementById('example');
anchor.addEventListener('click',eventListener,false);
```

W3C DOM2事件模型也定义了我们前面讨论过的下列Event对象的方法。

- initEvent(eventType,canBubble,cancelable)，用于初始化通过document.createEvent('Event')方法创建的事件对象。
- preventDefalt()，用于取消对象的默认动作（如果可以取消），例如取消浏览器重定向到一个锚元素的href属性的动作。
- stopPropagation()，用于停止事件流的进一步执行，包括捕获阶段、目标对象和冒泡阶段。当调用这个方法时，所有侦听器仍然会在当前层次上执行，但事件流不会继续超出currentTarget。

### 2. DOM2事件规范定义的MouseEvent对象

对于所有W3C DOM鼠标事件而言，传递到事件侦听器中的事件对象都是MouseEvent对象的实例。鼠标事件对象中包含与鼠标指针的位置有关的属性，也包含在鼠标事件发生的同时可能会按住的键盘中某个键的信息。MouseEvent对象的属性如下。

**altKey**、**ctrlKey**和**shiftKey**，altKey、ctrlKey和shiftKey都是布尔值，分别表示在鼠标事件发生时是否按住了键盘上的Alt、Ctrl或Shift键。

**button**，button中会包含表示哪个鼠标键被按下的一个整数值。鼠标上每个键与整数的对应关系如下：

- 0表示鼠标左键。
- 1表示鼠标中键（如果有）。
- 2表示鼠标右键（或Apple单键鼠标中的Command-click）。

理想的情况下，W3C还应该包含表示每种可能性的常量，因此对于下面的代码：

```
if(W3CEvent.button==0){
    // 左键单击的代码
}
```

还可以让它变得意思更明确一些，例如：

```
if(W3CEvent.button==MouseEvent.BUTTON_LEFT){
    // 左键单击的代码
}
```

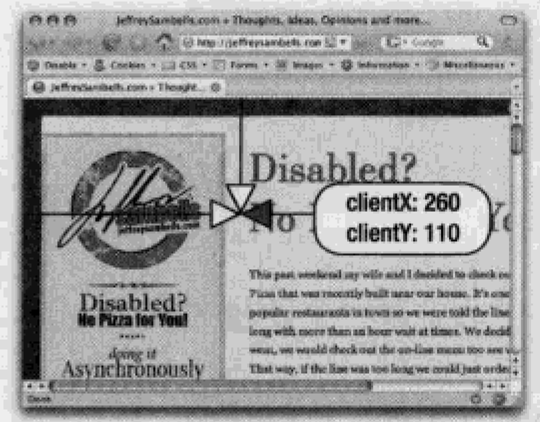
**clientX**和**clientY**，根据W3C规范，clientX和clientY是指“事件发生位置相对于DOM实现的客户端区域的[水平和垂直]坐标”。虽然表达没有问题，但实际上并不十分清楚。“客户端

www.TopSage.com

区域”的意思是浏览器窗口还是整个浏览器窗体？如果在符合W3C标准的浏览器中为document对象添加以下click事件：

```
document.addEventListener('click',function(W3CEvent) {
    alert('client: (' + W3CEvent.clientX + ',' + W3CEvent.clientY + ')');
},false);
```

你会发现clientX和clientY表示的位置是相对于浏览器窗口而不是文档的，如图4-19所示。当你在滚动页面之后仍然在窗口中的同一个位置上单击时，所得到的坐标值是相同的。



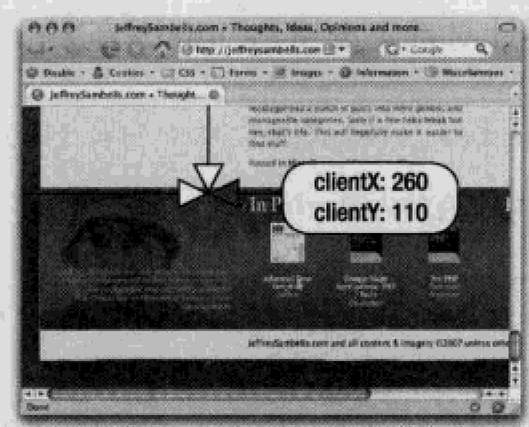




图4-19 滚动网页前后clientX和clientY表示的位置

**screenX**和**screenY**，screenX和screenY与前面两个客户端属性类似，只不过它们分别表示事件发生位置相对于客户端所在屏幕的水平和垂直坐标，如图4-20所示。

**MouseEvent**，MouseEvent对象有一个relatedTarget属性，该属性引用的是与事件相关的“次要目标”。在多数情况下，这个属性值都是null，但在mouseover事件中，它引用的是鼠标离开的前一个对象。同样地，在mouseout事件中，它引用的是鼠标之前进入的那个对象。

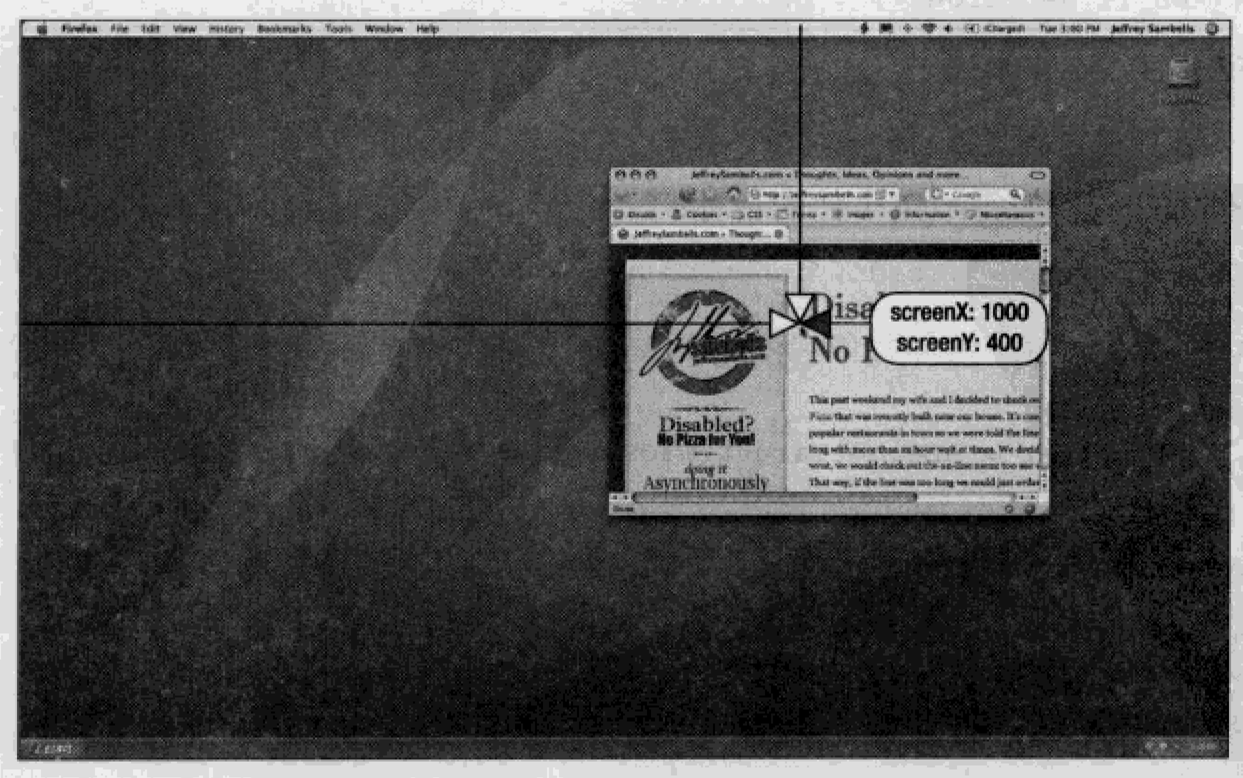



图4-20 screenX和screenY在网页中表示的位置

www.TopSage.com

### 3. 处理诸多浏览器不兼容性问题

虽然坚持使用W3C方法是理想的解决方案，但如果你想要的是不针对特定浏览器的解决方案，那么仅使用W3C方法是不现实的。下面我们介绍一些处理浏览器间不兼容性的技术。

**访问事件的目标元素**。这可能是你遇到的第一件麻烦事。IE没有提供`target`或`currentTarget`属性，但却提供了一个`srcElement`属性。而且，Apple中的Safari浏览器当DOM元素包含一个文本节点时也会制造点小问题。对于Safari而言，文本节点会代替包含它的元素变成事件的目标对象。要解决这些问题，需要把下面的`ADS.getTarget()`方法加入到你的ADS库中，以便提供不针对特定浏览器的解决方案：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

function getTarget(eventObject) {
    eventObject = eventObject || getEventObject(eventObject);

    // 如果是W3C或MSIE的模型
    var target = eventObject.target || eventObject.srcElement;

    // 如果像Safari中一样是一个文本节点
    // 重新将目标对象指定为父元素
    if(target.nodeType == ADS.node.TEXT_NODE) {
        target = node.parentNode;
    }

    return target;

}
window['ADS']['getTarget'] = getTarget;

……以下是库中已有的内容……

})();
```

通过这个方法可以取得给定事件的目标：

```
ADS.addEvent(window, 'load', function() {

    function eventListener(W3CEvent) {

        // 取得目标
        var target = ADS.getTarget(W3CEvent);

        // target现在引用的是一个适当的元素
        window.open(target.href);
    }
    var anchor = document.getElementById('example');
    addEvent(anchor, 'click', eventListener);

});
```

**确定单击了哪个鼠标键**。这也会涉及一些问题。回忆一下在上一节中，W3C规定：

www.TopSage.com

- 0表示按下了左键。
- 1表示按下了中键。
- 2表示按下了右键。

如果你熟悉数组（关键是它们的索引是从0开始的），那么可以将第一个键（从左往右）理解为索引0，因此左键=0是有意义的。同样地，将鼠标的3个键分别用0、1、2来表示也就有意义了。但是，Microsoft则另辟蹊径，在它的方案中不仅包含了多个键的组合，甚至还提出了“没按键”的概念。Microsoft认为：

- 0表示没有键被按下。
- 1表示按下了左键。
- 2表示按下了右键。
- 3表示同时按下了左右键。
- 4表示按下了中键。
- 5表示同时按下了左中键。
- 6表示同时按下了右中键。
- 7表示同时按下了所有三个键。

这就让我们更加头痛了。例如，W3C用1表示中键，而IE用1表示左键。而且，W3C和Microsoft还都使用了相同的button属性保存这些信息，因此围绕button属性不可能有太好的能力检测方案出台。

为了在不进行浏览器检测的基础上创建通用的方法，必须要有点创意并使用事件对象的toString()方法的返回值。首先，检测是否存在toString()方法，以及该方法返回的结果是不是MouseEvent。如果toString()方法存在并且是MouseEvent，那算你走运，你可以不考虑浏览器而直接使用W3C的方法。否则，如果仍然存在button属性，那就可以假设是IE了。现在请把下面的ADS.getMouseButton()方法添加到你的ADS库中：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

function getMouseButton(eventObject) {
    eventObject = eventObject || getEventObject(eventObject);

    // 使用适当的属性初始化一个对象变量
    var buttons = {
        'left':false,
        'middle':false,
        'right':false
        };

    // 检查eventObject对象的toString()方法的值
    // W3C DOM对象有toString方法并且此时该
    // 方法的返回值应该是MouseEvent
    if(eventObject.toString && eventObject.toString().➥
indexOf('MouseEvent') != -1) {
```

www.TopSage.com

```
        // W3C方法
        switch(eventObject.button) {
            case 0: buttons.left = true; break;
            case 1: buttons.middle = true; break;
            case 2: buttons.right = true; break;
            default: break;
        }
    } else if(eventObject.button) {
        // MSIE方法
        switch(eventObject.button) {
            case 1: buttons.left = true; break;
            case 2: buttons.right = true; break;
            case 3:
                buttons.left = true;
                buttons.right = true;
            break;
            case 4: buttons.middle = true; break;
            case 5:
                buttons.left = true;
                buttons.middle = true;
            break;
            case 6:
                buttons.middle = true;
                buttons.right = true;
            break;
            case 7:
                buttons.left = true;
                buttons.middle = true;
                buttons.right = true;
            break;
            default: break;
        }
    } else {
        return false;
    }

    return buttons;

}
window['ADS']['getMouseButton'] = getMouseButton;

……以下是库中已有的内容……

})();
```

这个ADS.getMouseButton方法将会返回一个对象，如果相应的键被按下，那么这个对象的left、middle或right属性的值就是true；否则，就是false。

> 作为替代方案，也可以检查某些其他与事件相关的方法，例如addEventListener，不过也不能排除别的脚本会在Object对象的原型中添加自定义的addEventListener()方法的可能性。无论如何，只要避免浏览器检测，你的方案都将使用正确的兼容功能（如果可能），不用考虑浏览器。

www.TopSage.com

**处理鼠标的位置**。你的大部分时间都应该花在寻找光标相对于文档原点的位置，而不是相对于屏幕或者浏览器窗口的位置上。

为此，仍然需要进行一些能力检测并根据访问页面的浏览器来调整你要使用的属性。虽然W3C和IE都定义了`clientX`和`clientY`属性，但它们在确定针对浏览器滚动后的位移属性时却各有不同。W3C使用`document.documentElement.scrollTop`，而IE则使用`document.body.scrollTop`。同样是这种情况，Safari也有自己的一套，它把位置信息放在了事件的`pageX`和`pageY`属性中。下面的`ADS.getPointerPositionInDocument()`方法一并处理了这些情况：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……
function getPointerPositionInDocument(eventObject) {
    eventObject = eventObject || getEventObject(eventObject);

    var x = eventObject.pageX || (eventObject.clientX +
        (document.documentElement.scrollLeft
            || document.body.scrollLeft));

    var y= eventObject.pageY || (eventObject.clientY +
        (document.documentElement.scrollTop
            || document.body.scrollTop));

    // 现在x和y中包含着鼠标
    // 相对于文档原点的坐标
    return {'x':x,'y':y};
}
window['ADS']['getPointerPositionInDocument'] = ➥
getPointerPositionInDocument;

……以下是库中已有的内容……

})();
```

有了`ADS.getPointerPositionInDocument()`方法，就可以通过`mousemove`事件（如chapter4/follow/follow.js中的源代码）让一个元素跟随鼠标指针移动：

```
ADS.addEvent(window, 'load', function() {

    // 定义要移动的对象
    var object = document.getElementById('follow');

    // 为其进行绝对定位
    object.style.position = 'absolute';

    // 为文档的mousemove事件创建事件侦听器
    function eventListener(W3CEvent){
        var pointer = ADS.getPointerPositionInDocument(W3CEvent);

        // 相对于鼠标指针定位对象
        object.style.left = pointer.x + 'px';
        object.style.top = pointer.y + 'px';
    }
    // 将事件侦听器指定给文档
```

www.TopSage.com

```
    // 对象的mousemove事件
    ADS.addEvent(document,'mousemove',eventListener);

});
```

在浏览器中打开chpater4/follow/follow.html看一下，其中已经注册了鼠标移动事件。你会发现网页中的Follow Pointer元素会随着鼠标在文档中移动，如图4-21所示。

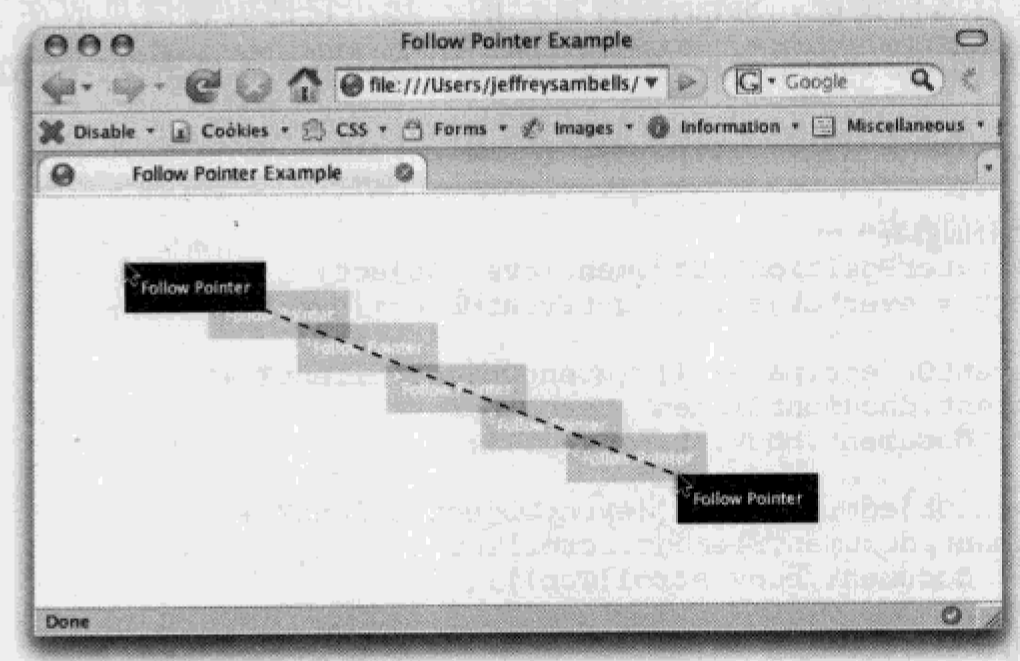



图4-21　一个跟随鼠标指针移动的元素

这个例子示范了我们将在第6章中深入探索的拖放界面中"拖"的部分。

**4. 访问键盘命令**

除了在MouseEvent对象的属性中包含了几个键的组合之外，DOM2事件规范没有对访问键盘作出任何规定。虽然DOM3事件规范中已经引入了键盘命令，但在本书编写时，该规范仍然处于草拟状态，尚未完成。因此，目前你只能依赖于浏览器专有的方法（其中有些方法实际上早已经被W3C标准所采用）。

可喜的是，现代的浏览器都已经决定使用相同的属性了（万岁！），因此你可以通过事件对象的keyCode属性取得按键的Unicode值。但像从前一样编写一个辅助函数仍然还是个好主意，这样一旦W3C方法实现以后，你就可以轻松地切换过去了。下面是需要添加到ADS库中的一个简单的ADS.getKeyPressed()方法，可以通过它取得按键代码及相关的ASCII值：

```
(function(){
if(!window.ADS) { window['ADS'] = {} }

……以上是库中已有的内容……

function getKeyPressed(eventObject) {
    eventObject = eventObject || getEventObject(eventObject);

    var code = eventObject.keyCode;
    var value = String.fromCharCode(code);
    return {'code':code,'value':value};
```

```
}
window['ADS']['getKeyPressed'] = getKeyPressed;
```

……以下是库中已有的内容……

```
})();
```

通过这个简单的小函数，就可以确定在键盘事件中按下的键了。例如，如果你在文档的keydown事件中使用它，则每个键都会如表4-2所示的显示出键值和键代码：

```
ADS.addEvent(document, 'keydown', function(W3CEvent){
    var key = ADS.getKeyPressed(W3CEvent);
    ADS.log.write(key.code + ':' + key.value);
});
```

表4-2 **ADS.getKeyPressed()键/码表**

| 键 | 代　码 | 键 | 代　码 |
|---|---|---|---|
| A | 65 | 1 | 49 |
| B | 66 | 2 | 50 |
| C | 67 | 3 | 51 |
| D | 68 | 4 | 52 |
| E | 69 | 5 | 53 |
| F | 70 | 6 | 54 |
| G | 71 | 7 | 55 |
| H | 72 | 8 | 56 |
| I | 73 | 9 | 57 |
| J | 74 | Backspace（回格） | 8 |
| K | 75 | Tab（制表） | 9 |
| L | 76 | Enter（回车） | 13 |
| M | 77 | Shift（上档） | 16 |
| N | 78 | Ctrl（控制） | 17 |
| O | 79 | Alt/Command（换档/命令）（Apple键盘中的Option） | 18 |
| P | 80 | Pause/Break（暂停/中断） | 19 |
| Q | 81 | Esc（退出） | 27 |
| R | 82 | Page Up（上页） | 33 |
| S | 83 | Page Down（下页） | 34 |
| T | 84 | End（结束） | 35 |
| U | 85 | Home（起始） | 36 |
| V | 86 | Left arrow（向左键） | 37 |
| W | 87 | Up arrow（向上键） | 38 |
| X | 88 | Right arrow（向右键） | 39 |
| Y | 89 | Down arrow（向下键） | 40 |
| Z | 90 | Insert（插入） | 45 |
| 0 | 48 | Delete（删除） | 46 |

www.TopSage.com

（续）

| 键 | 代码 | 键 | 代码 |
|---|---|---|---|
| Left window key（左Window键） | 91 | Scroll Lock（滚动锁定） | 145 |
| Right window key（右Window 键） | 92 | Semicolon (;)（分号;） | 59 |
| Select key（选择） | 93 | Equal sign (=)（等号=） | 61 |
| Multiplication symbol (*)（乘号*） | 106 | Comma (,)（逗号,） | 188 |
| Addition symbol (+)（加号+） | 107 | Period (.)（句号.） | 190 |
| Subtraction symbol (–)（减号–） | 109 | Forward slash (/)（正斜杠/） | 191 |
| Decimal point (.)（小数点.） | 110 | Opening bracket ([)（左方括号[） | 219 |
| Division symbol (\)（反斜杠\） | 111 | Closing bracket ([)（右方括号]） | 221 |
| Num Lock（数字锁定） | 144 | Single quotation mark (')（单引号'） | 222 |

## 4.4 小结

本章需要记忆的东西很多，所以希望你还有能力记住更多的内容。处理事件是当前Web开发中最复杂的部分，而且我们还没有接触到自定义事件。之所以如此复杂，是因为各种浏览器之间的差别很大，而且它们有的实现了，而有的并没实现DOM2事件规范。

本章的内容为在不同浏览器中处理事件打下了良好的基础，从中你学到了如何进行事件注册、访问事件对象，以及事件流和事件侦听器的内容。而且，你也向自己的ADS库中添加了许多新的不针对特定浏览器的事件方法，这些方法将会为你节省大量时间和精力。本章还重申了在涉及到使用不同的事件注册和访问事件对象的方法时，运用能力检测的重要意义。

在下一章中，我们将研究操纵文档样式的不同方式，包括解决Microsoft IE中存在的PNG图像透明度的问题，以及在浏览器中动态访问并编辑样式表等。而在第6章，我们会把前几章所学的内容综合到一块，通过渐进的行为增强技术来构建一个不唐突的图像裁剪和缩放工具。

www.TopSage.com

第 5 章

# 动态修改样式和层叠样式表

在第1章，我们强调了标记、行为与表现分离的重要性。而分离之后的问题是，这三层之间的很多方面会相互交迭，甚至在急剧增长的混乱代码中会发生冲突。你在第3和第4章已经学习了如何操纵DOM文档的结构以及行为化的事件侦听器。现在该介绍表现层及如何编写修改表现的脚本了。我们将探索无须在脚本中嵌入样式而修改页面外观的各种方式，但有时在脚本中硬编码样式也是不可避免的。

在本章中，我们将简单地介绍W3C DOM2级样式规范中的重点内容和能够用来保持表现与DOM脚本分离的多种方法。此外，还将讨论有关style属性的一个最大的误解，以及修复某些版本的Microsoft IE浏览器中存在的PNG图像透明度的问题。本章最后一部分还将创建一个渐变效果的例子，而第10章将会对该例子进一步加以扩展。

## 5.1 W3C DOM2 样式规范

虽然W3C DOM2样式规范（http://w3.org/TR/DOM-Level-2-Style/）乍一看可能有点令人望而却步，但如果你理解CSS的基本知识，其实你就已经知道它的大部分内容了。在此我们不会详细讨论CSS，而是要聚焦于规范中那些更复杂并且更有用的部分，聚焦于你在日常编写DOM脚本时会遇到的问题上。

### 5.1.1 `CSSStyleSheet`对象

`CSSStyleSheet`对象表示的是所有CSS样式表，包括外部样式表和使用`<style type="text/css"></style>`标签指定的嵌入式样式表。`CSSStyleSheet`同样构建于其他的DOM2 CSS对象基础之上，而`CSSStyleRule`对象表示的则是样式表中的每条规则。

通过`document.styleSheets`属性可以取得文档中`CSSStyleSheet`对象的列表。而其中每个`CSSStyleSheet`对象都具有以下属性。

- `type`，这个属性值始终是`text/css`，就和在HTML文档中处理层叠样式表时一样。
- `disabled`，是一个布尔值，表示相应的样式表是应用于当前文档（`false`），还是被禁用（`true`）。
- `href`，是一个表示样式表相对于当前文档所在位置的URL。如果是嵌入式样式，则是当

www.TopSage.com

前文档的URL。
- title，是一个可以用来分组样式表的标签，本章稍后会介绍。
- media，表示样式表应用的目标设备类型，例如screen或print。
- ownerRule，是一个只读的CSSRule对象，如果样式表是使用@import等类似方式导入的，该属性即表示其父规则。
- cssRules，是一个只读的CSSRuleList列表对象，包含样式表中所有的CSSRule对象。

每个CSSStyleSheet对象都包含下列方法。
- insertRule(rule,index)，用于添加新的样式声明。
- deleteRule(index)，用于从样式表中移除规则。

本章我们将在学习如何修改和编辑文档表现的过程中，详细地讨论这些属性和方法。

### 5.1.2 CSSStyleRule 对象

每个CSSStyleSheet对象内部包含着一组CSSStyleRule对象。 这些对象分别对应着类似下面这样一条规则：

```
body {
    font: 62.5%/1.2em "Lucida Grande", Lucida, Verdana, sans-serif;
    background: #c7f28e;
    color: #1a3800;
}
```

CSSStyleRule对象具有下列属性。
- type，是继承自CSSRule对象的一个属性，以0～6中的一个数字表示规则类型。对于CSSStyleRule类型的规则而言，该属性值始终为1。但对于其他规则对象，例如CSSImportRule(表示@import规则)，则是其他值。所有规则对象类型及相应的值如下。
  - 0：CSSRule.UNKNOWN_RULE
  - 1：CSSRule.STYLE_RULE（表示一个CSSStyleRule对象）
  - 2：CSSRule.CHARSET_RULE（表示一个CSSCharsetRule对象）
  - 3：CSSRule.IMPORT_RULE（表示一个CSSImportRule对象）
  - 4：CSSRule.MEDIA_RULE（表示一个CSSMediaRule对象）
  - 5：CSSRule.FONT_FACE_RULE（表示一个CSSFontFaceRule对象）
  - 6：CSSRule.PAGE_RULE（表示一个CSSPageRule对象）
- cssText，包含以字符串形式表示的当前状态下的全部规则。如果这些规则被其他DOM方法修改了，那么这个字符串也会相应改变。
- parentStyleSheet，引用父CSSStyleSheet对象。
- parentRule，如果规则位于另一个规则中，该属性则引用另一个CSSRule对象。例如，位于特定的@media规则中的规则，将把这个@media规则作为其父规则。

而且，CSSStyleRule对象还具有以下针对CSS的属性。
- selectorText，包含规则的选择符。

www.TopSage.com

- style，与HTMLElement.style类似，是CSSStyleDeclaration对象的一个实例。

同样，本章也将详细讨论这些属性和方法。

### 5.1.3 CSSStyleDeclaration 对象

我们要介绍的最后一个，而且也可能是你用得最多的对象，是CSSStyleDeclaration对象，这个对象用于表示一个元素的style属性。与CSSStyleRule对象类似，CSSStyleDeclaration对象具有下列属性。

- cssText，包含以字符串形式表示的全部规则。
- parentRule，将引用CSSStyleRule对象。

此外，CSSStyleDeclaration对象还具有下列方法。

- getPropertyValue(propertyName)，返回一个字符串形式的CSS样式属性值。
- removeProperty(propertyName)，从声明中移除特定的属性。
- setProperty(propertyName,value,priority)，用于设置特定CSS属性的值。

还有一个通过CSS2Properties访问CSSStyleDeclaration对象实例（例如HTMLElement.style属性）的快捷方法，我们将在本章后面介绍style属性时再深入讨论该方法。

> 通过本章的学习你会发现，style属性引用的并不是经过层叠机制计算的样式。

### 5.1.4 支持的匮乏

遗憾的是，有一些浏览器不支持DOM2样式规范的全部特性。虽然这些浏览器中有表现类似特性的类似对象，但访问这些对象的方法与DOM2样式规范中规定的方法是不同的。在本章其余部分中，我们主要讨论W3C方法，但你将向自己的ADS库中添加的所有与样式访问和操作相关的方法，都是不针对特定浏览器的，而且还会包含必要的针对浏览器的访问方法。虽然我会指出不同于W3C访问方法的情况，但不会有太多内容涉及那些专有的方法。唯一的例外是用于解决PNG图像透明度问题的Microsoft IE CSS的filter属性，该属性会在本章快结束时谈到。

## 5.2 当 DOM 脚本遇到样式

Web应用程序中的表现层主要是由CSS样式组成的。我会假设你熟悉CSS并掌握了相应的基础知识，但若事实并非如此，甚至你还没有接触过CSS，我向你高度推荐下面几本书。

- Dan Cederholm编著的*Web Standards Solutions: The Markup and Style Handbook*（friends of ED，ISBN-13: 978-1-59059-381-3）。
- Andy Budd、Simon Collison和Cameron Moll编著的*CSS Mastery: Advanced Web Standards Solutions*[①]（friends of ED，ISBN-13: 978-1-59059-614-2）。
- Andy Clark和Molly Holzschlag编著的*Transcending CSS: The Fine Art of Web Design*（New Riders Press，ISBN-13: 978-0-32141-097-9）。

① 此书中文版《精通CSS：高级Web标准解决方案》已由人民邮电出版社出版。——译者注

www.TopSage.com

同DOM脚本一样，CSS对网站而言也不是绝对可靠的。为了利用CSS的威力和灵活性，需要在一个外部样式表中适当地实现所有CSS定义。这样虽然能够将样式与标记分离开来，但也会导致如下两个问题。

- 正如我所说的，CSS并不是绝对可靠的。它通常要求在语义化的标记中添加更多标签，以便让CSS规则操纵这些额外的元素以达到应用样式的目的。
- CSS还会对行为层造成影响。在很多（即使不是最多）情况下，行为增强的脚本都会涉及操纵元素的表现，但任何情况下表现都不应该混合到DOM脚本中。

但只要经过一番预先设计和规划，文档的表现就可以从行为元素中分离出来，并且还能够同时保持CSS和DOM脚本的灵活性。

## 为应用样式而改动标记

虽然语义纯粹论者可能会拒绝使用任何与文档语义没有直接关系的标记，但有时候这也是无法避免的。当使用CSS时，你很快会发现由于它的局限性，有时候为了使页面元素的外观看起来和你想像的一样，必须使用一点额外的标记——只是为了表现的需要——但那也不能算是坏事。从个人角度来说，我相信只要是提前规划的结果，一点额外的标记并不会伤及任何人，而且你也不会总依靠堆砌新标记而不考虑替代方案。

CSS图像替换就是需要使用额外标记的一种常见情况。不仅可以使用图像来替换标题中简单的文本，而且还可以替换更复杂的元素，例如符号列表或者表格。不管是如何使用图像替换，目标都将文本替换成更加赏心悦目的页面元素，但同时也应该维护对信息的可访问性。

而最终的标记仍然应该：

- 对屏幕阅读器而言是可访问的。
- 当禁用图像而CSS有效时是可以理解的（很多解决方案中都存在这个问题）。
- 维护与可访问性相关的特性，如`alt`和`title`属性。
- 避免使用不必要的标记——如果可能的话。

CSS图像替换的经典技术被称为FIR（Fahrner Image Replacement，Fahrner图像替换技术）。这种技术涉及到使用无关的`<span>`标签在某个元素（如标题）中包含文本内容：

```
<h1 id="advancedHeader"><span>Advanced DOM Scripting</span></h1>
```

然后再通过CSS在标题中插入背景同时隐藏`<span>`元素：

```
/* 为父元素添加背景图像 */
#advancedHeader {
    height:60px;
    background: transparent url(http://advanceddomscripting.com/ ➥
images/advancED-replace.png) no-repeat;
}
/* 隐藏文本 */
#advancedHeader span {
    display:none;
}
```

随着CSS的应用，结果是用一幅自定义的图像代替了浏览器中纯粹的文本，如显示了在浏览

www.TopSage.com

器中打开chapter5/image-replacement/fir.html文件的图5-1所示——纯文本标题AdvancED DOM Scripting已经被一幅图像所替换。

如果没有CSS，标题将平稳地退化为图5-2所示的纯文本效果。

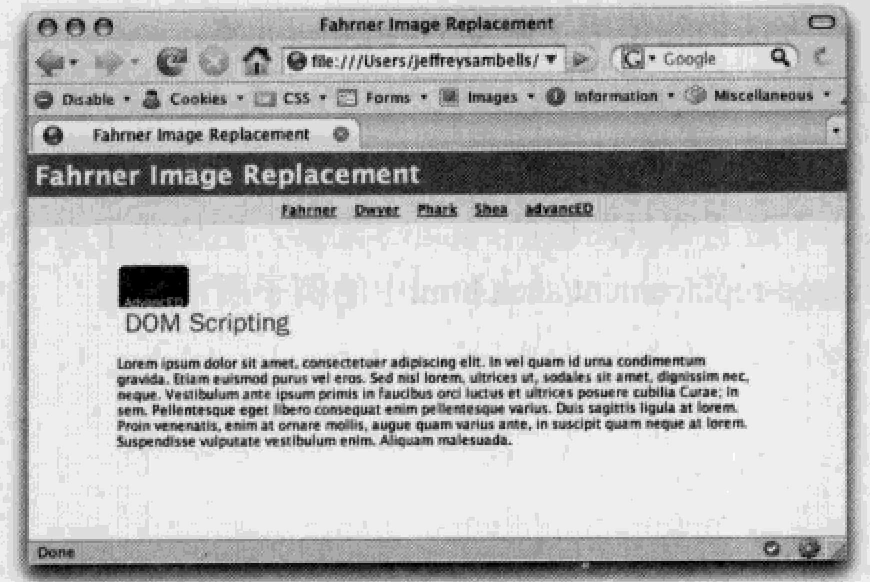



图5-1 标题被一幅图像所替换

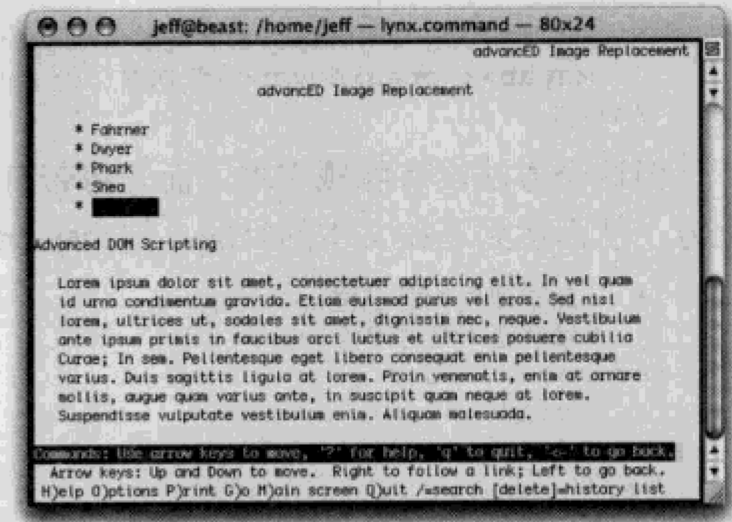



图5-2 在基于文本的lynx浏览器中看到的标题

这种经典的FIR方法存在两个问题。首先，如果禁用了图像将什么也不会显示，因为span仍旧被隐藏着。第二，display:none属性会对支持CSS的屏幕阅读器隐藏内容，从而完全破坏了图像替换中的可访问性原则。

在其他已经发现的方法中，有代表性是以下3种。

Leon Dwyer发明的Dwyer方法使用0尺寸的附加<span>标签，不过与FIR相似，它在CSS有效而图像禁用的情况下仍然不具有可访问性，如chapter5/image-replacement/dwyer.html中的例子所示：

```
/* 为父元素添加背景图像 */
#advancedHeader {
    height:60px;
    background: transparent url(http://advanceddomscripting.com/ ➥
images/advancED-replace.png) no-repeat;
}
/* 使用0尺寸的盒子隐藏文本*/
#advancedHeader span {
    display:block;
    width:0;
    height:0;
    overflow:hidden;
}
```

由Mike Rundle发明的Phark方法不需要任何额外的标签，而是使用了负文本缩进来隐藏内容。但是，这种方法在CSS有效而图像被禁用的情况还是会损害可访问性，如chapter5/image-replacement/phark.html中的例子所示：

```
/* 使用背景图像和负
文本缩进来隐藏文本*/
#advancedHeader {
    text-indent: -100em;
```

www.TopSage.com

```
    overflow:hidden;
    height:60px;
    background: transparent url(http://advanceddomscripting.com/ ➥
images/advancED-replace.png) no-repeat;
}
```

另一种方法是Dave Shea创造的Shea方法，该方法仍然使用了附加的`<span>`元素：

```
<h1 id="advancedHeader" title="AdvancED DOM Scripting">
    <span></span>Advanced DOM Scripting
</h1>
```

但这个方法没有隐藏文本，而是将带有实心不透明背景图像的`<span>`元素，定位在了文本上方，从而达到遮盖文本的目的，如chapter5/image-replacement/shea.html中的例子所示：

```
/* 父元素使用相对定位
   子元素使用绝对定位 */
#advancedHeader {
    height:60px;
    position:relative;
}
/* 通过在绝对定位的span元素上使用
   不透明背景来隐藏文本 */
#advancedHeader span {
    background: transparent url(http://advanceddomscripting.com/ ➥
images/advancED-replace.png) no-repeat;
    position:absolute;
    display:block;
    width:100%;
    height:100%;
}
```

当图像被禁用时，Shea的方法会使文本保持可见。而且，当鼠标悬停在标题元素上时，标题元素的`title`属性也能够充当图像元素中等效的`alt`属性。

**去掉额外的标记**

在前面图像替换的方案中，额外的`<span>`标签没有提供更多的语义，而仅仅是一个额外的标签。但事情并不总是那么糟，还可以使用DOM脚本来完成同样的任务。如果你愿意让禁用JavaScript的浏览器取得退化的只包含文本的版本，那么通过`load`事件侦听器可以轻易地添加额外的标记。为此，标题中可以只包含必要的识别标记，如源文件chapter5/image-replacement/advance-ED.html所示：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>advancED Image Replacement</title>
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="../chapter.css" />

    <!--ADS Library (full version from source linked here) -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>

    <!--Image replacement -->
    <link rel="stylesheet" type="text/css"
```

www.TopSage.com

```
        href="advancED.css" media="screen">

    <!--The load script -->
    <script type="text/javascript" src="advancED.js"></script>

</head>
<body>
    <h1>advancED Image Replacement</h1>
    <h2 id="advancedHeader">Advanced DOM Scripting</h2>
</body>
</html>
```

然后，在链接的advancED.css文件中加入适当的样式规则。这里，我选择将Shea方法作为基本的CSS方法，同时在选择符中添加了额外的.advanceED类：

```
/* 在图像被禁用的情况下为文本添加的样式 */
#advancedHeader {
    color: #1A5B9D;
}
/* 根据图像设置标题的大小 */
#advancedHeader.advancED {
    height:60px;
    position:relative;
    overflow:hidden;
}
/* 使用不透明图像隐藏文本*/
#advancedHeader.advancED span {
    background: white url(http://advanceddomscripting.com/ ➡
images/advancED-replace.png) no-repeat;
    display:block;
    width:100%;
    height:100%;
    position:absolute;
}
```

其中.advanceED类是关键所在，因为它没有出现在chapter5/image-replacement/advancED.html的源标记中。

有了这些CSS规则之后，即使是无法（或没有）运行JavaScript的设备都会看到（如果CSS无效则没有）应用CSS样式的标题，如图5-3所示。

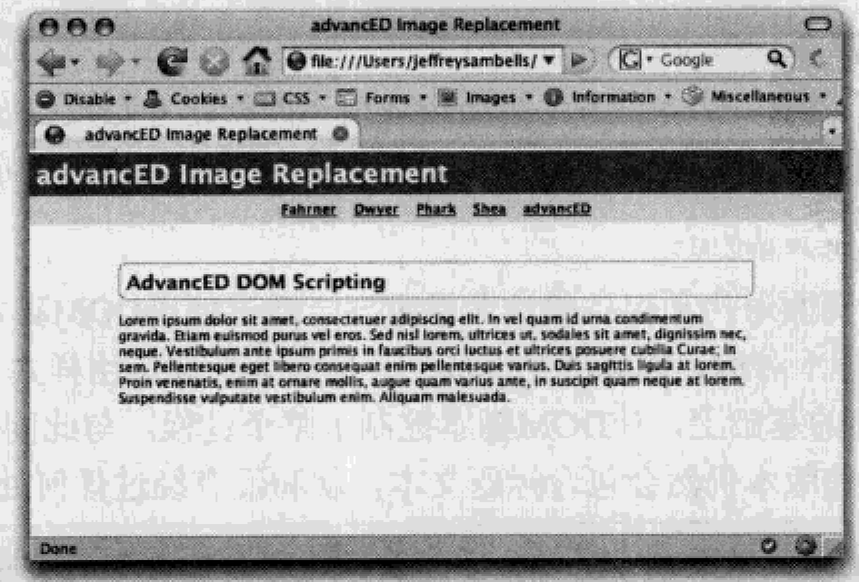



图5-3 脚本未运行时的高级图像替换方案

www.TopSage.com

为了让页面真正有吸引力，chapter5/image-replacement/advancED.js文件中的load事件会向标题元素中加入额外的<span>标签以及额外的.advancED类名，这个事件的代码如下：

```
ADS.addEvent(window, 'load', function() {
    // 取得标题
    var header = ADS.$('advancedHeader');
    // 创建图像元素
    var image = document.createElement('IMG');

    // 当图像载入成功后再添加span和类名
    ADS.addEvent(image, 'load', function() {

        var s = document.createElement('SPAN');
        // 将span添加为标题的子元素
        ADS.prependChild(header,s);

        // 创建必要的title属性
        if(!header.getAttribute('title')) {
            var i, child;
            var title = '';
            // 循环遍历子元素以组合title属性
            for(i=0 ; child = header.childNodes[i] ; i++ ) {
                if(child.nodeValue) title += child.nodeValue;
            }
            header.setAttribute('title',title);
        }
        // 修改类名称以标明
        // 变更并应用CSS
        header.className = 'advancED';
    });

    // 载入图像
    // 这种硬编码的方式并不理想
    // 本意后面还将讨论这一点
    image.src = 'http://advanceddomscripting.com/ ➥
images/advancED-replace.png';

});
```

> 这个例子不是非常具有可移植性，因为如果想修改图像必须修改脚本代码。我们将在本章后面解决这个问题。

如果图像载入成功，图像的load事件将会生成适当的标记并触发CSS应用必要的样式：

```
<h1 id="advancedHeader" title="Advanced DOM Scripting" class="advancED">
    <span></span>
Advanced DOM Scripting</h1>
```

添加额外的类对于页面退化和保持整洁非常重要。通过使用DOM脚本添加一个新类，事实上是在通知CSS表现层脚本对标记的修改已经成功，应该为标题应用样式了。这样就在保持实际的表现与DOM脚本分离的基础上，让DOM脚本控制了整个过程。与此同时，无论是CSS还是图像不可用，平稳退化后的版本仍然会包含标题文本，就和没有经过任何处理一样。

> 如果你原封不动地测试这个例子，那么当覆盖标题元素中原有的类时将会遇到问题。在本章后面，你将为自己的ADS库创建一个能够解决这个问题的类名操纵方法。

www.TopSage.com

使用这个方法时，唯一需要注意的就是其中的替换操作依赖于DOM脚本。根据计算机及网络连接的速度，在浏览器呈现文本和载入脚本（图像载入）期间可能会出现短暂的闪烁。所以，必须权衡这个方案同时具有的清洁、可访问性以及可退化的优点和依赖JavaScript的缺点，并作出自己正确的选择。

## 5.3 把样式置于 DOM 脚本之外

从标记中分离样式，可以通过使用适当的类和ID属性加上一个外部样式表轻松地实现。不过，同样的思想在DOM脚本中却经常被忽略。事实上，从行为改变中分离出样式改变的成分同样重要。在适当的情况下，DOM脚本会影响文档的外观——但它们不应该在调整设计时以强迫你修改DOM脚本的方式来影响外观[①]。

在上一节中，我们看到了使用CSS和DOM脚本的组合操纵标记来完成图像替换的过程。在本节中，你将看到很多能够影响外观但不用直接改变样式属性的方式。我们还是从DOM脚本编程中一个最被误解的属性——DOM `style`属性说起吧。

### 5.3.1 `style` 属性

当在小范围内修改表现时（例如每次只涉及一或两个元素），你可能会在网上（甚至在本书中）碰到许多使用`HTMLElement`元素的`style`属性来修改其表现的例子。但修改`style`属性存在如下两个主要问题。

- 使用`style`属性将设计样式嵌入到行为层的DOM脚本中，并不比在语义标记中使用`style`属性更好。但有时候，对于与行为相关的属性使用`style`也确实是有意义的，例如定位或者与网站整体设计没有关系的样式变化[②]等。
- 而且`style`属性并不像你想象的那样。我的意思是说，`style`属性本身是一个表示特定元素的所有不同CSS样式的`CSSStyleDeclaration`对象。但你可能不知道的是，通过`style`属性只能访问到在元素的`style`属性中以嵌入方式声明的CSS属性。仅此而已，没有别的。换句话说，通过`style`属性无法访问由多重样式表层叠而来或者从父元素继承的任何CSS属性。这意味着你无法通过它来取得元素的完整的计算样式。对此，必须使用本章后面将要讨论到的几个不同对象。

> 如果你认为`style`属性会包含元素的计算样式，其实你并不孤单。这是一个大多数人都容易弄错的常见假设。有些人会假设样式表还没有载入完成，或者某些浏览器载入的怪异模式导致`style`属性受到了干扰，但这些都不对。因为在设计上，`style`属性根本就不会为你提供那些信息。这也是一个很好的例子，说明只要花点时间从头到尾看一遍规范，就能避免把本来的特性当成几小时都调试不好的“bug”来看待的挫折。

① 即DOM脚本中不能直接嵌入样式规则，而应该采取向标记中添加/删除类或其他更灵活的方法。——译者注

② 比如第2章最后的实例研究中创建的`ADS.log`对象，就在脚本中使用了嵌入的样式。因为日志窗口属于开发工具，与调试的网站设计风格没有关系。——译者注

www.TopSage.com

记住了这两个问题以后，我们就来看看style属性和它的其他一些怪异表现。

你在前面已经看到了，可以使用CSS的color属性设置像元素的前景色之类的简单属性：

```
element.style.color = 'red';
```

也可以使用下面的代码设置元素的背景颜色（background-color）：

```
element.style.backgroundColor = 'red';
```

所有CSS属性都遵循相同的模式，不过在设置background-color的例子中使用的是驼峰形大小写形式，而不是CSS中带连字符的形式。将background-color转换为驼峰形大小写形式的backgroundColor（删除连字符并将后续单词的首字母变成大写）是必需的，因为你实际上是在使用一种快捷方式。为了适当地访问style属性，DOM2样式规范为CSSStyleDeclaration对象定义了相应的方法，比如setProperty()就使用固有的带连字符的CSS属性名称和值：

```
element.style.setProperty('background-color','red');
```

同样，问题的根源在于非标准的浏览器（比如Microsoft IE）不支持setProperty()方法，因此为了保证跨浏览器的兼容性，就必须依靠以快捷方式表示的属性名称。

为了保证不针对特定的环境，将下面的setStyleById()、setStylesByClassName()和setStylesByTagName()方法添加到你的ADS库中。这些方法都以一个JavaScript对象作为第2个参数，这样就可以使用JavaScript的对象表示法来通过适当的CSS属性名称定义多个样式，不过方法内部仍然利用了驼峰形大小写形式的属性名称：

```
ADS.setStyleById('example',{
    'background-color': 'red',
    'border': '1px solid black',
    'padding': '1em',
    'margin':'1em'
});
```

在setStyle()方法内部，会按照需要将带连字符版的CSS样式转换成驼峰形大小写形式，转换工作由在第3章编写HTML到DOM转换工具时添加的ADS.camelized()方法负责：

```
(function(){
window['ADS'] = {};
```

……以上是库中已有的内容……

```
/* 通过ID修改单个元素的样式 */
function setStyleById(element, styles) {
    // 取得对象的引用
    if(!(element = $(element))) return false;
    // 循环遍历styles对象并应用每个属性
    for (property in styles) {
        if(!styles.hasOwnProperty(property)) continue;

        if(element.style.setProperty) {
            // DOM2样式规范方法
            element.style.setProperty(
            uncamelize(property,'-'),styles[property],null);
        } else {
```

www.TopSage.com

```
                // 备用方法
                element.style[camelize(property)] = styles[property];
            }
        }
        return true;
    }
    window['ADS']['setStyle'] = setStyleById;
    window['ADS']['setStyleById'] = setStyleById;

    /* 通过类名修改多个元素的样式 */
    function setStylesByClassName(parent, tag, className, styles) {
        if(!(parent = $(parent))) return false;
        var elements = getElementsByClassName(className, tag, parent);
        for (var e = 0 ; e < elements.length ; e++) {
            setStyleById(elements[e], styles);
        }
        return true;
    }
    window['ADS']['setStylesByClassName'] = setStylesByClassName;

    /* 通过标签名修改多个元素的样式 */
    function setStylesByTagName(tagname, styles, parent) {
        parent = $(parent) || document;
        var elements = parent.getElementsByTagName(tagname);
        for (var e = 0 ; e < elements.length ; e++) {
            setStyleById(elements[e], styles);
        }
    }
    window['ADS']['setStylesByTagName'] = setStylesByTagName;

    ……以下是库中已有的内容……

    })();
```

这些方法也提供了通过类名来修改相关元素样式属性的、不针对浏览器的友好方式：

```
ADS.setStylesByClassName(
    'findClass',
    '*',
    document,
    {'background-color':'red'}
);
```

或通过标签名

```
ADS.setStylesByTagName('a',{'text-decoration':'underline'});
```

不过同样，在脚本中修改background-color之类的属性仍然意味着在行为代码中嵌入样式。即使你没有共享的意图并且你的代码仅供你自己使用，坚持不在DOM脚本中混合CSS样式仍然是一个好习惯。当你准备更新当前设计并编辑CSS样式时，只摆弄几个CSS文件一定会比逐行扫描脚本代码，以防遗漏每个backgroundColor='red'[①]的实例令人更愉快。

唯一可以接受样式与脚本混合的时候，就是在定位的情况下。其中，像拖放式交互功能的关

① 原文background-color=red有误。——译者注

www.TopSage.com

键就是要在页面中移动元素。像下面这样通过position属性将元素设置为绝对定位，在脚本中是有意义的。因为此时的定位方式和元素的位置并不属于视觉设计范畴，而是直接建立在响应鼠标在页面上移动这种交互性事件基础之上的：

```
ADS.setStyleById('example',{
    'position':'absolute',
    'top':'10px',
    'left':'20px'
});
```

另一方面，操纵字体和改变颜色之类的修改则不应归入脚本的范畴——至少没有直接关系。

## 5.3.2　基于 className 切换样式

对于中小范围的表现变化，例如只影响少数元素的颜色、边框、背景和字体样式，可以通过切换className来避免在代码中苦苦寻觅样式属性。换句话说，简单地使用DOM脚本修改目标元素的className属性，就可以基于在样式表中预定义的规则来修改相应元素的外观。

从实现的角度来看，这样需要与脚本配合着手工或动态地添加样式表。而对于设计者来说，事实上增强了可用性和亲和力。此外，这种模式也支持在多种设计之间重用同样的DOM脚本，因为站点的设计并不是DOM脚本的一部分。例如，不用直接通过style属性修改元素的background-color，而是要在CSS文件中定义两类规则：

```
.normal {
    background-color: black;
}
.normal. modified {
    background-color: red;
}
```

然后通过修改元素的className属性来应用改进后的样式：

```
var element = ADS.$('example');
element.className += ' modified';
```

### 1. 在className切换中使用公共的类

在实现className切换时，有必要提炼出一组只用于表示文档变化的公共类。这些类就像是伪类选择符，而且应该只将它们用于与其他选择符组合起来触发基于DOM脚本操作的变化。例如，你可能会将hover视为一个保留字。如果像下面这样直接声明一个hover类：

```
.hover {
    position:absolute:
    top:100px;
    left:200px;
}
```

那么将会破坏DOM脚本的效果。但如果只将它与其他选择符组合使用，如

```
#cart.hover {
    background-color:yellow;
}
#sidebar.hover a {
```

www.TopSage.com

```
    text-decoration:underline;
}
```

那么这个hover类将如同一个伪类，与CSS中a:hover伪类如出一辙。

使用className切换方法可以维护适当的分离，并且为CSS设计者打开了使用样式表设计网站表现的大门。

为了让这个过程更简便一些，将下面这些操纵className的方法添加到你的ADS库中：

```
(function(){
window['ADS'] = {};

……以上是库中已有的内容……

/* 取得包含元素类名的数组 */
function getClassNames(element) {
    if(!(element = $(element))) return false;
    // 用一个空格替换多个空格
    // 然后基于空格分割类名
    return element.className.replace(/\s+/,' ').split(' ');
};
window['ADS']['getClassNames'] = getClassNames;

/* 检查元素中是否存在某个类 */
function hasClassName(element, className) {
    if(!(element = $(element))) return false;
    var classes = getClassNames(element);
    for (var i = 0; i < classes.length; i++) {
        // 检测className是否匹配，如果是则返回true
        if (classes[i] === className) { return true; }
    }
    return false;
};
window['ADS']['hasClassName'] = hasClassName;

/* 为元素添加类 */
function addClassName(element, className) {
    if(!(element = $(element))) return false;
    // 将类名添加到当前className的末尾
    // 如果没有className，则不包含空格
    element.className += (element.className ? ' ' : '') + className;
    return true;
};
window['ADS']['addClassName'] = addClassName;

/* 从元素中删除类 */
function removeClassName(element, className) {
    if(!(element = $(element))) return false;
    var classes = getClassNames(element);
    var length = classes.length
    // 循环遍历数组删除匹配的项
    // 因为从数组中删除项会使
    // 数组变短，所以要反向循环
    for (var i = length-1; i >= 0; i--) {
        if (classes[i] === className) { delete(classes[i]); }
    }
```

www.TopSage.com

```
        element.className = classes.join(' ');
        return (length == classes.length ? false : true);
    };
    window['ADS']['removeClassName'] = removeClassName;
    ……以下是库中已有的内容……

    })();
```

这些方法简单而适当地操纵了className属性：

- ADS.getClassNames(element)返回包含与element元素关联的类名的数组。
- ADS.hasClassName(element, class)检查element元素的类名中是否包含class类。
- ADS.addClassName(element, class)向element元素的className中添加一个class类。
- ADS.removeClassName(element, class)从element元素的类名中删除特定的class类。

可以使用这些方法方便地取得与一个元素关联的所有类：

```
var element = document.getElementById('example');
var classes = ADS.getClassNames(element);
var class;
for (var i=0; class=classes[i]; i++) {
    // 对每个类进行操作
}
```

或者在向元素添加类名之前检查是否存在该类名，以便实现切换效果：

```
function toggleClassName(element,className) {
    if(!ADS.hasClassName(element,className)) {
        ADS.addClassName(element,className);
    } else {
        ADS.removeClassName(element,className);
    }
}
ADS.addEvent('toggleButton','click',function() {
    toggleClassName(this,'active');
});
```

甚至还能通过使用适当的事件侦听器和hover类生成翻转效果：

```
var element = document.getElementById('example');
ADS.addEvent(element,'mouseover',function() {
    ADS.addClassName(this,'hover');
});
ADS.addEvent(element,'mouseout',function() {
    ADS.removeClassName(this,'hover');
});
```

### 2. 使用className切换的缺点

使用动态类名来表示样式改变是个很好的方法，但它同样也有缺点：

第1个缺点是元素的样式可能会影响到脚本中的交互操作。比如声明不同的内边距（padding）或外边距（margin）样式可能会改变元素的大小和位置，因此会导致迷人的可拖放对象无法适当对齐。而为交互性元素建立一个公共的标记方案可以解决这个问题。例如，可以制定一条方针，

www.TopSage.com

让所有可拖放元素内部都包含一个带有公共类名的二级容器元素，CSS设计者可以随意操纵这个二级容器元素，但外部的容器元素则仍然按照预期进行定位。

第2个缺点是我们在前面例子中所看到的，在没有事先检查className中原有类名的情况下，向其中添加修改后的类名会导致问题。虽然大多数情况下，你都会基于期望的操作使用更有意义的类名，但你也会因为要完成不同的操作而添加或删除多个类名。因此，这就涉及必须通过检查来事先确定className中已有的内容，以防止造成不必要的混乱。同样地，在删除修改后的类名时，仍然需要通过某些字符串操作来保证你删除的是className字符串中正确的部分。而使用刚才添加到ADS库中的类名方法有助于解决这个问题。

**3. 为什么不使用setAttribute方法来设置类名**

此时此刻，你可能会奇怪为什么这些例子都直接修改className属性，而不使用setAttribute()方法。由于className引用的是HTMLElement对象的class属性，所以实际上可以通过3种不同的方式来定义类属性，但在任何一种浏览器中如下这3种方式的工作情况都不相同。

- `element.setAttribute('class','newClassName')`在所有兼容W3C标准的浏览器中有效。
- `element.setAttribute('className','newClassName')`在IE中有效，但不是符合W3C标准的设置该值的方式。
- `element.className = 'newClassName'`则在所有浏览器中都有效。

第3种方法在所有浏览器中都能奏效，而且也是修改（读/写）HTMLElement元素className属性的一种完全有效的方式，所以我们坚持使用这种简单的方式。

### 5.3.3 切换样式表

对于更大范围的改变，例如改变整个页面的布局，就不可能通过迭代整个DOM树并手工修改其中每个元素的样式来实现了。而最容易做到的则是将当前在用的样式表切换为另外一个完全不同的样式表。根据设置CSS样式表和规则的方式，也可以通过下面几种不同的手段实现样式表的切换。

- 可以使用<link>元素的rel属性定义备用的样式表并在它们之间进行切换。
- 可以为body标签应用一个类，并根据这个类修改CSS选择符——实际上是以body标签作为根元素的className切换。
- 可以动态添加或移除样式表。

无论采取哪种方法，所期望的效果都将是彻底地改变布局或者文档中（如果不是全部）大多数元素的表现。

**1. 使用备用的样式表**

在备用的样式表之间进行切换最早是由Paul Sowden在他的文章"Alternative Style: Working with Alternate Style Sheets(http://alistapart.com/stories/alternate/)"中提出来的。该方法利用了DOM2 HTML <link>元素的某些特性。

如果你始终遵守表现与标记分离的原则，那么应该熟悉<link>元素。该元素用于在文档头

www.TopSage.com

部包含样式表：

```
<link type="text/css" href="/path/to/style.css" media="screen" />
```

<link>元素包含以下属性。

- type，用于表示样式文件的MIME类型，即这里的text/css。
- href，用于指定样式表的位置。
- media，用于限制执行样式表的设备类型。但是，对media类型的检查由设备和软件完成。而某些设备可能会忽略media或者使用不正确的类型。

这些属性都是最经常用到的，但是<link>的属性还不止这些。<link>元素还包括以下属性（除此之外还有别的属性）。

- rel，表示样式表与文档之间的关系。
- disabled，表示样式是否起作用。
- title，表示与样式表关联的标题。

> 我们注意到这些属性与本章开始时介绍的CSSStyleShees对象的属性有些类似。

可以使用rel="stylesheet"属性指定将一个样式表立即应用到文档：

```
<link rel="stylesheet" type="text/css" ➥
href="/path/to/style.css" media="screen">
```

或者使用rel="alternate stylesheet"将其作为备用样式表而在默认情况下禁用它：

```
<link rel="alternate stylesheet" type="text/css" ➥
href="/path/to/style.css" media="screen">
```

当关系[①]设置为alternate stylesheet时，浏览器会载入该样式表，但也会将其disabled标记为true，使它不会即时生效。

虽然title属性对样式表本身没有影响，但却可以在脚本中利用它。比如可以使用它作为标识字符串来动态创建可用样式的列表，以便用户任意在这些样式表之间切换。在chapter5/switcher/example.html的示例源代码中，包含了几种不同的CSS布局样式表，从中可以看到备用样式表的应用：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <title>Style Switcher</title>
    <!--
        The common styles that won't change
        Leave the title off so they won't be included in the list
    -->
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css" href="../chapter.css" />
```

---

① <link>元素的rel属性是relationship（关系）的意思。——译者注

www.TopSage.com

```
    <link rel="stylesheet" type="text/css"
        href="common.css" media="screen">

    <!--AdvancED DOM Scripting Simple Style (the default) -->
    <link rel="stylesheet" title="AdvancED DOM Scripting"
        type="text/css" href="ads.css" media="screen">
    <!--[if true]>
    <link rel="alternate stylesheet" title="AdvancED DOM Scripting"
        type="text/css" href="adsIE.css" media="screen">
    <![endif]-->

    <!--A friends of ED style -->
    <link rel="alternate stylesheet" title="friends of ED"
        type="text/css" href="foed.css" media="screen">

    <!--An Apress style -->
    <link rel="alternate stylesheet" title="Apress"
        type="text/css" href="apress.css" media="screen">


    <!--ADS Library (full version from source linked here) -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>
    <!-- The load script -->
    <script type="text/javascript" src="styleSwitcher.js"></script>


</head>
<body>
<h1>Style Switcher</h1>
    <div id="content">
        <h2>It's Easy!</h2>
        <ul id="styleSwitcher"></ul>
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
        Quisque iaculis elit in mauris. Mauris euismod tempor tortor.
        Integer fringilla, orci at venenatis consequat, lorem ipsum.
        Morbi ornare sollicitudin justo. Nulla est. Cras lorem.</p>
    </div>
</body>
</html>
```

在这个例子中，公共的CSS文件没有设置title属性，而friends of ED和Apress样式都有一个相应的样式表文件，并且AdvancED DOM Scripting的样式包含两个样式表（一个针对IE）。标记中的空列表元素：

```
<ul id="styleSwitcher"></ul>
```

包含的内容是由chapter5/switch/styleSwitcher.js文件通过load事件处理程序，在备用样式表文件及其title属性的基础上动态生成的：

```
ADS.addEvent(window,'load',function() {
    // 取得所有link元素
    var list = ADS.$('styleSwitcher');
    var links = document.getElementsByTagName('link');
    var titles = [];

    for (var i=0 ; i<links.length ; i++) {
        // 跳过不带title属性的<link>元素
```

www.TopSage.com

```
            if(links[i].getAttribute("rel").indexOf("style") != -1
                && links[i].getAttribute("title")) {

                // 如果该样式表还未添加则
                // 向列表中添加一个新项
                var title = links[i].getAttribute("title");
                if(!titles[title]) {

                    var a = document.createElement('A');
                    a.appendChild(document.createTextNode(title));
                    a.setAttribute('href','#');
                    a.setAttribute('title','Activate ' + title);
                    a.setAttribute('rel',title);
                    ADS.addEvent(a,'click',function(W3CEvent) {
                        // 当单击链接时激活锚的rel属性
                        // 中的标题所表示的样式表文件
                        setActiveStyleSheet(this.getAttribute('rel'));
                        ADS.preventDefault(W3CEvent);
                    });

                    var li = document.createElement('LI');
                    li.appendChild(a);

                    list.appendChild(li);

                    // 将titles数组中的这个标题项设置为true
                    // 以便在多个样式表使用相同标题时跳过
                    titles[title] = true;
                }
            }
        }
    });
```

以上styleSwitcher.js文件会创建一个常规的列表，也可以按照需要对其应用样式，如图5-4所示。

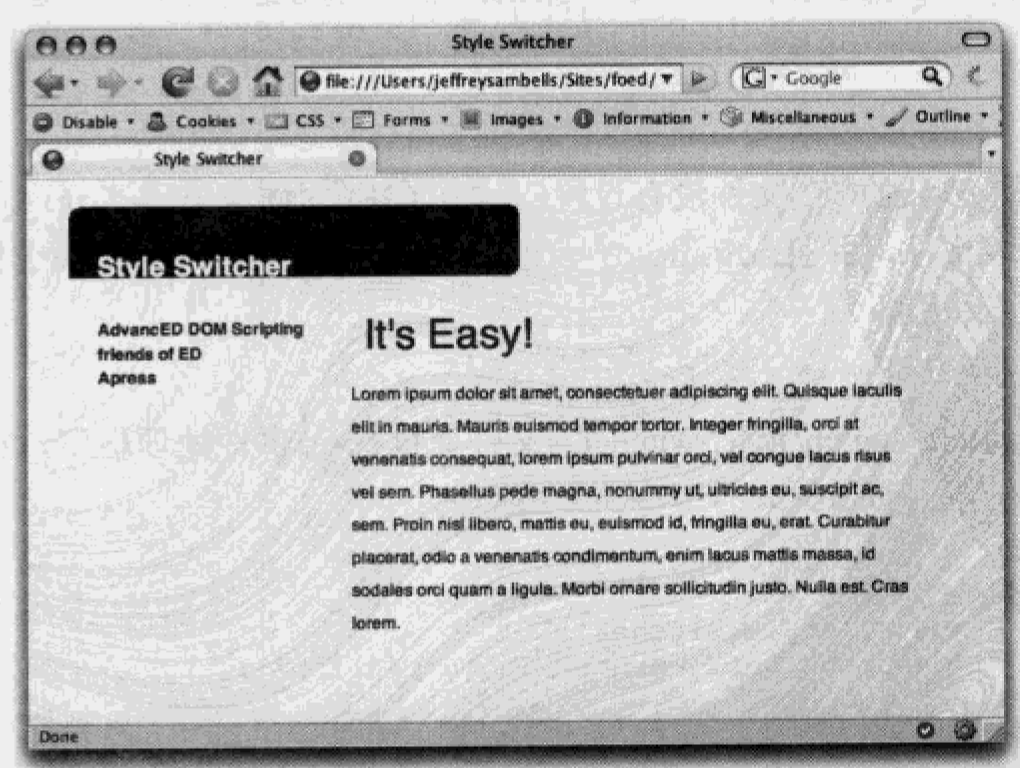


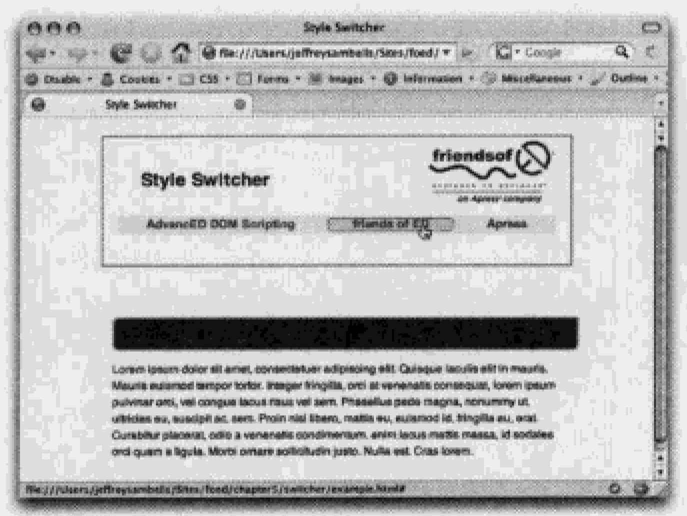

图5-4 基于link元素的title属性动态生成的样式切换列表

www.TopSage.com

生成的样式列表通过使用锚的`click`事件侦听器以及Paul Sowden原文中的`setActiveStyleSheet()`函数，可以让用户在备用的样式表之间进行切换。`setActiveStyleSheet()`函数的代码如下：

```
function setActiveStyleSheet(title) {
    var i, a, main;
    for(i=0; (a = document.getElementsByTagName("link")[i]); i++) {
        if(a.getAttribute("rel").indexOf("style") != -1
           && a.getAttribute("title")) {
          a.disabled = true;
          if(a.getAttribute("title") == title) a.disabled = false;
        }
    }
}
```

切换的结果如图5-5所示。



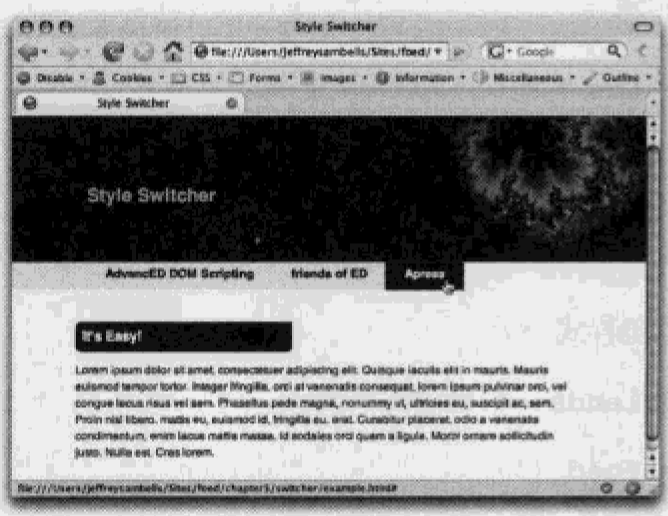



图5-5 当单击每个链接时页面会应用不同的样式表

这个例子并不完整，因为它不支持在网站的多个页面间维护样式，不过它却示范了启用和禁用多个样式表文件的能力。

**2. 切换body元素的className**

这种方法的指导思想遵循了与前面讨论的切换`className`方法的相同原理，只不过这里要切换的是body标签的`className`。Andy Clarke和James Edwards在他们的文章"Invasion of the Body Switchers（http://alistapart.com/articles/bodyswitchers）"中详细地描述了这种基于body元素的类名切换样式的技术。这种技术与前面应用到元素的`className`切换技术唯一真正的区别在于CSS规则的定义。为了基于body标签应用样式，必须在包含的样式表的所有相关声明中使用该body标签的类作为选择符：

```
/* 公共样式 */
body {
    font: 62.5%/1.2em sans-serif;
    color: #1a3800;
    text-align:center;
}

#container {
```

```
    text-align:left;
}

/* AdvancED DOM Scripting 样式 */

body.ads {
    background-image: url(images/ads-bg.jpg);
}

body.ads h1 {
    height: 12px;
    margin: 1em 3em 0 30px;
    padding: 2em 0 0 20px;
    padding-bottom: 0em ;
    background: #f06;
    width: 300px;
    overflow: hidden;
    white-space: nowrap;
    -moz-border-radius: 0.5em;
    -moz-border-radius-bottomleft: 0em
}
body.ads h2 {
    border: 0;
    font-size: 3em;
    color: #1A5B9D;
    font-weight: normal;
}

/* 其他样式 */

/* friends of ED 样式 */

body.foed {
    font-family: sans-serif;
    background-color: #fffbf2;
    line-height: 1.8em;
}

body.foed h1 {
    height: 100px;
    color: #f06;
    padding: 40px 0 0 40px;
    background: transparent url(images/foed.png) no-repeat top right;
    border: 2px solid #999;
    width: 460px;
    margin: 1em auto 1em auto;
}

body.foed h2 {
    border-width: 2px;
    background: #f06;
    padding: 0.5em;
    width: 100%;
}

/* 其他样式 */

/* Apress样式 */
body.apress {
    font-family: sans-serif;
```

www.TopSage.com

```
    background-color: white;
    line-height: 1.8em;
}

body.apress h1 {
    height: 80px;
    background: black url(images/fractal.jpg) no-repeat top right;
    color: #ffCC00;
    padding: 80px 0 0 80px;
}

body.apress h2 {
    border: 0;
    background: #900;
    padding: 0.5em;
    color: white;
    width: 40%;
}
/* 其他样式 */
```

然后，使用在本章前面添加到ADS库中的类名方法或者你自己创建的其他函数，可以动态地切换body标签的类名，而CSS规则会相应地修改页面表现：

```
ADS.addEvent('ads-anchor','click',function() {
    ADS.addClassName(document.body,'asd');
});
ADS.addEvent('foed-anchor','click',function() {
    ADS.addClassName(document.body,'foed');
});
ADS.addEvent('apress-anchor','click',function() {
    ADS.addClassName(document.body,'apress');
});
```

在使用这种方法时，如果要自动生成可选择的样式列表需要一些技巧性。因为没有简单的办法可以将某个标题与不同的样式集合建立关联。

其中的类名和CSS层叠技术适用于文档的任何层次，而不仅仅是个别的body元素。

**3. 动态载入和移除样式表**

涉及在文档中动态载入和卸载样式表的第3种技术相当直观。使用这种技术所要做的就是通过document.createElement()及适当的属性创建新的<link>元素。把下面这两个方法添加到ADS库中，可以帮你实现这一技术：

```
(function(){

window['ADS'] = {};

……以上是库中已有的内容……

/* 添加新样式表 */
function addStyleSheet(url,media) {
    media = media || 'screen';
    var link = document.createElement('LINK');
    link.setAttribute('rel','stylesheet');
    link.setAttribute('type','text/css');
    link.setAttribute('href',url);
    link.setAttribute('media',media);
```

www.TopSage.com

```
    document.getElementsByTagName('head')[0].appendChild(link);
}
window['ADS']['addStyleSheet'] = addStyleSheet;

/* 移除样式表 */
function removeStyleSheet(url,media) {
    var styles = getStyleSheets(url,media);
    for(var i = 0 ; i < styles.length ; i++) {
        var node = styles[i].ownerNode || styles[i].owningElement;
        // 禁用样式表
        styles[i].disabled = true;
        // 移除节点
        node.parentNode.removeChild(node);
    }
}
window['ADS']['removeStyleSheet'] = removeStyleSheet;

……以下是库中已有的内容……

})();
```

ADS.addStyleSheet()方法会根据给定的URL和media参数向文档中添加样式表：

```
ADS.addStyleSheet('/path/to/style.css','screen');
```

同样地，ADS.removeStyleSheet()方法则会根据给定的URL和media移除样式表：

```
ADS.removeStyleSheet('/path/to/style.css','screen');
```

> ADS.removeStyleSheet()方法依赖于将在后面添加的ADS.getStyleSheets()方法。

通过载入和卸载不同的样式表，你可以在自己喜欢的任意多个样式表之间进行切换。

### 5.3.4 修改 CSS 规则

当需要通过脚本修改页面上一些相关元素的外观时，修改样式表中实际的CSS规则，往往要比查找所有元素并分别修改其样式属性来得更容易一些。这种技术特别适合于只想修改几个特殊的属性，而仍然希望声明的层叠机制继续应用到像锚的:hover伪类这样的选择符之上的情况。

以下面这些针对锚的简单CSS声明为例：

```
/* 锚样式 */
a{
    font-weight:normal;
}
a:link {
    text-decoration:underline①;
    color: black;
}
a:visited {
    text-decoration:none;
}
a:hover {
    color: #248030;
```

① 原代码中的text-decoration:none表示链接在默认状态下没有下划线，这与图5-6不符。故将其修改为text-decoration:underline。当然，去掉该条声明也可以。——译者注

www.TopSage.com

```
    text-decoration: none;
}
```

如果将这些CSS规则应用到一个包含几个链接的简单文本段落上，例如

```
<p><a href="http://lipsum.com" title="Go to lipsum.com">Lorem ➡
ipsum</a> dolor sit amet, consectetuer adipiscing elit. <a ➡
href="http://lipsum.com" title="Go to lipsum.com">Aliquam ➡
tempor</a> risus ac elit. Nullam consectetuer. Sed feugiat ➡
pharetra enim. Mauris et velit in felis ultricies suscipit. ➡
Proin quam arcu, <a href="http://lipsum.com" title="Go to ➡
lipsum.com">mattis vitae</a>, consectetuer non, cursus non, ➡
mauris. Fusce tristique magna id diam. Mauris sit amet lacus a ➡
elit <a href="http://lipsum.com" title="Go to lipsum.com">auctor ➡
dapibus</a>. Aliquam eros sem, nonummy vitae, mollis et, tempus ➡
vel, neque.</p>
```

那么就会看到与图5-6所示的类似页面。

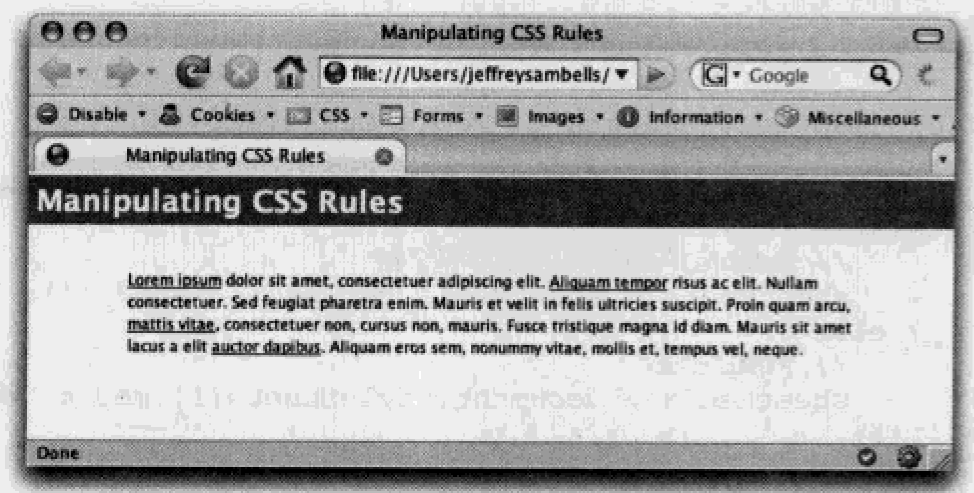



图5-6 带有不同链接的文本页面

如果想通过DOM脚本在页面中每个锚的后面显示出相应的URL，有以下几种方法可以选择。

首先，可以循环遍历页面中的所有链接并使用适当的DOM方法读取每个元素的href属性，然后再将获得的内容添加到相应的锚中。这种方法虽然有效，但它需要使用脚本搜索DOM树中所有适当的锚标签并直接修改标记。而如果浏览器支持相应的CSS2属性和选择符，例如content和:after伪类，那么就可以使用简单的CSS规则实现同样的目的：

```
a[href]:after {
    content: " (" attr(href) ") ";
    font-size: 40%;
    color: #16009b;
}
```

其次，作为修改DOM文档标记的替代方案，可以使用DOM脚本在样式表中添加或编辑CSS规则，不过得需要一点技巧。在document.styleSheets属性中，可能会包含几个样式表。除非你知道自己想要操作的是哪一个，否则就必须对这些样式表进行循环遍历以查找匹配的选择符。即使你知道想要操作的样式表的URL，仍然需要对所有样式表进行循环遍历，才能找到想要操纵的样式规则，因为document.styleSheets是通过数字索引的列表。下面是一个可以添加

www.TopSage.com

到ADS库中的方法，它能够帮你从document.styleSheets列表中查找出带有适当的href和media属性的样式表：

```
(function(){

window['ADS'] = {};

……以上是库中已有的内容……

/* 通过URL取得包含所有样式表的数组 */
function getStyleSheets(url,media) {
    var sheets = [];
    for(var i = 0 ; i < document.styleSheets.length ; i++) {
        if (url && document.styleSheets[i].href.indexOf(url) == -1) {
            continue;
        }
        if(media) {
            // 规范化media字符串
            media = media.replace(/,\s*/,',');
            var sheetMedia;

            if(document.styleSheets[i].media.mediaText) {
                    // DOM方法
                    sheetMedia = document.styleSheets[i].media. ➡
mediaText.replace(/,\s*/,',');
                    // Safari会添加额外的逗号和空格
                    sheetMedia = sheetMedia.replace(/,\s*$/,'');
                } else {
                    // MSIE方法
                    sheetMedia = document.styleSheets[i].media. ➡
replace(/,\s*/,',');
            }
            // 如果media不匹配则跳过
            if (media != sheetMedia) { continue; }
        }
        sheets.push(document.styleSheets[i]);
    }
    return sheets;
}
window['ADS']['getStyleSheets'] = getStyleSheets;

……以下是库中已有的内容……

})();
```

这个getStyleSheets()方法会根据给定的CSSStyleSheet对象的href属性和可选的media属性，返回一个包含所有匹配的样式表的数组。对于文档头部或主体中的嵌入<style>样式代码块（不应该这样使用样式），其href属性是当前页面的URL。

现在，可以找到目标样式表了。但还需要向你的ADS库中添加以下ADS.editCSSRule()和ADS.addCSSRule()方法才能修改其中的样式规则：

```
(function(){

window['ADS'] = {};

……以上是库中已有的内容……
```

www.TopSage.com

```
/* 编辑一条样式规则 */
function editCSSRule(selector,styles,url,media) {
    var styleSheets = (typeof url == 'array' ? url : ➡
getStyleSheets(url,media));

    for ( i = 0; i < styleSheets.length; i++ ) {

        // 取得规则列表
        // DOM2样式规范方法是styleSheets[i].cssRules
        // MSIE方法是styleSheets[i].rules
        var rules = styleSheets[i].cssRules || styleSheets[i].rules;
        if (!rules) { continue; }

        // 由于MSIE默认使用大写故转换为大写形式
        // 如果你使用的是区分大小写的id，则可能会
        // 导致冲突
        selector = selector.toUpperCase();

        for(var j = 0; j < rules.length; j++) {
            // 检查是否匹配
            if(rules[j].selectorText.toUpperCase() == selector) {
                for (property in styles) {
                    if(!styles.hasOwnProperty(property)) { continue; }
                    // 设置新的样式属性
                    rules[j].style[camelize(property)] = ➡
styles[property];
                }
            }
        }
    }
}
window['ADS']['editCSSRule'] = editCSSRule;

/* 添加一条CSS规则 */
function addCSSRule(selector, styles, index, url, media) {
    var declaration = '';

    // 根据styles参数（样式对象）构建声明字符串
    for (property in styles) {
        if(!styles.hasOwnProperty(property)) { continue; }
        declaration += property + ':' + styles[property] + '; ';
    }

    var styleSheets = (typeof url == 'array' ? url : ➡
getStyleSheets(url,media));
    var newIndex;
    for(var i = 0 ; i < styleSheets.length ; i++) {
        // 添加规则
        if(styleSheets[i].insertRule) {
            // DOM2样式规范的方法
            // index = length是列表末尾
            newIndex = (index >= 0 ? index : ➡
styleSheets[i].cssRules.length);
            styleSheets[i].insertRule(
                selector + ' { ' + declaration + ' } ',
                newIndex
            );
        } else if(styleSheets[i].addRule) {
            // Microsoft的方法
            // index = -1 是列表的末尾
            newIndex = (index >= 0 ? index : -1);
```

www.TopSage.com

```
                styleSheets[i].addRule(selector, declaration, newIndex);
            }
        }
    }
    window['ADS']['addCSSRule'] = addCSSRule;

    ……以下是库中已有的内容……

})();
```

记住，在编辑规则时只能编辑在样式表中已经明确声明的规则。而且，ADS.addCSSRule()方法在Safari中无效，但它却能在Webkit.org最新发布的WebKit中工作，因此可以期待该方法在将来Safari的某个版本中能够运行。

将这些方法应用到前面包含的文本和链接页面，就可以完成一些简单的修改，例如使用一种背景颜色突出显示所有锚：

```
ADS.editCSSRule('a',{'background-color':'yellow'});
```

当编辑锚的背景颜色时，除了层叠过程中会覆盖这个颜色的样式，所有锚都会取得这个新的颜色样式。例如，如果a:hover声明了一个不同的背景颜色，则悬停状态下锚的背景仍然会是原先的颜色。但是，只修改元素自身的style属性则无法实现类似这样的改变。

当然，也可以通过向样式表中添加一个新选择符来显示URL（如图5-7所示）。不过，这种方式只对于Firefox、Safari和Opera这样的实现了CSS 2.1标准的浏览器才有效：

```
ADS.addCSSRule('a[href]:after',{
    'content':'" (" attr(href) ") "',
    'font-size': '40%',
    'color': '#16009b'
});
```

对于ADS.addCSSRule()方法而言，其第3个参数index用于指定将新规则添加到CSS文档中的什么位置，使用null则表示添加到文件末尾。

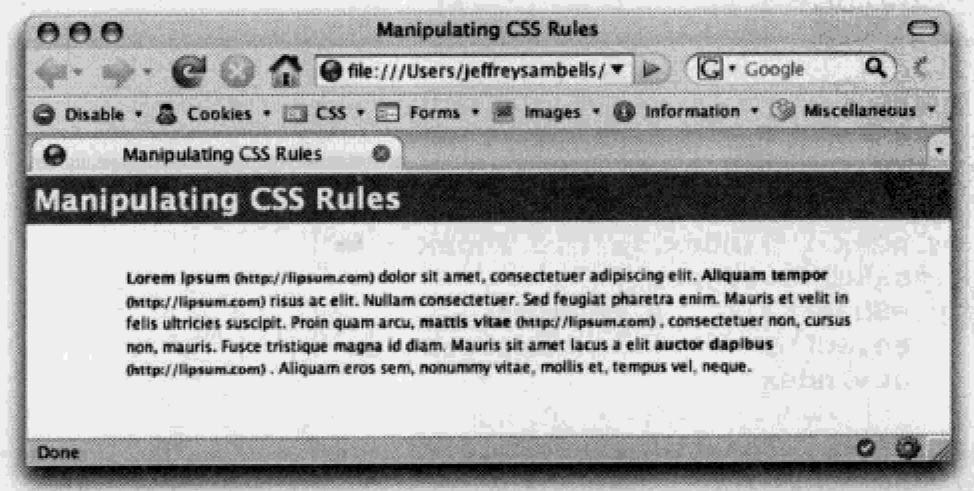



图5-7　修改后显示href属性的锚

www.TopSage.com

在以上例子的文本页面中，只包含一个样式表，因此对于编辑和添加规则的方法来说，可选的url和media参数并不是必需的。如果页面中存在多个样式表，那么就可以通过包含可选的参数来只取得匹配url的那些样式表：

```
ADS.editCSSRule(
    'a:hover',
    {'text-decoration':'underline'},
    '/path/to/style.css'
);
ADS.addCSSRule(
    'a',
    {'font-weight':'bold'},
    null,
    '/path/to/style.css'
);
```

修改个别的样式属性、切换body标签的类以及编辑CSS规则都有各自的优点，因此需要根据自己开发的需求来决定最佳方案。

**改进高级图像替换**

本章前面，我们在chapter5/image-replacement/advancED.html的例子中创建了一个图像替换脚本，并在load事件中硬编码了图像的URL：

```
ADS.addEvent(window, 'load', function() {

    ……省略的代码……

    // 载入图像
    // 这种硬编码的方式并不理想
    image.src = 'http://advanceddomscripting.com/images/ ➥
advancED-replace.png';
});
```

现在，可以不必在DOM脚本中硬编码URL了，我们可以为实现图像替换指定一个样式表（例如advancED.css），然后从其样式规则中搜索必需的背景图像URL。

为此，需要按照前面CSS文件的相同逻辑来创建一个CSS文件，如chapter5/imagereplacement/advancED.css中所示：

```
/*在图像被禁用时为文本添加的样式 */
#advancedHeader {
    color: #1A5B9D;
}

/* 为放置图像设置标题大小 */
#advancedHeader.advancED {
    height:60px;
    position:relative;
    overflow:hidden;
}
/* 通过不透明图像来隐藏文本 */
#advancedHeader.advancED span {
    background: white url(http://advanceddomscripting.com/ ➥
images/advancED-replace.png) no-repeat;
```

www.TopSage.com

```
    display:block;
    width:100%;
    height:100%;
    position:absolute;
}
```

然后，为了取得图像的URL，需要通过组合的ID号、advancED类以及<span>标签来检查样式表中每条规则的selectorText属性：

```
#advancedHeader.advancED span
```

同样地，这里面也有一个小陷阱。这个选择符[1]也可以定义为

```
.advancED#advancedHeader span
```

或者带上一个标签，如<h2>

```
h2.advancED#advancedHeader span
```

而且，在IE中，无论CSS文件中的代码是如何编写的，这条规则的selectorText属性都会被转换为.class#id的形式，并且，还会将标签转换为大写形式：

```
.advancED#advancedHeader SPAN
```

你可以像chapter5/image-replacement-revisited/advancED.js中的脚本代码那样，将前面图像替换例子中的load事件处理程序重新包装成一个辅助方法，只向其中传递元素的ID作为参数：

```
function replaceImage(element) {
    // 取得元素
    var element = ADS.$(element);
    // 创建图像元素
    var image = document.createElement('IMG');

    // 当图像载入成功后再添加span和类名
    ADS.addEvent(image, 'load', function() {

        var s = document.createElement('SPAN');
        // 将span添加为元素的子元素
        ADS.prependChild(element,s);

        // 创建必要的title属性
        if(!element.getAttribute('title')) {
            var i, child;
            var title = '';
            // 循环遍历子元素以组合title属性
            for(i=0 ; child = element.childNodes[i] ; i++ ) {
                if(child.nodeValue) title += child.nodeValue;
            }
            element.setAttribute('title',title);
        }
        // 修改类名以标明
        // 变更并应用CSS
        ADS.addClassName(element,'advancED');
```

① 原文rule有误。——译者注

www.TopSage.com

```
    });

    // 载入图像
    var styleSheet = ADS.getStyleSheets('advancED.css')[0];
    if(!styleSheet) return;

    var list = styleSheet.cssRules || styleSheet.rules
    if(!list) return;

    var rule;
    for(var j = 0 ; rule = list[j] ; j++) {

        // 查找规则：
        // 可能是 #element-id.advancED span
        // 或者 .advancED#element-id span
        // 或者MSIE中的：.advancED#element-id SPAN
        // 其中element-id是传递到方法中的参数
        if(
            rule.selectorText.indexOf('#' + ➥
element.getAttribute('id')) !== -1
            && rule.selectorText.indexOf('.advancED') !== -1
            && rule.selectorText.toUpperCase().indexOf(' SPAN') !== -1
        ){
            // 使用正则表达式：/url\(([^\)]+)\)/
            // 在CSS规则中查找URL
            var matches = rule.style.cssText.match(/url\(([^\)]+)\)/);
            // matches[1]中包含的是与正则表达式
            // 匹配的捕获圆括号中的值
            if(matches[1]) {
                image.src = matches[1];
                break;
            }
        }
    }
}

ADS.addEvent(window, 'load', function() {
    replaceImage('advancedHeader');
});
```

通过以上load事件和replaceImage()方法，在取得advancED.css文件并使用正则表达式从适当的规则中解析出URL的基础上完成了相同的效果。这样，对象的样式就实现了与DOM脚本完全分离，而且也可以按照需要自由地修改CSS文件了。

如果在此基础上再深入一步，可以通过在标记中添加一个类作为标识来使得DOM脚本更加不唐突：

```
<h2 id="advancedHeader" class="replaceMe">Advanced DOM Scripting</h2>
```

然后，通过load事件查找所有匹配该类的元素，并按照预期以图像进行替换：

```
ADS.addEvent(window, 'load', function() {
    var replacements = ADS.getElementsByClassName('replaceMe');
    for(var i=0 ; i< replacements.length ; i++) {
        replaceImage(replacements[i]);
    }
});
```

www.TopSage.com

最后一步改进使脚本具有了高度的可维护性，因为你所要做的只是包含脚本并对文档作出相应的标记和添加CSS。不需要对DOM脚本进行任何多余的编辑。

## 5.4 访问计算样式

在修改一个元素的表现之前，你可能希望首先确定它当前的样式属性。正如前面所提到的，元素的style属性只适用于以嵌入方式定义的样式，因此无法通过style取得计算样式。虽然可以从CSS规则自身中取得附加的样式信息，但由于这个过程冗长而复杂，最好还是交给浏览器去完成。如果希望访问一个元素所有的计算样式（如由层叠决定的样式），则需要使用其他替代属性。

DOM2样式规范在document.defaultView中包含了一个名叫getComputedStyle()的方法，恰好是为这个目的而设计的。该方法返回一个只读的CSSStyleDeclaration对象，其中包含特定元素的所有计算样式，而不仅仅是以嵌入方式定义的样式。

在取得了给定元素的计算样式之后，可以通过与操作元素的style属性一样的方式来取得样式信息：

```
var element = ADS.$('example');
var styles = document.defaultView.getComputedStyle(element);
```

要取得background-color只需像下面这样简单：

```
var color = styles.getProperty('background-color');
```

同样地，使用这种方法的问题在于Microsoft有自己的使用元素的currentStyle属性的版本，因此必须要在下面这个添加到ADS库中的ADS.getStyle()方法中使用这个专有的方法：

```
(function(){
window['ADS'] = {};

……以上是库中已有的内容……

/* 取得一个元素的计算样式 */
function getStyle(element,property) {
    if(!(element = $(element)) || !property) return false;
    // 检测元素style属性中的值
    var value = element.style[camelize(property)];
    if (!value) {
        // 取得计算的样式值
        if (document.defaultView && document.defaultView. ➥
getComputedStyle) {
            // DOM方法
            var css = document.defaultView.getComputedStyle(
                element, null
            );
            value = css ? css.getPropertyValue(property) : null;
        } else if (element.currentStyle) {
            // MSIE的方法
            value = element.currentStyle[camelize(property)];
        }
    }
    // 返回空字符串而不是auto
    // 这样就不必检查auto值了
```

www.TopSage.com

```
        return value == 'auto' ? '' : value;
    }
    window['ADS']['getStyle'] = getStyle;
    window['ADS']['getStyleById'] = getStyle;
    ……以下是库中已有的内容……

    })();
```

## 5.5 Microsoft 的 filter 属性

虽然我并不提倡使用跨浏览器间非标准的专有特性，但如果是利用这些特性来修复它们所属浏览器的不兼容性，那就另当别论了。在这里，我指的是IE的滤镜属性和该属性“修复”Microsoft IE 6或更低版本中透明PNG文件的事实。

我要介绍的这个属性没有遵循任何标准，而且只能在IE 6及更低版本中运行，因此相关代码中的主要部分都是针对Microsoft的。假如这些代码遇到了比IE 6版本高的浏览器或者其他浏览器将不能运行，因为在这些浏览器中不需要这些代码。

IE 6的问题是，它不能正确地显示PNG文件中的透明区域；相反，它会将透明区域显示为亮蓝色的方块，如图5-8所示。

图5-8　在IE 6中显示的透明PNG文件

可能出现这种问题的情况主要有：

- 以PNG作为源文件的<img>元素。
- 使用<link>元素或@import规则导入的样式表中，使用透明PNG图像作为元素背景。
- 在嵌入式的样式属性中将透明的PNG图像定义为元素的背景。

要解决这个问题，可以使用Microsoft.AlphaImageLoader滤镜，通过该滤镜可以将透明PNG文件设置为HTML元素的背景。为此可以使用条件注释来包含只针对Microsoft的CSS文件，而在这个CSS文件中应用该滤镜：

```
<!--[if lte IE 6]>
    <link rel="stylesheet" href="style-lte-IE-6.css" type="text/css"/>
<![endif]-->
```

在以上样式表中包含针对特定元素的适当样式：

```
#example {
    /* IE透明PNG hack */
    background-color: transparent;
    background-image: url(blank.gif);
    width: 100px;
    height: 100px;
    /* 在滤镜中必须使用完整路径 */
    filter: progid:DXImageTransform.Microsoft.AlphaImageLoader( ➡
```

www.TopSage.com

```
src="translucent-image.png", sizingMethod="scale");
}
```

在例子源文件chapter5/fixMSIEPng/test.html中，使用了包含在chapter5/fixMSIEPng/fixMSIEPng.js中的`fixMSIEPng()`方法，该方法会在页面载入时运行并遍历文档和CSS文件，以便查找相关的PNG文件并对其进行修复：

```
function fixMSIEPng() {
    if(!document.body.filters) {
        // 不是MSIE
        return;
    }
    if(7 <= parseFloat(navigator.appVersion.split("MSIE")[1])) {
        // 7.0以上支持PNG
        return;
    }
    // 修复嵌入的图像
    if(document.images) {
        var images = document.images;
        var img = null;

        for(var i=images.length-1; img=images[i]; i--) {

            // 检查是不是PNG图像
            if(img.src
                && img.src.substring(
                    img.src.length-3,
                    img.src.length
                ).toLowerCase() !== 'png'
            ) {
                // 跳过
                continue;
            }

            // 为外部元素构建style属性
            var inlineStyle = '';
            if (img.align == 'left' || img.align == 'right') {
                inlineStyle += 'float:' + img.align + ';';
            }
            if (img.parentElement.nodeName == 'A') {
                // 这幅图像位于锚中故显示手形光标
                inlineStyle += 'cursor:hand;';
            }

            // 将display设置为inline-block以便拥有width
            // 和height属性，且仍具有适当的定位
            inlineStyle += 'display:inline-block;';

            // 取得应用到这个元素的其他CSS样式
            if(img.style && img.style.cssText) {
                inlineStyle += img.style.cssText;
            }

            // 通过带有适当样式和信息（如className
            // 和ID）的<span>标签包围这幅图像
            img.outerHTML = '<span '
            + (img.id ? ' id="' + img.id + '"' : '' )
            + (img.className ? 'class="' + img.className + '" ' : '')
            + ' style="width:' + img.width + 'px; height:' + img.height + 'px;'
```

www.TopSage.com

```
            + inlineStyle
            + ';filter:progid:DXImageTransform.Microsoft.'+'.AlphaImageLoader(src=\''
            + img.src
            + '\', sizingMethod=\'scale\');"></span>';
        }
    }

    // 创建一个用于下一组循环的私有方法
    // 这个方法会为元素设置适当的样式
    function addFilters(e) {
        // 检查元素是否有style，进而包含background
        // 并确保还没有应用滤镜
        if(
            e.style
            && e.style.background
            && !e.style.filter
        ) {
            // 检查是不是PNG
            var src=null;
            if(src = e.style.backgroundImage. ➥
match(/^url\((.*\.png)\)$/i)) {
                e.style.backgroundColor = 'transparent';
                e.style.backgroundImage = 'url()';
                e.style.filter = 'progid:DXImageTransform.Microsoft.'
                    + 'AlphaImageLoader(src=\''
                    + src[1]
                    + '\',sizingMethod=\''
                    + (( e.style.width && e.style.height ) ? ➥
'scale' : 'crop' )
                    + '\')';
            }
        }
    }

    // 创建一个私有的递归处理方法
    // 将addFilters()方法应用到样式表
    function processRules(styleSheet) {
        for (var i in styleSheet.rules) {
            addFilters(styleSheet.rules[i]);
        }

        // 递归由@import规则引入的stylesheets
        if(styleSheet.imports) {
            for (var j in styleSheet.imports) {
                processRules(styleSheet.imports[j]);
            }
        }
    }

    // 处理每个样式表
    var styleSheets = document.styleSheets;
    for(var i=0; i < styleSheets.length; i++) {
        processRules(styleSheets[i]);
    }

    // 修复嵌入的样式属性
    if(document.all) {
        var all = document.all;
        for(var i=0; i < all.length; i++) {
            addFilters(all[i]);
```

www.TopSage.com

```
        }
    }
}
if(window.attachEvent) window.attachEvent("onload", fixMSIEPng);
```

这个函数只能在IE中运行，因为它使用了只针对IE的attachEvent()方法。通过这个函数来转换所有PNG图像的效果如图5-9所示。

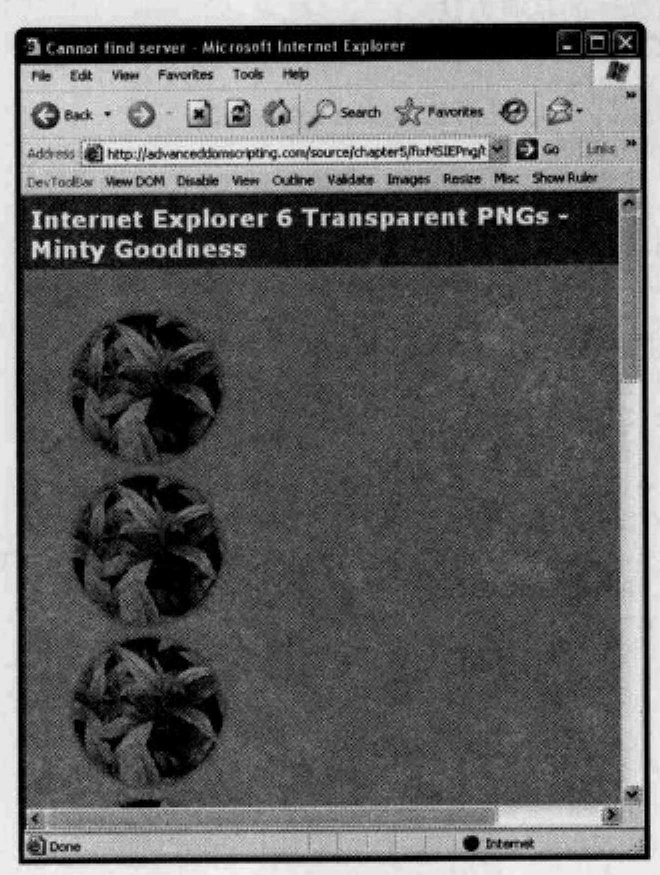



图5-9　当修复方法运行后PNG图像的透明背景正常显示了出来

请注意，这个滤镜属性中存在一个bug，即在某些情况下，位于应用了滤镜的元素内部的锚以及其他可单击的元素会变得不起作用。对于下面的例子，如果#background元素被应用了滤镜样式，那么嵌套在其中的锚将会失效：

```
<div id="background">
    <div id="content">
        <p>The content with <a href="http://example.com">anchors</a></p>
    </div>
</div>
```

为了避免这个问题，我建议将以上标记修改为类似下面这样，同时通过CSS把#content置于应用了滤镜的透明#background元素之上：

```
<div id="background"></div>
<div id="content">
    <p>The content with <a href="http://example.com">anchors</a></p>
</div>
```

## 5.6　实例：简单的渐变效果

在进一步学习新内容之前，下面这个例子将会给我们一个提醒：应该以不唐突的方式使用渐

www.TopSage.com

变效果增强用户体验，而不仅仅是因为你能做到。JavaScript正是由于被某些人用来开发唐突的动画才落下了不好的名声，事实上那些动画只有少数人认为很有趣也很酷，而多数人都感觉极其讨厌。在使用标准的DOM脚本编程方法的情况下，你仍然可能会犯同样的错误，因此在使用动画或渐变之类的效果时，一定要多加小心。

当需要巧妙地突出页面中的一部分以表现变化时，渐变效果会给人良好的感觉。例如，当用户在界面中完成了某个操作后，你可能希望向用户传达操作成功或失败的信息。同样地，如果某个操作影响到了页面中另一个看似无关的元素，你可能也希望指出相应的变化。

为了实现渐变效果，你编写的脚本必须随着时间推移运行一个周期性地修改期望元素的方法。所以，要使用JavaScript的`setTimeout()`函数，该函数会在给定的毫秒数之后调用一个方法。通过像下面这样创建一系列顺序调用`setTimeout()`方法的代码，随着时间推移修改同一个元素，就可以创建出期望的渐变效果：

```
setTimeout(modifyElement,10);
setTimeout(modifyElement,20);
setTimeout(modifyElement,30);
setTimeout(modifyElement,40);
setTimeout(modifyElement,50);
// 其他代码
```

渐变效果属于高度自定义的效果，因此在很多情况下，你必须使用自己的解决方案。但在第10章中，我们会介绍一些JavaScript库，其中不少库都有多种内置的渐变方法可以用来简化操作。不过在此之前，让我们通过实现一个在两种颜色之间淡入的简单效果来开掘出你的创意源泉。

使用包含在chapter5/fadeColor/test.html例子文件中的`fadeColor()`方法，可以指定两种颜色以及回调方法：

```
function fadeColor( from, to, callback, duration, framesPerSecond) {

    // setTimeout()的包装函数，
    // 用于基于帧数计算延迟时间
    function doTimeout(color,frame) {
        setTimeout(function() {
            try {
                callback(color);
            } catch(e) {
                // 去掉下面的注释可以对异常进行调试
                // ADS.log.write(e);
            }
        }, (duration*1000/framesPerSecond)*frame);
    }

    // 以秒表示的渐变效果持续时间
    var duration = duration || 1;
    // 在给定持续时间内动画的帧数
    var framesPerSecond = framesPerSecond || duration*15;

    var r,g,b;
    var frame = 1;

    // 在第0帧设置渐变的开始颜色
    doTimeout('rgb(' + from.r + ',' + from.g + ',' + from.b + ')',0);
```

www.TopSage.com

```
    // 计算两帧之间RGB值的改变量
    while (frame < framesPerSecond+1) {
        r = Math.ceil(from.r
            * ((framesPerSecond-frame)/framesPerSecond)
            + to.r * (frame/framesPerSecond));
        g = Math.ceil(from.g
            * ((framesPerSecond-frame)/framesPerSecond)
            + to.g * (frame/framesPerSecond));
        b = Math.ceil(from.b
            * ((framesPerSecond-frame)/framesPerSecond)
            + to.b * (frame/framesPerSecond));
        // 为这一帧调用延时函数
        doTimeout('rgb(' + r + ',' + g + ',' + b + ')',frame);

        frame++;
    }
}
```

为了调用这个渐变方法，需要定义两个包含起始和结束颜色的r、g、b值的JavaScript对象，并提供一个回调方法以便将新的颜色渐变效果应用到期望的任何元素上。这里的回调方法会取得一个包含格式为rgb(#,#,#)的颜色值的参数：

```
fadeColor(
    {r:0,g:255,b:0}, // 起始颜色
    {r:255,g:255,b:255}, // 结束颜色
    function(color) {
        // 将颜色应用到元素中
        ADS.setStyle('element',{'background-color':color});
    }
);
```

下面我们再来回顾一下第4章中那个在地址表单中填写邮政编码的例子。在那个例子中，用户填写一个邮政编码就可以引发预先装载其余地址字段的过程。而通过醒目的变化来表现预装载成功对用户是有帮助的。使用这里定义的fadeColor()方法会为那个例子的用户界面提供一种精致而又与信息提示相关的功能。而且，要实现这一功能只需向与邮政编码验证相关的XMLHttpRequest对象的onreadystatechange方法中添加几行代码即可：

```
req.onreadystatechange = function() {
    if (req.readyState == 4) {
        eval(req.responseText);

        if(ADS.$('street').value == '') {
            ADS.$('street').value = street;
            fadeColor(
                {r:0,g:255,b:0},{r:255,g:255,b:255},
                function(color) {
                    ADS.setStyle('street',
                      {'background-color':color}
                    );
                }
            );
        }

        if(ADS.$('city').value == '') {
```

www.TopSage.com

```
            ADS.$('city').value = city;
            fadeColor(
                {r:0,g:255,b:0},{r:255,g:255,b:255},
                function(color) {
                    ADS.setStyle('city',
                        {'background-color':color}
                    );
                }
            );
        }

        if(ADS.$('province').value == '') {
            ADS.$('province').value = province;
            fadeColor(
                {r:0,g:255,b:0},{r:255,g:255,b:255},
                function(color) {
                    ADS.setStyle('province',
                        {'background-color':color}
                    );
                }
            );
        }
    }
}
```

当装载完成后，这些字段的背景颜色都将从绿色逐渐淡化为白色，从而表明信息预装载成功。

## 5.7 小结

动态修改元素的表现是一个相对简单的任务，但要正确地完成这个任务也是一个挑战。在本章中，你学习了一些将会用到的常见的DOM2样式规范中规定的对象，并向你的ADS库中添加了许多能够跨浏览器访问这些对象的方法。

本章也自始至终地强调了将CSS表现从DOM脚本中分离出来的重要性。相关的一些方法和技术包括在变化范围较小的情况下，修改style属性和className切换，在涉及到全局性的变化时切换样式表和修改CSS规则。

记住这些概念和思想，无论对有众多设计者和开发者参与的大项目，还是只涉及到你本人的小项目而言，都是至关重要的。将表现从文档结构和行为增强中分离出来有助于你的Web应用程序更好地适应未来，而且坚持这一路线必将在你更新内容和重新设计Web应用时为你节省可观的时间、精力和金钱。

接下来，我们会在第6章中学习一个案例——届时将运用本书到目前为止介绍的所有知识，创建一个不唐突的图像裁剪和缩放工具。

www.TopSage.com

第 6 章

# 案例研究：图像裁剪和缩放工具

既然你的ADS库中又增加了一些新方法，下面我们就来创建一个能够用来裁剪和缩放照片的WYSIWYG工具。在你过于兴奋之前，我要提醒你一下，只使用DOM脚本实际上不可能对照片进行裁剪和缩放。完整的工具中还必须包含能够对实际的图像文件进行缩放的服务器端程序。在这个案例学习中，我们是要在浏览器中模拟裁剪和缩放操作，以便在看起来合适时将新的设置提交给服务器，并由服务器完成实际的修改。本章将要完成这一工具的基本功能，其操作界面类似图6-1所示。

图6-1　使用图像编辑工具操作示例图像的界面

## 6.1 测试文件

首先，打开本书源文件中包含的chapter6/imageEditor-start文件夹。该文件夹中包含下列测试

www.TopSage.com

页面及CSS、脚本和图像文件。

- test.html
- style.css
- imageEditor.js
- images/
- interface/handles.gif

> 别忘了，如果想获得这些作为起点的代码和文件，可以从http://advanceddomscripting.com上面下载。

在test.html文件中包含两幅测试图像和一些用于保存缩放信息的表单标记：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Image Editor Test Page</title>
    <!-- Include some CSS style sheet to make
    everything look a little nicer -->
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css"
        href="../chapter.css" />
    <link rel="stylesheet" type="text/css"
        href="style.css" />

    <!-- Your ADS library with the common JavaScript objects -->
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>
    <!-- Log object from Chapter 2 -->
    <script type="text/javascript"
        src="../../chapter2/myLogger-final/myLogger.js"></script>

    <!-- The imageEditor object -->
    <script type="text/javascript" src="imageEditor.js"></script>
</head>
<body>
<h1>Edit Your Images</h1>
<ul>
    <li>
        <form action="/path/to/server/script" method="post">
            <img id="mintImage" src="images/mint.jpg"/>
            <fieldset>
                <legend>New Size</legend>
                <div>
                    <label>Width</label>
                    <input type="text" name="newWidth">
                </div>
                <div>
                    <label>Height</label>
                    <input type="text" name="newHeight">
                    <p>The width and height applies to the
```

TopSage.com

www.TopSage.com

```
                original uncropped image</p>
            </div>
        </fieldset>
        <fieldset>
            <legend>Trim Edges</legend>
            <div>
                <label>Top</label>
                <input type="text" name="cropTop">
            </div>
            <div>
                <label>Rigt</label>
                <input type="text" name="cropRight">
            </div>
            <div>
                <label>Bottom</label>
                <input type="text" name="cropBottom">
            </div>
            <div>
                <label>Left</label>
                <input type="text" name="cropLeft">
            </div>
        </fieldset>
        <div class="buttons">
            <input type="submit" value="Apply">
            <input type="reset" value="Reset">
        </div>
    </form>
</li>
<li>
    <form action="/path/to/server/script" method="post">
        <img id="babyImage" src="images/baby.jpg"/>
        <fieldset>
            <legend>New Size</legend>
            <div>
                <label>Width</label>
                <input type="text" name="newWidth">
            </div>
            <div>
                <label>Height</label>
                <input type="text" name="newHeight">
                <p>The width and height applies to the
                original uncropped image</p>
            </div>
        </fieldset>
        <fieldset>
            <legend>Trim Edges</legend>
            <div>
                <label>Top</label>
                <input type="text" name="cropTop">
            </div>
            <div>
                <label>Rigt</label>
                <input type="text" name="cropRight">
            </div>
            <div>
                <label>Bottom</label>
                <input type="text" name="cropBottom">
```

www.TopSage.com

```
                </div>
                <div>
                    <label>Left</label>
                    <input type="text" name="cropLeft">
                </div>
            </fieldset>
            <div class="buttons">
                <input type="submit" value="Apply">
                <input type="reset" value="Reset">
            </div>
        </form>
    </li>
</ul>
</body>
</html>
```

test.html文件中也包含了一个CSS样式表，以便使界面更加美观，在浏览器中打开这个文件的结果如图6-2所示。

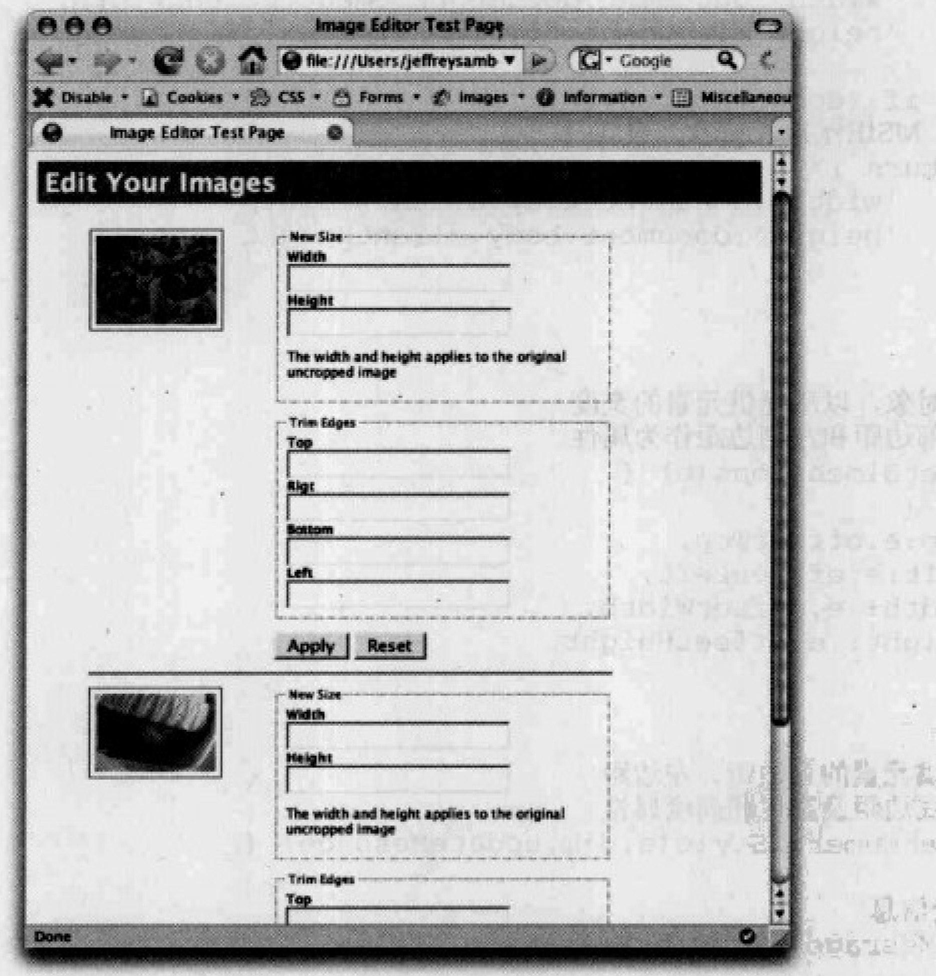



图6-2 DOM脚本代码运行之前的带有两幅图像的图像编辑器测试文件界面

在test.html文件的头部也包含了ADS库的最终版本，该版本中包括了本书迄今为止已添加的所有方法。如果你也自定义了自己的库，当然也可以使用你自己的库来代替这个版本。

这个页面实际上不会对表单提交的设置作任何处理，这些都要由你来决定。不过，在页面中包含表单可以为无法访问精美的拖放界面的浏览器提供一种备用方案。

在本章剩下的内容中，我们要实现的就是imageEditor.js文件。

## 6.2 imageEditor 对象

在准备好测试文件之后，我们就来看一看包含在chapter6/imageEditor-start/imageEditor.js中的将要实现的图像编辑器工具的初始结构。其中包含的空方法将在你学习本章剩余内容的过程中补充完整：

```
(function(){

// 返回一个数组，浏览器窗口的宽度和高度
// 分别保存在这个数组的第0和第1个元素中
function getWindowSize(){
    if (self.innerHeight) {
        // 最常用
        return { 'width':self.innerWidth,'height':self.innerHeight };
    } else if (document.documentElement
        && document.documentElement.clientHeight) {
        // MSIE严格模型
        return {
            'width':document.documentElement.clientWidth,
            'height':document.documentElement.clientHeight
        };
    } else if (document.body) {
        // MSIE怪异（quirk）模式
        return {
            'width':document.body.clientWidth,
            'height':document.body.clientHeight
        };
    }
};

// 返回一个对象，以所提供元素的宽度
// 高度、顶部边距和左侧边距作为属性
function getDimensions(e) {
    return {
        top:e.offsetTop,
        left:e.offsetLeft,
        width: e.offsetWidth,
        height: e.offsetHeight
    };
};

// 设置所提供元素的上边距、左边距
// 右边距、底边距及宽度和高度属性
function setNumericStyle(e,dim,updateMessage) {

    // 检查信息
    updateMessage = updateMessage || false;

    // 分配一个新对象
    // 原对象保持不变
    var style = {};
    for(var i in dim) {
        if(!dim.hasOwnProperty(i)) continue;
        style[i] = (dim[i]||'0') + 'px';
    }
    ADS.setStyle(e,style);

    // 如果存在信息则更新
    if(updateMessage) {
```

www.TopSage.com

```
            imageEditor.elements.cropSizeDisplay.firstChild.nodeValue =
                dim.width
                + 'x' + dim.height;
        }
};

function imageEditor() { };

// 一个在编辑图像时保存信息的属性
imageEditor.info = {
    resizeCropArea:false,
    pointerStart:null,
    resizeeStart:null,
    cropAreaStart:null,
    imgSrc:null
};

// 一个保存编辑器中DOM对象实例的属性
imageEditor.elements = {
    'backdrop': null,
    'editor': null,
    'resizeHandle': null,
    'cropSizeDisplay': null,
    'resizee': null,
    'resizeeCover': null,
    'cropArea': null,
    'resizeeClone': null,
    'cropResizeHandle': null,
    'saveHandle':null,
    'cancelHandle':null
};

// 这个方法会按照需要注册事件及修改DOM
// 它会在window载入时自动运行
imageEditor.load = function(W3CEvent) {

    // 取得页面中所有带ADSImageEditor类名的表单元素

        // 在符合条件的表单中查找图像

        // 为图像添加imageEditor.imageClick事件

        // 修改类名以便CSS按照需要修改其样式

        // 如果表单的类名被修改，则会应用CSS
        // 文件中包含的修改页面样式的额外规则
    }
};

imageEditor.unload = function(W3CEvent) {
    // 移除编辑及背景幕（backdrop）
};

imageEditor.imageClick = function(W3CEvent) {

    // 创建新的JavaScript Image对象
    // 以便确定图像的宽度和高度

    // this引用被单击的图像元素

    // 为放置背景幕和居中编辑器而取得页面大小
```

www.TopSage.com

```
    // 创建背景幕div，并使其撑满整个页面

    // 创建编辑器div以包含编辑工具的GUI（Graphical User Interface，图形用户界面）

    // 创建缩放手柄

    // 创建可缩放的图像

    // 创建半透明的蒙板（cover）

    // 创建裁剪大小显示区域

    // 创建裁剪区域容器

    // 在剪裁区域中创建图像的副本

    // 创建裁剪缩放手柄

    // 创建保存手柄

    // 创建取消缩放手柄

    // 向DOM元素添加事件

    // 缩放手柄的翻转图

    // 裁剪手柄的翻转图

    // 保存手柄的翻转图

    // 取消手柄的翻转图

    // 启动图像缩放事件流

    // 启动裁剪区域拖动事件流

    // 启动裁剪区域缩放事件流

    // 防止保存手柄启动裁剪拖动事件流

    // 在单击保存手柄或双击
    // 裁剪区域时保存图像

    // 防止取消手柄启动裁剪拖动事件流

    // 在单击时取消改变

    // 如果窗口大小改变则调整背景幕的大小
};

imageEditor.resizeMouseDown = function(W3CEvent) {

    // 保存当前位置和尺寸

    // 添加其余事件以启用拖动

    // 停止事件流

};
imageEditor.resizeMouseMove = function (W3CEvent) {
```

www.TopSage.com

```
        // 取得当前鼠标指针所在位置

        // 基于鼠标指针来计算
        // 图像新的宽度和高度

        // 最小尺寸是42平方像素

        // 计算基于原始值的百分比

        // 如果按下了Shift键，则按比例缩放

        // 计算裁剪区域的新尺寸

        // 缩放对象

        // 停止事件流
    };
    imageEditor.resizeMouseUp = function (W3CEvent) {
        // 移除事件侦听器以停止拖动
        // 停止事件流
    };
    // 裁剪区域上的mousedown事件侦听器
    imageEditor.cropMouseMove = function(W3CEvent) {
        // 包含缩放以限制裁剪区域的移动
        // 停止事件流
    };
    // 裁剪区域上的mousemove事件侦听器
    imageEditor.cropMouseMove = function(W3CEvent) {
        var pointer = ADS.getPointerPositionInDocument(W3CEvent);
        if(imageEditor.info.resizeCropArea) {
            // 缩放裁剪区域

            // 如果按下了Shift键，则按比例缩放
            // 计算基于原始值的百分比

            // 检查新位置是否超出了边界
        } else {
            // 移动裁剪区域

            // 检查新位置是否超出了边界
            // 如有必要则加以限制
        }
        // 停止事件流

    };
    imageEditor.cropMouseUp = function(W3CEvent) {
        // 移除所有事件
        // 停止事件流
    };
    imageEditor.saveClick = function(W3CEvent) {
        // 此处只是发出一个警告
        // 如果成功则卸载编辑器
    };
    imageEditor.cancelClick = function(W3CEvent) {

    };
    window['ADS']['imageEditor'] = imageEditor;
    })();
```

www.TopSage.com

```
// 因为该页面中可能包含许多图像，所以这里使用
// ADS.addLoadEvent()方法为window添加load事件
ADS.addLoadEvent(ADS.imageEditor.load);
```

通过观察以上代码，我们发现其中包含了如下一些新的实用方法。

- getWindowSize()返回一个对象，对象的width和height属性表示窗口大小。
- getDimensions(element)取得给定元素的顶边距、左边距、宽度和高度属性（分别以返回对象的top、left、width和height属性表示）。
- setNumericStyle(element,dimensions,message)用于设置给定的样式属性。该方法与ADS.setStyle()方法类似，但只能使用以px作为后缀的数字属性值。该方法会为指定的属性设置数字值。而且，如果第3个参数为true，图像编辑器的裁剪大小信息显示区域也会被更新为给定的尺寸（稍后将会讨论到）。

> 将要构建的imageEditor工具是作为一个实例化对象的实例而不是构造函数存在的。要了解有关JavaScript对象的更多内容，请参考第2章。

### 6.2.1 调用 imageEditor 工具

当在浏览器中查看test.html文件时（如图6-2所示），你会注意到为每一幅图像指定新尺寸及裁剪位置的所有必需的基本输入字段。无需通过任何增强，这个页面也可以轻松地实现其功能——只不过这个界面对用户并不十分友好，因为计算大小和裁剪区域需要一些数学知识。为了让这个界面对用户更友好一些，我们要隐藏手工输入的表单元素，同时为缩放和裁剪图像提供一个可拖动的界面。

我们的DOM脚本对开发者也应该是友好的。如果对于开发者来说，整合这个工具需要数小时的额外工作量，开发者很可能会弃之不用。为了提供无缝的整合，就需要考虑通过某种简单的方式让开发者能够以最小的努力来调用这一工具。对你而言，实现这一点最简单的方式，就是使这个工具能够通过扫描DOM文档来找到特殊的类名，然后按照需要修改相应的元素。通过定义一个特殊的类名，开发者可以标识出具有适当格式并应该被修改以包含裁剪工具的那些元素。如果没有这个类名或者JavaScript被禁用，表单将继续如图6-2所示的那样运行。不过该工具一旦通过load事件被激活，用户界面就会转变为图6-3（如6.2.2节所示）。

在本章这个例子当中，必要条件是与<form>标签关联的ADSImageEditor类。为确保正确运行，还要求表单的<form>标签中包含一个单独的<img>子元素，如下所示：

```
<form action="/path/to/server/script"
    method="post" class="ADSImageEditor">
    <img id="imageID" src="images/image.jpg"/>
    ……省略的代码……
</form>
```

除了以上<img>元素之外，如果还需要在表单中包含必要的输入元素，那么开发者可以按照自己认为合适的方式向其中添加其余的标记。

www.TopSage.com

> 对于我们要开发的这个图像增强工具而言，当前存在的一个限制就是它只能对每个表单中包含一幅图像的情况加以处理。等到开发完成之后，你可以通过修改事件处理程序使该工具能够处理更多幅图像。

现在，需要为chapter6/imageEditor-start/test.html文件中每个`<form>`标签都添加上`class="ADSImageEditor"`属性。这是你或者任何其他开发者在调用该工具时需要对测试文件所做的唯一一处修改——仅此而已。

### 6.2.2 imageEditor 载入事件

当在启用JavaScript的情况下载入页面时，脚本应该自动扫描适当的类名并按照需要修改标记。修改一旦完成，页面将会如图6-3所示——所有元素中除附加了适当事件的图像之外全都隐藏了起来。

图6-3 imageEditor.load()运行后的test.html文件

要做到这一点，载入事件侦听器需要

(1) 搜索DOM文档中的表单标签。

(2) 检测表单的className属性中是否包含ADSImageEditor类。

(3) 通过CSS属性隐藏表单中所有的输入元素和标记。

(4) 为<img>元素添加一个调用ADS.imageEditor.imageClick()方法的单击事件。

花点时间向imageEditor.load()方法中添加以下代码：

```
(function(){
……省略的代码……
imageEditor.load = function(W3CEvent) {

    // 取得页面中所有带ADSImageEditor类名的表单元素
    var forms = ADS.getElementsByClassName('ADSImageEditor', 'FORM');

    // 在符合条件的表单中查找图像
    for( var i=0 ; i < forms.length ; i++ ) {
```

```
            // 查找表单中的图像
            var images = forms[i].getElementsByTagName('img');
            if(!images[0]) {
                // 这个表单中不包含图像，跳过
                continue;
            }

            // 为图像添加imageEditor.imageClick事件
            ADS.addEvent(images[0],'click',imageEditor.imageClick);

            // 修改类名以便CSS按照需要修改其样式
            forms[i].className += ' ADSImageEditorModified';

            // 如果表单的类名被修改，则会应用CSS
            // 文件中包含的修改页面样式的额外规则
        }
    };
    ……省略的代码……
    })();
```

通过代码中的注释可以了解到，这个load()方法通过ADS.getElementsByClassName('ADSImageEditor','FORM')取得了页面中所有符合条件的表单，并检查表单中是否包含一个子图像元素。如果有图像，则将imageClick()事件指定给该图像，同时通过修改表单的类名以标识表单被修改过了。

在被修改之后，表单取得了另一个类名ADSImageEditorModified，CSS规则通过这个类名将现有的表单元素从视图中隐藏了起来：

```
form.ADSImageEditorModified * {
    position: absolute;
    left: -2000px;
}
form.ADSImageEditorModified img {
    position: inherit;
    left: auto;
}
```

在脚本文件的最后一行，将图像编辑器的载入事件注册给了window的载入事件：

```
// 向window对象添加载入事件
ADS.addLoadEvent(imageEditor.load);
```

在这里，使用的是第4章添加的ADS.addLoadEvent()方法，因为页面中很可能包含多幅图像，而我们不希望等到所有图像都下载完成再调用载入事件侦听器。在imageEditor.load()事件处理程序运行之后，重新载入的页面将如图6-3所示，其中所有表单元素都被隐藏了起来。此时，你可以单击任何一幅图像，以调用相应图像上的imageEditor.imageClick()方法。不过，此时调用该方法什么也不会发生，因为我们还没有为该方法添加任何代码。

### 6.2.3　创建编辑器标记和对象

编辑器工具自身是由一些在当前页面上以绝对定位方式分层叠放的对象组成的，这些未经样式化的标记如下（标记中的注释表明了每个元素的用途）：

www.TopSage.com

```
<div><!-- 背景幕 --></div>
<div>
    <!-- 编辑器 -->
    <div><!-- 缩放手柄 --></div>
    <img src="/path/to/image.jpg">
    <div><!-- 半透明的蒙板 --></div>
    <div>
        <!-- 裁剪区域 -->
        <img src="/path/to/image.jpg">
        <div><!-- 裁剪大小显示区域 --></div>
        <div><!-- 裁剪手柄 --></div>
        <div><!-- 保存手柄 --></div>
        <div><!-- 取消手柄 --></div>
    </div>
</div>
```

当通过此工具操纵图像时，拖动缩放手柄可以缩放图像，而拖动裁剪手柄则可以缩放被裁剪的区域。此外，还需要让裁剪区域自身能够被拖动，因此裁剪区域能够在图像上面被重新定位。为保存或取消更改，还需要几个可单击的元素。图6-4显示了不同元素的标记经解析后呈现的分解示意图。

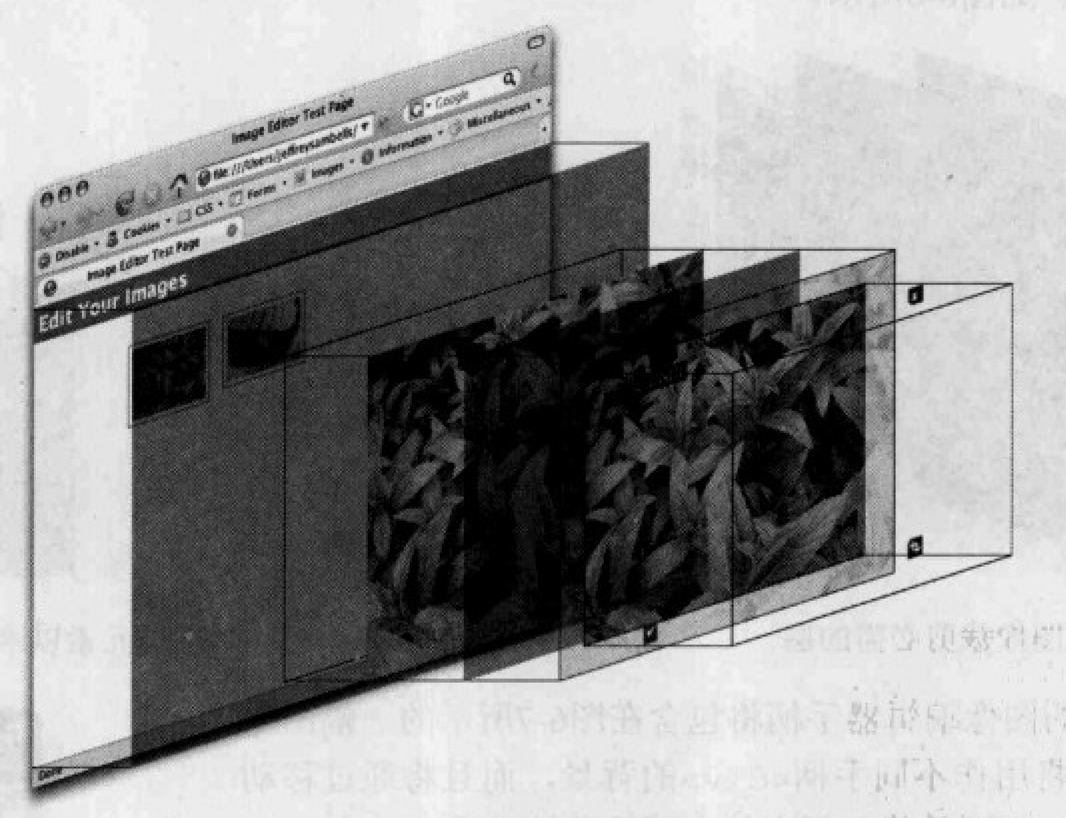



图6-4 图像编辑器工具中解析后的标记分解示意图

> 图6-4表现的是工具中的标记在视觉上分层的结果，并没有表现出这些标记之间的父-子嵌套关系。而在文档中，这些“层”的主体部分实际上是相互嵌套的元素。

如图6-4所示，底层是半透明的覆盖整个窗口区域的盒子。这一层起到了从视觉上把编辑器与页面其余部分隔开的作用。而且，它也充当了捕获图像区域外部任何单击事件的安全装置的作用。因此，在编辑图像期间不会意外地发生位于半透明区域下方的页面元素的鼠标事件。

www.TopSage.com

接下来的几层对于图像编辑器工具中的缩放功能是必需的。缩放图像很容易，因为你可以修改width和height属性，而让浏览器负责完成剩下的显示工作。但是，事情并没有那么简单。因为所有与裁剪相关的元素也需要与图像一同被缩放。此时的缩放，几乎要以某种方式修改每个元素的大小。

编辑器中复杂的部分是模拟裁剪。为了使裁剪功能在视觉上更真实（代码也更难编写），我们要在被裁剪的图像下方以灰色溢出的形式显示完整的图像。为了实现这种效果，需要在完整的图像上面放置一个（相同大小的）半透明的对象，以这个对象作为图像的“灰色溢出”部分。在这个半透明对象上方，还需要使用另一个元素来模拟被裁剪的图像，如图6-5所示。

在浏览器中无法真正裁剪图像——不过，可以将图像放在另一个元素中，并将该元素设置为相对或绝对定位，同时设置其overflow:hidden样式属性。当图像被以负的top和left值进行定位时，这个元素（即<div>）将使图像看起来像是被裁剪了一样。但实际上，图像的其余部分只是通过溢出隐藏了起来。这中间关键的部分是在图像上面拖动<div>的时候：当裁剪<div>沿某一路径移动时，需要向相反的方向调整其内部图像的位置，以获得在裁剪区域移动时图像保持固定不动的效果，如图6-6所示。

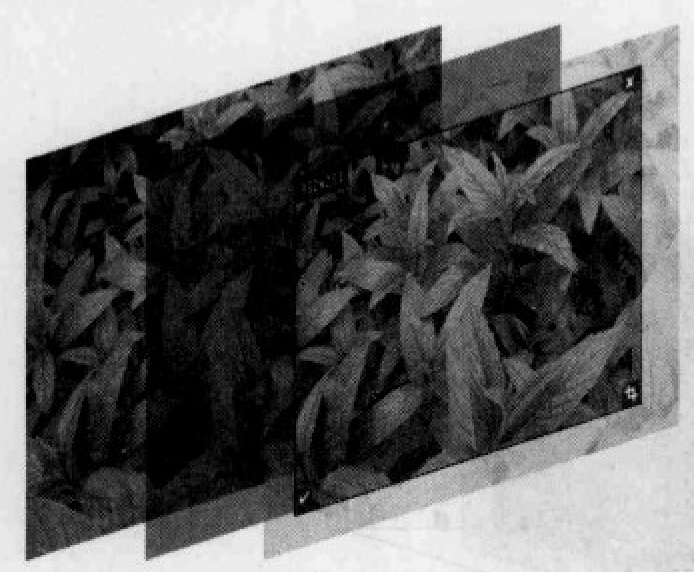
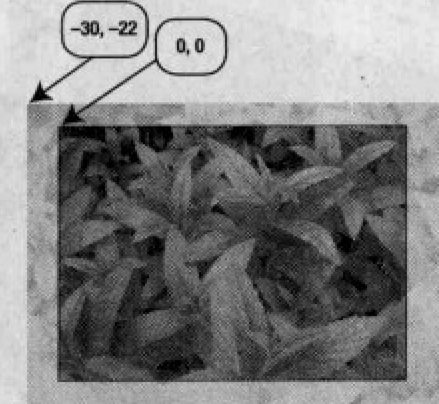


图6-5　模拟图像裁剪必需的层



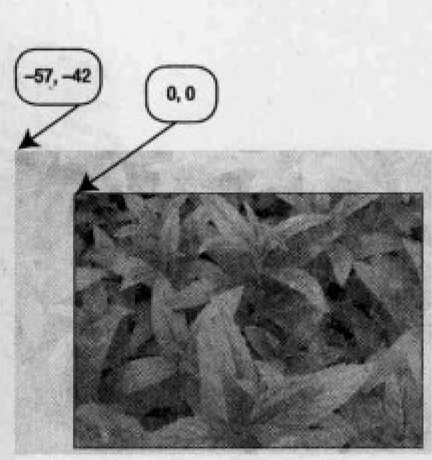




图6-6　向负方向移动子图像元素以维持其相对位置

最后，不同的图像编辑器手柄将包含在图6-7所示的一幅图像之中。这幅图像将用作不同手柄<div>的背景，而且将通过移动背景图像的位置，而不是载入新文件来创建翻转效果。

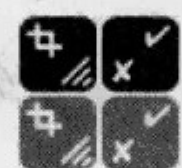


图6-7　完整的handles.png图像

在解释完这些之后，我们可以实际地进行编码了。用来新创建图像编辑器标记的必需的DOM代码相对简单，而且只涉及创建一些<div>和<img>元素。这些元素将在用户单击图像并调用imageEditor.imageClick()方法时被添加到DOM文档结构中。我们不会详细解释如何定位所有这些元素，因为通过代码你可以很容易地看出这些过程。此外，原始图像的大小将被作为编辑器区域的起始大小，而其他各层都将据之进行相应定位。

下面花点时间完成imageEditor.imageClick()方法。当稍后需要注册某些事件侦听器

www.TopSage.com

时，会再看到这个方法的后半部分代码：

```
(function(){
……省略的代码……
imageEditor.imageClick = function(W3CEvent) {

    // 创建新的JavaScript Image对象
    // 以便确定图像的宽度和高度
    var image = new Image();

    // this引用被单击的图像元素。
    image.src = imageEditor.info.imgSrc = this.src;

    // 为放置背景幕和居中编辑器而取得页面大小
    var windowSize = getWindowSize();

    // 创建背景幕div，并使其撑满整个页面
    var backdrop = document.createElement('div');
    imageEditor.elements.backdrop = backdrop;
    ADS.setStyle(backdrop,{
        'position':'absolute',
        'background-color':'black',
        'opacity':'0.8',
        'width':'100%',
        'height':'100%',
        'z-index':10000,
        // 对于MSIE需要使用滤镜
        'filter':'alpha(opacity=80)'

    });
    setNumericStyle(backdrop,{
        'left':0,
        'top':0,
        'width':windowSize.width,
        'height':windowSize.height
    });

    document.body.appendChild(backdrop);

    // 创建编辑器div以包含编辑工具的GUI（Graphical User Interface，图形用户界面）
    var editor = document.createElement('div');
    imageEditor.elements.editor = editor;
    ADS.setStyle(editor,{
        'position':'absolute',
        'z-index':10001
    });
    setNumericStyle(editor,{
        'left': Math.ceil((windowSize.width-image.width)/2),
        'top': Math.ceil((windowSize.height-image.height)/2),
        'width':image.width,
        'height':image.height
    });
    // 将编辑器添加到文档中
    document.body.appendChild(editor);
```

www.TopSage.com

```
    // 创建缩放手柄
    var resizeHandle = document.createElement('div');
    imageEditor.elements.resizeHandle = resizeHandle;
    ADS.setStyle(resizeHandle,{
        'position':'absolute',
        'background':'transparent url(interface/handles.gif) ➡
no-repeat 0 0'
    });
    setNumericStyle(resizeHandle,{
        'left':(image.width -18),
        'top':(image.height -18),
        'width':28,
        'height':28
    });
    // 将缩放手柄添加到编辑器中
    editor.appendChild(resizeHandle);

    // 创建可缩放的图像
    var resizee = document.createElement('img');
    imageEditor.elements.resizee = resizee;
    resizee.src = imageEditor.info.imgSrc;

    // 去掉应用给img元素的任何CSS
    ADS.setStyle(resizee,{
        'position':'absolute',
        'margin':0,
        'padding':0,
        'border':0
    });
    setNumericStyle(resizee,{
        'left':0,
        'top':0,
        'width':image.width,
        'height':image.height
    });

    editor.appendChild(resizee);

    // 创建半透明的蒙板（cover）
    var resizeeCover = document.createElement('div');
    imageEditor.elements.resizeeCover = resizeeCover;
    ADS.setStyle(resizeeCover,{
        'position':'absolute',
        'background-color':'black',
        'opacity':0.5,
        // 对MSIE需要使用滤镜
        'filter':'alpha(opacity=50)'
    });
    setNumericStyle(resizeeCover,{
        'left':0,
        'top':0,
        'width':image.width,
        'height':image.height
    });

    editor.appendChild(resizeeCover);
```

www.TopSage.com

```
// 创建裁剪大小显示区域
var cropSizeDisplay = document.createElement('div');
imageEditor.elements.cropSizeDisplay = cropSizeDisplay;
ADS.setStyle(cropSizeDisplay,{
    'position':'absolute',
    'background-color':'black',
    'color':'white'
});

setNumericStyle(cropSizeDisplay,{
    'left':0,
    'top':0,
    'font-size':10,
    'line-height':10,
    'padding':4,
    'padding-right':4
});

cropSizeDisplay.appendChild(document.createTextNode('size'));

// 创建裁剪区域容器
var cropArea = document.createElement('div');
imageEditor.elements.cropArea = cropArea;
ADS.setStyle(cropArea,{
    'position':'absolute',
    'overflow':'hidden',
    'background-color':'transparent'
});

// 设置尺寸并更新显示盒子的大小
setNumericStyle(cropArea,{
    'left':0,
    'top':0,
    'width':image.width,
    'height':image.height
},true);

editor.appendChild(cropArea);

// 在剪裁区域中创建图像的副本
var resizeeClone = resizee.cloneNode(false);
imageEditor.elements.resizeeClone = resizeeClone;

cropArea.appendChild(resizeeClone);
cropArea.appendChild(cropSizeDisplay);

// 创建裁剪缩放手柄
var cropResizeHandle = document.createElement('div');
imageEditor.elements.cropResizeHandle = cropResizeHandle;
ADS.setStyle(cropResizeHandle,{
    'position':'absolute',
    'background':'transparent url(interface/handles.gif) ➥
no-repeat 0 0'
});
setNumericStyle(cropResizeHandle,{
    'right':0,
    'bottom':0,
```

```
        'width':18,
        'height':18
    });

    cropArea.appendChild(cropResizeHandle);

    // 创建保存手柄
    var saveHandle = document.createElement('div');
    imageEditor.elements.saveHandle = saveHandle;
    ADS.setStyle(saveHandle,{
        'position':'absolute',
        'background':'transparent url(interface/handles.gif) ➥
no-repeat -40px 0'
    });
    setNumericStyle(saveHandle,{
        'left':0,
        'bottom':0,
        'width':16,
        'height':18
    });

    cropArea.appendChild(saveHandle);

    // 创建取消缩放手柄
    var cancelHandle = document.createElement('div');
    imageEditor.elements.cancelHandle = cancelHandle;
    ADS.setStyle(cancelHandle,{
        'position':'absolute',
        'background':'transparent url(interface/handles.gif) ➥
no-repeat -29px -11px'
    });
    setNumericStyle(cancelHandle,{
        'right':0,
        'top':0,
        'width':18,
        'height':16
    });
    cropArea.appendChild(cancelHandle);
    ……省略的代码……
};
……省略的代码……
})();
```

如果此时重载页面并单击其中一幅图像，图像编辑器的标记就会按照代码中设计的那样，以恰当的定位显示出来。不过此时的图像编辑器还远未开发完成。下面还需要使用适当的事件侦听器，向其中添加所有拖动及单击的交互功能。

### 6.2.4 向 imageEditor 对象添加事件侦听器

当用户使用这个图像编辑器时，会因为拖动或单击不同的元素而引发以下不同的事件。

- 当用户的鼠标指针在不同的手柄图像上移过时，需要重新定位每个手柄的背景图像，以显示适当的鼠标悬停和鼠标移出状态。这有助于用户在使用不同的手柄时作出正确的判断。
- 当用户单击并拖动缩放手柄时，与图像相关的所有元素都需要保持相对的比例。静态方

法imageEditor.resizeMouseDown()、imageEditor.resizeMouseMove()和imageEditor.resizeMouseUp()将负责完成这些任务。

- 当用户单击并拖动裁剪手柄时，应该只影响裁剪<div>元素，而且裁剪手柄应该跟随指针移动。此外，当用户在裁剪区域内单击并拖动该区域时，还应该调整裁剪区域及其内部图像的位置。静态方法imageEditor.cropMouseDown()、imageEditor.cropMouseMove()和imageEditor.cropMouseUp()将会完成这些操作。
- 当单击取消[①]手柄时，需要将图像编辑器对象从文档中移除并重新设置适当的值。
- 当单击保存手柄时，会通过原始的<form>表单动作生成一个Ajax请求，以提交裁剪和缩放信息。就本例而言，我们在此将只添加一个粗略的方法，等到你在第7章学习完如何添加Ajax之后可以再回过头来完成它。

下面我们来完成imageEditor.imageClick()方法，请把下列事件侦听器添加到你刚才在该方法中创建的所有DOM元素的后面。这些侦听器中有不少现在什么也不会做，因为在其中指定的方法尚未完成，不过这只是开始：

```
(function(){
……省略的代码……
imageEditor.imageClick = function(W3CEvent) {
    ……省略的代码……
    // 向DOM元素添加事件

    // 缩放手柄的翻转图
    ADS.addEvent(resizeHandle,'mouseover',function(W3CEvent) {
        ADS.setStyle(this,{'background-position':'0px -29px'});
    });
    ADS.addEvent(resizeHandle,'mouseout',function(W3CEvent) {
        ADS.setStyle(this,{'background-position':'0px 0px'});
    });

    // 裁剪手柄的翻转图
    ADS.addEvent(cropResizeHandle,'mouseover',function(W3CEvent) {
        ADS.setStyle(this,{'background-position':'0px -29px'});
    });
    ADS.addEvent(cropResizeHandle,'mouseout',function(W3CEvent) {
        ADS.setStyle(this,{'background-position':'0px 0px'});
    });

    // 保存手柄的翻转图
    ADS.addEvent(saveHandle,'mouseover',function(W3CEvent) {
        ADS.setStyle(this,{'background-position':'-40px -29px'});
    });
    ADS.addEvent(saveHandle,'mouseout',function(W3CEvent) {
        ADS.setStyle(this,{'background-position':'-40px 0px'});
    });

    // 取消手柄的翻转图
    ADS.addEvent(cancelHandle,'mouseover',function(W3CEvent) {
        ADS.setStyle(this,{'background-position':'-29px -40px'});
    });
    ADS.addEvent(cancelHandle,'mouseout',function(W3CEvent) {
```

① 原文close有误。——译者注

www.TopSage.com

```
        ADS.setStyle(this,{'background-position':'-29px -11px'});
    });

    // 启动图像缩放事件流
    ADS.addEvent(resizeHandle,'mousedown',imageEditor.resizeMouseDown);

    // 启动裁剪区域拖动事件流
    ADS.addEvent(cropArea,'mousedown',imageEditor.cropMouseDown);

    // 启动裁剪区域缩放事件流
    ADS.addEvent(cropResizeHandle,'mousedown',function(W3CEvent) {
        imageEditor.info.resizeCropArea = true;
    });

    // 防止保存手柄启动裁剪拖动事件流
    ADS.addEvent(saveHandle,'mousedown',function(W3CEvent) {
            ADS.stopPropagation(W3CEvent);
    });

    // 在单击保存手柄或双击
    // 裁剪区域时保存图像
    ADS.addEvent(saveHandle,'click',imageEditor.saveClick);
    ADS.addEvent(cropArea,'dblclick',imageEditor.saveClick);

    // 防止取消手柄启动裁剪拖动事件流
    ADS.addEvent(cancelHandle,'mousedown',function(W3CEvent) {
            ADS.stopPropagation(W3CEvent);
    });

    // 在单击时取消改变
    ADS.addEvent(cancelHandle,'click',imageEditor.cancelClick);

    // 如果窗口大小改变则调整背景幕的大小
    ADS.addEvent(window,'resize',function(W3CEvent) {
        var windowSize = getWindowSize();
        setNumericStyle(backdrop,{
            'left':0,
            'top':0,
            'width':windowSize.width,
            'height':windowSize.height
        });
    });
};
……省略的代码……
})();
```

位于前面的一组实现翻转效果的事件侦听器都非常直观。其中的mouseover和mouseout侦听器只是重新定位了前面图6-7所示的背景图像的位置，从而创建出期望的翻转效果。

> 你可能会奇怪为什么不用JavaScript来修改类名，并通过CSS样式表创建翻转图或定位这些不同的元素。不错，你可这样实现，但是不要忘了这些元素的大小很大程度上要依赖于脚本操纵元素的大小来确定。你可以使用第5章介绍的方法动态载入一个样式表，也可以在主样式表中包含相应的样式。总之，如果你喜欢使用样式表，那么可以随意修改这个例子。

www.TopSage.com

## 6.2.5 缩放图像

下一组事件侦听器相对要麻烦一些，因为它们涉及了一点拖放逻辑，如图6-8所示。

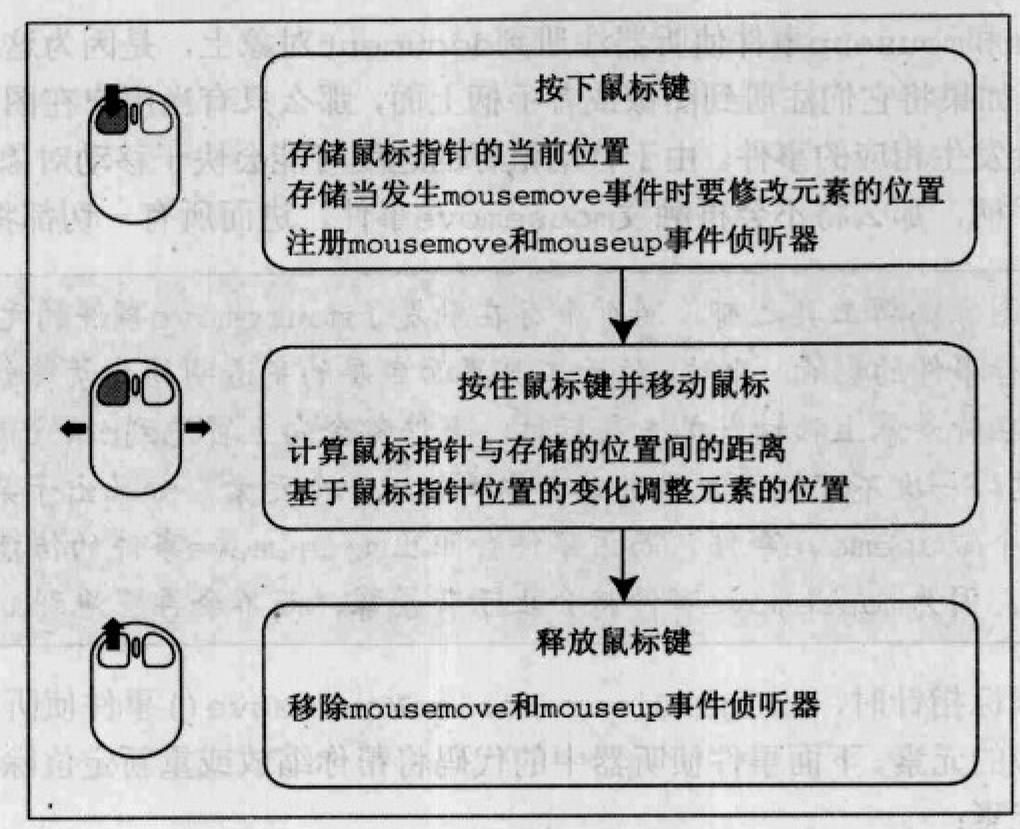



图6-8　拖动事件中涉及的步骤

首先，在imageEditor.resizeMouseDown()事件侦听器中，需要保存鼠标指针当前的位置以及所有后面需要改变的元素的大小。包括图像、图像的半透明蒙板、裁剪区域和所有相关手柄的大小。而且，imageEditor.resizeMouseDown()事件侦听器中还要在document对象上注册imageEditor.resizeMouseMove()和imageEditor.resizeMouseUp()事件侦听器：

```
(function(){
……省略的代码……
imageEditor.resizeMouseDown = function(W3CEvent) {

    // 保存当前位置和尺寸
    imageEditor.info.pointerStart = ADS.getPointerPositionInDocument(
        W3CEvent
    );
    imageEditor.info.resizeeStart = getDimensions(
        imageEditor.elements.resizee
    );
    imageEditor.info.cropAreaStart = getDimensions(
        imageEditor.elements.cropArea
    );

    // 注册其余事件以启用拖动
    ADS.addEvent(document,'mousemove',imageEditor.resizeMouseMove);
    ADS.addEvent(document,'mouseup',imageEditor.resizeMouseUp);

    // 停止事件流
    ADS.stopPropagation(W3CEvent);
```

www.TopSage.com

```
        ADS.preventDefault(W3CEvent);
    };
    ……省略的代码……
    })();
```

把mousemove和mouseup事件侦听器注册到document对象上，是因为这是保证事件始终会发生的唯一方式。如果将它们注册到图像或者手柄上面，那么只有当用户在图像或手柄之上移动或释放鼠标时才会发生相应的事件。由于移动鼠标的速度可能会快于移动对象的速度，所以如果鼠标指针超过了手柄，那么将不会再触发mousemove事件，进而所有一切都将停止。

> 如果在调用图像编辑工具之前，网页中存在触发了mousemove事件的元素，这些元素也不会受到以上鼠标事件的影响。此时，位于工具界面底层的半透明的背景幕就如同一个包罗万象的大容器。当在背景幕上移动或单击鼠标时，事件都会向上冒泡到document对象。我们在第4章曾经讨论过，一次不可能同时单击或者影响超过一个元素。如果出于某种原因，你向背景幕上添加了一个mousemove事件，而该事件会阻止mousemove事件的传播，那么可拖放的界面将不再工作，因为mousemove事件将会止于背景幕，而不会再冒泡到document对象。

当用户移动鼠标指针时，imageEditor.resizeMouseMove()事件侦听器将根据指针的当前位置来缩放相应的元素。下面事件侦听器中的代码将帮你缩放或重新定位除了编辑器盒子及背景幕之外每一个元素：

```
    (function(){
    ……省略的代码……
    imageEditor.resizeMouseMove = function (W3CEvent) {
        var info = imageEditor.info;

        // 取得当前鼠标指针所在位置
        var pointer = ADS.getPointerPositionInDocument(W3CEvent);

        // 基于鼠标指针来计算
        // 图像新的宽度和高度
        var width = (info.resizeeStart.width
            + pointer.x -info.pointerStart.x);
        var height = (info.resizeeStart.height
            + pointer.y -info.pointerStart.y);

        // 最小尺寸是42平方像素
        if(width < 42) { width = 42; }
        if(height < 42) { height = 42; }

        // 计算基于原始值的百分比
        var widthPercent = (width / info.resizeeStart.width);
        var heightPercent = (height / info.resizeeStart.height);

        // 如果按下了Shift键，则按比例缩放
        if(ADS.getEventObject(W3CEvent).shiftKey) {
            if(widthPercent > heightPercent) {
                heightPercent = widthPercent;
                height = Math.ceil(info.resizeeStart.height
                    * heightPercent);
            } else {
                widthPercent = heightPercent;
                width = Math.ceil(info.resizeeStart.width * widthPercent);
            }
```

www.TopSage.com

```
    }
    // 计算裁剪区域的新尺寸
    var cropWidth = Math.ceil(info.cropAreaStart.width
        * widthPercent);
    var cropHeight = Math.ceil(info.cropAreaStart.height
        * heightPercent);
    var cropLeft = Math.ceil(info.cropAreaStart.left
        * widthPercent);
    var cropTop = Math.ceil(info.cropAreaStart.top
        * heightPercent);

    // 缩放对象
    setNumericStyle(
        imageEditor.elements.resizee,
        {'width':width,'height':height}
    );
    setNumericStyle(
        imageEditor.elements.resizeeCover,
        {'width':width,'height':height}
    );
    setNumericStyle(
        imageEditor.elements.resizeHandle,
        {'left':(width - 18),'top':((height - 18))}
    );
    setNumericStyle(
        imageEditor.elements.cropArea,
        {'left':cropLeft,'top':cropTop,
        'width':cropWidth,'height':cropHeight},
        true
    );
    setNumericStyle(
        imageEditor.elements.resizeeClone,
        {'left':(cropLeft * -1),'top':(cropTop * -1),
        'width':width,'height':height}
    );

    // 停止事件流
    ADS.stopPropagation(W3CEvent);
    ADS.preventDefault(W3CEvent);
};
……省略的代码……
})();
```

你会发现在缩放元素的同时我们也添加了其他一些功能。一是当方形的尺寸收缩到42×42像素以下时，会由于被视为过小而停留在42像素的大小上。二是通过检查事件来发现用户是否按下了Shift键，如果是，则基于缩放之前图像的比例对图像进行缩放：

```
// 如果按下了Shift键，则按比例缩放
if(ADS.getEventObject(W3CEvent).shiftKey) {
    if(widthPercent > heightPercent) {
        heightPercent = widthPercent;
        height = Math.ceil(info.resizeeStart.height * heightPercent);
    } else {
        widthPercent = heightPercent;
        width = Math.ceil(info.resizeeStart.width * widthPercent);
    }
}
```

现在，拖动功能已经具备了，而且当鼠标移动时图像也会相应缩放，因此最后一步就是通过imageEditor.resizeMouseUp()事件侦听器来终止拖动。这里，只需移除imageEditor.resizeMouseMove()事件侦听器，所有元素就都会保持为修改后的状态。同时，还必须移除imageEditor.resizeMouseUp()事件侦听器，以防止mouseup事件与其他不用的事件混淆：

```
(function(){
……省略的代码……
imageEditor.resizeMouseUp = function (W3CEvent) {

    // 移除事件侦听器以停止拖动
    ADS.removeEvent(document,'mousemove',imageEditor.resizeMouseMove);
    ADS.removeEvent(document,'mouseup',imageEditor.resizeMouseUp);

    // 停止事件流
    ADS.stopPropagation(W3CEvent);
    ADS.preventDefault(W3CEvent);
};
……省略的代码……
})();
```

如果此时试验图像编辑器工具，你应该能够拖动缩放手柄并通过拖动来缩放图像。而与当前图像大小相同的裁剪区域，应该随着图像同步缩放。

### 6.2.6 裁剪图像

与裁剪相关的事件侦听器同缩放事件侦听器的运行原理大致相同，但也存在一个截然不同的差别：即通过imageEditor.imageClick()方法添加到裁剪区域中的3个手柄（cropResizeHandle、saveHandle和cancelHandle）都是cropArea元素的子元素。当单击这些手柄时，事件流的冒泡阶段将从手柄元素开始并传播到cropArea元素。虽然cropArea元素必须拥有能够使其在图像上方被拖动的事件侦听器，但我们不希望用户为缩放裁剪区域、保存改变或者取消改变而单击任何手柄时调用与鼠标移动相关的事件。

以cropResizeHandle为例，我们会通过mousedown事件将一个特殊的缩放标志设置为true：

```
imageEditor.info①.resizeCropArea = true;
```

然后，imageEditor.cropMouseMove事件侦听器会根据这个标志来确定用户是想要拖动裁剪区域（当标志值为false时），还是想要缩放裁剪区域（当标志值为true时）。

如果你回忆一下第4章中有关事件流和ADS.addEvent()方法的讨论，应该记得ADS.addEvent()只在冒泡阶段注册事件侦听器。当用户在裁剪缩放手柄上单击鼠标时，mousedown事件就会从裁剪缩放手柄的事件侦听器开始冒泡：

```
ADS.addEvent(cropResizeHandle,'mousedown',function(W3CEvent) {
    imageEditor.opts.resizeCropArea = true;
});
```

① 原文opts有误。——译者注

www.TopSage.com

然后该事件会通过祖先元素一直传播到裁剪区域自身。之所以会这样，是因为在cropResizeHandle上注册的匿名事件侦听器并没有对事件流的传播给予任何形式的阻止。当事件离开第一个事件侦听器之后，它又冒泡到在裁剪区域上注册的imageEditor.cropMouseDown事件侦听器中，并从那里开始继续冒泡。

请像下面这样将代码添加到imageEditor对象与裁剪相关的事件侦听器中，这些代码可以使裁剪区域可缩放而且可以被拖动：

```
(function(){
……省略的代码……
imageEditor.cropMouseDown = function(W3CEvent) {

    imageEditor.info.pointerStart = ADS.getPointerPositionInDocument(
        W3CEvent
    );
    imageEditor.info.cropAreaStart = getDimensions(
        imageEditor.elements.cropArea
    );

    // 包含缩放以限制裁剪区域的移动
    var resizeeStart = getDimensions(imageEditor.elements.resizee);
    imageEditor.info.maxX = resizeeStart.left + resizeeStart.width;
    imageEditor.info.maxY = resizeeStart.top + resizeeStart.height;

    ADS.addEvent(document,'mousemove', imageEditor.cropMouseMove);
    ADS.addEvent(document,'mouseup', imageEditor.cropMouseUp);

    // 停止事件流
    ADS.stopPropagation(W3CEvent);
    ADS.preventDefault(W3CEvent);
};

imageEditor.cropMouseMove = function(W3CEvent) {

    var pointer = ADS.getPointerPositionInDocument(W3CEvent);

    if(imageEditor.info.resizeCropArea) {

        // 缩放裁剪区域
        var width = (
            imageEditor.info.cropAreaStart.width
            + pointer.x
            -imageEditor.info.pointerStart.x
        );
        var height = (
            imageEditor.info.cropAreaStart.height
            + pointer.y
            -imageEditor.info.pointerStart.y
        );

        // 如果按下了Shift键，则按比例缩放
        // 计算基于原始值的百分比
        var widthPercent = (width
            / imageEditor.info.cropAreaStart.width);
        var heightPercent = (height
```

```
            / imageEditor.info.cropAreaStart.height);
        if(ADS.getEventObject(W3CEvent).shiftKey) {
            if(widthPercent > heightPercent) {
                heightPercent = widthPercent;
                height = Math.ceil(
                    imageEditor.info.cropAreaStart.height
                    * heightPercent
                );
            } else {
                widthPercent = heightPercent;
                width = Math.ceil(imageEditor.info.cropAreaStart.width
                    * widthPercent);
            }
        }

        // 检查新位置是否超出了边界
        if(imageEditor.info.cropAreaStart.left
            + width > imageEditor.info.maxX
        ){
            width = imageEditor.info.maxX
                -imageEditor.info.cropAreaStart.left;
        } else if(width < 36) {
            width = 36;
        }
        if(imageEditor.info.cropAreaStart.top
            + height > imageEditor.info.maxY
        ){
            height = imageEditor.info.maxY
                -imageEditor.info.cropAreaStart.top;
        } else if(height < 36) {
            height = 36;
        }

        setNumericStyle(
            imageEditor.elements.cropArea,
            {'width':width,'height':height},
            true
        );
    } else {

        // 移动裁剪区域
        var left = (
            imageEditor.info.cropAreaStart.left
            + pointer.x
            -imageEditor.info.pointerStart.x
        );

        var top = (
            imageEditor.info.cropAreaStart.top
            + pointer.y
            -imageEditor.info.pointerStart.y
        );

        // 检查新位置是否超出了边界
        // 如有必要则加以限制
        var maxLeft = imageEditor.info.maxX
            -imageEditor.info.cropAreaStart.width;
```

www.TopSage.com

```
        if(left < 0) { left = 0; }
        else if (left > maxLeft) { left = maxLeft; }

        var maxTop = imageEditor.info.maxY
            -imageEditor.info.cropAreaStart.height;

        if(top < 0) { top = 0; }
        else if (top > maxTop) { top = maxTop; }

        setNumericStyle(
            imageEditor.elements.cropArea,
            {'left':left,'top':top}
        );
        setNumericStyle(
            imageEditor.elements.resizeeClone,
            {'left':(left * -1),'top':(top * -1)}
        );
    }
    // 停止事件流
    ADS.stopPropagation(W3CEvent);
    ADS.preventDefault(W3CEvent);
};

imageEditor.cropMouseUp = function(W3CEvent) {
    // 移除所有事件
    var eventObject = ADS.getEventObject(W3CEvent);
    imageEditor.info.resizeCropArea = false;

    ADS.removeEvent(document,'mousemove', imageEditor.cropMouseMove);
    ADS.removeEvent(document,'mouseup', imageEditor.cropMouseUp);

    // 停止事件流
    ADS.stopPropagation(W3CEvent);
    ADS.preventDefault(W3CEvent);
};
……省略的代码……
})();
```

同以前一样，我们将裁剪区域的大小限制在了外部图像的边界之内（将裁剪区域拖动到图像外面没有什么意义），而且，仍然以Shift键作为信号来维持缩放时的比例。

最后两个注册在saveHandle和cancelHandle元素上面的mousedown事件侦听器被阻止了传播，因此cropArea元素上的相应事件侦听器将忽略这两个事件。此外，saveHandle和cancelHandle元素也都有click事件侦听器。不过在这里，imageEditor.cancelClick事件侦听器只是简单地调用imageEditor.unload方法，删除构成图像编辑器的所有DOM元素。而当前的imageEditor.saveClick事件侦听器则什么也不做，因为我们还需要为此再创建适当的服务器端交互程序：

```
(function(){
……省略的代码……
imageEditor.saveClick = function(W3CEvent) {
    // 此处只是发出一个警告
```

www.TopSage.com

```
        alert('This should save the information back to the server.');

        // 如果成功则卸载编辑器
        imageEditor.unload();
    };

    imageEditor.cancelClick = function(W3CEvent) {
        if(confirm('Are you sure you want to cancel your changes?')) {
            // 卸载编辑器
            imageEditor.unload();
        }
    };
    ……省略的代码……
})();
```

### 6.2.7 未完成的图像编辑器

到现在为止，我们的图像编辑器开发就暂时告以段落了。你已经为此构建了良好的结构，在此基础上还可以继续扩充和添加新的功能。比如，可以通过imageEditor.saveClick()方法修改已有的表单，然后提交页面。或者，在学习完下面几章之后，再回过头来对它进一步加以改进，例如使用Ajax交互技术把修改保存到服务器。

> 可以从本书网站http://advanceddomscripting.com中下载缩放图像的服务器端脚本。

## 6.3 小结

在本章中，虽然没有为ADS库添加任何新方法，但却在建立图像编辑器的基础结构过程中发挥了它应有的作用。尽管图像编辑器本身的构思非常巧妙，但本章的关键，是为了向你展示如何从一个具有可访问性的而且与DOM脚本无关的文档开始，综合运用现有的库和一些DOM脚本来增强该文档。而同样的思想也适用于你所开发的Web应用程序的方方面面，只不过你必须通过自己大胆的想象找到一些有创意的想法。

接下来，在本书第二部分中，我们会探索Ajax以及浏览器与服务器的通信。在此之后，你可以再回到本章的例子中，继续完成图像编辑器的saveClick()方法，以便它能够将信息传送回服务器。

www.TopSage.com

# Part 2 第二部分

# 浏览器外部通信

本部分内容

■ 第 7 章　向应用程序中加入 Ajax
■ 第 8 章　案例研究：实现带进度条的异步文件上传功能

www.TopSage.com

第7章

# 向应用程序中加入Ajax

当谈论Web技术时，如果离开Ajax、JavaScript、Web 2.0、标准、退化能力或者渐进增强这些主题，很难想象还有什么可说的。把后两个概念（退化能力和渐进增强）放到其中，是因为我希望它们能得到更多的认同，而不是经常被忽视。除此之外，另一个老生常谈的混合搭配技术，也就是我们现在称之为Ajax（Asynchronous JavaScript and XML，异步JavaScript和XML）技术的出现，虽然不一定是件坏事（因为它体现了我们赖以进步的某种协作思想），但我们也不应该因此喜新厌旧，把刚刚学过的那些东西抛到九霄云外去。

将JavaScript和XML技术结合起来形成一种新技术如Ajax的问题在于，它经常会使人们忘记其本来面目，即Ajax只是现有技术的一种整合。在Jesse James Garrett给它起这个好记的名称之前，Ajax并不是什么新鲜事物——Google等一些公司早已开始使用它了，但这个名称则是随着一些花哨的例子，如Google Suggest（http://www.google.com/webhp?complete=1&hl=en）和Google Maps（http://maps.google.com）一起流行起来的。而且，它的流行也具备充分的理由，即通过Ajax可以使用户的体验得到某些难以置信的增强。

即使都是一些已有的东西，但Ajax还是成为了一种新技术。不过因而，它也同样保留了原有技术中固有的跨浏览器及技术性的不兼容问题。事实上，你将在本章中看到，Ajax背后的异步思想以及这种技术的实现方式又开启了一个全新的问题领域。

当然，千万不要误解——Ajax的普及及其背后的思想是非常值得称道的，而且我也完全支持人们致力于创建充满奇思妙想的Ajax应用程序。但是，如果执行出现偏差或者没有事先规划就贸然行动，即使是一件好事也可能被做得严重走样。

在本章中，我们将重温一下`XMLHttpRequest`对象，并介绍一些你可能还不知道的新东西。而且，你还会看到在实现Ajax驱动的界面时最常遇到的一些问题以及相应的解决方案（即使我们的解决方案意味着使用一种更加传统的通信流）。

## 7.1 组合的技术

适当的Ajax是对已有技术和下列思想的组合：

- 语义化(X)HTML标记
- 文档对象模型（DOM）

www.TopSage.com

- JavaScript
- XML

之所以说“适当的Ajax”，是因为这4个组成部分中的几乎任何一个都可能由于被替代物所替代而导致更好或者更差的结果。

### 7.1.1 语义化 XHTML 和 DOM

对于Ajax而言，在文档中使用有效的语义正确的(X)HTML不是必需的。任何旧式HTML都能够在浏览器中得到很好的呈现。语义化的标记只是能够使DOM脚本编程更容易而已。由于大多数情况下，你都将使用第3章中介绍的DOM方法来基于Ajax请求的结果操纵标记和文档结构，所以，如果你能做到标记语义化，同时为各种元素使用适当的标签并指定类名和ID，那么你在识别和操纵文档的相应内容时就会更轻松一些。而且，由适当标记构成的文档通常会比其非语义化的版本更加清晰，复杂性也更低。这也就意味着当你的脚本在文档结构中导航遍历时会经历更少的周折。

### 7.1.2 JavaScript 和 `XMLHttpRequest` 对象

所有Ajax请求的核心都是JavaScript中的`XMLHttpRequest`对象。虽然W3C针对`XMLHttpRequest`对象有一份工作草案（http://www.w3.org/TR/XMLHttpRequest），但该草案并没有纳入当前的任何标准或规范中。Microsoft IE 5最早以ActiveX插件的形式引入了这一思想。是的，没错，从Microsoft IE 5时代起就可以进行Ajax应用的开发了。而要使用这个ActiveX插件，必须使用IE的`Microsoft.XMLHTTP` AcitveX组件来实例化一个对象：

```
var request = new ActiveXObject("Microsoft.XMLHTTP");
```

从那时起，其他浏览器厂商（包括Microsoft，直到IE 7）也都实现了具有相同功能的`XMLHttpRequest`对象（不过在IE 7中仍然可以使用原来的ActiveX对象）：

```
var req = new XMLHttpRequest();
```

在不同的浏览器之间，公共的`XMLHttpRequest`方法包括：

- `open(method, URL[, asynchronous[, userName[, password]]])`用于指定请求URL、方法以及与请求相关的其他可选属性。
- `setRequestHeader(label, value)`用于以给定的`label`和`value`为请求应用一个头部信息。该方法必须在请求的`open()`方法之后且在`send()`方法之前调用。
- `send(content)`用于发送请求，可以包含可选的内容，例如`POST`请求的内容等。
- `abort()`用于停止当前的请求。
- `getAllResponseHeaders()`返回字符串形式的完整的头部信息集合。
- `getResponseHeader(label)`返回指定头部的一个单独的字符串值。

在Prototype和jQuery之类的库中，Ajax对象通常还包含其他一些方法，但这些方法都是以上方法或者请求结果的包装方法。在第9章中，你会看到其中一些库的实际应用，而在本章后面，你将创建添加到ADS库中的你自己的Ajax对象。

www.TopSage.com

#### 1. 生成一次新请求

要生成一次请求，需要通过浏览器能力检测来实例化一个新对象，并调用open()和send()方法：

```
function stateChangeListener() {
    // 某些代码
}

var request = false;
if(window.XMLHttpRequest){
    var request = new window.XMLHttpRequest();
} else if (window.ActiveXObject) {
    var request = new window.ActiveXObject('Microsoft.XMLHTTP');
}
if(request) {
    request.onreadystatechange = stateChangeListener;
    request.open('GET', '/your/script/?var=value&var2=value', true);
    request.send(null);
}
```

Microsoft IE 7也引入了本地的XMLHttpRequest对象。而这是展示适当的能力检测（在第1章讨论过）有效性的一个例子。假如你使用浏览器检测来检查IE并使用其ActiveX版的组件，那么新增的本地对象就会被忽略。首先检查XMLHttpRequest对象，则可以让任何浏览器（包括IE 7）都使用适当的对象。

这个例子发送了一个GET请求，而请求URL的后面还附加了几个必需的变量：

```
request.open('GET', '/your/script/?var=value&var2=value', true);
```

对于GET请求而言，send()方法的参数应该是null。如果要执行一次POST请求，那么应该将POST作为open()的方法参数，并且通过使用send()方法在请求主体中包含变量，而不是将变量包含在URL中：

```
request.open('POST', '/your/script/', true);
request.send('var=value&var2=value');
```

遗憾的是，或者从安全角度讲幸运的是，由于浏览器施加限制，JavaScript不能访问用户系统中的文件。这意味着将无法使用file输入框，通过包含选中的文件来执行一次经过多重加密后的POST请求。在第8章中，你将看到围绕这一主题的几种不同解决方案。

#### 2. 处理响应

在前面的例子中，你会注意到为请求对象的onreadystatechange属性指定了stateChangeListener()方法。无论你为该属性指定什么方法，这个方法都将在请求的不同阶段被调用几次。这个方法可以让DOM脚本通过访问下列XMLHttpRequest对象的属性来与请求进行交互。

- readyState是一个表示下列状态的整数值。
  - 0: 尚未初始化
  - 1: 载入中

- 2: 载入完成
- 3: 交互
- 4: 完成

- responseText是一个在响应中返回的数据的字符串表示。
- responseXML是一个兼容DOM核心的文档对象（在响应是一个有效的XML文档并且设置了适当的头部信息的情况下）。
- status是一个表示请求状态的数字代码。这些数字代码是由服务器生成的HTTP协议状态代码（http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html），例如：404表示“Not Found”，200表示“OK”。
- statusText是与状态代码相关的一条信息。
- onreadystatechange应该包含在请求的不同readyState状态下被调用的方法。

使用指定给onreadystatechange属性的方法以及其他的请求属性，可以判断请求的状态（无论是否成功），以及从服务器返回的响应是什么类型。下面就是一个典型的onreadystatechange方法：

```
function stateChangeListener() {
    // 根据请求的状态转换功能
    switch (request.readyState) {
        case 1:
            // 载入中
            break;
        case 2:
            // 载入完成
            break;
        case 3:
            // 交互
            break;
        case 4:
            // 完成
            if (request.status == 200) {
                // 对request.responseText
                // 或request.responseXML进行处理
            } else {
                // request.status中可能包含某个错误代码
                // 而request.statusText中则包含报告的错误信息
            }
            break;
    }
}
```

> 这个指定给onreadystatechange属性的方法不接收任何参数。为了访问请求对象并取得readyState和status的值，需要引用作用域链中的请求对象。在这个例子的stateChangeListener()方法中，请求变量引用XMLHttpRequest对象的实例，而且请求变量必须在与stateChangeListener()方法相同的作用域中定义。

在任何情况下，如果请求完全成功，则responseText属性将被响应返回的一个字符串填充。

www.TopSage.com

但如果响应的Content-Type头部信息是application/xml，并且响应是一个有效的XML文档，那么responseXML属性中将会保存一个XML DOM文档。这种DOM表示法的用处在于，你可以使用第3章介绍的所有DOM2核心方法遍历并读取responseXML中保存的文档。

> 假如你希望使用XML格式，就必须将响应的Content-Type头部信息设置为application/xml；否则，即使响应是一个有效的XML文档，XMLHttpRequest对象仍然会假定响应中只包含文本。同样也必须记住，responseXML中保存的文档将被解释为XML文档，而非HTML文档，因此DOM2 HTML规范中规定的所有方法都不适用，即使文档包含有效的HTML代码也是如此。

当通过XMLHttpRequest对象发起请求时，需要牢记的重要一点是，作为请求结果而采取的操作必须通过onreadystatechange侦听器来调用。如果你之前使用过Ajax，那么有可能会写过类似下面这样的代码：

```
……省略的代码……
request.open('GET', '/yourscript/?var=value', true);
request.send(null);
alert(request.responseText);
```

但是这些代码并不能按照计划运行，因为发起请求的模式是异步的（第3个参数为true）。当XMLHttpRequest对象处于异步模式时，请求会被异步发送，因此alert()函数可能会在响应返回之前被执行（显示一个空值）。

> 无论你看过什么例子，我都强烈建议你始终将第3个参数设置为true，并且适当地处理响应的结果。通过指定true值，就可以使请求处于异步模式而非阻塞（blocking）模式。在异步模式下，脚本在执行完send()命令到从服务器返回响应之前会继续执行。但在阻塞模式下，脚本会停止并一直等到响应返回后再继续执行。使用异步模式不仅能够消除意外挂起脚本的可能性，而且还能在连接中存在延迟的情况下，避免浏览器功能丧失。不过，异步模式也会带来一些其他的问题，这些问题在本章后面我们更详细地讨论异步请求时会涉及。

如果你希望脚本中剩余的操作等到请求完成后再执行，那么它们就应该在响应返回后通过onreadystatechange方法被执行：

```
// 创建请求对象
……省略的代码……

// 创建你希望运行的方法
function alertAndDoWhatever(r) {
    // r是通过onreadystatechange
    // 侦听器传递进来的请求对象
    alert(r.responseText);
    // 继续进行其他操作
}
request.onreadystatechange = function() {
    // 当请求完成时运行相应的方法
    if(request.readyState == 4 && request.status == '200') {
```

www.TopSage.com

```
            // 请求已经成功完成，需要调用你创建的方法
            alertAndDoWhatever(request);
        }
    }
    // 启动请求
    request.open('GET', '/yourscript/?var=value', true);
    // 发送请求
    request.send(null);
```

当在指定给onreadystatechange属性的方法内部使用this关键字时，this引用方法自身，而不是XMLHttpRequest对象。在前面例子中，按照我们在第1章和第2章的讨论，对匿名的onreadystatechange方法中的request进行求值的结果，将是作用域链中request的引用。而且，alertAndDoWhatever()方法也可以通过相同的方式来引用request，因为该方法是在作用域链中相同的位置被定义的。不过这里是将request作为参数传递给了该方法。在本章后面，当你创建ADS.ajaxRequest对象时，将会以this关键字引用request而不是引用方法本身的方式来应用该方法。

### 3. 在服务器上识别Ajax请求

Ajax请求没有什么特别之处。在服务器端看来，Ajax请求与其他任何请求实际上都是一样的。为了简单起见，可以通过XMLHttpRequest对象发送一个特殊的头部信息，以便在服务器上识别该请求。

在一次典型的XMLHttpRequest事务中，浏览器发送的请求中会包含多种头部信息，例如：

```
Host=advanceddomscripting.com
User-Agent=Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; ➥
en-US; rv:1.8.1.1)
Gecko/20061204 Firefox/2.0.0.1
Accept=text/xml,application/xml,application/xhtml+xml,text/html;q=0.9, ➥
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language=en-us,en;q=0.5
Accept-Encoding=gzip,deflate
Accept-Charset=ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive=300
Connection=keep-alive
Referer=http://advanceddomscripting.com/example/example.html
```

为了标识来自XMLHttpRequest对象的请求，可以使用请求对象的setRequestHeader()方法来发送自定义的头部信息。可以在该方法中指定任何头部信息，但却只能在启动请求事务之后并且在调用send()方法之前调用该方法：

```
request.open('GET', '/yourscript/?var=value', true);
request.setRequestHeader('My-Special-Header','AjaxRequest');
request.setRequestHeader('Sent-By','Jeff');
request.send();
```

经过这样设置以后，执行的请求中将会包含更多的头部信息：

```
Host=advanceddomscripting.com
User-Agent=Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; ➥
en-US; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1
Accept=text/xml,application/xml,application/xhtml+xml,text/html; ➥
q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
```

www.TopSage.com

```
Accept-Language=en-us,en;q=0.5
Accept-Encoding=gzip,deflate
Accept-Charset=ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive=300
Connection=keep-alive
My-Special-Header=AjaxRequest
Sent-By=Jeff
Referer=http://advanceddomscripting.com/example/example.html
```

然后，就可以在服务器端检查特殊的头部信息并采取相应的操作。在PHP中，这一过程简单到只需从全局$_SERVER数组中读取头部信息：

```
<?php
if (isset($_SERVER['HTTP_MY_SPECIAL_HEADER'])) {
    // 对XMLHttpRequest对象的响应
} else {
    // 对传统请求的响应
}
?>
```

> 在PHP中，头部信息被保存在关联的$_SERVER全局数组中，而头部信息的名称对应着数组的键。键的格式会在名称基础上稍微有所改动，名称的文本会转换为大写，而所有连字符(-)都会转换成下划线（_），正如前面例子中所示。同时，每个表示头部信息的键还以HTTP作为前缀来表示该信息来自HTTP请求。

同理，通过请求对象的getResponseHeader()方法可以从响应中取得一个特殊的头部信息，因而可以基于响应中的信息（如Content-Type）来修改脚本：

```
switch(request.getResponseHeader('Content-Type')) {
    case 'text/javascript':
        // 按request.responseText中包含JavaScript代码处理
        break;
    case 'text/xml':
    case 'application/xml':
        // 使用request.responseXML或将request.responseText读作XML
        break;
    case 'text/html':
        // 按request.responseText中包含HTML代码处理
        break;
}
```

取得哪种Content-Type取决于服务器的响应内容。如果服务器返回XHTML而不是HTML，那么Content-Type可能是application/xhtml+xml。而且，Content-Type头部信息中也可能包含字符编码：

```
Content-Type: text/html; charset=ISO-8859-4
```

而这会导致前面的切换代码失效，因为该代码只查找text/html。

也可以使用这种方法来验证服务器是否返回了适当的响应。比如按照如下所示发送带自定义头部信息的请求：

```
request.onreadystatechage = processSpecialRequest;
request.open('GET', '/yourscript/?var=value', true);
```

www.TopSage.com

```
request.setRequestHeader('My-Ajax-Request','SpecialValue');
request.send();
```

并在响应中的另一个自定义的头部信息中使用相同的值：

```
<?php
if (isset($_SERVER['HTTP_MY_AJAX_REQUEST'])) {
    header('My-Ajax-Response: '.$_SERVER['HTTP_MY_AJAX_REQUEST']);
    echo 'Hello';
}
?>
```

那么，请求的onreadystatechange侦听器可以在进一步处理之前检查响应中是否包含有效的头部信息：

```
function processSpecialRequest(request) {
    if(this.readyState == 4) {
        var header = request.getResponseHeader('My-Ajax-Response');
        if(header == 'SpecialValue') {
            // 服务器响应中包含了你发送的预期值
            alert(request.responseText);
        } else {
            // 服务器响应了其他的值
        }
    }
}
```

如果你想看到附加在请求中的输出和输入的头部信息，我建议你在Firefox（http://getfirefox.com）中使用Firebug（http://getfirebug.com）插件。如图7-1所示，你可以从中看到浏览器生成的XMLHttpRequest请求所包含的全部输出和输入的头部信息。

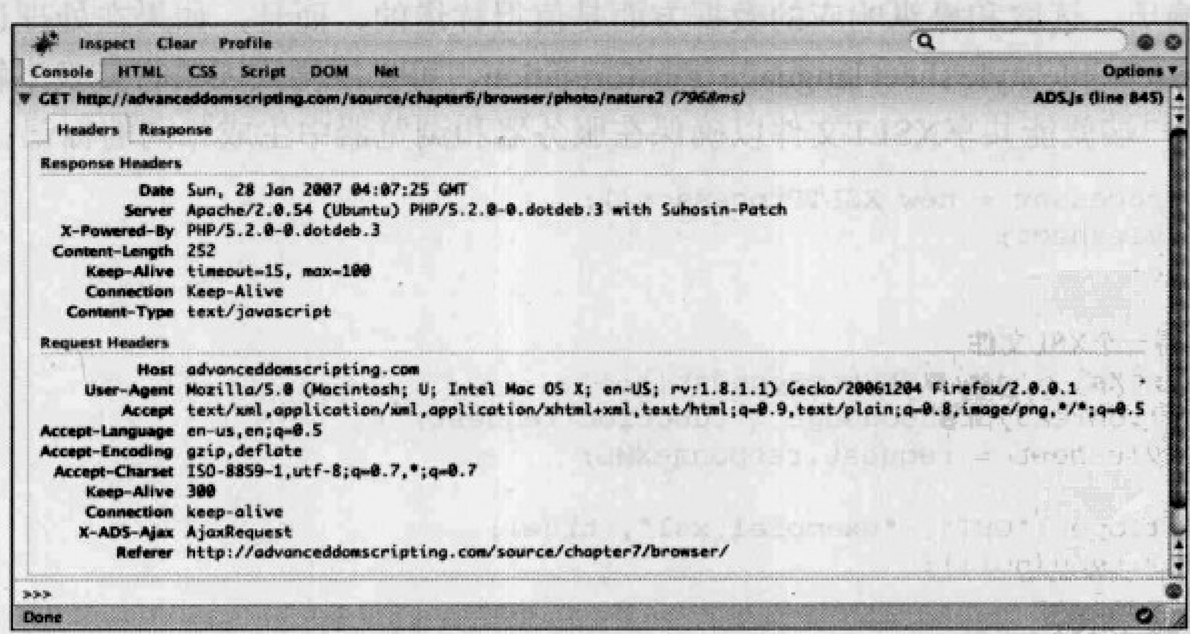



图7-1 当在Firebug中查看Ajax照片浏览器程序时显示的一个XMLHttpRequest请求所包含的输入和输出的头部信息

#### 4. 不限于GET和POST

XMLHttpRequest对象的最后一个也是经常被忽略的特性，就是它发送除了GET和POST之外的请求的能力。open()方法可以接受如下方法：

- GET，用于从服务器取得头部信息及其他信息的请求。
- POST，用于修改服务器上信息的请求。
- HEAD，用于与GET相同的请求，但HEAD中只包含与请求关联的头部信息，而不包含请求的主体。
- PUT，用于希望在服务器上某个特殊位置存储信息的请求。
- DELETE，用于希望删除服务器上的文件或资源的请求。
- OPTIONS，用于列出服务器上可用选项的请求。

支持何种请求类型取决于你从中请求信息的服务器。大多数服务器都支持GET、POST和HEAD请求，而PUT、DELETE和OPTIONS请求则通常被视为等同于GET请求。

典型的GET或POST请求返回的所有信息如果并非全都需要，可以使用以上替代方法。例如，如果只想检查服务器上一个文件的状态，而并不想实际取得该文件，那么就可以使用HEAD方法：

```
request.onreadystatechange = function() {
    if(this.readyState == 4) {
        alert(this.status);
    }
}
request.open("HEAD", '/some/file.pdf');
request.send(null);
```

不过，这样可能会因为网络原因，而不是因为缺少文件而导致不明确的响应，甚至发生错误。

### 7.1.3 XML

Ajax混合技术中的最后一部分是XML响应。作为一种数据传输机制，XML在允许开发者从DOM层次上遍历、读取和操纵响应的数据方面是值得称道的。而且，如果在处理过程中整合了一个XSLT（extensible style sheet language transformation，可扩展样式表语言转换）解析机制，那么服务器和客户端就能共享XSLT文件以确保在服务器和浏览器中生成相同的标记：

```
var xsltProcessor = new XSLTProcessor();
var xslStylesheet;
var xmlDoc;

// 异步取得一个XSL文件
var requestXsl = new XMLHttpRequest();
requestXsl.onreadystatechange = function(request) {
    xslStylesheet = request.responseXML;
}
requestXsl.open("GET", "example1.xsl", true);
requestXsl.send(null);

// 异步取得一个XML文件
var requestXml = new XMLHttpRequest();
requestXml.onreadystatechange = function(request) {
    xmlDoc = request.responseXML;
}
requestXml.open("GET", "example1.xml①", true);
```

① 原文.xsl有误。——译者注

```
requestXml.send(null);


var processor = function() {
    if(xslStylesheet && xmlDoc) {
        clearInterval(this);
        // 通过XSLT转换XML
        xsltProcessor.importStylesheet(xslStylesheet);
        var fragment = xsltProcessor.transformToFragment(
            xmlDoc, document
        );
        ADS.$('example').appendChild(fragment);
    }
}
// 每200毫秒检查一次文件是否载入完成
setInterval(processor,200);
```

正如我们前面所提到的，XML意味着可以对XML响应使用DOM2核心方法：

```
var messages = request.responseXML.getElementsByTagName('messages');
for(var i=0 ; i < messages.length ; i++) {
    ADS.$('example').appendChild(messages[i]);
}
```

不过，XML也有一些不足的地方。通常，围绕数据的XML标记都将是响应中的大部分内容，而在高通信流量的情况下使用体积更小的方法会提高效率。而且，在跨浏览器的环境中处理XSLT也是有问题的。如果你在使用XML时遇到了问题，或者它不具有什么优势，那么可以考虑使用其他替代方案。不过在选择时一定要认真谨慎，因为错误的方案有可能会导致应用程序的品质降低。

#### 1. 纯文本

对于确实简单明了的请求，可以只返回未经格式化或任何特殊处理的纯文本字符串。而发送纯文本作为响应的问题是缺少元数据。如果是在一个句子中返回信息，那么没有任何方法能够指示信息发送成功还是出现了错误。但是，如果只需要取得表示是或否的响应，则可以响应true或false，甚至只返回t或f，然后再据此采取措施：

```
request.onreadystatechange = function() {
    if(this.readyState == 4 && request.status == 200) {
        if(request.responseText == 't') {
            // 服务器处理成功
        } else {
            // 服务器处理失败
        }
    }
}
```

#### 2. HTML

另一个常见的选择是以常规的HTML代码作为响应：

```
<p class="success">The response was successful</p>
```

这样就具备了通过元素属性添加元数据的能力，而且还能使用innerHTML属性将responseText中的代码插入到文档中：

www.TopSage.com

```
request.onreadystatechange = function() {
    if(request.readyState == 4 && request.status == 200) {
        ADS.$('example').innerHTML = request.responseText;
    }
}
```

在实践中，这样做并不是一个好习惯，原因如下：

- 与XML相似，响应可能会因为标记而变得无谓的臃肿，而那些标记都可以通过其他方式提取并放到可重用的JavaScript对象中。
- 作为一个字符串，HTML不能为脚本提供指向其内部元素的任何直接关联。必须在通过innerHTML属性将HTML插入到文档后，使用document.getElementById()或类似方法创建相应的关联。
- 以HTML作为响应不能使应用程序中的各个层之间实现很好的分离。因为不能把服务器或脚本的功能重用于多个实例，除非这些实例都使用同样的标记和方法。而保持标记尽可能多的本地化并且只在响应中发送必要的信息，然后再通过DOM方法将这些信息转换为适当的结构则是更好的方式。
- 在IE中，innerHTML属性在处理表格和选择列表时也存在问题。如果不使用它们作为目标容器当然没有问题，但这也意味着限制了DOM脚本的功能。

在某些情况下，HTML可能是达成你期望结果的最简便快捷的方式，但同时请求也会变得臃肿而且对用户不够友好。所以，为了尽可能快地得到响应，XMLHttpRequest对象应该尽可能地精简。

> 如果响应的是XHTML并且具有适当的Content-Type信息，则responseXML中会生成该HTML的一个DOM文档表示。不过，要记住的是此时生成的DOM文档仅适用于DOM2核心（而不是DOM2 HTML），也就是说对responseXML中的DOM文档无法使用HTML元素中的许多属性。

### 3. JavaScript代码

作为另外一个选项，可以在响应中传输任何JavaScript代码。这里我指的并非JSON（下一小节会谈到JSON），而是指类似如下调用简单的alert()函数的完整脚本或方法：

```
alert('The response was successful.');
```

对此，可以通过在responseText属性上调用eval()方法来执行相应的代码：

```
request.onreadystatechange = function() {
    if(request.readyState == 4 && request.status == 200) {
        eval(request.responseText);
    }
}
```

使用JavaScript也会导致一些问题。首先，使用eval()被认为是一种坏习惯，因为一不小心就可能造成安全隐患。比如，恶意用户可能会利用你的Ajax界面输入信息，而对这些信息求值后，则会执行恶意代码。其次，之所以使用JavaScript不是一个好主意，还因为与使用HTML一样的另一个原因，即JavaScript将因此分散在应用程序不同部分的一些文件中。而JavaScript应该出现且仅出现在一个地方，即包含在文档头部的.js文件中。

www.TopSage.com

### 4. JSON

最后一个替代方案是只使用JSON（JavaScript Object Notation，JavaScript对象表示法）。本质上说，JSON语法是JavaScript对象字面量表示法的一个子集，因此可以说你对它已经不陌生了。要了解有关JSON语法的详细信息，请参考http://json.org。不过，一般而言在响应中使用JSON对象时都意味着返回一个简单的JavaScript对象：

```
{
    message : 'The response was successful',
    type : 'success'
}
```

随后，可以使用eval()来将JSON解析为本地JavaScript对象，从该对象中再取得任何想要的信息：

```
request.onreadystatechange = function() {
    if(this.readyState == 4 && request.status == 200) {

        // 对JSON求值以生成response对象
        var response = eval('(' + this.responseText + ')');

        // 通过innerHTML对response执行某些操作
        ADS.$('example').innerHTML = '<p class="'
            + response.type
            + '">'
            + response.message
            + </p>;

        // 或者直接对用户给出提示
        alert(response.message);

        // 或使用DOM方法
        var p = document.createElement('P')
        p.className = response.type;
        p.appendChild(document.createTextNode(response.message));
        ADS.$('example').appendchild(p);

    }
}
```

使用eval()解析JSON虽然快捷而简单，但同样也存在不安全的问题，所以这比使用常规的JavaScript代码和eval()也强不到哪去。因为恶意代码很容易被包含在不规范的JSON对象中，从而在对JSON求值时被执行。为了解决这个问题，最好的办法就是使用一种只识别有效JSON语法的解析程序，即将JSON的清晰简单与更加安全的获取方法结合起来。解析程序一旦发现了不规范的对象，应该直接中断或者根本不执行其中的恶意代码。

如果你觉得编写JSON解析程序有点困难，不用担心。你可以从http://www.json.org/json.js（此链接指向JavaScript版）[①]免费下载到公众域（Public Domain）的解析程序。在下面要创建的可重

---

① 此版本已经被2007年11月6日发布的json2.js所取代，故新链接为http://www.json.org/json2.js。——译者注

www.TopSage.com

用的Ajax对象中，包含了一个在json.org提供的parseJON()方法基础上的改进版。如果你使用JSON的频率很高，那么也可能需要包含其他一些方法。

> 至于是使用eval()还是JSON解析程序当然取决于你自己。如果你能够对响应中的信息做到完全控制（并且不会被你的用户所修改），那么使用eval()大概是最好的选择。从另一方面来说，虽然使用解析程序要比使用eval()方法稍慢一些，但却能够以少许的性能损失换得可靠的安全保障。

总而言之，在响应中使用JSON对象是一种明智的选择，因为它不仅能够包含相应的信息，而且还可以在附加标记最少化的条件下包含任何元数据。而响应侦听器则可以在随后以任何适当的方式对响应内容进行转换。这样，服务器端只需负责响应信息，而对响应的解释工作则交给了你的DOM脚本完成。

### 7.1.4　一个可重用的对象

为了使处理XMLHttpRequest对象更加容易，请在你的ADS库中创建下列ADS.getRequest-Object()和ADS.ajaxRequest()对象。其中，getReqeustObject()方法只是用于设置XMLHttpRequest对象及其onreadystatechange侦听器的一个辅助方法：

```
(function(){

window['ADS'] = {};

……以上是库中已有的内容……

/*
parseJSON(string,filter)
这是一个在公共域方法http://www.json.org/json2.js①
基础上进行了少量修改的版本。该方法解析JSON文本
以生成一个对象或数组。它可能抛出SyntaxError异常
*/
function parseJSON(s,filter) {
    var j;

    function walk(k, v) {
        var i;
        if (v && typeof v === 'object') {
            for (i in v) {
                if (v.hasOwnProperty(i)) {
                    v[i] = walk(i, v[i]);
                }
            }
        }
        return filter(k, v);
    }
```

① 原文中http://www.json.org/json.js已经被http://www.json.org/json2.js取代了。——译者注

www.TopSage.com

```
// 解析通过3个阶段进行。第1阶段，通过正则表达式
// 检测JSON文本，查找非JSON字符。其中，特别关注
// “()”和“new”，因为它们会引起语句的调用，还有“=”，
// 因为它会导致变量的值发生改变。不过，为安全起见
// 这里会拒绝所有不希望出现的字符

    if (/^("(\\.|[^"\\\n\r])*?"|[,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t])+?$/.
                test(s)) {

// 第2阶段，使用eval函数将JSON文本编译为JavaScript
// 结构。其中的“{”操作符具有语法上的二义性：即它可
// 以定义一个语句块，也可以表示对象字面量。这里将
// JSON文本用括号括起来是为了消除这种二义性

        try {
            j = eval('(' + s + ')');
        } catch (e) {
            throw new SyntaxError("parseJSON");
        }
    } else {
        throw new SyntaxError("parseJSON");
    }

// 在可选的第3阶段，代码递归地遍历了新生成的结构
// 而且将每个名/值对传递给一个过滤函数，以便进行
// 可能的转换

    if (typeof filter === 'function') {
        j = walk('', j);
    }
    return j;
};

/* 设置XMLHttpRequest对象的各个不同的部分 */
function getRequestObject(url,options) {

    // 初始化请求对象
    var req = false;
    if(window.XMLHttpRequest) {
        var req = new window.XMLHttpRequest();
    } else if (window.ActiveXObject) {
        var req = new window.ActiveXObject('Microsoft.XMLHTTP');
    }

    if(!req) return false;

    // 定义默认的选项
    options = options || {};
    options.method = options.method || 'GET';
    options.send = options.send || null;

    // 为请求的每个阶段定义不同的侦听器
    req.onreadystatechange = function() {
        switch (req.readyState) {
            case 1:
            // 载入中
            if(options.loadListener) {
                options.loadListener.apply(req,arguments);
            }
            break;
```

www.TopSage.com

```
        case 2:
            // 载入完成
            if(options.loadedListener) {
                options.loadedListener.apply(req,arguments);
            }
            break;
        case 3:
            // 交互
            if(options.ineractiveListener) {
                options.ineractiveListener.apply(req,arguments);
            }
            break;
        case 4:
            // 完成
            // 如果失败则抛出错误
            try {
            if (req.status && req.status == 200) {


                    // 针对content-type的特殊侦听器
                    // 由于Content-Type头部中可能包含字符集，如：
                    // Content-Type: text/html; charset=ISO-8859-4
                    // 因此通过正则表达式提取出所需的部分
                    var contentType = req.getResponseHeader( ➥
'Content-Type');
                    var mimeType = contentType.match( ➥
/\s*([^;]+)\s*(;|$)/i)[1];

                    switch(mimeType) {
                        case 'text/javascript':
                        case 'application/javascript':
                            // 响应是JavaScript，因此以
                            // req.responseText作为回调的参数
                            if(options.jsResponseListener) {
                                options.jsResponseListener.call(
                                    req,
                                    req.responseText
                                );
                            }
                            break;
                        case 'application/json':
                            // 响应是JSON，因此需要用匿名函数对
                            // req.responseText进行解析
                            // 以返回作为回调参数的JSON对象
                            if(options.jsonResponseListener) {
                                try {
                                    var json = parseJSON(
                                        req.responseText
                                    );
                                } catch(e) {
                                    var json = false;
                                }
                                options.jsonResponseListener.call(
                                    req,
                                    json
                                );
                            }
                            break;
                        case 'text/xml':
```

www.TopSage.com

```
                        case 'application/xml':
                        case 'application/xhtml+xml':
                            // 响应是XML，因此以
                            // req.responseXML作为
                            // 回调的参数
                            // 此时是Document对象
                            if(options.xmlResponseListener) {
                                options.xmlResponseListener.call(
                                    req,
                                    req.responseXML
                                );
                            }
                            break;
                        case 'text/html':
                            // 响应是HTML，因此以
                            // req.responseText作为
                            // 回调的参数
                            if(options.htmlResponseListener) {
                                options.htmlResponseListener.call(
                                    req,
                                    req.responseText
                                );
                            }
                            break;
                    }
                    // 针对响应成功完成的侦听器
                    if(options.completeListener) {
                        options.completeListener.apply(req,arguments);
                    }
                } else {
                    // 响应完成但却存在错误
                    if(options.errorListener) {
                        options.errorListener.apply(req,arguments);
                    }
                }
            } catch(e) {
                // 忽略错误
            }
            break;
        }
    };
    // 开启请求
    req.open(options.method, url, true);
    // 添加特殊的头部信息以标识请求
    req.setRequestHeader('X-ADS-Ajax-Request','AjaxRequest');
    return req;
}
window['ADS']['getRequestObject'] = getRequestObject;

/* 通过简单地包装getRequestObject()和send()方法
发送XMLHttpRequest对象的请求 */
function ajaxRequest(url,options) {
    var req = getRequestObject(url,options);
    return req.send(options.send);
}
window['ADS']['ajaxRequest'] = ajaxRequest;

……以下是库中已有的内容……

})();
```

www.TopSage.com

这个对象能够根据在options参数中指定的各种不同的方法，来完成对响应所有必要的处理：

- method，是用于请求的方法，默认为GET。
- send，是一个包含在XMLHttpRequest.send()中的可选的字符串，默认为null。
- loadListener，是当readyState为1时调用的onreadystatechange侦听器。
- loadedListener，是当readyState为2时调用的onreadystatechange侦听器。
- interactiveListener，是当readyState为3时调用的onreadystatechange侦听器。
- jsResponseListener，是当请求成功并且响应的Content-Type为text/javascript或application/javascript时调用的侦听器，这个侦听器将从响应中取得的JavaScript字符串作为其第一个参数。如果要执行该JavaScript字符串，必须使用eval()方法。
- jsonResponseListener，是当请求成功并且响应的Content-Type为application/json时调用的侦听器。这个侦听器将从响应中取得的JSON对象作为其第一个参数。
- xmlResponseListener，是当请求成功并且响应的Content-Type为application/xml或application/xhtml+html时调用的侦听器。这个侦听器将从响应中取得的XML DOM文档作为其第一个参数。
- htmlResponseListener，是当请求成功并且响应的Content-Type为text/html时调用的侦听器。这个侦听器将从响应中取得的HTML字符串作为其第一个参数。
- completeListener，是当上面所列的针对Content-Type的响应侦听器调用之后被调用的侦听器。这个方法总是在成功的响应最后被调用，也就是说如果响应中没有适当的Content-Type头部信息，那么你可以指定这个方法作为兜底儿（catchall）的侦听器。
- errorListener，是当响应状态值不是200也不是0时被调用的侦听器。如果是在不会提供适当响应代码的系统（如硬盘驱动器中的本地文件系统）上运行XMLHttpRequest，那么状态值将始终为0。在这种情况下，只有completeListener会被调用。

ADS.ajaxRequest方法则在处理请求的同时，设置了一个X-ADS-Ajax-Request头部信息，因此可以通过这个头部信息在服务器端脚本中识别该请求。

至于在脚本中使用ADS.ajaxRequest()方法，则与定义URL和适当的侦听器一样简单：

```
ADS.ajaxRequest('/path/to/script/',{
    method:'GET',
    completeListener:function() {
        alert(this.responseText);
    }
});
```

但是，此时要注意在侦听器内部使用了this关键字来引用请求对象。由于对ADS.ajaxRequest对象的侦听器应用了call方法[①]，所以这里与常规的onreadystatechange方法不同的是，this关键字引用的是请求对象而不是onreadystatechange方法。也就是说，可以通过this来访问XMLHttpRequest对象的任何其他属性，如this.responseXML或this.status等。

① 或apply方法修改了作用域。——译者注

www.TopSage.com

在本章后面，当我们介绍通过Ajax增强的一个简单的图像浏览器时，你还会看到这个对象的更多用法。不过，在使用这个可重用的对象之前，你应该问问自己Ajax是不是正确的选择，以及自己还应该理解哪些需要考虑到的其他问题。

### 7.1.5 Ajax 是正确的选择吗

如果你已经掌握了诸如语义标记、JavaScript、DOM脚本编程以及XML等这些必需的技术，那么可以说已经具备了一个良好的开端。不过，还应该记住的是，可以使用Ajax不等于应该使用。恰当使用Ajax方法可以提供类似桌面应用程序的体验，而且还能提高Web应用程序的效率——不过，我要着重强调的是这只是“恰当使用”。假如使用不当，Ajax则很容易破坏一个站点的可用性和可访问性。

稍后你将会看到，在某些情况下，Ajax跟你所了解的那个Web应用程序完全不同，而且在很多方面都把事情搞得更复杂了。因此，在实现一个Ajax应用之前，最好问自己几个问题：

- Ajax会不会因为权限受限、破坏已有功能或者需要额外的训练和技术而以某种方式妨碍最终用户的访问？
- Ajax会不会破坏你的网站整体界面的一致性？
- Ajax导致在应用程序开发过程中增加的开发时间及成本，是否超过了最终用户因此可以获得的潜在利益的价值？
- Ajax会不会增加用户体验或者开发需求的复杂性？
- 你计划使用Ajax是不是仅仅因为你希望应用程序“看上去更酷”？

如果对这些问题中的任何一个问题回答“是”，就说明你需要对使用Ajax的动机进行重新评估。如果对所有这些问题都回答“否”，那么Ajax可能是你的正确选择。即便如此，考虑到下一节讨论的一些问题，难题依然存在。

## 7.2 为什么 Ajax 会破坏网站及如何解决

很多开发者（也包括我自己）在最初接触Ajax时，都不会注意到它固有的一些问题。抛开其不易访问性不谈，异步Ajax界面的要求本身会带来一些问题，比如竞态条件（多个请求“竞争”着去做同一件事）和跨站点安全限制等，而绝大多数这些问题在传统的模型中是不存在的。而且，Ajax经常破坏浏览器产品的功能，例如后退按钮和书签等用户已经离不开或者期望以某种方式运行的功能。而不解决这些问题，就谈不上提供增强（而不是降级）Web应用程序质量的流畅且没有问题的Ajax体验。本节我们就来讨论一下如何解决这些问题。

### 7.2.1 依赖 JavaScript 生成内容

你可能会犯的一个最大的错误，就是依赖Ajax显示网页中的主要内容，但却未提供Ajax失效情形下的替代方案。因而，开发Ajax应用的首要规则，应该是先开发不包含Ajax功能的网站，然后再通过Ajax来增强网站的界面。即在载入事件中使用能力检测，以便在特定的技术不存在时，可以舒缓地退化到一个不太满意但仍可访问的版本。

www.TopSage.com

有些人可能会觉得在要求用户首先登录的Web应用中，这并不是个大问题。搜索引擎的蜘蛛程序，这个可能因为技术上不可访问而受影响的最大目标，不会访问Web应用的后台管理区域，因此提供可退化的界面似乎并不重要——但在这种情况下，你却忽视了那些残障的Web用户。有些人在访问你的网站时，可能会使用专门为他们特定的残疾而设计的替代浏览器。W3C Web Accessibility Initiative's AlternateWeb Browsing（http://www.w3.org/WAI/References/Browsing）页面上公布了许多这样的替代浏览器，而其中大多数都不能通过迷人的Ajax或者基于鼠标的界面进行导航。因此为了让残障用户能够轻松地访问并在你的网站中遍历，唯一的途径就是提供可退化的、由语义标记构成的版本。换句话说就是依赖传统的非JavaScript技术和单击重载模型构建的版本。

> 有一段时期，曾经流行过针对不同浏览器开发同一网站不同版本的做法。也就是说，如果用户使用了某种特殊的浏览器，那么可以访问支持该浏览器中特殊技术的替代版本。虽然以退化后的格式重复开发网站也是一种选择，但同时也意味着双倍的工作量。如果某种方法能够照顾到所有情况，那么在你2倍甚至3倍的开发工作中就会显得尤其突出。前面我们已经看到过了，开发具有可退化功能的渐进增强的Web应用并不困难，它所需要的只是一些在整合应用过程中的换位思考，从而为用户在简单的可访问性基础上提供更多的好处，比如增强的可用性、开发者友好性以及更好的搜索引擎优化等。

### 7.2.2 通过<script>标签绕过跨站点限制

在使用XMLHttpRequest对象时，可能会遇到的一个最大限制，就是无法向当前主机所在域之外的其他源请求信息。因为所有Web浏览器都执行限制JavaScript只能与提供网页的主机通信的同源安全策略。同源策略的典型特征，就是所有的请求都将发生在客户端浏览器与应用程序所在的服务器之间。不过有时候，恐怕还需要从你控制范围之外的服务器上面获取信息，比如某台管理服务器或专门的信息服务器。而当向XMLHttpRequest对象传递一个指向其他域的完整URL时，这个请求会显示出一个错误并被拒绝访问，如图7-2中的Firebug JavaScript控制台所示。

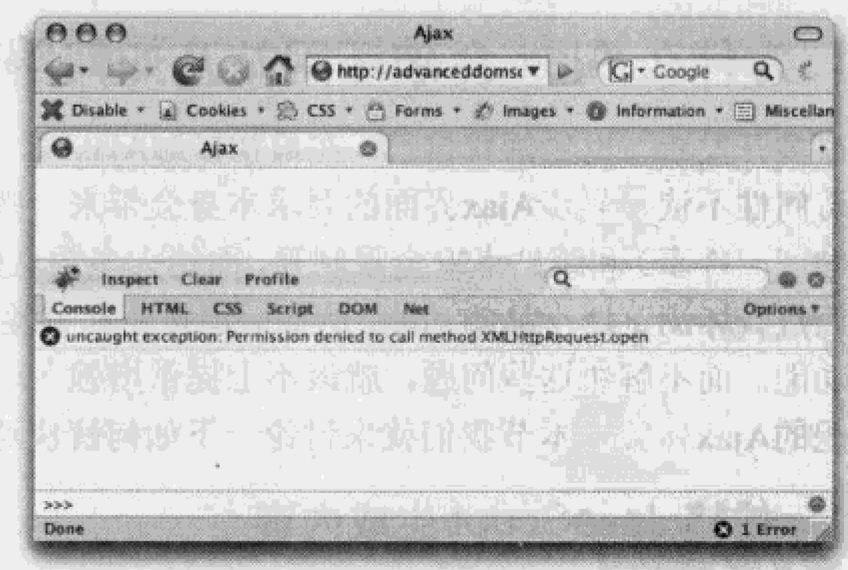



图7-2　当尝试访问位于当前域外部的URL时出现的安全异常

不过，有人已经就此采取了巧妙的对策，即使用嵌入式的<iframe>从其他服务器中获取数据。这种方式在某种意义上讲是成功的，因为使用<iframe>可以载入外部域的数据，而且还能传递任意参数。但是，以这种方式只能提交数据却无法检索数据。因为当<iframe>中的其他域的信息载入完成后，JavaScript是无法跨越<iframe>

www.TopSage.com

的界限与其中的内容进行通信的，而这依然要归因于同源策略的限制。

绕过这个障碍的一种方法是忘掉炫目的XMLHttpRequest对象，而只使用动态生成的<script>元素。即可以在页面中载入来源指向其他域的<script>元素，而得到的脚本将会如同你在页面中包含的其他JavaScript脚本一样执行。你应该能够想象得到，使用这种方法的注意事项是只能载入JavaScript，而不能像使用XMLHttpRequest对象那样载入任意内容。表7-1简单地列出了XMLHttpRequest方法与<script>元素方法之间的特性对比。

表7-1 XMLHttpRequest对象与动态生成的<script>标签之间的简单比较

| 特 性 | XMLHttpRequest对象 | 动态生成的<script>元素 |
|---|---|---|
| 跨浏览器兼容性 | 对A类浏览器[①]而言是 | 是，但存在一些小bug |
| 异步通信 | 是 | 是 |
| 同步通信 | 是 | 否 |
| 被“同源”策略限制 | 是 | 否 |
| 支持多种HTTP方法 | 是 | 否，只支持GET |
| 访问HTTP状态代码 | 是 | 否 |
| 发送自定义的头部信息 | 是 | 否 |
| 支持自定义的头部信息 | 是 | 否 |
| 支持将XML转换为DOM对象 | 是 | 否* |
| 支持JavaScript/JSON | 是，但必须使用eval()函数 | 是 |
| 支持HTML | 是 | 否* |
| 支持纯文本 | 是 | 否* |

* 不过，这种特性可以通过在JavaScript响应中嵌入相关内容然后再从中提取出来实现。

虽然不是一种完美的解决方案，但动态生成<script>元素的方法却是一个有益的起点，而且也能够催生出有用的第三方工具，比如Google在其Google Maps API（http://google.com/apis/maps）中对这种方法的运用（我们将在第11章介绍有关这个API的更多例子）。

对这种方法我下面未作太多介绍，但为了让你有一个初步的认识，下面列出了一组使用动态生成的<script>元素来载入信息的JavaScript对象。为便于使用，我对这些对象的API进行了设计，使它们看起来与XMLHttpRequest对象以及前面的ADS.ajaxRequest()对象有些相似。这样，你就可以通过基本相同（只有少许差别）的方式来使用它们了：

```
(function(){

window['ADS'] = {};

……以上是库中已有的代码……

// XssHttpRequest对象的计数器
var XssHttpRequestCount=0;
```

① 由Yahoo公司在其浏览器分级策略中划分出来，详见http://developer.yahoo.com/yui/articles/gbs/index.html#history。——译者注

www.TopSage.com

```
// XMLHttpRequest对象的一个
// 跨站点<script>标签的实现*/
var XssHttpRequest = function(){
    this.requestID = 'XSS_HTTP_REQUEST_' + (++XssHttpRequestCount);
}
XssHttpRequest.prototype = {
    url:null,
    scriptObject:null,
    responseJSON:null,
    status:0,
    readyState:0,
    timeout:30000,
    onreadystatechange:function() { },

    setReadyState: function(newReadyState) {
        // 如果比当前状态更新
        // 则只更新就绪状态
        if(this.readyState < newReadyState || newReadyState==0) {
            this.readyState = newReadyState;
            this.onreadystatechange();
        }
    },

    open: function(url,timeout){
        this.timeout = timeout || 30000;
        // 将一个名为XSS_HTTP_REQUEST_CALLBACK
        // 的特殊变量附加给URL，其中包含本次请求的
        // 回调函数的名称
        this.url = url
            + ((url.indexOf('?')!=-1) ? '&' : '?' )
            + 'XSS_HTTP_REQUEST_CALLBACK='
            + this.requestID
            + '_CALLBACK';
        this.setReadyState(0);
    },

    send: function(){
        var requestObject = this;
        // 创建一个载入外部数据的新script对象
        this.scriptObject = document.createElement('script');
        this.scriptObject.setAttribute('id',this.requestID);
        this.scriptObject.setAttribute('type','text/javascript');
        // 尚未设置src属性，也先不将其添加到文档……

        // 创建一个在给定的毫秒数之后触发的
        // setTimeout()方法。如果在给定的时间
        // 内脚本没有载入完成，则取消载入
        var timeoutWatcher = setTimeout(function() {
            // 在脚本晚于我们假定的停止时间之后载入的情况下
            // 通过一个空方法来重新为window方法赋值
            window[requestObject.requestID + '_CALLBACK'] = function() { };

            // 移除脚本以防止进一步载入
            requestObject.scriptObject.parentNode.removeChild(
                requestObject.scriptObject
            );
```

www.TopSage.com

```
            // 将状态设置为错误
            requestObject.status = 2;
            requestObject.statusText = 'Timeout after '
                + requestObject.timeout
                + ' milliseconds.'

            // 更新就绪状态
            requestObject.setReadyState(2);
            requestObject.setReadyState(3);
            requestObject.setReadyState(4);

        },this.timeout);

        // 在window对象中创建一个与
        // 请求中的回调方法匹配的方法
        // 在调用时负责处理请求的其他部分
        window[this.requestID + '_CALLBACK'] = function(JSON) {
            // 当脚本载入时将执行这个方法
            // 同时传入预期的JSON对象

            // 在请求载入成功时
            // 清除timeoutWatcher方法
            clearTimeout(timeoutWatcher);

            // 更新就绪状态
            requestObject.setReadyState(2);
            requestObject.setReadyState(3);

            // 将状态设置为成功
            requestObject.responseJSON = JSON;
            requestObject.status=1;
            requestObject.statusText = 'Loaded.'

            // 更新就绪状态
            requestObject.setReadyState(4);
        }

        // 设置初始就绪状态
        this.setReadyState(1);

        // 现在再设置src属性并将其添加
        // 到文档头部。这样会载入脚本
        this.scriptObject.setAttribute('src',this.url);
        var head = document.getElementsByTagName('head')[0];
        head.appendChild(this.scriptObject);

    }
}
window['ADS']['XssHttpRequest'] = XssHttpRequest;

/* 设置XssHttpRequest对象的不同部分 */
function getXssRequestObject(url,options) {
    var req = new XssHttpRequest();

    options = options || {};
    // 默认中断时间为30秒
    options.timeout = options.timeout || 30000;
```

www.TopSage.com

```
    req.onreadystatechange = function() {
        switch (req.readyState) {
            case 1:
                // 载入中
                if(options.loadListener) {
                    options.loadListener.apply(req,arguments);
                }
                break;
            case 2:
                // 载入完成
                if(options.loadedListener) {
                    options.loadedListener.apply(req,arguments);
                }
                break;
            case 3:
                // 交互
                if(options.interactiveListener) {
                    options.interactiveListener.apply(req,arguments);
                }
                break;
            case 4:
                // 完成
                if (req.status == 1) {
                    if(options.completeListener) {
                        options.completeListener.apply(req,arguments);
                    }
                } else {
                    if(options.errorListener) {
                        options.errorListener.apply(req,arguments);
                    }
                }
                break;
        }
    };
    req.open(url,options.timeout);

    return req;

}
window['ADS']['getXssRequestObject'] = getXssRequestObject;

/* 发送XssHttpRequest请求 */
function xssRequest(url,options) {
    var req = getXssRequestObject(url,options);
    return req.send(null);
}
window['ADS']['xssRequest'] = xssRequest;


……以下是库中已有的内容……

})();
```

在使用以上对象和方法从外部主机域中载入脚本时，有一个必要条件，服务器对请求的响应必须返回一个JSON对象，而这个JSON对象则被包装在一个以GET请求的XSS_HTTP_REQUEST_CALLBACK变量的值定义的函数中。作为一个例子，下面就是位于http://advanceddomscripting.

www.TopSage.com

com/source/chapter7/xssRequest/responder.php中的简单的服务器端响应程序的代码：

```
<?php
header('Content-type: text/javascript');

// 只允许回调函数中存在数字、字母和下划线
$callback = preg_replace(
    '/[^A-Z0-9_]/i',
    '',
    $_GET['XSS_HTTP_REQUEST_CALLBACK']
);
echo "/* XSS request for callback: $callback */\n";

if($callback) {
    $date = date('r');
    echo
<<<JSON
{$callback}({
    message:'response on {$date}'
});
JSON;

}
?>
```

这样，当使用ADS.xssRequest()对象发送请求时：

```
ADS.xssRequest(
    'http://advanceddomscripting.com/source/chapter7/xssRequest/responder.php', {
        completeListener:function() {
            alert(this.responseJSON.message);
        },
        errorListener:function() {
            alert(this.statusText);
        }
    }
);
```

响应将会作为脚本被载入，而且返回的函数也将被执行：

```
XSS_HTTP_REQUEST_1({
    message:'It Worked!'
})
```

这个在响应中调用的函数，与前面XssHttpRequest对象中创建的函数是相匹配的。当脚本载入时，该函数会执行并通过为请求侦听器设置特殊的responseJSON属性来完成请求。

虽然与响应交互的方式与使用XMLHttpRequest对象的方式相同，但可用的属性却不完全一样。此处的XssHttpRequest对象在模拟了某些XMLHttpRequest对象属性的基础上，还添加了以下一些新属性。

- responseJSON，包含响应的结果。这个属性已经是一个JavaScript对象了，因此不需要再使用eval()。
- status，在这里可能的值有如下两个。
    - 1：表示成功

www.TopSage.com

■ 2：表示存在错误

- statusText，包含错误的原因。

可以通过在下列侦听器内部使用this关键字来访问以上属性。

- loadedListener，当对象处于载入完成状态时调用。
- waitListener，当对象等待响应时调用。
- completeListener，当对象取得成功的响应后调用。
- errorListener，当脚本载入外部脚本失败或载入的脚本格式不正确时调用。

> 为了保证对象正确运行，响应必须要调用GET变量中所引用的JavaScript函数。一般而言，最好的方式就是向这个函数中传递一个JSON对象。但如果你想传递XML文件或其他信息，也可以简单地将这些信息包含在JSON对象中，然后通过响应侦听器来按照需要解析这些信息。

### 7.2.3　后退按钮和书签功能

即使适当的地实现了能够平稳退化的行为增强，Ajax仍然会带来一些基本的问题。而且，这些问题并不限于Ajax请求，而是存在于任何JavaScript行为增强中。这些问题中最有代表性的就是后退按钮和书签的问题。

当用户在传统的单击重载式的网站中导航时，单击后退按钮可以返回网站中的上一个页面，这一过程如图7-3所示。

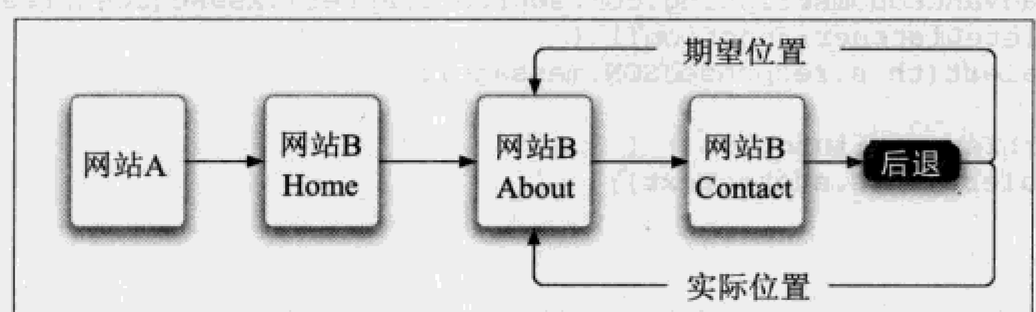



图7-3　在传统的单击重载式的工作流中，用户单击后退按钮时期望的位置和实际返回的位置

图7-3所示的工作流是有意义的，而且这也是普通的Web用户所期望发生的行为。然而，当在一个启用了Ajax功能而且没有了传统的请求重载行为的网站中导航时，对后退按钮行为的预期依然是存在的。当在Ajax导致了错误的行为之后单击后退按钮时，返回的并不是前一次Ajax请求的状态，而是用户在当前页面之前所访问的页面。这一过程如图7-4所示。

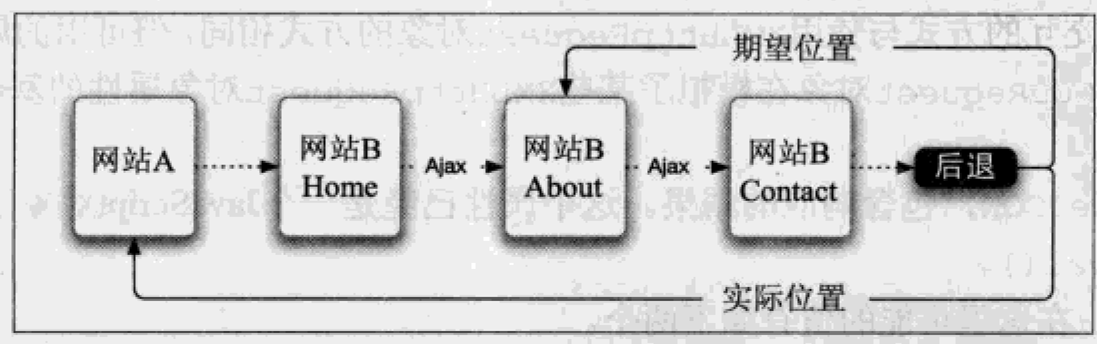



图7-4　在间断的Ajax工作流中，用户单击后退按钮时期望的位置和实际返回的位置

www.TopSage.com

假如你通过Ajax简单地改变了Web应用程序中某些对象的状态，例如选中或取消选中列表中的某个项，那么后退按钮则不是什么问题。当用户在这种情况下按下后退按钮时，返回动作并不意味着一次“撤销”操作，因为即使是在传统的请求重载式的工作流中，也不存在对该动作会恢复列表项状态的期待。这一过程如图7-5所示。

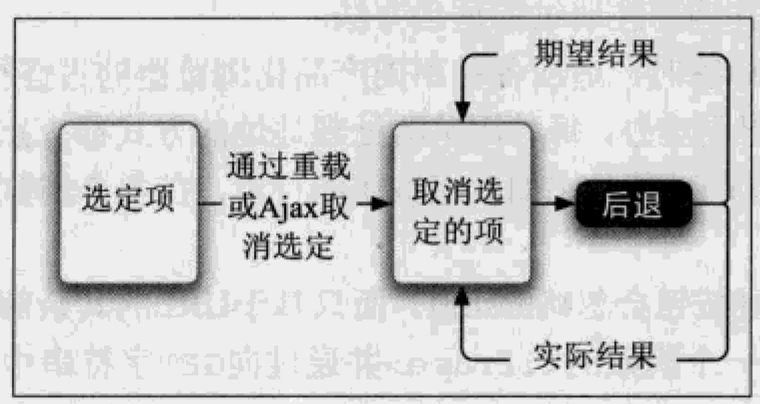



图7-5 在修改页面元素状态的情况下，用户单击后退按钮时期望的结果与实际的结果

前面图7-4所示的问题，会在你决定通过Ajax来导航在传统的方法中被认为是页面的内容时公开出来。比如，让用户通过单击导航选项卡来展示新的内容页，甚至只是浏览一组图像，都可能被用户认为请求了多个页面，与此同时也就产生了使用后退按钮（或者更少用的前进按钮）可以回到适当页面的期待。

书签功能[①]也会以类似的方式受到影响。为使用Ajax请求构建的任何页面添加的书签，都只会指向初始的页面，而非在添加书签时在Ajax导航中处于活动状态的当前内容。

通过一些DOM脚本并利用URL的hash值可以补救后退按钮和标签的问题。例如，在浏览器中通过下列URL打开图像浏览器：

```
http://advanceddomscripting/source/chapter7/browser/
```

而许多链接中都会用到hash（也称为锚），比如：

```
http://advanceddomscripting/source/chapter7/browser/#photo/1
```

> 在本章后面，你会重新创建这个图像浏览器，届时你将看到如何对一个基本的Web应用程序进行多方面的修复。

其中，hash值表示浏览器应该保持在当前页面上，但同时必须通过重新定位页面使得与hash匹配的命名的锚（或带有相同ID属性的元素），在浏览器的视口（view port）中可见。如果文档中不存在与hash匹配的项，则浏览器只会改变地址栏中的URL。这里的hash可以连同URL一起被作为标签使用，而且通过一些DOM脚本及hack技术，hash还可以在用户单击后退按钮时用于更新页面。

#### 1. 不那么简单的修复

修复后退按钮和标签涉及监视和识别URL中hash值的变化，并通过该变化来调用Ajax请求。

① 在IE中是收藏夹功能。——译者注

www.TopSage.com

要做到这一点，需要创建一个检测地址中hash值变化的对象，同时以预先定义的适当方法进行响应。这个对象需要做以下几件事。

- 使用不唐突的DOM脚本增强文档以跟踪页面的变化。
- 允许注册根据不同hash值作出反应的不同方法。
- 监视地址栏的变化并调用注册过的适当方法。
- 在处理后退按钮和标签时，适应各种不同产品化浏览器的古怪行为。

好像是为了把事情弄得更难办，每种浏览器处理地址的方式都有点不一样，因此对它们都需要特殊对待。虽然是产品化的功能，比如后退按钮的行为，在各个浏览器间也不是标准的，而且也不受任何规范的控制：

- 在IE中，后退和前进按钮会忽略hash值，而只基于URL的其余部分进行导航。为了解决这个问题，需要使用一个隐藏的`<iframe>`并通过向`GET`字符串中添加hash来模拟导航到不同的网页。
- 在Safari中，当通过带hash的URL向前或向后导航时，浏览器的`history`及`history.length`会相应改变。但是，`window.location.href`的值会保持为在通过后退和前进按钮导航之前，浏览器中打开的最后一个地址。考虑到安全原因，我们无法访问`history`中的URL，因此需要相对于Safari的`history`的长度保持跟踪访问过的hash值，并且能够从存储列表中找回适当的hash。
- Firefox和Opera的行为则都更符合常理，它们会在使用后退和前进按钮通过hash来导航时，同步更新`window`对象的`location`值。

此外，访问和修改地址栏中URL的方式也是因浏览器而异的。为此，我想拿出的解决方案可能够不上优雅的标准，因为这个解决方案要采取我曾经要求你避免的一种不良行为——浏览器嗅探。

#### 2. 针对产品功能的浏览器嗅探

在处理具体浏览器的产品功能时，浏览器嗅探是一种可以接受的方案，而且也是唯一可能的方案，因为产品的差异并不针对JavaScript中的能力或对象。虽然每种浏览器都有访问地址栏中URL的`window.location`对象，但在通过`hash`进行导航时，每种浏览器都会受到不同的影响或者说反应都有所不同。对于在这种情形下保持浏览器的一致性而言，对象检测是派不上用场的。例如，你可以使用类似下面的代码：

```
if (window.attachEvent) {
    // 说明这是Microsoft IE
}
```

但是，`window.attachEvent`方法与你要解决的问题没有任何直接的关系，因此这也不是一个适当的对象。所以，浏览器检测才是合理的解决方案。

#### 3. 跟踪地址变化

将下面的`ADS.actionPager`对象添加到你的ADS库中，然后我们再来看看它是如何解决后退按钮和标签问题的：

```
(function(){
```

www.TopSage.com

```
window['ADS'] = {};

……以上是库中已有的内容……

/* 生成回调函数的一个辅助方法 */
function makeCallback(method, target) {
    return function() { method.apply(target,arguments); }
}

/* 一个用来基于hash触发注册的
方法的URL hash侦听器 */
var actionPager = {
    // 前一个hash
    lastHash : '',
    // 为hash模式注册的方法列表
    callbacks: [],
    // Safari历史记录列表
    safariHistory : false,
    // 对为IE准备的iframe的引用
    msieHistory: false,
    // 应该被转换的链接的类名
    ajaxifyClassName: '',
    // 应用程序的根目录。当创建hash时
    // 它将是被清理后的URL
    ajaxifyRoot: '',

    init: function(ajaxifyClass,ajaxifyRoot,startingHash) {

        this.ajaxifyClassName = ajaxifyClass || 'ADSActionLink';
        this.ajaxifyRoot = ajaxifyRoot || '';

        if (/Safari/i.test(navigator.userAgent)) {
            this.safariHistory = [];
        } else if (/MSIE/i.test(navigator.userAgent)) {
            // 如果是MSIE，添加一个iframe以便
            // 跟踪重写（override）后退按钮
            this.msieHistory = document.createElement('iframe');
            this.msieHistory.setAttribute('id', 'msieHistory');
            this.msieHistory.setAttribute('name', 'msieHistory');
            setStyleById(this.msieHistory,{
                'width':'100px',
                'height':'100px',
                'border':'1px solid black',
                'visibility':'visible',
                'zIndex':'-1'
            });
            document.body.appendChild(this.msieHistory);
            this.msieHistory = frames['msieHistory'];

        }

        // 将链接转换为Ajax链接
        this.ajaxifyLinks();

        // 取得当前的地址
        var location = this.getLocation();
```

www.TopSage.com

```
        // 检测地址中是否包含hash（来自书签）
        // 或者是否已经提供了hash
        if(!location.hash && !startingHash) { startingHash = 'start'; }

        // 按照需要保存hash
        ajaxHash = this.getHashFromURL(location.hash) || startingHash;
        this.addBackButtonHash(ajaxHash);

        // 添加监视事件以观察地址栏中的变化
        var watcherCallback = makeCallback(
            this.watchLocationForChange,
            this
        );
        window.setInterval(watcherCallback,200);
    },
    ajaxifyLinks: function() {
        // 将链接转换为锚以便Ajax进行处理
        links = getElementsByClassName(
            this.ajaxifyClassName,
            'a',
            document
        );
        for(var i=0 ; i < links.length ; i++) {
            if(hasClassName(links[i],'ADSActionPagerModified')) {
                continue;
            }

            // 将href属性转换为#value形式
            links[i].setAttribute(
                'href',
                this.convertURLToHash(links[i].getAttribute('href'))
            );
            addClassName(links[i],'ADSActionPagerModified');

            // 注册单击事件以便在必要时添加历史记录
            addEvent(links[i],'click',function() {
                if (this.href && this.href.indexOf('#') > -1) {
                    actionPager.addBackButtonHash(
                        actionPager.getHashFromURL(this.href)
                    );
                }
            });
        }
    },
    addBackButtonHash: function(ajaxHash) {
        // 保存hash
        if (!ajaxHash) return false;
        if (this.safariHistory !== false) {
            // 为Safari使用特殊数组
            if (this.safariHistory.length == 0) {
                this.safariHistory[window.history.length] = ajaxHash;
            } else {
                this.safariHistory[window.history.length+1] = ajaxHash;
            }
            return true;
        } else if (this.msieHistory !== false) {
```

www.TopSage.com

```
            // 在MSIE中通过导航iframe
            this.msieHistory.document.execCommand('Stop');
            this.msieHistory.location.href = '/fakepage?hash='
                + ajaxHash
                + '&title='+document.title;
            return true;
        } else {
            // 通过改变地址的值
            // 使用makeCallback包装函数
            // 以便在超时方法内部使this
            // 引用actionPager
            var timeoutCallback = makeCallback(function() {
                if (this.getHashFromURL(window.location.href) != ajaxHash) {
                    window.location.replace(location.href + '#'
                        + ajaxHash);
                }
            },this);
            setTimeout(timeoutCallback, 200);
            return true;
        }
        return false;
    },
    watchLocationForChange: function() {

        var newHash;
        // 取得新的hash值
        if (this.safariHistory !== false) {
            // 在Safari中从历史记录数组中取得
            if (this.safariHistory[history.length]) {
                newHash = this.safariHistory[history.length];
            }
        } else if (this.msieHistory !== false) {
            // 在MSIE中从iframe的地址中取得
            newHash = this.msieHistory.location.href.split('&')[0].split('=')[1];
        } else if (location.hash != '') {
            // 对其他浏览器从window.location中取得
            newHash = this.getHashFromURL(window.location.href);

        }

        // 如果新hash与最后一次的hash不相同，则更新页面
        if (newHash && this.lastHash != newHash) {
            if (this.msieHistory !== false
            && this.getHashFromURL(window.location.href) != newHash) {
                // 修复MSIE中的地址栏
                // 以便能适当地加上标签（或加入收藏夹）
                location.hash = newHash;
            }

            // 在发生异常的情况下使用try/catch
            // 结构尝试执行任何注册的侦听器
            try {
                this.executeListeners(newHash);
                // 在通过处理程序添加任何
                // 新链接的情况下进行更新
                this.ajaxifyLinks();
            } catch(e) {
```

www.TopSage.com

```
                // 这里将捕获到回调函数中的任何异常JS
                alert(e);
            }

            // 将其保存为最后一个hash
            this.lastHash = newHash;
        }
    },
    register: function(regex,method,context){
        var obj = {'regex':regex};
        if(context) {
            // 一个已经指定的环境
            obj.callback = function(matches) {
                method.apply(context,matches);
            };
        } else {
            // 以window作为环境
            obj.callback = function(matches) {
                method.apply(window,matches);
            };
        }

        // 将侦听器添加到回调函数数组中
        this.callbacks.push(obj)
    },
    convertURLToHash: function(url) {
        if (!url) {
            // 没有url，因而返回一个磅字符（#）
            return '#';
        } else if(url.indexOf("#") != -1) {
            // 存在hash，因而返回它
            return url.split("#")[1];
        } else {
            // 如果URL中包含域名（MSIE）则去掉它
            if(url.indexOf("://") != -1) {
                url = url.match(/:\/\/[^\/]+(.*)/)[1];
            }
            // 按照init()中的约定去掉根目录
            return '#' + url.substr(this.ajaxifyRoot.length)
        }
    },
    getHashFromURL: function(url) {
        if (!url || url.indexOf("#") == -1) { return ''; }
        return url.split("#")[1];
    },
    getLocation: function() {
        // 检查hash
        if(!window.location.hash) {
            // 没有则生成一个
            var url = {host:null,hash:null}
            if (window.location.href.indexOf("#") > -1) {
                parts = window.location.href.split("#")[1];
                url.domain = parts[0];
                url.hash = parts[1];
            } else {
                url.domain = window.location;
            }
            return url;
```

www.TopSage.com

```
        }
        return window.location;
    },
    executeListeners: function(hash){
        // 执行与hash匹配的任何侦听器
        for(var i in this.callbacks) {
            if((matches = hash.match(this.callbacks[i].regex))) {
                this.callbacks[i].callback(matches);
            }
        }
    }
}
window['ADS']['actionPager'] = actionPager;
```

……以下是库中已有的内容……

```
})();
```

通过查看ADS.actionPager对象中的注释，你会发现对每种浏览器的处理都稍有不同。

对于IE，ADS.actionPager()会向文档中添加一个<iframe>元素。IE虽然会在后退按钮中忽略带有hash的URL，但与此同时，如果你也在嵌入的iframe中导航到了另一个页面，那么后退按钮会首先应用于嵌入的iframe，然后才是其所在的父页面。当iframe后退到不能再退时，其父页面才会发生改变。通过在hash每次改变的时候都修改iframe，ADS.actionPager()对象巧妙地使浏览器保持在了正确的页面上。

使用隐藏的iframe的唯一问题，就是需要引用网站中的一个真实页面。但这个页面不需要做什么特别的事情，使用它只是为了不让IE尝试去搜索不存在的页面。虽然可以在这里使用任何页面，但我还是建议你将链接指向一个简单的空HTML页面：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>iframe target page</title>
</head>
<body>Nothing to see here</body>
</html>
```

在前面ADS.actionPager()的源代码中，已经将链接指向了advanceddomscripting.com网站根目录中的fakepage脚本。

处理Safari同样也需要一点技巧，因为它需要通过一个数组以及浏览器的history.length属性，才能找到应该为当前URL使用什么hash。

这个对象实现起来相对是比较容易的，因为只要使用ADS.actionPager对象中的如下3个方法就可以了。

- ADS.actionPager.init(ajaxifyClass,startingHash)：初始化页面（pager）的功能。
- ADS.actionPager.register(hash,listener,context)：用于根据特殊的hash来注册侦听器。
- ADS.actionPager.ajaxifyLinks()：会自动将任何带标识的锚标签转换为页面（pager）可以识别的链接。

www.TopSage.com

> 其他的方法都是作为公有方法存在的，以便你能够将更多的脚本整合到页面（pager）中，进而提供更高级的功能。

首先，就像是Ajax功能不存在一样编写Web应用程序。同样，如果可能的话，最好以“优质URL”的格式[1]来编写与页面相关的链接，以便于将来处理。例如，如果每个页面的链接类似pages.php?page=1，那么可以使用URL重定向[2]或者其他某些机制把这个链接修改为pages/1或pages_1的格式。虽然这一步不是必需的，但却能够使下一步更方便。

> 当我们在本章后面介绍图像浏览器的例子时，还会更详细地介绍URL重定向。

然后，通过一个唯一的类名把希望转换为Ajax请求的锚标签标识出来，比如ADSActionLink：

```
<a href="pages/1" class="ADSActionLink">...</a>
<a href="pages/2" class="ADSActionLink">...</a>
<a href="pages/3" class="ADSActionLink">...</a>
```

这里的类名可以根据你自己的意愿指定，ADSActionLink只是在不指定其他类名情况下的一个默认值。这样，当init()方法运行时，它会自动调用ADS.actionPager.ajaxifyLink()方法，而后者则会将以上链接中的href属性转换为hash，同时再为其追加一个ADSActionPagerModified类以标识该锚元素已经转换完成：

```
<a href="#pages/1" class="ADSActionLink ADSActionPagerModified">...</a>
<a href="#pages/2" class="ADSActionLink ADSActionPagerModified">...</a>
<a href="#pages/3" class="ADSActionLink ADSActionPagerModified">...</a>
```

而且，actionPager.ajaxifyLinks()方法还将为链接添加一个引用ADS.actionPager对象的单击事件侦听器，以便适当地跟踪单击事件：

```
……ADS.ActionPager.ajaxifyLinks()方法中的代码片段……
addEvent(links[i],'click',function() {
    if (this.href && this.href.indexOf('#') > -1) {
        actionPager.addBackButtonHash(
            actionPager.getHashFromURL(this.href)
        );
    }
});
……ADS.ActionPager.ajaxifyLinks()方法中的代码片段……
```

如果你愿意，也可以在运行ADS.actionPager.init()方法之后的任何时候，通过手工调用ADS.actionPager.ajaxifyLinks()。这在应用程序的其他部分添加的链接格式不正确的情况下可能会有用。如果多次调用ADS.actionPager.ajaxifyLinks()方法，那么该方法会忽略已经转换过的链接。同样地，每当注册的事件发生时也会自动调用这个方法，从而保证添加的链接也会被转换。

在将链接转换完成之后，单击这些链接时浏览器将会保持在同一个页面上，而且只有地址栏

---

① 所谓“优质URL”，可以理解为静态链接，而不是动态链接，比如page_1.html而不是pages.php?page=1这种格式。之所以说静态链接“优质”是因为容易记忆，也容易被搜索引擎收录。——译者注

② 应该是像在WordPress这样的系统中常用的URL重写机制，但需要Web服务器的支持（如Apache）。——译者注

www.TopSage.com

的URL中会显示一个新hash值。此时虽然后退和前进按钮以及书签功能都会有效，但主要的功能仍然欠缺。

对每个不同的hash，还需要分别为它们注册一个当检测到匹配的hash时调用的方法。此时就需要使用ADS.actionPager.register()方法了。以前面讨论的page/#格式的URL为例，需要像下面这样来注册它：

```
ADS.actionPager.register('pages/1',function(hash) {
    ADS.ajaxRequest('pageBits.php?page=1',{
        completeListener = function() {
            // 将this.responseText中的内容添加DOM树
        }
    });
});
```

这个注册的方法将只会匹配包含page/1的URL hash。

> 由于ADS.actionPager对象的设置，匹配将仅限于hash值，而不包括磅字符（#）。

为了使匹配更加通用，也可以在注册时使用正则表达式。这样一来，就可以把相同的侦听器应用到多个hash上面了：

```
ADS.actionPager.register(/^pages\/([0-9])$/,function(hash,page){
    ADS.ajaxRequest('pageBits.php?page=' + page ,{
        completeListener = function() {
            // 将this.responseText中的内容添加DOM树
        }
    });
});
```

在这个例子中，所用的正则表达式会匹配hash值为page/0到page/9的情况。而且，由于正则表达式中包含了捕获圆括号，所以侦听器函数将会取得另外一个参数（即这里的page），该参数是正则表达式圆括号中的模式所匹配的内容[①]。

此外，你在注册的侦听器内部所看到的this关键字，引用的是window对象。假如需要this引用其他对象，那么可以使用ADS.actionPager.register()方法可选的第3个参数来修改。比如，在下面的例子中，通过在为hash注册的侦听器中指定document.body对象，任何与正则表达式匹配的hash（即hash中包含page）都会使文档主体的background-color属性变为blue：

```
ADS.actionPager.register(/page/i,function(hash,page) {
    ADS.setStyle(this,{
        'background-color':'blue'
    });
},document.body);
```

当URL hash改变时，每个注册的侦听器都会被求值，因此可以为同一个hash注册多个侦听器。而且，侦听器会按照注册它们的顺序被处理。也就是说，如果你愿意，可以注册具有先后依赖关系的侦听器。

以上方案解决了在Ajax导航的情况下，书签以及后退/前进按钮的功能问题。然而，如果你

① 即0~9。——译者注

www.TopSage.com

在侦听器中整合了Ajax，则又会导致下面我们将要讨论的同步性问题。

### 7.2.4　完成请求的赛跑

无论你曾听到过有关Web 2.0和Ajax的什么样的介绍，但有一点可以肯定：即它是异步的。但是，你可能还不知道它同样也是一场赛跑，而且你有可能输掉这场赛跑。

传统的Web应用程序会遵循如图7-6所示的一种常规模式。

但是，在使用`XMLHttpRequest`对象时，则只会载入一个页面和众多的异步请求，如图7-7所示。

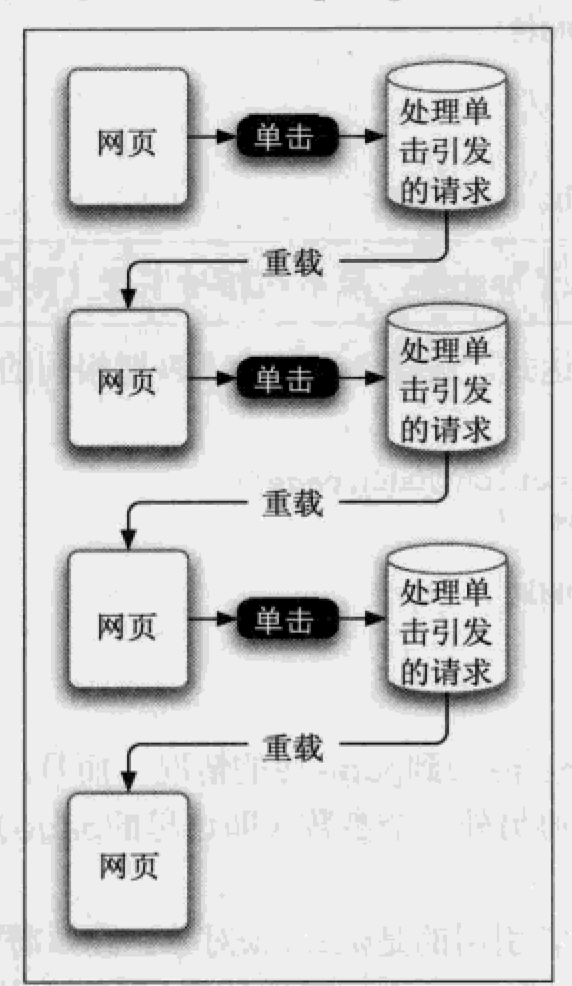



图7-6　传统的“载入-单击-重载”的请求工作流

图7-7　Ajax请求的工作流

所有这些异步通信会导致应用程序流程的本质发生变化。由于页面只载入一次，我们不能再依赖“载入-单击-重载”①的循环。但问题在哪呢？当然，如果你理解并认同异步性的潜在含义，其实这并不是一个问题。不过，必须要知道的一点是，现在你能够在短暂的时间内做很多的事了。虽然每次新的`XMLHttpRequest`请求都会被发出，但这些请求却不一定按照你发送它们的顺序返回。

作为一个简单的示范，你可访问一下http://advanceddomscripting.com/source/chapter7/latency/中的例子。如果你多单击几次Submit按钮，就会发现这个问题（见图7-8）。

图7-8中的请求是按照“Request+数字”所显示的顺序发送的，但是响应则没按照该顺序被返回，这一点从它们在列表中的位置可以看出来。在这个例子中，每当单击一次提交按钮，都会向

① 原文load-click-repeat有误。——译者注

www.TopSage.com

服务器发送一次请求。而在服务器端，则通过使PHP脚本休眠[①]0～3秒钟的随机间隔来模仿服务器的通信延迟：

```
<?php
header("Cache-Control: no-cache, must-revalidate");
sleep( $time = (rand(0,6)/2) );
echo "$time";
die();
?>
```

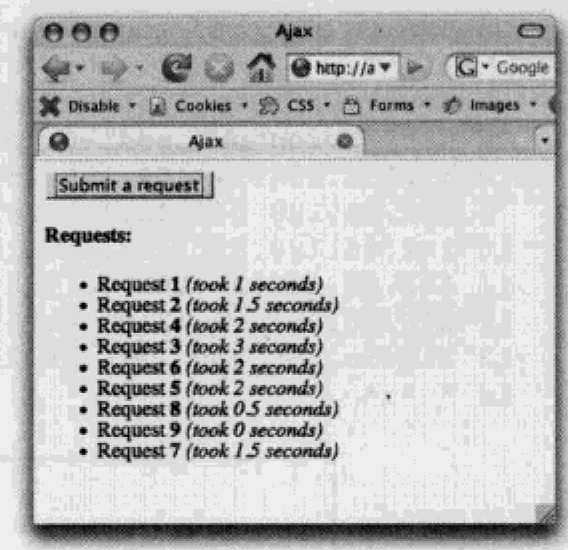



图7-8 一次延迟测试显示从服务器返回的请求列表顺序发生了错乱

如果每次单击按钮的时间间隔超过了3秒钟，那么每次请求可能都会按照发送它们的顺序收到响应，因为通信过程不会超过3秒钟。但是，如果你连续多次快速地发送了多个请求，那么很有可能会在收到第2秒的请求之后才收到第0.5秒的请求。在这种情况下，由于第二个请求的响应会在第一个请求之前结束并完成，所以响应的顺序就会颠倒过来。这个例子示范了异步请求的实际效果——它们几乎同时发生，但却不会同步而且次序颠倒。

### 1. 延迟决定胜利者

在赛跑过程中，无论是服务器还是脚本都无法使某个请求更快地得到响应。请求过程中的延迟会出现在几个阶段，而其中的多个阶段都是你无法控制的。图7-9展示了我们都很熟悉的Web应用程序中的基本通信工作流。

所有通信过程都将遵循如下相同的模式：

(1) 客户端计算机对服务器发起获取或修改信息的请求。

(2) 将请求通过一条计算机网络发送到服务器。

(3) 服务器处理请求。

(4) 服务器将对请求的响应通过另一条计算机网络发送回客户端计算机。

在这个请求/响应循环的过程中，每个阶段都存在外部因素的影响，因此在任何时候，一个阶段可能会导致整个循环减慢速度并造成延迟，从而使Web应用程序增加等待时间。大量的因特网通信会导致请求和响应的速度都变慢，因为这两个过程都需要通过网络实现。而且，服务器上的另外一个过程或者请求自身的要求也可能耽搁服务器的响应。

为了看清你的计算机发送的请求到达服务器的过程，可以在计算机和服务器之间作一次简单的traceroute[②]。例如，下面是在我家里的MacBook与本书网站之间进行的一次路由追踪测试的结果：

```
macbook:~ jeffreysambells$ traceroute advanceddomscripting.com
traceroute to advanceddomscripting.com (216.16.243.44), 64 hops max, 40 byte packets
 1 192.168.1.1 (192.168.1.1) 7.318 ms 0.908 ms 1.025 ms
 2 192.168.2.1 (192.168.2.1) 1.523 ms 1.553 ms 1.589 ms
 3 64.230.197.224 (64.230.197.224) 10.074 ms 9.635 ms 10.294 ms
 4 dis26-toronto63_vlan101.net.bell.ca (64.230.229.81) 7.740 ms 7.470 ms 7.835 ms
```

① 即延时执行脚本。——译者注

② Window用户可以命令提示符窗口中使用tracert命令。——译者注

www.TopSage.com

```
   5 core3-toronto63-gigabite4-0.in.bellnexxia.net (206.108.107.169) 7.700 ms 7.751 ms ➥
7.870 ms
   6 core1-toronto63_pos0-1.net.bell.ca (64.230.242.94) 8.853 ms 8.314 ms 8.530 ms
   7 dis1-torontoxn_pos1-0.net.bell.ca (64.230.229.46) 8.182 ms 8.355 ms 8.078 ms
   8 69.156.254.94 (69.156.254.94) 8.112 ms 8.686 ms 7.986 ms
   9 142.46.128.6 (142.46.128.6) 8.967 ms 8.764 ms 8.728 ms
  10 142.46.128.82 (142.46.128.82) 12.384 ms 13.074 ms 12.263 ms
  11 142.46.130.2 (142.46.130.2) 13.644 ms 12.571 ms 12.729 ms
  12 216.16.255.90 (216.16.255.90) 12.745 ms 12.564 ms 12.943 ms
  13 216.16.255.210 (216.16.255.210) 12.529 ms 12.542 ms 12.552 ms
  14 * * *
```

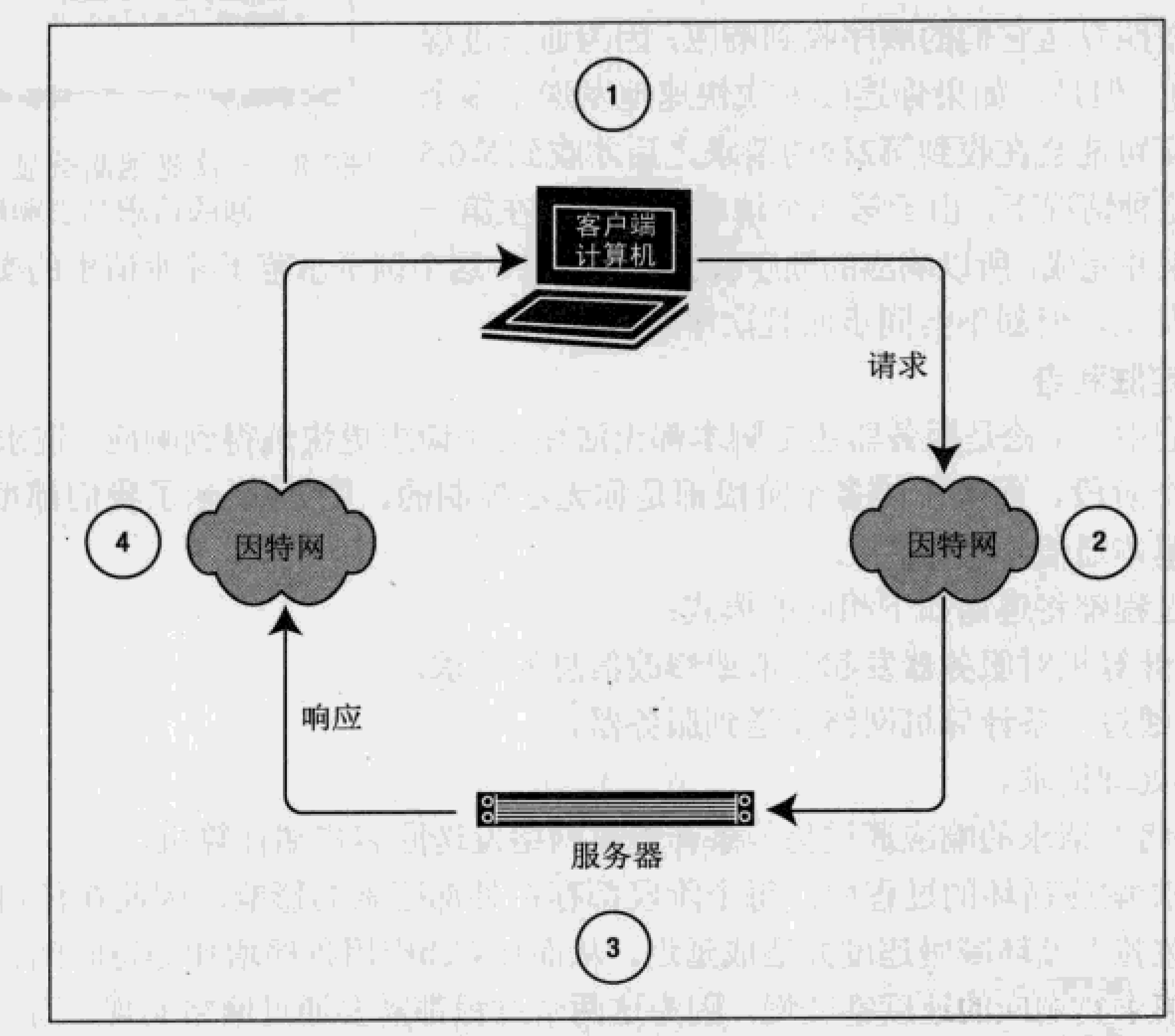



图7-9　基本的因特网通信流程所展示的请求/响应循环

第14行没有响应是因为网站所在的主机安装了防火墙的缘故，但这也展示了完整的路径。

这个路径中大约有14处可能会出现问题的地方。而且，如果我运行`traceroute advanced-domscripting.com`命令几次，通常都会得到相同的路径。因为每台交换机或服务器一般都认为这是最佳路径。假如由于某种原因，这条路径中某个点上面的通信量过大，或者其中一点断开了连接，那么每个信息包都会在不给出任何警示的情况下转向其他路径。对于从网站到我的MacBook之间，沿着另外一条相反方向的路径传输的响应信息而言，情况也是如此。

有了这些可能导致延迟的点，那么多个请求发生一点不协调的现象也就不足为奇了。假设你实现了某种简单的搜索建议框，它会在你输入关键词时通过Ajax请求取得建议项，而建议的结果就有可能发生次序颠倒的问题。如果用户要搜索我写过的书，并想输入我的名字（Sambells），那

www.TopSage.com

么他们所期望的建议项大概是以下这种模式：

(1) Books by S：“S”的建议项列表。

(2) Books by Sa：“Sa”的建议项列表。

(3) Books by Sam：“Sam”的建议项列表。

(4) Books by Samb：“Samb”的建议项列表。

事实上，前面请求/响应循环中的每个阶段都出现了严重的问题。于是，Books by S可能会在Books by Samb之后才会返回，也就是说程序所提供的建议项不再与用户在搜索框中输入的内容匹配。

这个搜索的例子似乎并不是什么大问题，怎么说？虽然可能会出现几个错误的建议项，但用户还是可以忍受的。然而，如果请求同时也会修改数据库，那可就是个大问题了。比如，你想创建一个可以保存拖放结果的列表，即在每次列表项被修改时都进行一次保存，而每次请求都会假设前一次请求已经修改了服务器上的数据库。如果此时处理请求的次序是错误的，那么列表项的顺序就会乱作一团。而且，服务器中保存的数据与用户在屏幕上所看到的情形也会不一致。不过，也有几个解决这一问题的方案。

**2. 处理异步请求**

处理请求/响应循环中的延迟问题有很多不同的方式。以下是其中几种主要的思路，但绝不是全部方案。

**置之不理**。对这个问题置之不理是最简单的方案，也是一种最常见的“实现”。搁置问题并且假设多数情况下自己的程序都能正常运行是毫不费力的，但这显然不是一个可取的方案。比如说，我就不认为多数情况下能够运行的程序就算完成了。

**关掉异步行为**。在Ajax对象中设置`asynchronous=false`是另外一种选择，但这个选择也可以从你的方案清单中划掉。如果你认为在`XMLHttpRequest`请求的`open()`方法中，将`asynchronous`标记设置为`false`（在阻塞模式下运行所有程序）就能解决所有问题，那你就错了。我在前面提到过，如果在`XMLHttpRequest`对象上设置了同步模式，那么它会按照次序处理请求，但它是通过把请求转换为一种更加激进的阻塞模式来实现这一点的。在这种情况下，你的脚本将被迫停止运行直至请求完成，其间可能会因为响应过慢而导致脚本和浏览器被挂起。

如果通过同步阻塞模式向服务器发送了10次处理时间介于1～5秒钟的请求，那么请求总共可能会花10～50秒钟，而且与此同时浏览器将始终处理停顿状态，直至所有这些请求全部完成。而且，在等待请求完成期间，你也无法提供必要的载入反馈或者进行其他交互操作。因此还不如使用传统的工作流——那也只不过需要重载整个页面而已。

**在客户端对请求排队**。排队是另一种可能的方案。与其一次发送多个`XMLHttpRequest`请求，不如等到前一个请求获得响应后再发送下一个。与同步请求相似，如果向服务器发送10个处理时间介于1～5秒钟之间的请求，请求的总时间也会花10～50秒钟，但这时的差别在于请求处于异步模式中，用户可以在某个请求载入期间继续自由地使用页面。

下面是一个用于包装`ADS.ajaxRequest()`方法的简单的`ADS.ajaxRequestQueue()`对象，你可把它添加到ADS库中。这个对象可以防止同一个队列中的请求，在前一次请求收到响应

www.TopSage.com

之前被发送：

```
(function(){

window['ADS'] = {};

……以上是库中已有的内容……

/* 一个复制JavaScript对象的辅助方法 */
function clone(myObj) {
    if(typeof(myObj) != 'object') return myObj;
    if(myObj == null) return myObj;
    var myNewObj = new Object();
    for(var i in myObj) {
        myNewObj[i] = clone(myObj[i]);
    }
    return myNewObj;
}

/* 用于保存队列的数组 */
var requestQueue = [];

/* 为使ADS.ajaxRequest方法启用排队功能的包装对象 */
function ajaxRequestQueue(url,options,queue) {
    queue = queue || 'default';

    // 这个对象将把可选的侦听器包装在另一个函数中
    // 因此，可选的对象必须唯一。否则，如果该方法
    // 被调用时使用的是共享的可选对象，那么会导致
    // 陷入递归中
    options = clone(options) || {};
    if(!requestQueue[queue]) requestQueue[queue] = [];

    // 当前一次请求完成时，需要使用completeListerner
    // 调用队列中的下一次请求。如果完成侦听器已经
    // 有定义，那么需要首先调用它

    // 取得旧侦听器
    var userCompleteListener = options.completeListener;

    // 添加新侦听器
    options.completeListener = function() {

        // 如果存在旧的侦听器则首先调用它
        if(userCompleteListener) {
            // this引用的是请求对象
            userCompleteListener.apply(this,arguments);
        };

        // 从队列中移除这个请求
        requestQueue[queue].shift();

        // 调用队列中的下一项
        if(requestQueue[queue][0]) {
            // 请求保存在req属性中，但为防止它是
```

www.TopSage.com

```
            // 一个POST请求，故也需包含send选项
            var q = requestQueue[queue][0].req.send(
                requestQueue[queue][0].send
            );
        }
    }

    // 如果发生了错误，应该通过调用相应的
    // 错误处理方法取消队列中的其他请求

    // 取得旧侦听器
    var userErrorListener = options.errorListener;

    // 添加新侦听器
    options.errorListener = function() {

        if(userErrorListener) {
            userErrorListener.apply(this,arguments);
        };

        // 由于已经调用了错误侦听器
        // 故从队列中移除这个请求
        requestQueue[queue].shift();

        // 由于出错需要取消队列中的其余请求，但首先要调用
        // 每个请求的errorListener。通过调用队列中下一
        // 项的错误侦听器就会清除所有排队的请求，因为在
        // 链中的调用是依次发生的

        // 检测队列中是否还存在请求
        if(requestQueue[queue].length) {

            // 取得下一项
            var q = requestQueue[queue].shift();

            // 中断请求
            q.req.abort();

            // 伪造请求对象，以便errorListener
            // 认为请求已经完成并相应地运行

            var fakeRequest = new Object();

            // 将status设置为0，将readyState设置为4
            // 就好像请求虽然完成但却失败了一样
            fakeRequest.status = 0;
            fakeRequest.readyState = 4

            fakeRequest.responseText = null;
            fakeRequest.responseXML = null;

            // 设置错误信息，以便需要时显示
            fakeRequest.statusText = 'A request in the queue received an error';

            // 调用状态改变。如果readyState是4，而
            // status不是200，则会调用errorListener
            q.error.apply(fakeRequest);
```

www.TopSage.com

```
            }
        }

        // 将这个请求添加到队列中
        requestQueue[queue].push({
            req:getRequestObject(url,options),
            send:options.send,
            error:options.errorListener
        });

        // 如果队列的长度表明只有一个
        // 项（即第一个）则调用请求
        if(requestQueue[queue].length == 1) {
            ajaxRequest(url,options);
        }
    }
    window['ADS']['ajaxRequestQueue'] = ajaxRequestQueue;

    ……以下是库中已有的内容……

})();
```

这个对象与ADS.ajaxRequest()方法的前两个参数相同，但ADS.ajaxRequestQueue()还有一个用于指定队列名称的可选的第3个参数。可以通过这个对象一次运行多个队列，而且如果有一个请求报告错误，那么其余排队的请求都将触发errorListener()方法。实现排队请求与使用ADS.ajaxRequest()方法没什么不同：

```
// 队列1中的请求1
ADS.ajaxRequestQueue('/your/script/',{
    completeListener: funcion() {
        alert(this.responseText);
    }
},'Queue1');

// 队列2中的请求2
ADS.ajaxRequestQueue('/your/script/',{
    completeListener: funcion() {
        alert(this.responseText);
    }
},'Queue2');

// 队列1中的请求3
// 要等到请求1完成
ADS.ajaxRequestQueue('/your/script/',{
    completeListener: funcion() {
        alert(this.responseText);
    }
},'Queue1');
```

在上面的例子中，请求1和请求2（位于不同的队列中）将会在同一时刻以异步方式运行。然而，请求3则会等到请求1完成之后才会运行，因为它们都处于同一个队列中。排队的请求都会在completeListener成功地执行后被触发。要测试这一点，可以运行位于本书源文件chapter7/ajaxRequestQueue中的例子。

www.TopSage.com

**令请求异步但禁用有冲突的功能**。禁用功能可能是避免不协调问题的最常见方法了。当执行某些异步请求时，让用户知道后台在干什么永远都是很重要的。而这通常是通过在请求等待响应期间显示“载入中”等信息或者动画来完成的。在等待期间，用户仍然可以自由地使用他们认为合适的应用程序。不过，假如他们由于急不可耐而在载入完成之前又执行了相同的操作（可能因为他们希望速度再快一点），那么就可能对程序造成潜在的破坏。

除了显示简单的“载入中”信息之外，还可以禁用程序中的某个部件，以防止用户在不耐烦的情况下重复操作。而实现这一点的唯一技巧，就是无论是响应成功，还是发生了错误都要重新启用所禁用的部件。比如说，如果Web应用程序的界面中包含如下提交按钮

```
<input type="submit" id="buttonID">
```

那么，就可以通过简单地使用类似下面这样的代码禁用提交表单的功能：

```
ADS.ajaxRequest('/your/script/',{
    loadListener:function() {
        // 在载入期间禁用按钮
        ADS.$('buttonID').disabled = 'disabled';
    },
    completeListener: funcion() {
        // 当响应成功时启用按钮
        ADS.$('buttonID').disabled = '';
        alert(this.responseText);
    },
    errorListener: function() {
        // 当发生错误时也要启用按钮
        ADS.$('buttonID').disabled = '';
        alert('Oops, please try again: ' + this.statusText);
    }
});
```

此时的请求本身仍然是异步的，但可以通过这种方法来保证应用程序在处理请求期间不会发生意外。因为按钮已经被禁用，而且直至收到响应它也无法接受单击，所以也就不可能再发出冲突的请求了。

这种方法的唯一问题，就是在某些情况下——比如拖放式的界面功能中——如果使用它，结果差不多会与传统的页面重载的工作流一样令人讨厌。因为可拖动的用户界面是为了提供一种易用的，而且像桌面应用程序一样流畅的体验，所以不能在脚本等待响应期间禁用拖动功能，以免强迫用户在每次操作之间都要等待几秒钟。

**提出你自己的解决方案**。提出自己有创意的方案也是切实可行的。本节提到的这些方案当然不是全部可能的方案，而且在这些方案之外也一定会有其他方案。另外，我在此没有提及如何在服务器端保持请求协调有序的问题，因为这超出了本书的范围。

> 如果你提出了其他有创意的方案，可以随时通过本书网站http://advanceddomscripting.com告诉我。而且，如果你同意的话，我也会把你的方案在该网站发表出来。

www.TopSage.com

### 7.2.5　增加资源占用

千万别简单地以为添加Ajax界面会减少应用程序对服务器端资源的占用。而且，如果有占用也是增加资源的占用。不过，如果你的服务器资源多的是，Ajax增加的资源占用也不一定是个问题。然而，要是你不想增加资源占用那恐怕就是个问题了。

当传统Web应用程序过多消耗资源时，也许不会引起等待浏览器下载并呈现页面的用户的注意。但当用户是以类似操作桌面应用程序的方式与页面交互时，他们期望的将是实时呈现结果，而不是等待。如果每次异步请求也需要同样多的资源耗用，那么用户等待页面中某个小元素改变的过程也会变得缓慢起来。并且，在一个页面中有可能会同时发出多个请求，因而某个用户的一次请求相应变成了同时发出的多个请求。由于所有请求都需要耗用同样的资源，所以服务器就需要拿出多倍的资源才能保证应用程序运行。

### 7.2.6　问题解决了吗

如果你能认真考虑如何着手实现自己的Ajax界面，那么有可能会避开本章提到的大多数问题。不过，可别忘了要适当地实现Ajax界面是要投入时间和精力的，因为Ajax不会让开发周期自动加快，也不会让开发过程更加简化。但是，无论付出多少成本，所换来的优势和效用一定也会更大。为此，我们下面就来看一个利用了本章所介绍方法的简单例子。

## 7.3　实例：Ajax 增强的相册

为了示范`ADS.ajaxRequest()`、`ADS.ajaxRequestQueue()`和`ADS.actionPager()`对象的实际应用，我们来看一个非常简单的、可退化的相册实例，即用户无论启用还是不启用JavaScript都可以在这个相册的网页中进行导航。由于Ajax跨站点的限制以及Web应用程序的动态性等原因，这里你只能通过访问http:/advanceddomscripting.com/source/chapter7/browser/，来对这个实例进行体验，该实例的界面如图7-10所示。

> 由于这个实例需要访问服务器端的脚本代码以生成适当的页面和响应，而探索服务器端程序超出了本书的范围，所以我们不会讨论这个实例的内部构造，以免对你理解本实例造成障碍。其中，我们有意对请求和`pager`对象的实现进行了简化，以便能够聚焦于服务器脚本所需的信息，以及为创建最终产品JavaScript与服务器脚本进行交互的过程。而且，作为一款图像浏览器程序，这个实例显然还不够完善，因为对于一个完备的应用程序而言，它至少还需要一个载入状态指示器。完整的服务器端PHP源代码位于chapter7/browser/index.php中，你可以自己配置使用这个脚本，但该脚本需要PHP 5支持。

通过查看下面照片浏览器中一个页面的HTML源代码，并尝试在禁用JavaScript的情况下浏览该页面，你会注意到几个问题：

www.TopSage.com

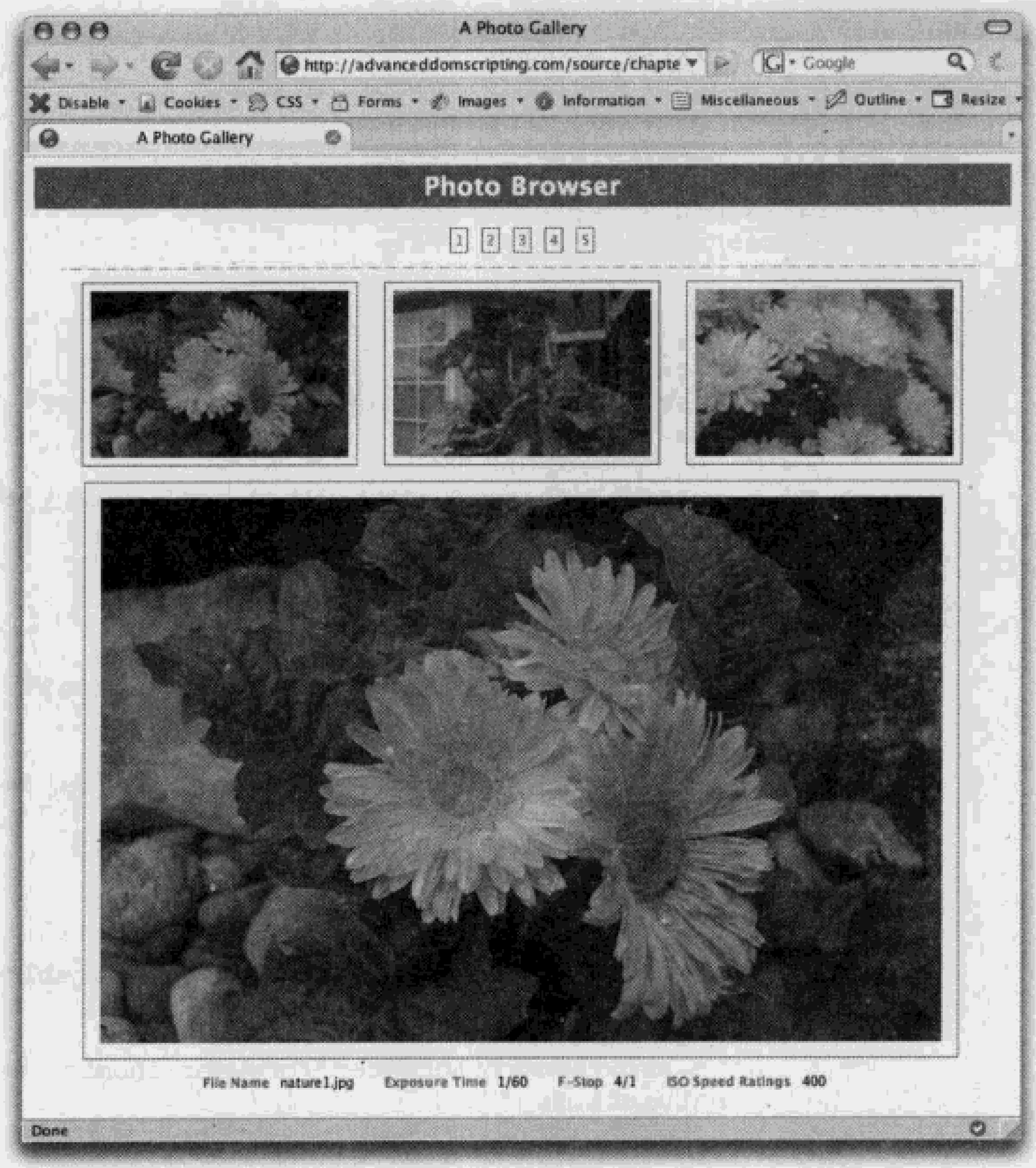



图7-10 一个Ajax增强的相册

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>A Photo Gallery</title>

    <link rel="stylesheet" title="Photo Gallery" type="text/css"
        href="/source/chapter7/browser/browser.css" media="screen">

    <script type="text/javascript" src="/source/ADS-final-verbose.js"></script>
    <script type="text/javascript"
        src="/source/chapter7/browser/browser.js"></script>

</head>
<body>
    <h1>Photo Browser</h1>
    <div id="content">
        <ul id="pages">
            <li><a href="/source/chapter7/browser/page/1/"
                    class="ajaxify">1</a></li>
```

www.TopSage.com

```
            <li><a href="/source/chapter7/browser/page/2/"
                    class="ajaxify">2</a></li>
            <li><a href="/source/chapter7/browser/page/3/"
                       class="ajaxify">3</a></li>
            <li><a href="/source/chapter7/browser/page/4/"
                       class="ajaxify">4</a></li>
            <li><a href="/source/chapter7/browser/page/5/"
                       class="ajaxify">5</a></li>
        </ul>
        <div id="gallery">
            <ul id="list">
                <li id="photo1">
                    <a href="/source/chapter7/browser/photo/nature2"
                            class="ajaxify">
                        <img id="photo1Thumb"
                                src="/source/chapter7/browser/thumbs/ ➥
                            nature2.jpg"
                                          alt="Photo: nature2.jpg"/>
                    </a>
                </li>
                <li id="photo2">
                    <a href="/source/chapter7/browser/photo/nature3"
                             class="ajaxify">
                        <img id="photo2Thumb"
                                src="/source/chapter7/browser/thumbs/ ➥
                            nature3.jpg"
                                          alt="Photo: nature3.jpg"/>
                    </a>
                </li>
                <li id="photo3">
                    <a href="/source/chapter7/browser/photo/natureb1"
                             class="ajaxify">
                        <img id="photo3Thumb"
                                src="/source/chapter7/browser/thumbs/ ➥
                            natureb1.jpg"
                                           alt="Photo: natureb1.jpg"/>
                    </a>
                </li>
            </ul>
            <div id="preview">
                <div id="previewPhotoFrame">
                    <img id="previewPhoto"
                            src="/source/chapter7/browser/photos/nature2.jpg"
                        alt="Photo: nature2.jpg">
                </div>
                <div id="photoInfo">
                    <dl>
                        <dt>File Name</dt>
                        <dd id="photoFile">nature2.jpg</dd>
                        <dt>Exposure Time</dt>
                        <dd id="photoExposure"></dd>
                        <dt>F-Stop</dt>
                        <dd id="photoFStop"></dd>
                        <dt>ISO Speed Ratings</dt>
                        <dd id="photoISO"></dd></dt>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```

www.TopSage.com

首先，你会发现HTML源代码中所有的锚都使用了指向站点根目录下相应文件的绝对URL：

```
<a href="/source/chapter7/browser/page/1/" class="ajaxify">1</a>
```

采用这种格式（称为“优质URL”）的URL的原因很多，包括有利搜索引擎优化、增强浏览器友好程度以及应用程序自身需要等。这个图像浏览器中所有未指向真实存在文件的URL，都会通过Apache的重写规则（RewriteRule）重定向到一个中心位置：

```
RewriteEngine On
# if it's not an existing file
RewriteCond %{REQUEST_FILENAME} !-f
# or an existing directory
RewriteCond %{REQUEST_FILENAME} !-d
# Redirect to index.php
RewriteRule . index.php [L]
```

> 要激活Apache RewriteRule，需要将以上规则放在与index.php位于同一目录下的名为.htaccess的文件（本书提供的源代码中有这个文件）中。假如它不起作用，那可能是因为服务器管理员不允许你覆盖服务器的重写设置。如果是这样，你需要与服务器管理联系，让他帮你在重写规则中添加具体的指令。

如果你愿意，可在自己的应用程序中使用传统的GET方法：

```
<a href="index.php?page=1" class="ajaxify">1</a>
```

但优质URL却能够使稍后注册actionPager()侦听器时，从URL中解析相关信息更方便。其次，你会注意到HTML中的一些锚拥有值为ajaxify的类属性：

```
<a href="/source/chapter7/browser/page/1/" class="ajaxify">1</a>
```

这些都是要在页面载入事件中被识别，而且会被ADS.actionPager方法转换的锚。

此外，你还会注意到整个实例在没有启用JavaScript的情况下也能如期运行。唯一的区别就是在每次请求时页面都会强制重载。这一点不仅对搜索引擎优化和可访问性非常重要，而且actionPager也会在这些现有URL的基础上，再为该应用构建不唐突的、启用Ajax的功能。

最后，你会注意到两种URL类型——页面和照片：

```
/source/chapter7/browser/page/1
/source/chapter7/browser/photo/nature2
```

而每种URL都将引起不同类型的响应：一个会取得新页面，而另一个则会取得一幅具体的照片。

### Ajax化照片浏览器

为增强这个照片浏览器的行为，同时为防止在每次查看新图像时都跳到页面顶部，通过Ajax请求载入页面、图像以及相关的信息是个好主意。而这是通过在照片浏览器的自定义DOM脚本browser.js（如下）中使用ADS库方法实现的：

```
http://advanceddomscripting.com/source/chapter7/browser/browser.js
```

添加Ajax增强的行为要求做到如下两点。

www.TopSage.com

- 通过适当的JavaScript载入事件按照需要处理页面：

```
function updatePhoto(info) {
    ADS.$('previewPhoto').src = info.webHref;
    ADS.removeChildren('photoFile').appendChild(
        document.createTextNode(info.file));
    ADS.removeChildren('photoExposure').appendChild(
        document.createTextNode(info.exposure));
    ADS.removeChildren('photoFStop').appendChild(
        document.createTextNode(info.fStop));
    ADS.removeChildren('photoISO').appendChild(
        document.createTextNode(info.iso));
}

function updateGalleryList(files) {
    // 按照需要修改页面
    var thumb;
    for(var i=0 ; i<files.length ; i++) {
        if((thumb = ADS.$('photo'+(i+1)+'Thumb'))) {
            var li = ADS.$('photo'+(i+1));

            if (files[i]) {
                // 用新图像更新缩略图
                thumb.src = '/source/chapter7/browser/thumbs/' + files[i];
                thumb.title = 'Photo: ' + files[i];
                thumb.alt = 'Photo: ' + files[i];
                ADS.removeClassName(li,'noFile');
                li.getElementsByTagName('A')[0].href = '#photo/'
                    + files[i].split('.')[0];
            } else {
                // 没有缩略图故隐藏它
                thumb.src = '';
                thumb.title = '';
                thumb.alt = '';
                ADS.addClassName(li,'noFile');
                li.getElementsByTagName('A')[0].href = '';
            }
        }
    }
}

ADS.addLoadEvent(function() {

    ADS.actionPager.register('start',function(hash) {
        // 你想要添加的任何启动事件
        // 它会在载入页面且没有任何
        // hash时被调用
    });

    // 照片变化侦听器
    ADS.actionPager.register(
        /photo\/([0-9a-z-]+)\/{0,1}$/i,
        function(hash,photo) {

            // 发送一个排队的ajaxRequest以取得照片
            ADS.ajaxRequestQueue(hash,{
```

www.TopSage.com

```
                // 服务器返回application/json响应
                jsonResponseListener:function(response) {
                    // 通过新列表（如果有）
                    // 更新缩略图导航区
                    updateGalleryList(response.currentPageFiles);
                    // 更新预览图
                    updatePhoto(response.currentPhoto);

                    // 更新文档标题
                    document.title = 'Photo Album Photo '
                        + response.currentPhoto.file;
                }
            },'photoBrowserQueue');
        }
    );

    // 页面变化侦听器
    ADS.actionPager.register(
        /page\/([0-9]+)\/{0,1}$/i,
        function(hash,page) {

            // 按照与照片侦听器相同的思路
            // 发送请求取得新页面信息
            ADS.ajaxRequestQueue(hash,{
                jsonResponseListener:function(response) {
                    updateGalleryList(response.currentPageFiles);
                    updatePhoto(response.currentPhoto);
                    document.title = 'Photo Album Page '
                        + response.currentPage;
                }
            },'photoBrowserQueue');

    });

    // 通过扫描带ajaxify类的链接启动actionPager
    // 同时设置hash的根目录位于browser目录之后
    ADS.actionPager.init('ajaxify','/source/chapter7/browser/');
});
```

- 在服务器端脚本中附加检测请求是否来自ADS.ajaxRequest()方法，并以适当的信息返回一个JSON对象：

```
<?php

……省略的代码……

if($_SERVER['HTTP_X_ADS_AJAX_REQUEST'] == 'AjaxRequest') {
    header('Content-type: application/json');
    //$currentPageFiles =  "'".join("','",$currentPageFiles)."'";

    echo json_encode(array(
        'numPages' => $numPages,
        'currentPage' => $currentPage,
        'currentPageFiles' => $currentPageFiles,
        'currentPhoto' => array(
            'webHref' => $currentPhoto['webHref'],
```

www.TopSage.com

```
            'file' => $currentPhoto['FileName'],
            'exposure' => $currentPhoto['ExposureTime'],
            'fStop' => $currentPhoto['FNumber'],
            'iso' => $currentPhoto['ISOSpeedRatings']
        )
    ));

    die();
}
```

……省略的代码……

```
?>
```

与我们在本章前面讨论的一样，JavaScript载入事件只是注册两个ADS.actionPager侦听器。一个侦听器使用下面的正则表达式匹配页面：

```
/photo\/([0-9a-z-]+)\/{0,1}$/i
```

这个正则表达式会查找hash中以photo/后跟一个包含字母数字的字符串以及一个可选的斜杠结尾。第2个侦听器使用下面的正则表达式匹配照片：

```
/page\/([0-9]+)\/{0,1}$/
```

其作用与前面那个一样，只不过它查找的是page/后跟一个数字。

> 如果你不熟悉正则表达式，我建议你最好掌握这一技术。因为正则表达式对于任何基于字符串的操作和匹配，都是非常强大的工具。而且，多数脚本语言都支持正则表达式。要了解有关正则表达式的更多信息，请参考http://www.regular-expressions.info/。

这两个正则表达式中都使用了捕获圆括号，因此匹配的页码或图像名称会作为第2个参数传递到侦听器中。

接下来，ADS.actionPager开始初始化，它负责转换带有ajaxify类的锚：

```
ADS.actionPager.init('ajaxify','/source/chapter7/browser/');
```

由于同时指定了应用程序的根目录，所以带有hash的URL可以被缩短到只包含该根目录之后的部分。虽然这不是必需的，但使用下面这个路径：

```
http://advanceddomscripting.com/source/chapter7/browser/#photos/page/2/
```

要比使用同样有效的完整路径好一些：

```
http://advanceddomscripting.com/source/chapter7/browser/#/source/chapter7/browser/ ➥
photos/page/2/
```

每个注册的侦听器的功能则相对直观，即它们会通过ADS.ajaxRequestQueue()简单地调用ADS.ajaxRequest()，以便取得针对特定hash的信息：

```
// 照片变化侦听器
ADS.actionPager.register(
    /photo\/([0-9a-z-]+)\/{0,1}$/i,
    function(hash,photo) {
```

www.TopSage.com

```
            // 发送一个排队的ajaxRequest以取得照片及相关信息
            ADS.ajaxRequestQueue(hash,{
                // 服务器返回application/json响应
                jsonResponseListener:function(response) {
                    // 通过新列表（如果有）
                    // 更新缩略图导航区
                    updateGalleryList(response.currentPageFiles);
                    // 更新预览图
                    updatePhoto(response.currentPhoto);

                    // 更新文档标题
                    document.title = 'Photo Album Photo '
                        + response.currentPhoto.file;
                }
            },'photoBrowserQueue');
        }
);
```

这个请求的结果是使服务器生成适当的JavaScript响应：

```
{
    "numPages":5,
    "currentPage":3,
    "currentPageFiles":["nature6.jpg","natureb4.jpg","nature7.jpg"],
    "currentPhoto":{
        "webHref":"\/source\/chapter7\/browser\/photos\/nature7.jpg",
        "file":"nature7.jpg",
        "exposure":1\/30",
        "fStop":"21\/5",
        "iso":"400"
    }
}
```

而文档的结构会通过辅助的`updateGalleryList()`和`updatePhoto()`方法被修改。就这些！

## 7.4　小结

Ajax是一样好东西——它对现有技术进行了出色的整合，而且还允许开发者提供比其他令人讨厌的应用程序平滑得多，也更加有魅力的解决方案。这些解决方案与桌面应用程序越来越相似，但与此同时也带来了很多问题。

在本章中，我们介绍了有关`XMLHttpRequest`对象以及它在Ajax增强的应用程序中常见用法的很多内容。其间，我们创建的`ADS.ajaxRequest()`方法，以及`ADS.ajaxRequestQueue()`方法，将对你构建自己的Ajax应用程序大有助益，但这两个方法并没有解决所有问题。此外，我们还讨论了涉及覆盖请求和浏览器怪癖的同步性问题，例如后退按钮和书签。不同于简单的调试bug和不一致性，你需要预先作出规划才能确保应用程序避免这类问题。

拥有了可行的想法和计划，其实使用Ajax并不难。但你应该在使用一种新概念或新方法之前，确定不是仅仅因为自己会用，或者认为它能让自己的程序看起来更酷。

在下一章，你会看到一种通过适当的Ajax也无法实现的Web应用程序的功能——通过网站上传文件。

www.TopSage.com

第 8 章

# 案例研究：实现带进度条的异步文件上传功能

你可能会对怎么通过第7章的ADS.ajaxRequest()方法上传文件感到困惑。当然，你不能——或者，我应该说，你不能通过它真正实现这一功能。本章的标题的确有点容易让人误解，你大概会认为我在暗示可以通过XMLHttpRequest对象上传文件。

虽然技术上没有限制使用XMLHttpRequest对象上传文件，但由于非常合理的安全方面的理由，浏览器不会允许JavaScript访问除cookie之外的本地文件。因此，当像下面这样在表单中使用文件输入元素时：

```
<input type="file" id="newFile1" name="newFile1" ➥
accept="image/jpeg,image/png" />
```

能获得的与选中文件相关的信息，只有它在本地驱动器中的路径——而且，还是在页面载入完成并选择了文件的情况下。也就是说，不能预先生成该输入元素的值。

事实上，通过XMLHttpRequest对象上传文件的真正局限，不是XMLHttpRequest对象本身，而是无法读取我们想要包含在请求中的文件。比如，可以使用XMLHttpRequest对象随意发送任何头部信息或其他数据，而且也可以发送完整的multipart/form-data类型的POST请求和PUT请求——问题只是我们无法访问自己发送的原始数据。

假如能够访问二进制文件信息，那么通过下面的操作（采用的是multipart/form-data请求）是可以上传文件的：

```
var request = false;
if(XMLHttpRequest) {
    var request = new XMLHttpRequest();
} else if (ActiveXObject) {
    var request = new ActiveXObject('Microsoft.XMLHTTP');
}

if(request) {
    var boundary = '--ExampleBoundaryString';
    var requestBody =
        boundary + '\n'
        + 'Content-Disposition: form-data; name="exampleText"\n\n'
```

www.TopSage.com

```
            + exampleText + '\n\n'
            + boundary + '\n'
            + 'Content-Disposition: form-data; name="myfile"; filename= ➥
    "example.zip"\n'
            + 'Content-Type: application/octet-stream\n\n'
            + escape(BinaryContent) + '\n'
            + boundary;
        request.open('POST', url, true);
        request.setRequestHeader('Content-Type',
            'multipart/form-data; boundary="'
            + boundaryString
            + '"');
        request.setRequestHeader('Connection', 'close');
        request.setRequestHeader('Content-Length', requestBody.length);
        request.send(requestBody);
    }
```

这里的关键是将文件信息保存到BinaryContent变量，进而包含在请求中。可以手工将文件转换为二进制格式，然后再将转换后的信息复制粘贴到文本框中。但是，那样不仅需要许多额外的工作，还要求你具备相应的技术经验。而且，虽然在Firefox这样的浏览器中，可以通过浏览器访问本地文件，但你必须在about:config中编辑特殊的安全设置以允许访问。虽然这样能提供一个纯粹的XMLHttpRequest方案，但它却只对少数浏览器有效。更何况，编辑安全设置增加的复杂性和给人带来的忧虑，也会成为实现上传功能的额外障碍。

TopSage.com

因此，一种简单（也是唯一）的不针对特定浏览器的提交带文件表单的方式，就是使用常规但陈旧的POST方法。如果你希望避免重新载入页面，那么就需要将表单POST请求的目标设置为另外一个框架，例如嵌入到页面中的iframe元素：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <link rel="stylesheet" href="style.css"
    type="text/css" media="screen" /> <title>Form with an iframe as the target</title>
</head>
<body>
<h1>Form with an iframe as the target</h1>
<form action="script/" target="formTarget">
    <div>
        <label for="example">Some text</label>
        <input type="text" name="example" id="example" value="" />
    </div>
    <div>
        <input type="submit" value="Send it to the iframe" />
    </div>
</form>
<iframe name="formTarget" id="formTarget"></iframe>
</body>
</html>
```

在这个例子中，你会注意到DOCTYPE被设置为XMLH 1.0 Transitional。这是因为iframe虽然属于HTML 4.01规范（http://www.w3.org/TR/html401/present/frames.html#h-16.5），但也只是在过渡型（或松散型）DOCTYPE（http://www.w3.org/TR/html401/loose.dtd）中有效。

www.TopSage.com

> 如果在这里使用严格型DOCTYPE（http://www.w3.org/TR/html401/strict.dtd），那么iframe元素以及form中的target都将被视为无效。
>
> 而且，DOCTYPE也会影响到CSS及标记的呈现，因此当在这两种文档类型声明间转换时必须明确这一点。

当iframe接收到POST请求时，其父页面仍然可以自由地运行脚本。这样，就可以在上传文件的同时，使用Ajax和XMLHttpRequest对象来异步跟踪上传的进度，而这正是本章其余内容的核心所在。

在本章中，你不会再向ADS库中添加任何方法了——事实上，你已经在其中添加了本书后面要用到的所有内容。因此，剩下的事就是使用你迄今为止所学的一切，包括最佳实践、JavaScript、DOM、事件、样式以及Ajax为你的Web应用程序来添加一个真正的进度指示器。

## 8.1　信息载入时的小生命

好像表示发生了什么的真正的进度指示器已经超出了Web开发的范围。因此，大多数这类指示器实际上只是一幅GIF动画图像在那里不断地循环，永远都看不到终点，甚至让人怀疑是不是真的有事情发生。图8-1展示了Tango Icon Library（http://tango.freedesktop.org/）提供的一幅载入图标的各个阶段，如果不中断该图像的显示，那么它就会永远螺旋状地转下去。

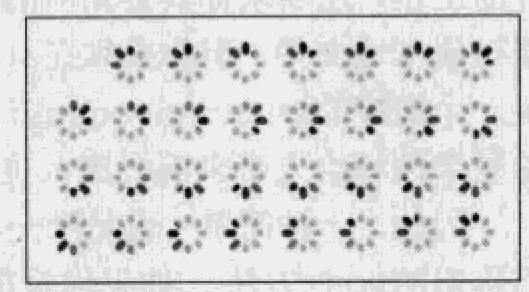


图8-1　开源Tango Icon Library中的进度动画图标的分帧状态

并不是说动画载入图像不好。在某些情况下确实无法说出某个操作需要多长时间，因此“载入中……”就是最好的选择。而在另外一些情况下，如果知道整个过程转眼即可完成，那么显示进度条可能是没有必要的。但是，对于耗时较长的请求（例如文件上传），或者Web服务器上需要花费一定时间的处理过程，都可以利用XMLHttpRequest对象的异步特点，来显示真正的实时进度指示器。

进度条的概念相当简单，只需确定两个关键的问题就可以实现进度条：

- 最终目标
- 监视进度向目标移动的能力

多数情况下，问题都出现在监视进度的能力方面。在XMLHttpRequest对象出现之前，确实没有什么好办法通过查询服务器知道处理操作的进度。虽然可以使用第7章提到的<script>标签的替代方案，但那种办法对开发人员并不非常友好。因此，如果知道某个操作需要一些时间，最好的办法就是显示“这可能要花几分钟时间”，从而提醒人们不要在操作完成之前刷新页面或者单击其他链接。另外，像文件上传这样需要根据因特网连接速度及上传文件的尺寸来估算时间的操作，还可能会导致其他的问题。

除了浏览器中的技术性障碍之外，在服务器上记录进度则比较简单。只需一些共享的存储空间（例如一个文件），在进度开始时存储最终目标，并在处理操作过程中存储接近目标的进度即

www.TopSage.com

可。例如，如果服务器代码中有一个for循环要花费特别长的时间，那么可以像下面这些伪代码所示的那样，通过为每次迭代进行计数来存储操作进度接近终点的过程：

```
count = 200;
storeProgress('End goal is 200');
for( i = 0 ; i < count ; i++) {
    storeProgress ('I am working with number i. ➥
Only count-i left to go!');
    // 进行某些密集性的操作
}
storeProgress('I have reached 200 and I'm done');
```

这并非真正有效的代码，但它可以示范我们所说的概念。只要服务器上某个附加的过程（比如一次XMLHttpRequest请求）能够获得共享的进度信息，那么计算完成百分比和适当地显示这些数据就是小事一桩了。

在DOM脚本中，可以通过使用setInterval()和ADS.ajaxRequest()方法，周期性地查询服务器来检查这一状态：

```
// 每秒钟检查一次
var watcher = function() {
    ADS.ajaxRequestQueue('/getProgressScript/',{
        jsonResponseListener(response) {
        if(response.done){
                // 你的代码
            } else {
                // 更新进度条
                ADS.setStyle('progressBar',{
                    'width' : (response.current/response.total) + '%';
                });
                // 等待1秒钟并发送另一次请求
                setInterval(watcher,1000);
            }
        }
    }
}
// 初始化第一次请求
watcher();
```

要完成这个例子还需要许多工作，但其中基本的要素已经都具备了：

(1) 每1秒钟取得一次响应。

(2) 检查进度是否完成。

(3) 如果完成，继续执行代码。

(4) 否则，更新进度条并发送另一次请求。

这种技巧不仅适用于执行得较慢的脚本，适用于监视信用卡认证过程，也适用于上传文件——而这正是本章的中心话题。

## 在服务器上处理上传

对于上传文件而言，最大的技术障碍就是当POST请求到达服务器时如何跟踪其进度。DOM

www.TopSage.com

脚本对于整个过程的这一部分无能为力，而且也不是本书的焦点。然而，通过添加基于Ajax的进度条来渐进增强令人讨厌的表单，显然属于DOM脚本的范畴，但只有同时整合这两个部分才能实现进度条的功能。

对于服务器端的部分，由于本书源文件中已经包含了必要的文件，并且也可以从http://advanceddomscripting.com中下载到它们，所以这里不会讲述其中的细节。本章只讨论对服务器端脚本而言必要的输入和输出，你可以选择使用本书提供的一种版本，也可以编写你自己的版本。

由于多数服务器端脚本编程语言都在取得完整的请求后才开始执行，所以没有办法周期性地存储请求的进度。但对于Perl以及PHP 5.2（http://php.net）与最新的APC扩展（http://pecl.php.net/apc）的结合来说则另当别论。

> 如果不能通过文件上传部分上传文件，可以使用chapter8/start/simulation/中（以PHP编写）的模拟脚本，使用该脚本也可以完成本章的学习。模拟脚本中设定的请求是20次，因而上传持续的时间都一样，并且文件实际上也不会被存储到服务器中。虽然模拟脚本仍然需要PHP环境，但PHP 4和5都没问题，而且也不需要安装APC扩展。

在查询Perl脚本时，Perl会在请求传输完成之前的请求起点处开始执行。这样我们就可以手工处理请求并从中提取必需的文件——也就是说，有机会存储所有进度存储信息。对于PHP 5.2来说，APC缓存扩展会在请求的起点处开始执行，并且会在`POST`请求中包含一个特殊隐藏字段的情况下自动跟踪接收到的上传文件。

> PHP5.2/APC方案需要APC 3.53或更高版本的支持，因为必须在`php.ini`中启用`apc.rfc1867`。要了解更多信息，请参考http://viewcvs.php.net/viewvc.cgi/pecl/apc/INSTALL?revision=3.53& view=markup。

这些服务器端脚本的作用都一样，即存储来自`POST`请求的文件并报告相应的进度。脚本的代码会直接取得一个只包含文件输入和少量隐藏字段的`POST`请求。当通过`POST`方法向脚本发送一个传统的请求时，脚本会处理其中的文件并告诉浏览器重定向回原始的页面。这样，即使在JavaScript被禁用的情况下，上传过程也能够完成——尽管没有那么优雅。但是，如果在请求中发现了适当的头部信息，脚本则会以下面这个JSON对象的形式返回进度报告：

```
{
    "total":845,                      // 以字节计的请求大小
    "current":845,                    // 已经处理的字节数
    "currentFileName":"file.jpg",     // 处理中的文件名
    "currentFieldName":"newFile3",    // 处理中的字段
    "filesProcessed":[],              // 包含所有被处理文件的数组
    "error":'',                       // 错误消息（如果有）
    "done":1                          // 0未完成，1完成
    "debug":"message"                 // 为开发准备的调试信息
};
```

www.TopSage.com

如果你希望在表单中包含更多信息（你很可能愿意这么做），可以按照需要来修改这些脚本，以便在存储文件的同时接受并存储更多信息。作为另外一种选择，还可以将这个工具中负责上传的部分转移到页面中一个单独的表单上，并通过组合本章的脚本以及你在第6章中所学的方法，构建一个完整的Ajax文件浏览器。

> 出于安全考虑，我在脚本中作了限制，因此脚本只能接受.jpg或.png文件，并且图像大小不能超过200 KB。当然，这并不是上传程序的限制，只是为防止恶意文件进入你的系统。永远也不要允许用户上传可能会在服务器上执行的潜在恶意文件，而且也要始终做到对上传的文件进行适当的过滤和验证。

**有魔力的字符串**

PHP服务器端解决方案依赖于APC扩展及其在多个服务器进程间提供共享缓存的能力。APC 3.5引入了一个特殊的选项，该选项只有当专门的APC_UPLOAD_PROGRESS键在文件上传之前存在于POST请求中时，才会自动跟踪POST请求中文件的上传进度：

```
<form action="script/" target="formTarget">
    <input type="hidden" name="APC_UPLOAD_PROGRESS"
        value="This_Should_Be_Unique"/>
    ……省略的代码……
</form>
```

相同的唯一值可以在后续的请求中用于识别相应的进行程。为方便起见，在使用Perl的服务器端脚本中，也使用了相同的APC_UPLOAD_PROGRESS变量，因此你的DOM脚本对于服务器而言不必考虑任何针对性。

## 8.2 起点

在本章剩下的内容中，我们将专注于构建不唐突的DOM脚本。作为一个起点，你可以从本书提供的源文件中选择一种方案：

- chapter8/start/php5-APC/
- chapter8/start/perl/
- chapter8/start/simulation/

> 由于这个例子的服务器端部分不能在常规的客户端中运行，所以需要一台服务器来运行相应的脚本语言——除非你的客户端计算机中安装了适当的Web服务器软件。

其中每个文件夹中都包含一些文件。例如，php5-APC文件夹中包含如下文件：

```
start/php5-APC
    /actions
        /index.php
    /index.php
    /load.js
    /style.css
```

www.TopSage.com

```
/uploader.js
/uploads
```

除了运行例子的服务器端语言之外，每个起点方案都是相同的。actions/子文件夹中的index文件中包含着我们前面讨论的服务器端脚本。之所以如此设置目录结构，是为了每个版本中的uploader.js文件（你将在其中编写代码）都通过相同的URL引用服务器端脚本：

```
actions/?var=value
```

对于php5-APC和Perl方案而言，唯一的附加条件就是要修改上传文件夹的权限。由于这两个版本的例子会将上传的文件储存在uploads/文件夹中，因此该文件夹对于运行Web服务器的用户而言必须是可写入。在Apache/Linux搭配的设置中，可以在起点文件夹中运行下列命令来修改权限（其中的username和groupname要以你用来运行Web服务器的用户名来代替）：

```
chown username.groupname uploads
```

如果你对这些服务器端的文件有疑问，请到http://advanceddomscripting/help/中查询更多的帮助信息。

在起点文件就绪之后，就可以着手显示某些进程了——注意，这里是一语双关。

## 8.3 完成整合：上传进度指示器

看一看你选择的起点方案中的index文件，就会发现它只是从uploads文件夹中取得文件列表，并输出一个相对简单的XHTML 1.0过渡型页面。下面就是php5-APC版中的index.php文件的代码：

```
<?php

// 循环遍历uploads文件夹
// 以便取得已经上传的文件
$uploads = new DirectoryIterator('./uploads');
$files = array();
foreach($uploads as $file) {

    // 跳过，并……
    if(!$file->isDot() && $file->isFile()) {

        // 添加到数组。稍后，该数组
        // 将在HTML中被连接起来
        $files[] = sprintf(
            '<li><a href="uploads/%s">%s</a> <em>%skb</em></li>',
            $file->getFilename(),
            $file->getFilename(),
            round($file->getSize()/1024)
        );
    }
}

// 输出页面
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

www.TopSage.com

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Image Uploader with Progress (php5-APC)</title>

    <!-- Inclue some CSS style sheet to make
    everything look a little nicer -->
    <link rel="stylesheet" type="text/css"
        href="../../../shared/source.css" />
    <link rel="stylesheet" type="text/css"
        href="../../chapter.css" />
    <link rel="stylesheet" type="text/css" href="style.css" />

    <!-- Your ADS library with the common JavaScript objects -->
    <script type="text/javascript"
        src="../../../ADS-final-verbose.js"></script>

    <!-- Progress bar script -->
    <script type="text/javascript" src="uploader.js"></script>

    <!-- load script -->
    <script type="text/javascript" src="load.js"></script>

</head>
<body>
    <h1>Image Uploader with Progress (php5-APC)</h1>
    <div id="content">
        <form action="actions/" enctype="multipart/form-data"
            method="post" id="uploadForm">

            <fieldset>
                <legend>Upload a new image</legend>
                <p>Only jpg/gif/png files less than 100kb allowed.</p>
                <div class="fileSelector">
                    <label for="newFile1">File 1</label>
                    <input type="file" id="newFile1" name="newFile1"
                        accept="image/jpeg,image/gif,image/png"/>
                </div>
                <div class="fileSelector">
                    <label for="newFile2">File 2</label>
                    <input type="file" id="newFile2" name="newFile2"
                        accept="image/jpeg,image/gif,image/png"/>
                </div>
                <div class="fileSelector">
                    <label for="newFile3">File 3</label>
                    <input type="file" id="newFile3" name="newFile3"
                        accept="image/jpeg,image/gif,image/png"/>
                </div>
                <input id="submitUpload" name="submitUpload"
                    type="submit" value="Upload Files" />
            </fieldset>

        </form>

        <div id="browserPane">
            <h2>
                <span id="fileCount">
                    <?php echo count($files); ?>
```

www.TopSage.com

```
                </span>
                Existing Files in <em>uploads/</em>
            </h2>
            <ul id="fileList">
                <?php echo join($files,"\n\t\t\t\t"); ?>
            </ul>
        </div>
    </div>
</body>
</html>
```

对于DOM脚本而言，HTML标记如何组织并不重要。你要创建的脚本会对任何带有文件输入元素的表单进行修改。在输出的index.html文件中不会包含除了显示页面所必需的元素之外的任何无关标记。其中既没有iframe，也没有用于显示进度条的特殊标记——有的只是毫无修饰的必要元素。

此时此刻，uploader.js文件中包含着尚未修改的代码。如果你在浏览器中打开相应页面并上传几幅JPEG或PNG图像，该页面会在刷新之后上传这些文件。除了浏览器内置的一些功能（相当有限）页面中不会有任何进度指示器。这个结果与你禁用JavaScript后在完整版的例子中看到的完全一样。同样，如果上传的是一个不符合要求的文件，那么你会看到在没有任何警告的情况下，请求仍然会被处理，如图8-2所示。

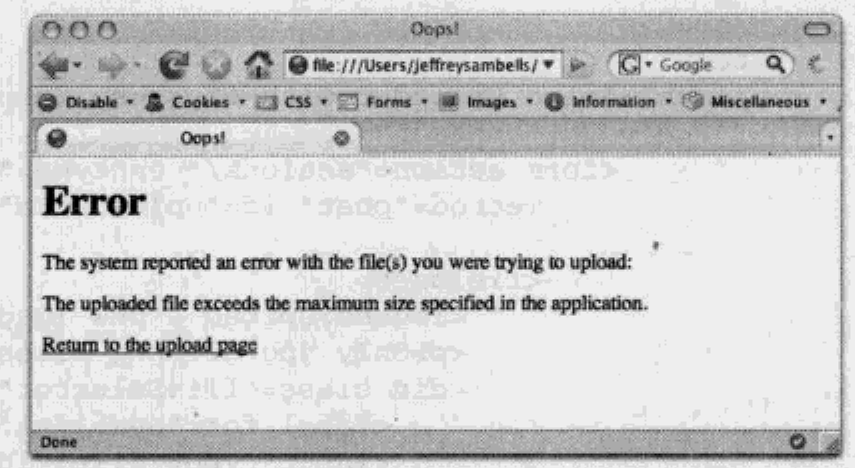



图8-2　表明文件类型不正确的错误页面

理想情况下，如果你的DOM脚本有效，那么除了提供上传文件的进度指示之外，还可以通过包含表单验证功能来避免这些错误。

但是，不需要编辑主index文件中的任何代码，因为我们要做的一切都只涉及uploader.js这一个文件。现在就来看看这个文件。

## 8.3.1　addProgressBar()对象的结构

通过浏览以下脚本代码，你会看到其中已经包含了对象的初始结构、每个待完成方法的必要代码以及每一阶段的注释文本：

```
function verifyFileType(fileInput) {
    if(!fileInput.value || !fileInput.accept) return true;
    var extension = fileInput.value.split('.').pop().toLowerCase();
    var mimetypes = fileInput.accept.toLowerCase().split(',');
    var type;
    for(var i in mimetypes) {
        type = mimetypes[i].split('/')[1];
        if(type == extension || (type=='jpeg' && extension=='jpg')) {
            return true;
        }
    }
    return false;
```

www.TopSage.com

```
}

var addProgressBar = function(form,modificationHander) {

    // 检查表单是否存在

    // 查找所有文件输入元素

    // 如果没有文件输入元素则停止脚本

    // 添加change事件以基于MIME类型验证扩展名

    // 为上传而附加目标iframe元素
    // 在IE中，不能像下面这样通过DOM设置name属性，例如：
    // var uploadTargetFrame = document.createElement('iframe');
    // uploadTargetFrame.setAttribute('id','uploadTargetFrame');
    // uploadTargetFrame.setAttribute('name','uploadTargetFrame');
    // 为解决这个问题，需要创建一个div并使用其innerHTML属性
    // 从而确保在IE和其他浏览器中都能正确地设置name属性


    // 将表单的target属性修改为新ifarme元素
    // 这样可以避免页面重载

    // 创建一个唯一的ID以跟踪上传进度

    // 为APC_UPLOAD_PROGRESS键添加这个唯一ID。
    // 这个字段必须添加到文件输入字段之前，以便
    // 服务器首先取得该键并触发存储进度信息的操作。

    // 创建进度条的不同部分

    // 进度条

    // 内部的背景容器

    // 检查已有的定位点
    // 必须是带有progressContainer类的span元素

    // 如果该定位点不存在则创建一个并将其添加到表单中
    // 设置容器为块级元素

    // 添加进度条的其余部分

    // 同时也添加一个进度信息显示区域

    // 创建一个将由后面的进度监视方法使用的
    // 私有方法，以方便更新进度条和相应信息

    // 从0%和waiting开始初始化进度条

    // 为表单添加提交事件侦听器，用于
    // 验证表单信息和更新进度条
}
```

在为这个对象中的每个方法编写代码时，我们将会看到其中所有的细节。

www.TopSage.com

### 8.3.2 载入事件

为了在页面载入时进行必要的初始化，index.php[①]输出的HTML文件中也包含了相应的load.js脚本。如果你看到这个载入脚本，就会发现这个不唐突的载入事件非常简单，因为所有关键的处理代码都包含在uploader.js中：

```
ADS.addEvent(window,'load',function(W3CEvent) {
var fileList = ADS.$('fileList');
    // 按照需要修改uploadForm
    addProgressBar('uploadForm',function(response) {
        var files = response.filesProcessed;
        for(var i in files) {
            // 跳过空文件
            if(files[i] == null) continue;

            // 创建一个新的文件列表元素
            var li = document.createElement('li');
            var a = document.createElement('a');
            a.setAttribute('href','uploads/' + files[i]);
            a.appendChild(document.createTextNode(files[i]));
            li.appendChild(a);
            fileList.appendChild(li);
        }
        // 更新文件计数器
        var countContainer = ADS.$('fileCount');
        ADS.removeChildren(countContainer);
        var numFiles = fileList.getElementsByTagName('LI').length;
        countContainer.appendChild(document.createTextNode(numFiles));
    });
});
```

以上代码只为window载入事件注册的侦听器调用了一个方法，以便初始化进度条：

```
addProgressBar('uploadForm',function(response) { ... });
```

其中，第2个参数中提供的匿名函数负责在上传成功后操作DOM文档——在这里，它会将上传的文件（名）附加到页面中的文件列表内，同时更新numFiles元素中的数字。

> 要想在不同的文档中使用uploader.js脚本，需要重新规划载入事件，以便按照需要修改标记。如果你觉得这种情形很普遍，那么还可以对用于显示文件列表的标记进行标准化，并将这些更改直接包含在addProgressBar()对象中。

### 8.3.3 addProgressBar()对象

接下来，就是要编写uploader.js文件中的addProgressBar()对象。

为了保证未来的可重用能力并为开发人员提供便利，addProgressBar()[②]需要你刚才看到的两个参数：

① 原文index.html有误。——译者注

② 原文addUploadProgress有误。——译者注

www.TopSage.com

- 一个是对要修改的表单的引用。
- 另一个是在提交成功后负责操纵文档结构的处理方法。

本例的index文件生成的HTML页面的底部提供了一个文件列表。你应该学会总是提前考虑并将脚本设计得尽可能会被重用，而提取出生成和更新这个列表的代码，可以使同样的方法也能够用于标记完全不同的另一个文档中。

如果你浏览了addProgressBar()对象结构中的注释，那么就会看到如下要完成的任务列表：

(1) 在给定的表单中查找文件输入字段。

(2) 为每个文件输入字段添加事件侦听器，以便验证选中的文件类型是否正确。

(3) 创建一个<iframe>元素，并将其添加到文档中，以便表单可以将目标设置这个<iframe>。

(4) 添加一个隐藏的元素，以便启用并跟踪进程。

(5) 以对开发人员友好的方式创建进度条元素。

(6) 为表单添加提交事件侦听器，以便在提交之前验证表单。

(7) 初始化必需的进度跟踪信息。

(8) 在请求成功或失败的情况下修改页面。

为了方便开发人员使用，我们从输入开始介绍。在本例中，addProgressBar()只接收一点信息——要操纵的表单ID。因此，可以将该信息传递到第1章中构建的ADS.$方法中以取得相应对象的引用。如果不存在该对象，整个过程即悄然告终：

```
// 检查表单是否存在
if(!(form = ADS.$(form))) { return false; }
```

如果你愿意，可以在此时弹出一个警告框告诉用户表单不存在，而悄然地失败会让载入事件还存在于动态生成的可能不包含表单的页面中。这样，页面会在存在相应表单时得到增强，而在其他情况下会被忽略。

#### 1. 修改文件输入字段

文件输入字段在DOM树中的实际位置并不重要，因为我们只想为它们添加几个事件侦听器。由于无需知道它们的位置，所以可以简单使用getElementsByTagName()方法来取得表单中所有的<input>元素，然后迭代这些元素以检查特殊的file类型：

```
// 查找所有文件输入元素
var allInputs = form.getElementsByTagName('INPUT');
var input;
var fileInputs = [];
for(var i=0 ; (input = allInputs[i]) ; i++) {
    if(input.getAttribute('type') == 'file') {
        fileInputs.push(input);
    }
}

// 如果没有文件输入元素则停止脚本
if(fileInputs.length == 0) { return false; }
```

在取得了包含在fileInputs中的输入元素列表后，可以通过执行一些错误检查来使表单对

用户更加友好。

可以为文件输入元素指定accept的属性，其中包含以逗号分隔的允许上传的MIME类型列表。大多数浏览器都会忽略这个信息，因此即使在其中指定了image/jpeg，仍然可以选择并上传其他类型的文件，并且浏览器也不会给出任何提示。而这也正是在服务器端仍然需要检查文件类型，以拒绝非法文件的重要原因。不过，至少我们还可以对文件输入字段中所选文件的扩展名，与期望的MIME类型是否匹配进行验证，从而提示用户他们要做的事不会成功。

为此，我们在uploader.js文件的一开始，包含了一个verifyFileType()方法：

```
function verifyFileType (fileInput) {
    if(!fileInput.value || !fileInput.accept) return true;
    var extension = fileInput.value.split('.').pop().toLowerCase();
    var mimetypes = fileInput.accept.toLowerCase().split(',');
    var type;
    for(var i in mimetypes) {
        type = mimetypes[i].split('/')[1];
        if(type == extension || (type=='jpeg' && extension=='jpg')) {
            return true;
        }
    }
    return false;
}
```

这个方法以文件输入元素作为参数，验证所选文件的扩展名是否与accept属性中的MIME类型匹配。该方法只对MIME类型中第2部分是扩展名的情况有效，例如：

```
image/png
```

和

```
Filename.png
```

对于JPEG文件，其MIME类型是image/jpeg，而其扩展名通常简写为.jpg。该方法中添加了检查这种特殊情况的代码。如果你希望增加允许上传的文件类型，那么必须要修改这个方法。

然后，在addProgressBar()方法中，添加下列循环以便为fileInputs数组中的每个文件输入元素创建change事件侦听器：

```
// 添加change事件以基于MIME类型验证扩展名
for(var i=0 ; (fileInput = fileInputs[i]) ; i++) {
    // 使用change事件侦听器进行文件类型检查
    ADS.addEvent(fileInput,'change',function(W3CEvent) {
        var ok = verifyFileType(this);
        if(!ok) {
        if(!ADS.hasClassName(this,'error')) {
            ADS.addClassName(this,'error');
        }
        alert('Sorry, that file type is not allowed. Please ➥
select one of: ' + this.accept.toLowerCase());
        } else {
            ADS.removeClassName(this,'error');
        }
    });
}
```

www.TopSage.com

当文件输入元素中的文件路径改变时，change事件侦听器就会被调用并根据MIME类型来验证扩展名。而且，在必要情况下，还会修改输入元素的类名以表示存在错误。正如我们在第5章所讨论过的，通过修改类名可以轻易地将视觉外观与DOM脚本分离。

只要看一看style.css文件，就会在Progress bar specific classes that are defiend by the addProgressBar() method（由addProgressBar()方法定义的针对进度条的类）中，发现与错误的类名关联的error类的样式规则[1]：

```
input.error {
    background-color: red;
    color: white;
}
```

如果文件名中存在确认的错误，那么文件输入元素的class属性中就会增加一个error类，而页面也将被修改为图8-3所示的外观：

```
<input class="error" type="file" id="newFile1" name="newFile1" ➡
accept="image/jpeg,image/gif,image/png"/>
```

如果试着浏览一下到目前为止的页面，将会发现即使是这些简单的增强，依然能够从整体上提升页面的优雅性和功能性，而且全部都是不唐突的。虽然表单仍然能够提交并且也会刷新页面，但至少你能在选择了错误的文件类型时得到一点反馈。

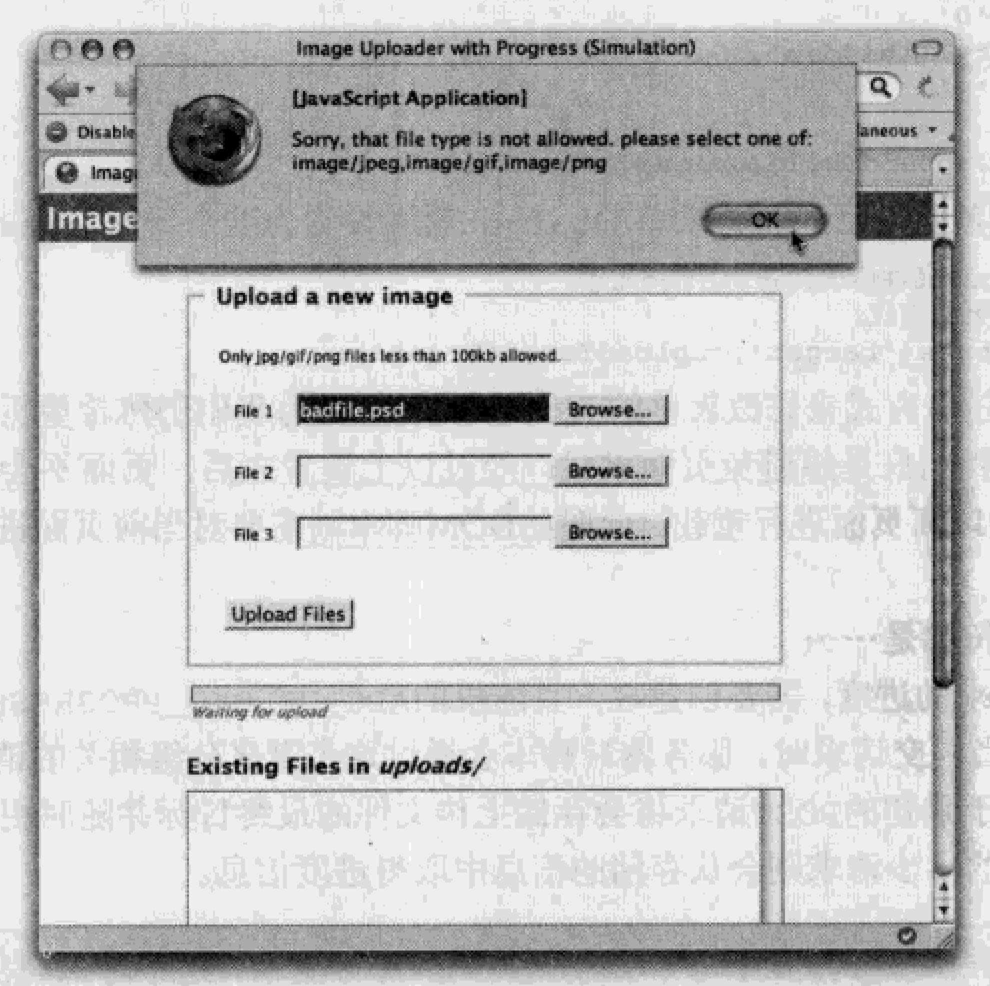



图8-3 当上传文件却选择了错误的文件类型时得到的错误提示

① 原文此处的代码与style.css中的规则不相符，现为正确的代码。——译者注

www.TopSage.com

### 2. 重定向表单

接下来的挑战，是我们前面所讨论的将表单重定向到嵌入的iframe中，以便在上传文件时避免页面重载。理想的情况下，可以使用document.createElement()创建一个新的iframe元素，但现实中却并非那么容易。在IE中，当修改以这种标准方式创建的iframe元素的name属性时，不会起作用。因此，也就不能将表单的目标设定为该iframe元素。为解决这个问题，必须在此使用innerHTML，通过它才能创建想要的对象。这同时也意味着需要一个盛放iframe的容器，所以还要创建一个<div>元素：

```
// 为上传而附加作为目标的iframe元素
// 在IE中，使用DOM方法不能适当地设置其name属性
// 例如：
// var uploadTargetFrame = document.createElement('iframe');
// uploadTargetFrame.setAttribute('id','uploadTargetFrame');
// uploadTargetFrame.setAttribute('name','uploadTargetFrame');
// 为解决这个问题，需要创建一个div然后再使用innerHTML属性，
// 这样才能保证在IE和其他浏览器中都正确地设置其name属性
var uploadTargetFrame = document.createElement('div');
uploadTargetFrame.innerHTML = '<iframe name="uploadTargetFrame" ➥
id="uploadTargetFrame"></iframe>';
ADS.setStyleById(uploadTargetFrame,{
    'width':'0',
    'height':'0',
    'border':'0',
    'visibility':'hidden',
    'zIndex':'-1'
});
document.body.appendChild(uploadTargetFrame);
```

当iframe元素就位之后，再将表单的target属性设置为这个新的iframe元素的name：

```
// 将表单的target属性修改为新ifarme元素。
// 这样可以避免页面重载。
form.setAttribute('target','uploadTargetFrame');
```

此处不需要以任何方式来修改表单的action属性。因为我们仍然希望页面像过去一样能够真正被提交，这里我们只是想避免页面重载。经过以上设置之后，页面不会再重载了，而新的iframe元素中则会以新页面进行重载。此时的DOM脚本则需要对当前页面进行修改，以反映适当的变化。

### 3. 有魔力的字符串是……

为跟踪POST请求的进度，需要创建唯一且随机的APC_UPLOAD_PROGRESS键，并将其放到每一次请求中发送。当提交请求时，服务器端脚本会通过这个键来分组相关的请求，以便存储和取得适当的信息。对于最初的POST请求将会存储上传文件的最终目标并随时更新处理进度，而对后来的带有相同键的异步请求则会从存储的信息中取得进度信息。

> 正如我们前面所提到的，在PHP和APC插件的组合方案中，这个特殊的键必须作为原始的POST请求的一部分，并且必须要叫做APC_UPLOAD_PROGRESS。为方便起见，这一脚本的Perl版本和模拟脚本中也都使用了相同的字段名称，这样服务器端脚本就能做到可互换了。

www.TopSage.com

然后，在addProgressBar()方法中，需要为此提前在表单中准备一个新的隐藏字段。此时的关键之处在于，这个隐藏字段必须位于文件输入字段之前，以便服务器在处理文件之前先取得这个特殊的键。此外，还必须将这个隐藏字段命名为APC_UPLOAD_PROGRESS，这个字段的值也必须是一个绝对唯一的字符串，比如一个很难预测的大随机数：

```
// 创建一个唯一的ID以跟踪上传进度
var uniqueID = 'A' + Math.floor(Math.random() * 1000000000000000);

// 为APC_UPLOAD_PROGRESS键添加这个唯一ID。
// 这个字段必须添加到文件输入字段之前，以便
// 服务器首先取得该键并触发存储进度信息的操作。
var uniqueIDField = document.createElement('input');
uniqueIDField.setAttribute('type','hidden');
uniqueIDField.setAttribute('value',uniqueID);
uniqueIDField.setAttribute('name','APC_UPLOAD_PROGRESS');
form.insertBefore(uniqueIDField,form.firstChild);
```

如果现在刷新浏览器中的页面，将发现表单会立即提交（页面并没有重载）。但是，看起来似乎什么也没有发生。因为，我们还没有创建进度指示器并从服务器上取得进度信息来更新页面。

**4. 进度条**

进度条本身由3个嵌套的DOM元素组成：容器、外部盒子和进度条本身，相应的HTML代码片断如下：

```
<span class="progressContainer" style="display: block;">
    <span class="progressBackground">
        <span class="progressBar"/></span>
        <span class="progressMessage"></span>
    </span>
</span>
```

请求的进度将通过把.progressBar元素的宽度调整为完成的百分比来表示。之所以又添加了两个附加的元素，是考虑到在格式化设计时放置内边距和外边距的需要。如有必要，可以在.progressContainer和.progressBackground元素上设置相应的样式。

但是，如果为.progessBar元素应用外边距样式，那么100%的宽度将导致它从一边突出出来，因为它的宽度是父元素的100%（忽略外边距）。图8-4显示了chapter8/progressbar/problems.html文件中为.progressBar元素设置了外边距之后的效果。

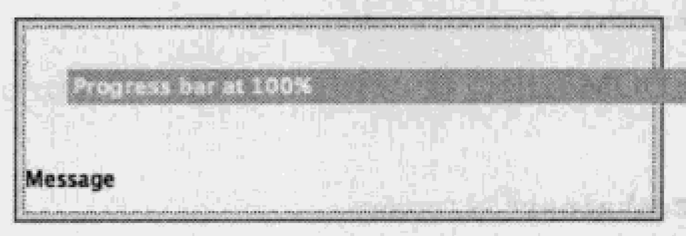



图8-4 如果为进度条应用外边距样式，它就会从父元素的一边突出出来

至于把进度条放在哪里、如何放则取决于你，不过为了体现对开发人员真正的友好，有必要允许他们在标记中随意指定放置进度条的位置。为此，需要向addProgressBar()方法中添加如下代码，检查是否存在一个带有progressContainer类名的DOM元素：

www.TopSage.com

```
// 创建进度条的不同部分

// 进度条
var progressBar = document.createElement('span')
progressBar.className = 'progressBar';
ADS.setStyle(progressBar,{
    'display':'block'
});

// 内部的背景容器
var progressBackground = document.createElement('span')
progressBackground.className = 'progressBackground';
ADS.setStyle(progressBackground,{
    'display':'block',
    'height':'10px'
});
progressBackground.appendChild(progressBar);

// 检查已有的定位点
// 必须是带有progressContainer类的span元素
var progressContainer = ADS.getElementsByClassName(
    'progressContainer',
    'span'
)[0];

// 如果该定位点不存在则创建一个并将其添加到表单中
if(!progressContainer) {
    progressContainer = document.createElement('span')
    progressContainer.className = 'progressContainer';
    form.appendChild(progressContainer);
}

// 设置容器为块级元素
ADS.setStyle(progressContainer,{
    'display':'block'
});

// 添加进度条的其余部分
progressContainer.appendChild(progressBackground);

// 同时也添加一个进度信息显示区域
var progressMessage = document.createElement('span')
progressMessage.className = 'progressMessage';
progressContainer.appendChild(progressMessage);
```

这样一来，脚本会首先在DOM树中检查一个用作容器的带有progressContainer类的特定对象，例如：

```
<span class="progressContainer"></span>
```

如果这个带有适当类的元素不存在，则将进度条添加到位于表单底部的默认位置。

接着，为减少一些冗余代码，在addProgressBar()方法中再添加下面这个私有的updateProgressBar()方法：

```
// 创建一个将由后面的进度监视方法使用的
// 私有方法，以方便更新进度条和相应信息
```

```
function updateProgressBar(percent,message,satus) {
    progressMessage.innerHTML = message;
    ADS.removeClassName(progressMessage,'error');
    ADS.removeClassName(progressMessage,'complete');
    ADS.removeClassName(progressMessage,'waiting');
    ADS.removeClassName(progressMessage,'uploading');
    ADS.addClassName(progressMessage,satus);

    // CSS样式和className将负责指示状态
    ADS.setStyle(progressBar,{
        'width':percent
    });
}
// 从0%和waiting开始初始化进度条
updateProgressBar('0%','Waiting for upload','waiting');
```

这个私有方法位于addProgressBar()方法的作用域中，因此在addProgressBar()方法内部的任何地方都可以调用它来更新进度条。这样会使下面步骤中的代码更整洁。

**5. 跟踪进度**

addProgressBar()方法中的最后一步，就是创建表单的submit事件侦听器。这里会涉及几个需要慎重对待的元素。先添加下列submit事件侦听器以便完成addProgressBar()方法，然后我们再来分析它的工作过程：

```
// 为表单添加提交事件侦听器，用于
// 验证表单信息和更新进度条
ADS.addEvent(form,'submit',function(W3CEvent){

    // 再次检查输入以确保
    // 其包含正确的扩展名
    var ok = true;
    var hasFiles = false;
    for(var i=0 ; (fileInput = fileInputs[i]) ; i++) {
        if(fileInput.value.length>0) {
            hasFiles = true;
        }
        if(!verifyFileType(fileInput)) {
            // 突出显示出错的文件输入元素
            if(!ADS.hasClassName(fileInput,'error')) {
                ADS.addClassName(fileInput,'error');
        }
        ok = false;
        }
    }

    if(!ok || !hasFiles) {
        // 如果检查未通过则提示用户解决问题
        ADS.preventDefault(W3CEvent);
        alert('Please select some valid files.');
        return false;
    }


// 通过发出警告信息来禁用表单元素
function warning(W3CEvent) {
```

www.TopSage.com

```
        ADS.preventDefault(W3CEvent);
        alert('There is an upload in progress. Please wait.');
    }
    for(var i=0 ; (input = allInputs[i]) ; i++) {
       //input.setAttribute('disabled','disabled');
       ADS.addEvent(input,'mousedown',warning);
    }

    // 创建一个函数以便在上传完成后重新启用表单
    // 该函数将在Ajax事件侦听器内部被调用
    function clearWarnings() {
        // 从表单元素移除警告侦听器
        for(var i=0 ; (input = allInputs[i]) ; i++) {
            ADS.removeEvent(input,'mousedown',warning);
        }

        // 以新ID数值更新原ID和表单
        // 以确保下次上传不影响本次上传
        uniqueID = Math.floor(Math.random() * 1000000000000000);
        uniqueIDField.setAttribute('value',uniqueID);
    }

    // 更新进度条
    updateProgressBar('0%','Beginning','waiting');

    // 为模拟脚本设置计数器
    var counter = 0;

    // 创建一个新方法以触发一次新的进度请求
    var progressWatcher = function() {

        // 使用唯一键来请求进度信息
        ADS.ajaxRequest(form.action
            + (form.action.indexOf('?') == -1 ? '?' : '&')
            + 'key=' + uniqueID + '&sim=' + (++counter) , {

            // 服务器端脚本将返回适当的头部信息
            // 因此我们可以使用JSON侦听器
            jsonResponseListener:function(response) {
                // 检测响应以确认服务器端
                // 脚本中是否存在错误

                if(!response) {
                // 没有有效的响应
                updateProgressBar(
                    '0%',
                    'Invalid response from progress watcher',
                    'error'
                );
                // 请求完成故清除警告提示
                clearWarnings();
            } else if(response.error) {
                // 服务器报告了错误
                updateProgressBar('0%',response.error,'error');
                // 请求完成故清除警告提示
                clearWarnings();
            } else if(response.done == 1) {
                // POST请求已经完成
                updateProgressBar(
                    '100%',
```

www.TopSage.com

```
                'Upload Complete',
                'complete'
                );
                // 请求完成故清除警告提示
                clearWarnings();
                // 为提供更改处理程序的
                // 用户传递新信息
                if(modificationHandler.constructor == Function) {
                    modificationHandler(response);
                }
            } else {
                // 更新进度条并返回结果
                // 由于结果是null，所以
                // 返回会简单地停止执行
                // 方法中其余的代码
                updateProgressBar(
                    Math.round(response.current/ ➥
            response.total*100)+'%',
                    response.current
                        + ' of '
                        + response.total
                        + '. '
                        + 'Uploading file: ' +
                        response.currentFileName,
                        'uploading'
                    );

                    // 再次执行进度监视程序
                    setTimeout(progressWatcher,1000);

                }

            },

            errorListener:function() {
                // Ajax请求发生了错误
                // 因此需要让用户知道
                updateProgressBar('0%',this.status,'error');

                // 并清除警告提示
                clearWarnings();

            }
        });
    };

    // 开始监视……
    setTimeout(progressWatcher,1000);

});
```

> 在学习本章内容的过程中，你可能已经注意到了，addProgressBar()方法在很大程度上依赖于JavaScript的闭包及作用域链特性。比如，在该方法的开始处，我们定义了一个包含文件输入元素的变量fileInput，而在整个方法的任何侦听器都可以直接引用这个相同的变量。虽然侦听器会附加到不同的对象上面执行（例如表单和文件输入元素），但是作用域链仍然会应用到在脚本中定义该变量的位置。

表单的提交事件侦听器一开始再次检查了文件输入路径中是否包含适当的扩展名。之所以要

www.TopSage.com

再检查一遍，是因为用户可能会忽略原先change侦听器给出的警告提示，而如果其中包含无效的文件则是没有必要提交的。

然后，通过将一个警告（warning()）方法作为表单中所有输入元素的mousedown事件侦听器，来防止在上传期间多次提交或者更改表单：

```
// 通过发出警告信息来禁用表单元素
function warning(W3CEvent) {
    ADS.preventDefault(W3CEvent);
    alert('There is an upload in progress. Please wait.');
}
for(var i=0 ; (input = allInputs[i]) ; i++) {
   ADS.addEvent(input,'mousedown',warning);
}
```

这也会让用户知道必须等到上传完成而且表单元素再度启用后，才能重新尝试。

对于进度自身的监视使用的是第7章中编写的ADS.ajaxRequest()方法，并且给每次请求都传递了唯一的键值：

```
ADS.ajaxRequest ('actions/?key=' + uniqueID,{ ... });
```

为了避免进度条生硬地变化，以及偶尔非常规甚至反向的冲突，请求自身在完成后会调用下一次请求——使用setTimeout()方法暂停一秒钟：

```
// 再次执行进度监视程序
setTimeout(progressWatcher,1000);
```

> 如果你预计通信量很大，那么可能会想到增加请求之间的延迟时间。而较长的延迟时间也不光意味着进度条的变动会比较突然，也会减轻服务器的通信量和负载。

这样也有助于消除最终“完成的”响应先于较早的请求到达的问题，从而避免在脚本中制造麻烦和导致进度条停滞于未完成状态的尴尬。

> 请求中还包含一个计数器，即'actions/?key=' + uniqueID + '&sim=' + (++counter)。这个计数器只用于模拟版的脚本，用来跟踪发送的请求数目。

来自服务器的每个进度响应中都将包含处理的总字节和当前字节数，以及与POST请求的当前状态有关的其他信息（正如你在本章前面所看到的那样）。而且，响应中使用的Content-Type为application/json，因此可以在ADS.ajaxRequest()请求中使用jsonResponse-Listener来检查响应，并按照响应更新页面中的元素。

> 如果你修改了服务器端脚本，或者在响应中使用了不同的Content-Type，那么就需要调整用于在ADS.ajaxRequest()中处理响应[1]的侦听器方法。在第7章可以看到已添加的所有侦听器方法的列表。

① 原文request有误，应为response。——译者注

www.TopSage.com

当响应无效或者响应中存在错误信息时，需要使用updateProgressBar()方法来显示适当的信息和变化。同样，在请求成功时，则会向提供的modificationHandler方法传递被成功处理的文件列表，以便更新页面：

```
if(!response) {
    // 没有有效的响应
    updateProgressBar(
    '0%',
    'Invalid response from progress watcher',
    'error'
    );
    // 请求完成故清除警告提示
    clearWarnings();
} else if(response.error) {
    // 服务器报告了错误
    updateProgressBar('0%',response.error,'error');
    // 请求完成故清除警告提示
    clearWarnings();
} else if(response.done == 1) {
    // POST请求已经完成
    updateProgressBar('100%','Upload Complete','complete');
    // 请求完成故清除警告提示
    clearWarnings();
    // 为提供更改处理程序的
    // 用户传递新信息
    if(modificationHandler.constructor == Function) {
        modificationHandler(response);
    }
} else {
    // 更新进度条并返回结果
    // 由于结果是null,
    // 返回会停止执行
    // 方法中其余的代码
    updateProgressBar(
        Math.round(response.current/response.total*100)+'%',
        response.current
            + ' of '
            + response.total
            + '. '
            + 'Uploading file: ' +
            response.currentFileName,
        'uploading'
    );

    // 再次执行进度监视程序
    setTimeout(progressWatcher,1000);

}
```

而且，还要在错误的或者“完成的”响应之后，调用clearWarnings()方法使表单恢复可用状态。与此同时，也要更新与表单关联的唯一ID键的值，以便下一次上传使用不同的ID：

```
// 创建一个函数以便在上传完成后重新启用表单
// 该函数将在Ajax事件侦听器内部被调用
function clearWarnings() {
    // 从表单元素移除警告侦听器
    for(var i=0 ; (input = allInputs[i]) ; i++) {
```

www.TopSage.com

```
            ADS.removeEvent(input,'mousedown',warning);
        }
        // 以新ID数值更新原ID和表单
        // 以确保下次上传不影响本次上传
        uniqueID = Math.floor(Math.random() * 1000000000000000);
        uniqueIDField.setAttribute('value',uniqueID);
    }
```

如果再次使用相同的ID，就会导致两个不同的、无关的请求相互之间发生冲突。因此，更新ID值确保了每次提交表单的请求都拥有自己唯一的ID。

在为页面添加了以上全部代码之后，虽然没有JavaScript也能上传文件，但在行为增强的条件下，用户会在等待请求完成的过程中看到一个实时的进度指示器，如图8-5所示。

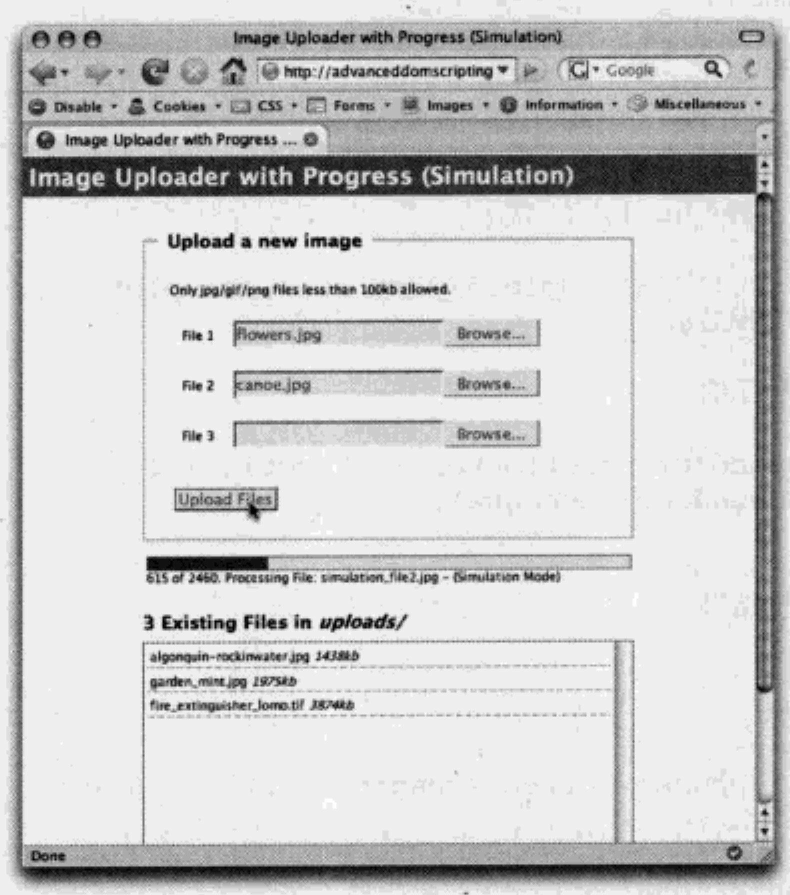



图8-5　在上传文件同时显示的进度条的最终效果

## 8.4　小结

进度条是一个充满技巧性的功能，特别是在跟踪文件上传的进度时更是如此。但是，你在本章中所学的方法能够适用于任何时间长度的服务器进程，并不仅限于文件上传。你只需构建一个能够存储并在将来取得进度信息的系统即可。通过仔细地分析随带的服务器端脚本，你会明白为了作出必要的反馈而怎样跟踪进度。而提供适当的反馈则会减少用户在时间不定的等待中可能会经历的挫折感。

本章是本书第二部分的终结，从此你不会再向ADS库中添加任何方法了。而你的这个个性化的库中也填满了每天都可能会用到的许多方法。在第三部分中，你将看到更多的第三方库，例如`Prototype`和`Script.aculo.us`。而且，还会看到通过最少的额外工作来提高开发效率，以及巧妙地增强你的Web应用程序的各种方式。

www.TopSage.com

# Part 3 第三部分

# 部分高级脚本编程资源

本部分内容


- 第 9 章　通过库来提高生产力
- 第 10 章　添加效果增强用户体验
- 第 11 章　丰富的 Mashup! 运用 API 添加地图、搜索及更多功能
- 第 12 章　案例研究：用 DOM 设计选择列表


www.TopSage.com

第 9 章

# 通过库来提高生产力

我们在前面曾经提到过，JavaScript程序开发的当前状态与四、五年之前相比，已经是不可同日而语了。其中一个主要的变化，就是开发者已经不需要自己编写全部代码了——因为已经出现了一些能够使编码效率更高也更轻松的手段。在第三部分中，我们将介绍一些库、效果以及第三方API，但本章的核心则是库。

库，指的是可以方便地应用到现有开发体系中的、现成的代码资源。库不仅为大部分日常的DOM脚本编程工作提供了快捷的解决方案，而且也提供了许多独特的工具。虽然库使用起来很方便，但它们也并非能够解决所有问题的“万金油”。

在深入钻研任何库之前，一定要保证花足够的时间真正理解JavaScript和DOM的原理。从本书第1章开始，我们就强调理解程序工作原理的必要性，不赞成你只是简单地知道它们能做什么。优秀的库有很多，其中一些我们还会在本章中进行讨论。但是，如果不能理解这些库背后的工作原理，那么对你和你的Web应用程序而言都是非常不利的。因为如果没有深入的理解，你很可能会对一些细节问题纠缠不清，而库的开发者通常会假定你已经明白了那些细节。

在本章中，我们要讨论如何使用下面这些库的特性来提高你的生产力。

- **DOMAssistant**：http://www.robertnyman.com/domass/
- **jQuery**：http://jquery.com
- **MochiKit**：http://mochikit.com
- **Prototype**：http://prototypejs.org
- **YUI（Yahoo User Interface）**：http://developer.yahoo.com/yui/

在第10章，我们还将看到通过这些库和其他一些库为Web应用程序添加的各种吸引眼球的效果。需要明确的一点是，我没有参与上面任何一个库的开发，因此不会有重视某个库而忽略其他库的个人偏见。不过，我倒是对其中几个库发表过一些看法——主要针对某些库文档中的遗漏或者需要改进之处提出建议。但也不是说这些库是包治百病的终极选择，而且优秀的库也远不止这些。它们出现在本章中，只是因为与本章介绍的某些标准（接下来会讨论到）恰好吻合，而且也对我将要讨论的每个方面提供了最清晰的解决方案。

提到生产力，我们就要重温一下本书已经讨论过的一些主题：

- 增强DOM操作能力。

www.TopSage.com

- 处理事件。
- 访问和操纵样式。
- 通过Ajax进行通信。

而且，你还会看到上面的每个库都是如何处理这些任务的。同样地，每个库在完成这些任务时也可能会采取几乎相同或者不同的方式，因此我会在讨论每个方面时，为你指出自己认为最好或者最有用的两三种方法，但我不会面面俱到地介绍每个库，要了解没有讨论到的内容请参考每个库的文档。

> 本章所有例子的源代码都可以在chapter9/中找到，在该文件夹中，例子代码是按照库而非功能进行划分的。

在学习完本章之后，你会更好地理解这些库提供的功能，以及它们之间的区别。届时，选择一款适合你的库也会变得更容易。

## 9.1 选择合适的库

当你决定要研究某个库时，面临的一个最大问题就是如何在数百个库中作出正确的选择。而在作出抉择之前，起码应该考虑以下标准。

- **它是否具有你需要的全部特性？** 在所选的库未设置适当的命名空间情况下，混合并协调多个库的问题会很多。例如像`$()`和`get()`等这样的常见方法，虽然使用的语法相同但处理的任务却可能各异。而且，如果同时使用了多个库，通常都会造成功能重复和代码冗余。
- **它具有的特性是否太多？** 特性太少是一个问题，而包含过多特性的大型库同样也是个问题。当一个库中的特性比你所需的多很多时，那么就应该考虑找一个更节省下载时间的轻量级版本。许多库，比如Prototype，都有删减了某些特性的“淡化”版。此外，也可以使用JavaScript代码压缩程序来减少代码文件的大小。此类压缩程序会使用更少的字符重新缩写原有的代码，因而会使文件变小——但是，压缩之后的代码是无法认读的。
- **它是基于模块的吗？** 为解决文件尺寸的问题，包含丰富特性的库一般都会将特性模块化，然后保存到不同的文件中。这样，通过只载入必要的文件和特性，可以保持页面文件最小化。为此，多数情况下，你都要确保包含所有必需的文件。但也有个别的库会提供一种动态载入机制，即只需包含一个文件，而该文件可以按照需要获得其他文件。
- **它具有完善的支持吗？** 没有活跃的开发者社区也就等于不会修改错误或者不会有特性改进。同样，一个库得到众多开发者的关注和共用意味着错误更少，结果更可靠。而且，背后拥有良好社区支持的库不仅会迅速修正错误和改进特性，而且你还能在碰到问题需要帮助时得到很多支持。
- **它有文档吗？** 没有文档，会令人迷惑。没错，你可能会看到其他人共享的某些例子，但缺少文档通常表明开发者缺乏激情，甚至认为这个项目是可有可无的。

www.TopSage.com

- **它有合适的许可吗？**能够在线查看源代码并不意味着可以自由地取得。在使用一个库之前，一定要确定它的许可涵盖了你的真实需求。而且，不要忘了查看许可本身的条款。很多许可并没有对你的工作结果作出限制，另外一些许可则要求你发布的代码也必须遵守同一许可，而这对于私人环境下的封闭源代码可能是不合适的。

> 当你选定了一个合适的库并且取得显著的成绩之后，可别忘了回报社区哟！这些库可都是那些为了让你日常所用的工具得到改进，而不惜占用自己宝贵休闲时间的开发者们奉献的结果。即使你对库的开发或者错误测试帮不上忙，但总可以提供一些例子和教程，或者帮着写写文档——不论什么努力都是有益的，而且都会使库变得更好。

## 几个库的简介

在本章中，我基于上述标准以及大体的流行程度和一点个人偏好，选择了几个库。这些库都有各自的优缺点，我们将在以下几节中分别介绍。

### 1. DOMAssistant①

DOMAssistant（http://robertnyman.com/domass）“为Web浏览器中基于DOM的脚本提供更简单也更一致的编写方式”。DOMAssistant是一个轻量级和模块化的库，对于保持文件的最小化很合适。虽然缺少Ajax通信方法意味着你可能还需要其他库（或者你自己的库）来获得相应功能，但考虑到第7章列举的问题，这也不一定是一件坏事。此外，使用这个库选择元素会有一点受限制——只包含“通过`id`”和“通过类名”的方法。而在这方面其他库则提供了基于更高级的CSS选择符表达式的方法，本章稍后将会介绍到。

- **许可②**，DOMAssistant使用的是Creative Commons Attribution ShareAlike 2.5许可（http://creativecommons.org/licenses/by-sa/2.5/deed.en）。
- **命名空间**，如果单个的`DOMAssistant`方法在`window`作用域中尚不存在，该库会注册其中很多方法。如果你在之前载入了另外一个含有冲突方法的库，`DOMAssistant`不会重写已有的方法，但你需要使用适当的命名空间来引用所有的`DOMAssistant`方法，例如`DOMAssistant.$()`。

### 2. jQuery

jQuery（http://jquery.com）“是一个快速、简明的JavaScript库，它能够简化遍历HTML文档、处理事件、生成动画以及为网页添加Ajax交互功能的过程”。通过jQuery极其强大的选择方法，可以组合使用ID、CSS和XPath选择符取得DOM元素。而简化的Ajax和事件方法与连缀语法的结合运用，则能够让代码更简洁也更容易理解。在jQuery背后有一个非常大的社区，其中许多插件

① 2007年9月19日DOMAssistant发布了2.0版（http://www.robertnyman.com/domassistant/index.htm），其中增强了Ajax模块和其他选择元素的方法。——译者注

② DOMAssistant 2.0使用的协议为Creative Commons Deed license（http://creativecommons.org/licenses/GPL/2.0/）。——译者注

www.TopSage.com

开发者为这个库添加了很多超出其基本功能之外的特性。

- **许可**，jQuery使用MIT许可（http://www.opensource.org/licenses/mit-license.php）和General Public License或GPL（http://opensource.org/licenses/gpl-license.php）。
- **命名空间**，如果只使用jQuery一个库，那么可以使用简写的`$()`方法代替`jQuery()`方法。如果想同时使用jQuery和另一个库，而另一个库中也在`window`作用域中定义了`$()`函数（比如Prototype库），那么为了防止jQuery覆盖`$()`函数，就需要在脚本开始处调用`noConflict()`方法：

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

要了解同时使用jQuery及其他库的更多注意事项，请参考http://docs.jquery.com/Using_jQuery_with_Other_Libraries。

#### 3. Mochikit

据其网站所说，MochiKit（http://mochikit.com）"让JavaScript不那么差劲"。经过了完备的测试而且文档也很完善（不过例子应该再多点）的MochiKit库，也提供了与其他库相同的DOM操作方法。而通过将色彩和视觉效果，以及它自己的调试用的日志面板整合起来，使得其他库变得不再必要。此外，这个库中的自定义事件和内置的拖放支持也会使创建Web应用程序更加容易。

- **许可**，这个库使用MIT许可（http://www.opensource.org/licenses/mit-license.php）和Academic Free License V2.1版（http://www.opensource.org/licenses/afl-2.1.php）。
- **命名空间**，MochiKit库是由位于MochiKit命名空间中的几个子对象构成的。例如，要调用`ifilter`方法可以使用

  ```
  ifilter(...)
  ```

  也可以使用

  ```
  MochiKit.Iter.ifilter(...)
  ```

在载入之后，这些方法全部都会注册到`window`命名空间中。如果不想注册到`window`命名空间中，则需要在网页头部包含该库的代码之前插入几行嵌入的JavaScript代码：

```
<script type="text/javascript">
    // 防止冲突
    MochiKit = {__export__: false};
</script>
<script src="/path/to/MochiKit.js" type="text/javascript"></script>
```

#### 4. Prototype

Prototype（http://prototypejs.org）"是一个针对简化动态Web开发的框架"。可能你还记得，我们在本书中曾经多次提到过Prototype。这个库中包含许多好用的DOM操作函数，以及一个比众所周知的`$()`函数更加流行的Ajax对象。Prototype唯一不足的地方就是它缺乏对命名空间的支持，也就是说在将它与其他的库共用时还不够方便。

- **许可**，Prototype使用MIT许可（http://www.opensource.org/licenses/mit-license.php）。
- **命名空间**，Prototype库不包含任何形式的命名空间，因此所有方法都会注册到`window`作用域中。

www.TopSage.com

#### 5. YUI

YUI（Yahoo! User Interface，Yahoo用户界面）库（http://developer.yahoo.com/yui）“是一组以JavaScript编写的，为使用诸如DOM脚本、DHTML以及Ajax等技术构建丰富的交互式Web应用而开发的实用工具和控件的集合。YUI库中也包含一些核心的CSS资源”。YUI库拥有庞大的开发者社区以及大量的文档。这个库中包含了从简单的DOM操作到高级效果及全功能饰件（widget）在内的、你所能够想象出来的各种特性。虽然这个库整体上划分成了许多小文件和命名空间，但有时为了确定要使用哪个文件和找到这个文件也是件令人烦恼的事——仅仅从其简单的说明就长达20页就能看出这个库有多大来。

- **许可**，YUI使用BSD许可（http://developer.yahoo.com/yui/license.txt）。
- **命名空间**，Yahoo的这个库在YAHOO命名空间中分成了各种各样的子对象。而且，这个库本身也分为许多文件。因此，无论使用哪个对象都必须保证事先包含了适当的文件。要了解每个对象的具体命名空间，可以查询其开发者文档http://developer.yahoo.com/yui。

> 并非所有的库都会遵循你在本书一开始设置的优美的命名空间约定。比如，Prototype、jQuery以及DOMAssistant这几个库中，都包含与你添加到ADS库中的`$()`函数相同的方法。不过，幸运的是，本节讨论的所有库（除了Prototype）都有自己备用的该方法的命名空间版。因此，本章中出现的所有例子代码中都将使用完整的命名空间。如果你想在自己的开发环境中使用其中一个库，并且只使用其中的一个，那么可以使用其简写版。要了解每个库都提供了什么样的简写方法，请查阅具体库的文档。

## 9.2 增强DOM操作能力

在提到DOM脚本编程时，每个库所提供的DOM增强操作能力通常都是它们最大的卖点。而提供的附加特性以及对现有特性的浏览器不兼容性的处理，则是这些库的闪光点所在。下面我们来看一看刚才提到的这些库中所包含的DOM增强特性，同时也看一下它们是怎样帮助你更方便地取得DOM元素的。

### 9.2.1 连缀语法

对于简单的调用方法的语法，你已经非常熟悉了，例如调用`ADS.$()`函数：

```
// ADS $()方法
var elementReference = ADS.$('browserList');
```

此外，你也曾看到过如何组合不同函数的用法，比如将使用`$()`函数取得的元素传递给`ADS.getElementsByClassName()`方法：

```
// ADS库的方法
var browserAnchors = ADS.getElementsByClassName(
    'browser',
    'a',
    ADS.$('browserList')
);
```

使用单个方法的思想同样也适用于其他库。MochiKit和YUI都遵循与此相同的模式，提供了以类似方式查找元素的方法：

```
// MochiKit库
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = MochiKit.DOM.getElementsByTagAndClassName(
    'a',
    'browser',
    MochiKit.DOM.$('browserList')
);

// YUI库
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = YAHOO.util.Dom.getElementsByClassName(
    'browser',
    'a',
    YAHOO.util.Dom.get('browserList')
);
```

> 前面两个例子都一样，都不用取得对browserList的DOM引用，而只需简单的传递字符串browserList即可实现同样的结果。展示这两例子只是为了说明不同的方法是如何协同运作的。

使用单个方法的思想常见于不同的库中，而这也正是你在学习本书并构建ADS库的过程中所体现出来的思想。另一种思想，或者说另一种更贴近JavaScript语法特性的方法，是连缀。

所谓连缀，就是在使用ADS.$()方法时，可以通过方法连缀的形式同时使用其他DOM操作方法，例如getElementsByTagName()：

```
var list = ADS.$('example').getElementsByTagName('a');
```

这种思想固然好，但它也仅限于针对DOM操作的方法，即不能以相同的方式使用其他自定义的方法。不过某些库，如DOMAssistant、jQuery以及Prototype（与库中的其他部分一起），都使用了自己的补救方法为返回的元素直接添加了附加的静态方法。在改进之后，我们就可以通过这些附加的方法，在相同的调用过程中，将库中的方法与标准的方法连缀起来使用了：

```
// DOMAssistant库（使用方法连缀）
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = DOMAssistant.$(
    'browserList'
).getElementsByClassName(
    'browser',
    'a'
);

// jQuery库（使用方法连缀）
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = jQuery('#browserList').find('a.browser');

// Prototype库（使用方法连缀）
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = $('browserList').getElementsByClassName(
```

www.TopSage.com

```
        'browser'
    ).findAll(
        function(e) {
            return e.nodeName == 'A';
        }
    );
```

使用连缀语法，可以在不降低可读性的基础上将代码精简为语义化的方法。而且，多数情况下，也可以组合使用现有的DOM方法：

```
// jQuery库（使用方法连缀）
// 查找ID为#browserList的元素中的所有a.browser锚元素
// 并取得第一个节点的nodeValue
var value = jQuery('#browserList').find('a')[0].firstChild.nodeValue;
```

当然，如有必要也可以通过把方法返回的结果赋给变量来将连缀的方法分开：

```
var browserList = jQuery('#browserList');
var browserAnchors = browserList.find('a.browser');
var value = browserAnchors[0].firstNode.nodeValue;
```

在使用方法连缀时，要确保不会在组合使用常规方法与库方法时作出过多的假设。比如，在前面的例子中，如果页面中不存在#browserList元素，那么jQuery方法在访问数组中索引为[0]的元素时，将会发生错误。因而，会出现如图9-1所示的错误提示。

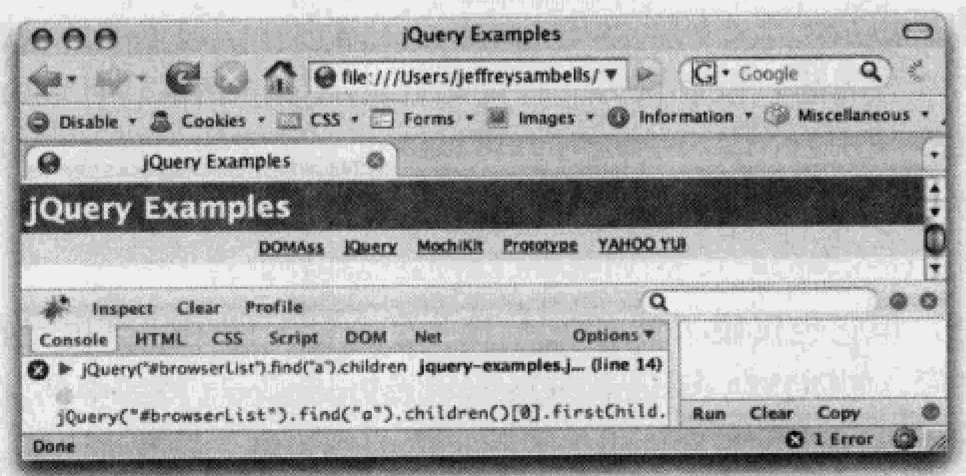



图9-1　在访问不存在的数组元素时发生的表示连缀语法失败的错误

### 1. 使用表达式的高级选择操作

通过ID选择元素是不错，但通过CSS选择符来选择元素则更方便。许多库都包含了如下类似的高级选择符方法。

- Prototype的$$()方法：

```
// Prototype库中的高级选择符方法
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = $$('#browserList a.browser');
```

- MochiKit.Selector.$$()方法：

```
// MochiKit库的高级选择符方法
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = MochiKit.Selector.$$('#browserList a.browser');
```

www.TopSage.com

- 基本的jQuery()库的方法:

```
// jQuery库的高级选择符方法
// 查找ID为#browserList的元素中的所有a.browser锚元素
var browserAnchors = jQuery('#browserList a.browser');
```

这3个方法为搜索和获取元素提供了难以言表的巨大灵活性。不仅可以使用多种CSS选择符（http://www.w3.org/TR/css3-selectors/#selectors）来选择具体的元素，而且在浏览器支持DOM 3级XPath（http://www.w3.org/TR/xpath）的情况下，这些方法的查询速度将会极快（取决于库的支持）。

> 其中MochiKit的$$()方法作为MochiKit.Selector对象的一个成员，只在1.4及更高版本中有效。在本书编写时，其1.4版仍处于开发过程中。因此，为使用这些高级的选择符方法，请下载该库的最新版本。

而且，其中每种方法都支持如下许多不同的CSS选择符。

- *：选择所有元素。
- tag：选择有效的HTML标签中所有是tag的元素。
- tagA tagB：选择作为tagA元素后代元素的所有tagB元素。
- tagA、tagB、tagC：选择所有tagA元素、tagB元素和tagC元素。
- #id或tag#id：选择ID为id的所有元素，或者以标签与ID组合形式表示的所有元素。
- .className和tag.className：选择类名中包含className的所有元素，或者以标签与类名组合形式表示的所有元素。
- 还可以将各种选择符通过空格组合起来，比如像#myList li或ul li a.selectMe，来选择更具体的后代元素。

此外，也可以使用如下CSS 2.1中规定的属性选择符。

- tag[attr]：选择带有attr属性的所有tag元素。
- tag[attr=value]：选择attr属性的值等于value的所有tag元素。
- tag[attr~=value]：选择attr属性的值中包含value的所有tag元素。
- tag[attr^=value]：选择attr属性的值以value开头的所有tag元素。
- tag[attr$=value]：选择attr属性的值以value结尾的所有tag元素。
- tag[attr|=value]：选择attr属性的值为带连字符的字符串且该值以value开头的所有tag元素。
- tag[attr!=value]：选择attr属性的值不是value的所有tag元素。

在使用属性选择符时，如果属性的值中包含空格，则必须将该值包含在一对引号中，例如：

```
$$('a[title="Hello World!"]');
```

此外，还可以使用下列子选择符或同辈选择符。

- tagA > tagB：选择作为tagA元素直接子元素的所有tagB元素。
- tagA + tagB：选择作为tagA元素紧邻同辈元素的所有tagB元素。
- tagA ~ tagB：选择前面的同辈元素为tagB元素的tagA元素。

www.TopSage.com

也可以使用下列伪类或伪元素选择符。

- tag:root：选择作为文档根元素的tag元素。
- tag:nth-child(n)：选择作为其父元素的正数第n个子元素的tag元素。
- tag:nth-last-child(n)：选择作为其父元素的倒数第n个子元素的tag元素。
- tag:nth-of-type(n)：选择作为同类型元素中的正数第n个同辈元素的tag元素。
- tag:nth-last-of-type(n):选择作为同类型元素中的倒数第n个同辈元素的tag元素。
- tag:first-child：选择作为其父元素的第一个子元素的tag元素。
- tag:last-child：选择作为其父元素的最后一个子元素的tag元素。
- tag:first-of-type：选择作为同类型元素中第一个同辈元素的tag元素。
- tag:last-of-type：选择作为同类型元素中最后一个同辈元素的tag元素。
- tag:only-child：选择作为其父元素的唯一子元素的tag元素。
- tag:only-of-type：选择作为同类型元素中唯一同辈元素的tag元素。
- tag:empty：选择没有子元素的所有tag元素。
- tag:enabled：选择用户界面中启用的所有tag元素[①]。
- tag:disabled：选择用户界面中禁用的所有tag元素[②]。
- tag:checked：选择用户界面中被选中的所有tag元素，例如复选框和单选按钮。
- tag:not(s)：选择所有与选择符s不匹配的tag元素。

> Prototype 1.5在使用选择符选择DOM元素时存在一定的限制，例如不支持子选择符（>）和相邻同辈选择符（+）。不过，其1.5.1版中则添加了对所有CSS 3选择符的支持，该版本在本书出版时很可能已经发布了[③]。

### 2. jQuery与XPath

对于jQuery而言，情况还略有不同，因为jQuery还增加了另一种强大的表达式方法——XPath。也就是说，jQuery的表达式引擎不仅可以解释前面列出的选择符（也有少数例外，本节后面将会谈到），而且还可以解释XPath表达式以及其他自定义的选择符。此外，在jQuery中也可以使用属性选择符，但它们的格式与常规的CSS属性选择符还有一些差别。出于同XPath语法保持一致的原因，所有属性都必须以@符号开头：

- jQuery('tag[@attr]')，选择带有attr属性的所有tag元素。
- jQuery('tag[@attr=value]')，选择attr属性的值为字符串value的所有tag元素。
- jQuery('tag[@attr^=value]')，选择attr属性的值以字符串value开头的所有tag元素。
- jQuery('tag[@attr$=value]')，选择attr属性的值以字符串value结尾的所有tag

① 原文中disabled有误，应为enabled。——译者注
② 原文中enabled有误，应为disabled。——译者注
③ Prototype 1.6于2007年11月7日发布。——译者注

www.TopSage.com

元素。

- jQuery('tag[@attr*=value]')，选择attr属性的值中包含字符串value的所有tag元素。

> 与Prototype的$$()函数相似，jQuery不支持某些伪元素和伪类选择符[①]，包括:link、:visited、:active、:hover、:focus、:target、::first-line、::first-letter、::selection、::before和::after。不过，jQuery之所以没有支持某些选择符是因为它们缺乏真正的实用价值，属于这一类的选择符有:nth-last-child()、:nth-of-type()、:nth-last-of-type()、:first-of-type、:last-of-type、:only-of-type和:lang(fr)。但是，其他表达式或表达式的组合则能够提供类似的功能。

除了简单的CSS选择符之外，在jQuery中也可以使用基本的XPath表达式。要了解有关XPath表达式语法的详细信息，请参阅W3C网站：http://www.w3.org/TR/xpath。XPath是一种查询XML文档的强大方式，而在下面使用jQuery的例子中可以看出这一点。

- 通过绝对路径选择作为文档主体后代元素的所有段落：

```
jQuery("/html/body//p")
jQuery("/*/body//p")
```

- 通过相对路径，相对于this引用的节点进行选择：

```
// 相对路径
jQuery("a",this)
jQuery("p/a",this)
```

- 通过不同的坐标进行选择：

```
// 选择带有后代元素P的后代元素DIV
jQuery("//div//p")
// 选择带有子元素P的子元素DIV
jQuery("//div/p")
```

- 基于属性（谓词）进行选择：

```
// 选择所有被选中的输入元素
jQuery("//input[@checked]")
// 选择所有ref属性为nofollow的锚
jQuery("//a[@ref='nofollow']")
```

- 通过语法稍有变化的其他谓词进行选择：

```
// [last()]或[position()=last()]变为:last
jQuery("p:last")
// [0]或[position()=0]变为:eq(0)或:first
jQuery("p:eq(0)")
// [position() < 5]变为:lt(5)
jQuery("p:lt(5)")
// [position() > 2]变为:gt(2)
jQuery("p:gt(2)")
```

① 原文pseudoselectors有误，应为pseudoclass selectors。——译者注

www.TopSage.com

此外，jQuery还支持XPath选择符与CSS选择符混合使用（不仅灵活性大大增加，而且速度也很快），从而可以更直接地访问到相应的元素。

比如，可以基于元素的世系（lineage）来选择元素，以选择一个无序列表的子元素为例：

```
jQuery('ul/li')
```

同样，也可以使用CSS选择符：

```
jQuery('ul > li')
```

而尤其令人赞叹的则是在选择了元素之后，进而再基于元素的属性取得元素值的能力。比如取得name属性为street的输入字段的值：

```
jQuery('input[@name=street]').val();
```

也可以像下面这样取得所有选中的单选按钮：

```
jQuery('input[@type=radio][@checked]')
```

除了CSS和XPath选择符之外，jQuery还包含如下一些自定义的表达式，也许对你比较有用。

- :even，从匹配的元素集合中间隔地选择偶数元素——对突出显示表格行恰好有用！
- :odd，从匹配的元素集合中间隔地选择奇数元素。
- :eq(0)和:nth(0)，从匹配的元素集合中选择第n个元素，例如页面中的第一个段落等。
- :gt(n)，选择索引值大于n的所有匹配的元素。
- :lt(n)，选择索引值小于n的所有匹配的元素。
- :first，等价于:eq(0)。
- :last，选择最后一个匹配的元素。
- :parent，选择带子元素（包括文本）的所有元素。
- :contains('test')，选择包含特定文本的所有元素。
- :visible，选择所有可见的元素（其中包括display属性值为block或inline、visibility属性值为visible的元素，以及type属性值不是hidden的表单元素）。
- :hidden，选择所有隐藏的元素（其中包括display属性值为none或visibility属性值为hidden的元素，以及type属性值为hidden的表单元素）。

使用这些自定义的表达式可以实现快速地修改元素，例如要修改页面中第一个段落的字体粗细：

```
jQuery("p:first").css("fontWeight","bold");
```

或者快捷地显示所有隐藏的<div>元素：

```
jQuery("div:hidden").show();
```

甚至可以直接隐藏包含单词“scared”的所有div元素：

```
jQuery("div:contains('scared')").hide();
```

最后，jQuery中还包含如下一些专门用于访问表单元素的表达式。

- :input，选择所有表单元素（input、select、text area、button）。

www.TopSage.com

- :text，选择所有文本字段（type="text"）。
- :password，选择所有密码字段（type="password"）。
- :radio，选择所有单选按钮字段（type="radio"）。
- :checkbox，选择所有复选框字段（type="checkbox"）。
- :submit，选择所有提交按钮（type="submit"）。
- :image，选择所有表单图像（type="image"）。
- :reset，选择所有重置按钮（type="reset"）。
- :button，选择其他所有按钮（type="button"）。

### 9.2.2 通过回调函数进行过滤

当高级表达式不能满足需要时，还可以通过回调函数来对DOM元素进行处理，并针对每个元素执行任何代码。在下面这些例子中，回调函数返回true就会包含相应的元素，而返回false则会从结果列表中排除相应的元素。

回调函数特别有用的时候就是用来创建反向选择符。由于所有CSS 3选择符识别的都是选择符中最右边的元素，故而无法选择“只有一个<img>子元素的所有锚标签”。然而，通过使用回调函数却非常容易实现这种操作。下面就是例子中使用的HTML代码片断：

```
<ul>
    <li>
        <a name="example1"><img src="example.gif" alt="example"/></a>
    </li>
    <li>
        <a name="example2">No Images Here</a>
    </li>
    <li>
        <a name="example3">
            Two here!
            <img src="example2.gif" alt="example"/>
            <img src="example3.gif" alt="example"/>
        </a>
    </li>
</ul>
```

如果使用YAHOO.util.Dom.getElementBy()方法，可以使用已有DOM元素的属性对结果列表进行过滤：

```
// 在YUI库中使用回调过滤函数
var singleImageAnchors = YAHOO.util.Dom.getElementsBy(function(e) {
    // 查找只有一个img子元素的A节点
    return (e.nodeName == 'A' && ➡
e.getElementsByTagName('img').length == 1);
});
```

结果singleImageAnchors变量中只包含对<a name="example1">的引用，因为它是唯一只包含一个img子元素的锚。

而MochiKit提供的MochiKit.Iter.ifilter()方法的原理也很类似：

```
// 在MochiKit库中使用回调过滤函数
```

www.TopSage.com

```
var singleImageAnchors = MochiKit.Iter.ifilter(
    function(e) {
        // 确保只有一个img子元素
        return (e.getElementsByTag('img').length === 1);
    },
    document.getElementsByTagName('a')
);
```

Prototype和jQuery也分别提供了`findAll()`和`filter()`方法。而这两个方法要通过方法连缀的形式对由表达式方法返回的元素进行过滤：

```
// 在Prototype库中使用回调过滤函数
var singleImageAnchors = $$('a').findAll(function(e) {
    return (e.descendants().findAll(function(e) {
        // 查找所有img元素
        return (e.nodeName == 'IMG');
    }).length == 1);
});

// 在jQuery库中使用回调过滤函数
var singleImageAnchors = jQuery('a').filter(function() {
    // 确保只有一个img子元素
    return (jQuery('img',this).length == 1)
});
```

虽然多数情况下，Prototype和jQuery所提供的表达式选择符方法对于过滤元素列表而言已经够用了，但当需要对元素进行深度分析时，回调函数就会显示出它强大的威力。

### 9.2.3 操纵 DOM 文档

由于每个库都提供了很多不同的DOM操纵方法，所以这里我们只是有选择地介绍其中几个，剩下的需要你自己阅读每个库的文档。

- **DOMAssistant**：http://www.robertnyman.com/domass/modules-domass-content.htm
- **jQuery**：http://docs.jquery.com/DOM/Manipulation
- **MochiKit**：http://www.mochikit.com/doc/html/MochiKit/DOM.html
- **Prototype**：http://www.prototypejs.org/api/element
- **YUI**：http://developer.yahoo.com/yui/dom/

这些库中的许多与DOM相关的方法和自定义的ADS库中的相应方法都很类似，不过这些库提供了更多的方法，比如下面几节要介绍的方法。

#### 1. 使用DOMAssistant创建元素

`DOMAssistant.$(id).create(name, attr, append, content)`方法可以为链中的元素创建一个新的DOM子元素：

```
// 在#content元素中创建一个id为myDiv
// class为justAdded的子<div>元素，并将
// 其文本内容设置为I'm a brand new div!
$("content").create("div", {
    id : "myDiv",
    className : "justAdded"
}, true, "I'm a brand new div!");
```

www.TopSage.com

### 2. 使用jQuery移动节点

jQuery(expression).appendTo(expression)方法可以查找大批元素，并将这些元素全都移动到另一个元素中，作为该元素的子元素：

```
// 查找ul#list1中的所有<li>元素并将
// 它们重新部署为ul#list2的子元素
$('ul#list1 li').appendTo("ul#list2");
```

### 3. 使用MochiKit创建元素

MochiKit.DOM.createDOM(name[, attrs[, node[, . . . ]]])方法可以按照给定的属性创建新的DOM元素：

```
var newDiv = MochiKit.DOM.createDOM(
    'DIV',
    {'class': 'justAdded'},
    'I\'m a brand new div!'
);
MochiKit.DOM.$('content').appendChild(newDiv);
```

这样会创建一个新的div元素：

```
<div class='"justAdded">I'm a brand new div!</div>
```

而且，通过将更多的createDOM()的结果作为第3个参数，可以创建DOM树。

### 4. 使用Prototype清理文档

$( id ).cleanWhitespace()方法可以清除给定元素的空白子节点。这个方法可以用来方便地创建不针对特定浏览器的脚本。在第3章中，我们曾讨论过浏览器以不同方式处理空白符的问题，因此一开始就把所有空白符全部去除可以保证得到更一致的结果。比如，对于下面这个无序列表

```
<ul id="example">
    <li>Item 1</li>
    <li>Item 2</li>
</ul>
```

如果像下面这样查找其第一个子元素，那么在某些情况下，很可能会得到空白的文本节点：

```
var node = $('example').firstChild
```

而在使用了cleanWhitespace()方法之后

```
var node = $('example').cleanWhitespace().firstChild
```

由于把HTML代码转换成了

```
<ul id="example"><li>Item 1</li><li>Item 2</li></ul>
```

所以可以确保firstChild属性始终会引用第一个<li>节点。

### 5. 使用YUI检查相交的元素

YAHOO.util.Dom.getRegion(String|HTMLElement|Array)方法能够返回一个YAHOO.util.Region对象，该对象会包含相应元素在页面中的上、左、下、右的位置。因此，可以将该方法返回的对象与另一个区域方法intersect()结合使用，来测试基于页面中的定位和样式

www.TopSage.com

规则计算的两个区域是否相交：

```
var region1 = YAHOO.util.Dom.getRegion('region1');
var region2 = YAHOO.util.Dom.getRegion('region2');
if(region1.intersect(region2)) {
    alert('The regions intersect!');
}
```

### 6. 迭代结果

我要说的另一个问题就是迭代操作。虽然迭代与DOM操作并不直接相关，但Prototype和jQuery的连缀语法都提供了迭代元素列表的非常清晰的方式。在Prototype中，如果要查找页面中所有的锚并在随后修改它们，可以使用each()方法，该方法的回调方法将获得这些元素及它们的索引：

```
// Prototype库通过each()方法迭代元素
// 迭代本章前面的singleImageAnchors变量中包含的
// 元素列表，并为每个元素添加一个hasOneImage类
singleImageAnchors.each(function(e,i){
    e.addClassName('hasOneImage');
});
```

在jQuery中，同样的操作也可以使用类似的each()方法来完成，不过，该方法的回调函数只取得每个元素的索引，而由于每个回调函数是在相应节点的环境中执行，所以通过this可以引用相应节点：

```
// Prototype库通过each()方法迭代元素
// 迭代本章前面的singleImageAnchors变量中包含的
// 元素列表，并为每个元素添加一个hasOneImage类
singleImageAnchors.each(function(i){
    jQuery(this).addClass('hasOneImage');
});
```

MochiKit等其他的库也提供了单独的方法，例如forEach()，用于迭代元素的列表：

```
// MochiKit库通过foEach()方法迭代元素
// 迭代本章前面的singleImageAnchors变量中包含的
// 元素列表，并为每个元素添加一个hasOneImage类
MochiKit.Iter.forEach(singleImageAnchors,function(e){
    MochiKit.DOM.addElementClass(e,'hasOneImage');
});
```

> 此外，MochiKit还提供了一个可以用来创建高级迭代对象的框架。要了解该框架的更多信息，请参阅http://www.mochikit.com/doc/html/MochiKit/Iter.html。

## 9.3 处理事件

正如我们在第4章中所讨论的，事件是用户交互的源头。如果没有事件，那么网页中的活力也就荡然无存。

在你创建的ADS库中，已经包含了一些基本的事件方法，但如果你选择使用其中一个库，将

www.TopSage.com

会发现这些库都有各自内置的事件处理方式。而且，这些库都提供了注册及调用自定义事件的能力，而自定义事件与内置的浏览器事件或者W3C事件都有所不同。

## 9.3.1 注册事件

在你的ADS库中，是通过`ADS.addEvent()`方法为元素添加事件的：

```
// ADS库中的事件注册
// 通过window的load事件来添加click事件侦听器
// 以便在一个新窗口中打开#source的链接
ADS.addEvent(window,'load',function() {

    ADS.addEvent(ADS.$('source'),'click',function(W3CEvent) {
        // 通过现有的href值打开新窗口
        window.open(this.href);
        // 防止链接的默认动作
        var event = ADS.getEventObject(W3CEvent);
        ADS.eventPreventDefault(event);
    });
});
```

而每个库也都有与此类似的方法。

### 1. DOMAssistant方式

在DOMAssistant库中，`window`的`load`事件要求开发者手工编辑`DOMAssistantLoad.js`文件，以包含需要运行的载入事件：

```
DOMAssistant.functionsToCall = [
    initPage
];
```

而针对元素的事件侦听器则可以使用`DOMAssistant.$( id ).addEvent()`方法，在选择的元素上面进行注册。注册的模式与`ADS.addEvent()`方法相同：

```
// DOMAssistant库中的事件注册
// 通过window的load事件来添加click事件侦听器
// 以便在一个新窗口中打开#source的链接
DOMAssistant.$('source').addEvent('click', function(event) {
    // 通过现有的href值打开新窗口
    window.open(this.getAttribute('href'));
    DOMAssistant.$(this).addClass('popup');
    // 防止链接的默认动作
    DOMAssistant.preventDefault(event);
});
```

遗憾的是，DOMAssistant库与你的ADS库类似，也没有可以手动调用事件的方法。

### 2. jQuery方式

jQuery处理事件的方式则略有不同。除了可以像使用`DOMAssistant`的`addEvent()`方法那样使用jQuery的`bind()`方法之外，jQuery中的元素还继承了下列与事件相关的方法：

- `blur(callback)`
- `change(callback)`
- `click(callback)`

www.TopSage.com

- dblclick(callback)
- error(callback)
- focus(callback)
- hover(mouseover-callback, mouseout-callback)
- keydown(callback)
- keypress(callback)
- keyup(callback)
- load(callback)
- mousedown(callback)
- mousemove(callback)
- mouseout(callback)
- mouseover(callback)
- mouseup(callback)
- ready(callback)
- resize(callback)
- scroll(callback)
- select(callback)
- submit(callback)
- unload(callback)

通过使用这些方法，可以为整批的DOM元素注册事件侦听器的回调函数，例如可以为页面上每个链接的click事件侦听器添加回调函数：

```
// jQuery中的事件注册
// 为每个锚添加click事件侦听器
// 以便在新窗口中打开锚的链接
$('a').click( function(event) {
    // 通过现有的href值打开新窗口
    window.open(this.getAttribute('href'));
    jQuery(this).addClass('popup');
    // 防止链接的默认动作
    return false;
});
```

而且，即使在调用以上方法时没有任何输入，也会调用相应的事件侦听器：

```
// jQuery中调用事件
// 调用页面上第一个锚的click事件
$('a:first').click();
```

在这些方法中，我特别欣赏的是hover()方法。我们在第1章结尾时研究的JavaScript翻转图的例子，如果使用hover()方法还可以得到进一步简化：

```
// 通过jQuery实现第1章的翻转图示例
jQuery(document).ready( function() {
    jQuery('a.multiStateAnchor').each(function() {
```

```
            // 保持anchorImage位于this的作用域中
            var anchorImage;
            if(!(anchorImage = jQuery('img:first',this))) return;

            // 解析扩展名
            var src = anchorImage.attr('src');
            var extensionIndex = src.lastIndexOf('.');
            var path = src.substr(0,extensionIndex);
            var extension = src.substring(
                extensionIndex,
                src.length
            );

            // 预载图像
            var imageMouseOver = new Image()
            imageMouseOver.src = path + '-over' + extension;
            var imageMouseDown = new Image()
            imageMouseDown.src = path + '-down' + extension;

            // 注册事件侦听器
            jQuery(this).hover(
                function() {
                    anchorImage.attr('src',imageMouseOver.src);
                },
                function() {
                    anchorImage.attr('src',path + extension);
                }
            );
            jQuery(this).mousedown(function() {
                anchorImage.attr('src',imageMouseDown.src);
            });
            jQuery(this).mouseup(function() {
                anchorImage.attr('src',path + extension);
            });
        });
    });
```

## 9.3.2 自定义事件

在处理浏览器与文档之间的交互作用时，内置的事件和事件侦听器可以大显威力。因为当用户与各种文档元素进行交互时，浏览器会根据用户的操作持续地调用相关的事件。然而，在DOM文档内部，由于无法控制鼠标指针，因而事件侦听器成了与之交互的唯一途径。不过，同样的思想也适用于在多个脚本之间进行交互。下面我们还以第6章案例研究中的图像编辑器为例。

当完成缩放和裁剪操作之后，图像编辑器的代码要负责通过当时只是简单填充的saveClick()方法，将修改后的结果传送回服务器：

```
// 将修改的结果保存回服务器
imageEditor.saveClick = function(W3CEvent) {
    alert('This should save the information back to the server.');
    imageEditor.unload();
}
```

这里的问题是，由于服务器处理过程不同，图像编辑器中的脚本需要基于每次安装进行定制

www.TopSage.com

开发。此外，为了与编辑后的图像保持一致，修改相应的表单输入并更新嵌入的图像也是必要的。为了创建一个更方便的、自包含的对象，而且这个对象不需要与服务器进行任何交互，那么saveClick()方法可以简单地在原始的图像上面调用一个自定义的事件。最后，由页面的开发者负责注册适当的侦听器，以完成对实际图像文件的缩放处理。

在注册和调用自定义的事件侦听器时，可以使用常规的事件注册方法，例如MochiKit的MochiKit.Signal.connect()方法。而使用该方法注册的事件侦听器可以在随后通过MochiKit.Signal.signal()方法来调用。如果将这个过程应用到imageEditor.saveClick()方法的环境中，我们可以调用一个名为editComplele的事件，并为该事件传递一个包含图像大小和裁剪信息的对象作为参数：

```
imageEditor.saveClick = function(W3CEvent) {
    MochiKit.Signal.signal(
        imageEditor.DOMObjects.originalImage,
        'editComplete',
        {
            imageWidth:imageEditor.DOMObjects.resizee.style.width,
            imageHeight:imageEditor.DOMObjects.resizee.style.height,
            cropTop:imageEditor.DOMObjects.cropArea.style.top,
            cropLeft:imageEditor.DOMObjects.cropArea.style.left,
            cropWidth:imageEditor.DOMObjects.cropArea.style.width,
            cropHeight:imageEditor.DOMObjects.cropArea.style.height
        }
    )
    imageEditor.unload();
}
```

这样，图像编辑器的唯一职责就是“发信号通知”原始的图像以指明更改结果。而为了服务器端部分运行的需要，原始的图像必须通过MochiKit.Signal.connect()方法，事先注册一个用以处理自定义的editComplete事件的侦听器：

```
MochiKit.Signal.connect(
    MochiKit.DOM.get('the-image'),
    'editComplete',
    function(event) {
        var properties = event.event();
        // 通过properties.imageWidth等属性完成相应操作
    }
);
```

下面是使用jQuery库实现同一思想的代码：

```
// jQuery库的自定义事件注册
// 为editComplete方法注册一个事件侦听器
$('the-image').bind(
    'editComplete',
    function(event,imageWidth,imageHeight,cropTop, ➥
cropLeft,cropWidth,cropHeight) {
        // 通过imageWidth等属性完成相应操作
    }
);

// jQuery库使用参数调用自定义的事件
```

www.TopSage.com

```
// 以适当的属性调用editComplete方法
imageEditor.saveClick = function(W3CEvent) {
    $(imageEditor.DOMObjects.originalImage).trigger(
        'editComplete',
        [
            imageEditor.DOMObjects.resizee.style.width,
            imageEditor.DOMObjects.resizee.style.height,
            imageEditor.DOMObjects.cropArea.style.top,
            imageEditor.DOMObjects.cropArea.style.left,
            imageEditor.DOMObjects.cropArea.style.width,
            imageEditor.DOMObjects.cropArea.style.height
        ]
    )
    imageEditor.unload();
}
```

利用像这样的自定义事件，可以分离出自定义的逻辑部分，进一步提高脚本的重用程度。而且，这个经过改进的图像编辑器脚本，现在也可以用于任何使用了同一个库的Web应用程序中了。

## 9.4 访问和操纵样式

所有这些库都没有提供比当前的ADS库中更多的CSS样式方法。不过，有些库还包含了与定位有关的方法。比如，Prototype中就包含下列方法。

- Element.cumulativeOffset()：返回元素相对于文档左上角位置的左上角位置。
- Element.relativize()：在不修改元素在页面上当前位置的前提下，将元素转换为相对定位的元素。这个方法特别适合于对某个元素的子元素进行绝对定位。

在第12章研究如何美化<select>列表中的案例中，你还会看到这两个方法。

## 9.5 通信

本章将要讨论的最后一个方面是浏览器通信。由于Ajax呈现出来的爆发态势，JavaScript库也变得越来越流行起来。许多库中为首的对象都是Ajax对象，至少Ajax对象是相应的库流行的因素之一。

在我们选定的这些库中，DOMAssistant是唯一一个没有内置Ajax对象的库。表面上看，这似乎给作为一个成品库的DOMAssistant打上了消极的印记，但事实上却并非如此。因为Ajax工作流是一个不该轻易触及的棘手的新事物。鉴于我们在本书第二部分中介绍的那些新问题，将Ajax对象排除在一个库之外，实际上在多数情况下都是一种积极的表现。不过，在学习完第7章和第8章之后，你已经熟悉了Ajax的所有非常规行为，那么再使用某个库对象也就不是那么不好接受了。

### 1. Prototype的Ajax对象

Prototype库中的Ajax对象大概最受欢迎了，该对象是随着Ruby on Rails框架流行起来的。Prototype库提供了如下几种不同风格的Ajax方法。

- Ajax.Request(url[, options])，执行一个基本的XMLHttpRequest请求。
- Ajax.Updater(element, url[, options])，是对一个请求的包装方法，用于自动地将请求的内容添加到给定的DOM节点中。

www.TopSage.com

- Ajax.PeriodicalUpdater(element, url[, options])，按照固定的间隔时间将请求的内容自动添加到给定的DOM节点中。它不是对Ajax.Updater()的包装方法，因此像evalJSON之类的方法是无效的。而且，在该方法内部也覆盖了onComplete回调方法，以便完成对DOM元素的更新。因而，必须通过onSuccess方法来运行附加的代码（不过onSuccess会发生在onComplete更新发生之前）。

其中，每个方法中的options参数，与我们在第7章中创建的ADS.ajaxRequest方法中的options相似，都包含以下属性。

- asynchronous，切换对象的异步和同步（阻塞）模式。不过，正如我们在第7章中所讨论过的，应该始终将这个属性指定为true。默认值是true。
- contentType，是请求的Content-Type头部信息。默认值是application/x-www-form-urlencoded。
- encoding，是对请求内容的编码方式。通常不必修改此属性。默认值是UTF-8。
- method，是请求的HTTP方法。Prototype会以post请求覆盖很多其他请求（如put和delete），并将原始的请求方法保存在请求的_method参数中。默认值是post。
- parameters，是随同请求一起发送的参数。可以像包含一个get请求那样，使用URL编码的字符串来定义这个属性；也可以使用任何兼容hash的对象，如数组或以属性名表示参数名的对象。
- postBody，默认值为null，是包含在POST请求主体中的内容。如果为空，请求主体中将会包含parameters属性的内容。
- requestHeaders，是一个表示在请求中包含的额外头部信息的对象或数组。如果是对象，则对象的属性名和值分别表示请求头部信息的名称和值。如果是数组，问题会稍微复杂一点，此时以偶数索引的项（从0开始算）表示头部信息的名称，而以奇数索引的项（从1开始算）表示头部信息的值。在默认情况下，如果不被覆盖的话，Prototype会包含以下几个头部信息。
- X-Requested-With：默认情况下，设置为XMLHttpRequest。
- X-Prototype-Version：设置为Prototype的当前版本号。
- Accept：默认情况下，设置为text/javascript, text/html, application/xml, text/xml,*/*。
- Content-type，根据contentType值和encoding选项构建。

除了这些属性之外，还有一些可以用于在请求过程的不同阶段或者基于服务器的响应，来运行代码的回调方法。下列每个回调方法都将以两个参数被调用，其中一个参数是XMLHttpRequest对象（或等价的IE ActiveX对象），另一个参数是当且仅当响应中包含X-JSON头部信息时由响应返回的JavaScript对象。如果不存在X-JSON头部信息，那么第2个参数的值为null。这种情况的唯一例外是onException回调方法，这个回调方法会取得Ajax.Request实例作为其第1个参数，而将exception对象作为其第2个参数。以下回调方法是按照它们在请求中被调用的顺序列出的。

www.TopSage.com

- onException(ajax.request,exception)，可能在请求或响应中发生错误时被调用，也可能会在以下回调方法中的任何一点被调用。
- onUninitialized(XHRrequest,json)，可能在创建请求对象时被调用，不过由于不一定总会成功调用该方法，所以应避免使用它。
- onLoading(XHRrequest,json)，可能在请求对象已建立且其连接已启动时被调用，不过同样，这个方法也不一定总能成功调用，所以要避免使用它。
- onLoaded(XHRrequest,json)，可能在请求对象已建立、连接已启动且已准备就绪发送请求时被调用。不过，仍然由于这个方法不一定总能成功调用，所以要避免使用它。
- onInteractive(XHRrequest,json)，可能在请求对象已获得部分响应且正在等待剩余请求完成时被调用。没错，你猜对了，这个方法也不一定总能成功调用，所以要避免使用它。
- on###(XHRrequest,json)，会在提供的适当响应代码已设置完成时被调用。###表示响应的HTTP状态代码。而且，这个回调方法会在请求刚刚完成且调用onComplete回调方法之前被调用。此外，它也将阻止onSuccess和onFailure回调方法的执行。
- onFailure(XHRrequest,json)，会在请求完成但其状态代码虽有定义却未处于200和300之间时被调用。
- onSuccess(XHRrequest,json)，会在请求完成且其状态代码未定义或有定义且处于200和300之间时被调用。
- onComplete(XHRrequest,json)，会在请求的最后作为链中的最后一个可能的回调方法被调用。

Prototype中也包含一个全局的Ajax.Responders方法，用来控制和访问进出于各种Ajax.Request方法中Ajax请求。要了解Ajax.Responders方法的更多内容，请参阅Prototype的在线文档http://www.prototypejs.org/api/ajax/responders。

如果把所有这些内容付诸实践，那么我们就可以通过示例文件chapter9/Prototype/prototype.html看到许多不同的请求：

```
// Prototype库的Ajax.Request对象
// 创建一个新的一次性请求并记录其成功状态
new Ajax.Request(
    '../ajax-test-files/request.json',
    {
        method:'get',
        onSuccess: function (transport) {
            var response = transport.responseText || ↵
"no response text";
            ADS.log.write('Ajax.Request was successful: ' + response);
        },
        onFailure: function (){
            ADS.log.write('Ajax.Request failed');
        }
    }
);
```

www.TopSage.com

```
// Prototype库的Ajax.Updater对象
// 创建一个一次性请求以responseText
// 的内容生成#ajax-updater-target元素
new Ajax.Updater(
    $('ajax-updater-target'),
    '../ajax-test-files/request.json',
    {
        method: 'get',
        // 添其添加到目标元素的顶部
        insertion: Insertion.Top
    }
);

// Prototype库的Ajax.periodicalUpdater对象
// 创建一个周期性的请求，每10秒钟
// 自动更新一次#ajax-target-element
new Ajax.PeriodicalUpdater(
    $('ajax-periodic-target'),
    '../ajax-test-files/periodic.json',
    {
        method: 'GET',
        // 将其添加到现在内容的顶部
        insertion: Insertion.Top,
        // 每10秒运行一次
        frequency: 10
    }
);
```

另一种非常简单但却非常实用的Ajax.Request对象的用法，就是间歇性地保存表单中的信息。这种用法对于博客网站非常实用，因为你可能会在页面上花较长时间地写一些东西，在此期间没有任何手段可以真正保存你的成果。而Ajax.Request()对象配合Prototype库的Form序列化方法，恰好可以每隔一定的时间取得一次表单中的当前信息，然后将其保存到服务器上，从而确保你不会意外丢失全部劳动成果：

```
// 使用Prototype库实现自动保存
// 每30秒钟保存一次#autosave-form的内容
// 并更新#autosave-status以标明已经保存
setTimeout(function() {
    new Ajax.Updater(
        $('autosave-status'),
        '../ajax-test-files/autosave.json',
        {
            method:'post',
            parameters : $('autosave-form').serialize(true)
        }
    );
},30000);
```

### 2. jQuery保持Ajax简单

jQuery也包含可以在其中指定各种各样属性的低层jQuery.ajax方法。不过，我更喜欢的则是可以通过最少的工作量获得所需信息的一些简捷易用的方法：

- jQuery.post(url, params, callback)，通过一个POST请求获得数据。
- jQuery.get(url, params, callback)，通过一个GET请求获得数据。

www.TopSage.com

- jQuery.getJSON(url, params, callback)，获得一个JSON对象。
- jQuery.getScript(url, callback)，获得并执行一个JavaScript文件。

以上这些方法全都包装了jQuery.ajax()方法及相应的回调方法，而回调方法也都是在jQuery.ajax()的成功回调方法中被调用。每个回调方法中取得的两个参数分别定义了请求的responseText和请求的状态：

```
jQuery.get('../ajax-test-files/request.json',
    { key: 'value' },
    function(responseText, status){
        // 你的代码
    }
);
```

其中的status可能的值如下：

- success
- error
- notmodified

对于getJSON()和getScript()这两个方法来说，都会对响应进行求值，因而getJSON()方法的参数将是一个JavaScript对象。

同样地，在示例文件chapter9/jQuery/jquery.html中，你也可以看到使用上面方法的实际效果：

```
// 简捷调用Ajax的jQuery.get()方法
// 创建一个一次性请求并记录其成功状态
jQuery.get('../ajax-test-files/request.json',
    { key: 'value' },
    function(responseText,status){
        ADS.log.header('jQuery.get()');
        ADS.log.write('status: ' + status);
        ADS.log.write('succesful: ' + responseText);
    }
);

// jQuery.getJSON() 载入一个JSON对象
// 创建一个一次性请求载入一个JSON文件并记录其成功状态
jQuery.getJSON('../ajax-test-files/request.json', function(json){
    ADS.log.header('jQuery.getJSON()');
    ADS.log.write('succesful: ' + json.type);
});
```

jQuery库中也包含一个附加的load()方法：

- jQuery(expression).load(url, params, callback)，将URL的结果载入到DOM元素中。

这个方法的用途与Prototype库的Ajax.updater()方法相同，也能够自动根据结果生成一个或多个元素：

```
// jQuery(..).load()方法自动生成元素
// 创建一个一次性的请求并根据responesText
// 的内容来生成#ajax-updater-target元素
jQuery("#ajax-updater-target").load(
```

www.TopSage.com

```
        '../ajax-test-files/updater.json',
        { key: 'value' },
        function(responseText,status) {
            ADS.log.header('jQuery(...).load()');
            ADS.log.write('status: ' + status);
            ADS.log.write('succesful: ' + responseText);
            ADS.log.write('jQuery(\'#ajax-updater-target\').load succesful');
        }
    );
```

而且，这个jquery()方法同样也能用来实现周期性的保存功能：

```
// 使用jQuery库实现自动保存
// 每10秒钟保存一次#autosave-form的内容
// 并更新#autosave-status以标明已经保存
setTimeout(function() {
    jQuery('autosave-status').load(
        '../ajax-test-files/autosave.json',
        jQuery.param({
            title:jQuery('#autosave-form input[@name=title]').val(),
            story:jQuery('#autosave-form textarea[@name=story]').val()
        })
    );
},10000);
```

此外，jQuery库还有许多可用的插件，例如Mike Alsup的Ajax Form插件（http://docs.jquery.com/Plugins），能够使处理表单和Ajax更加方便。需要通过Ajax来提交一个评论表单吗？只需如下这么简单：

```
$('#commentForm').ajaxForm(function() {
    alert("Thank you for your comment!");
});
```

这个方法会将表单的内容进行序列化，然后再将内容发送到表单的action属性中引用的脚本中去处理。

## 9.6 小结

在本章中，我们只是简单介绍了对日常脚本编程大有帮助的一些库的重要特性，这些库包括DOMAssistant、jQuery、MochiKit、Prototype和YUI等。由于没有完整地讨论任何一个库，所以我高度建议你浏览一下每个库的文档，看看它们还包含其他什么特性。

当对哪个库适合需要作出评估时，要保证全面仔细地分析每一个库。要考虑的因素包括适当的命名空间、特性太少还是太多、强大的社区以及良好的支持等。而当选定了某个库之后，要确保物尽其用——尽量用好、用足这个库所提供的全部特性。与此同时，还需要花时间掌握这个库的开发原理和过程。换句话说，只要不对其停留在想当然的层面上，依赖于一个库也是个明智的选择。

在第10章，我们会从另一个角度来看看本章讨论的库及另外几个库所提供的众多简单的视觉增强特性。而且，在使用适当的前提下，这些视觉增强能够使你的Web应用程序获得锦上添花般的提升。

www.TopSage.com

第 10 章

# 添加效果增强用户体验

HTML本身确实枯燥乏味。无论你怎么想，事实上每个上网者（包括你自己）都会希望在你的网站上看到光彩夺目的效果。不过，把握好炫目的效果与网站交互性之间的平衡是关键所在。在谈到效果及其有用性的时候，莎士比亚的名言“闪光的并不都是金子”总是给我警示。没错，因为可以添加某个效果，并不意味着就应该添加那个效果。还有人记得`<blink>`这个标签吗？所以，这里要问的一个最重要的问题是：“我的效果能否增强用户的体验？”而多数情况下的回答都是“能”。

如果使用得当的话，精细的效果能够对容易被人忽视的变化给出视觉上的提示。效果能唤起人们对界面上特定方面的注意，并引导交互按照正确的方向进行。或者效果只是使读者产生共鸣和愉悦感——为枯燥的老式HTML增添一点活力。

本章将讨论一些适合应用效果的技术及情形。同时，还会介绍`Script.aculo.us`及`Moo.fx`这两个库，它们都能省去创建效果的很多麻烦。而我们将利用这两个库的部分特性，在订货表单中创建更好的错误反馈。此外，还将创建一个可拖放的购物车。

## 10.1 自己动手实现效果

JavaScript中的所有效果都基于一点——时间。其他的动画插件，如Adobe Flash，是基于“每秒帧数”的单个视觉画面的。但是，常规的DOM脚本没有提供这种特性。因为其速度取决于很多可变的因素，包括浏览器、用户的计算机以及其他因素。要完成位移、动画或任何随时间推移而变化的效果，都要依赖JavaScript的核心方法`setTimeout()`或`setInterval()`。

在需要经过一定的时间间隔再处理某个事件的情况下，我们曾多次使用`setTimeout()`和`setInerval()`方法。这两个方法之间的唯一区别，在于调用回调函数的次数：

- `setTimeout(callback, milliseconds)`，会在经过给定的毫秒数之后调用一次回调函数。
- `setInterval(callback, milliseconds)`，会在经过给定的毫秒间隔之后无限次地调用回调函数，直至使用`clearInterval()`移除间隔函数。

这两个方法通过间隔一定的时间（在浏览器和计算机能够维持的条件下）执行回调函数，提

www.TopSage.com

供了“每秒帧数”的效果。如果想要实现动画、过渡或其他效果，只需编写一个渐进地操纵目标元素的函数，然后通过`setInterval()`来运行该函数即可。比如下面所示的chapter10/dom-animation/animation.html中的例子：

```
ADS.addEvent(window,'load',function() {
    // 将一个元素从其当前位置
    // 向右下方移动+300px
    var moveMe = document.getElementById('element-id');
    ADS.setStyle(moveMe,{
        position:'absolute',
        border:'1px solid black',
        width:'100px',
        height:'20px'
    });

    var startLeft = moveMe.offsetLeft;
    var startTop = moveMe.offsetTop;

    // 创建间隔
    var mover = setInterval(function() {
        var remove = false;
        var currentLeft = moveMe.offsetLeft;
        var currentTop = moveMe.offsetTop;

        // 移动以2像素递增
        var newLeft = currentLeft + 2;
        var newTop = currentTop + 2;
        if (newLeft > startLeft + 300 || newTop > startTop + 300) {
            // 如果新位置超出了期望的
            // 目标，则重置相应的值
            newLeft = startLeft;
            newTop = startTop;
        }

        // 重新定位元素
        moveMe.style.left = newLeft + 'px';
        moveMe.style.top= newTop + 'px';
    },10);
});
```

通过运行以上例子可以发现，完成随时间推移而变化的效果并非难事。其中的关键是编写由`setInterval()`调用的回调函数，该函数负责操纵文档中所有的元素。而提供操纵元素的方法则是我们提到的库的核心所在。由于这些库也提供了间隔方法的包装方法，所以，我们可以不必像上面这个例子中那样手工设置它们。

### 10.1.1 让我看到内容

回忆一下可访问性。在开始向Web应用程序中添加效果时，要保证不会降低应用程序自身的可访问性。在下面图10-1所示的例子中，元素由开始时的不透明度为0%的完全透明，过渡到了不透明度为100%的完全不透明。

www.TopSage.com

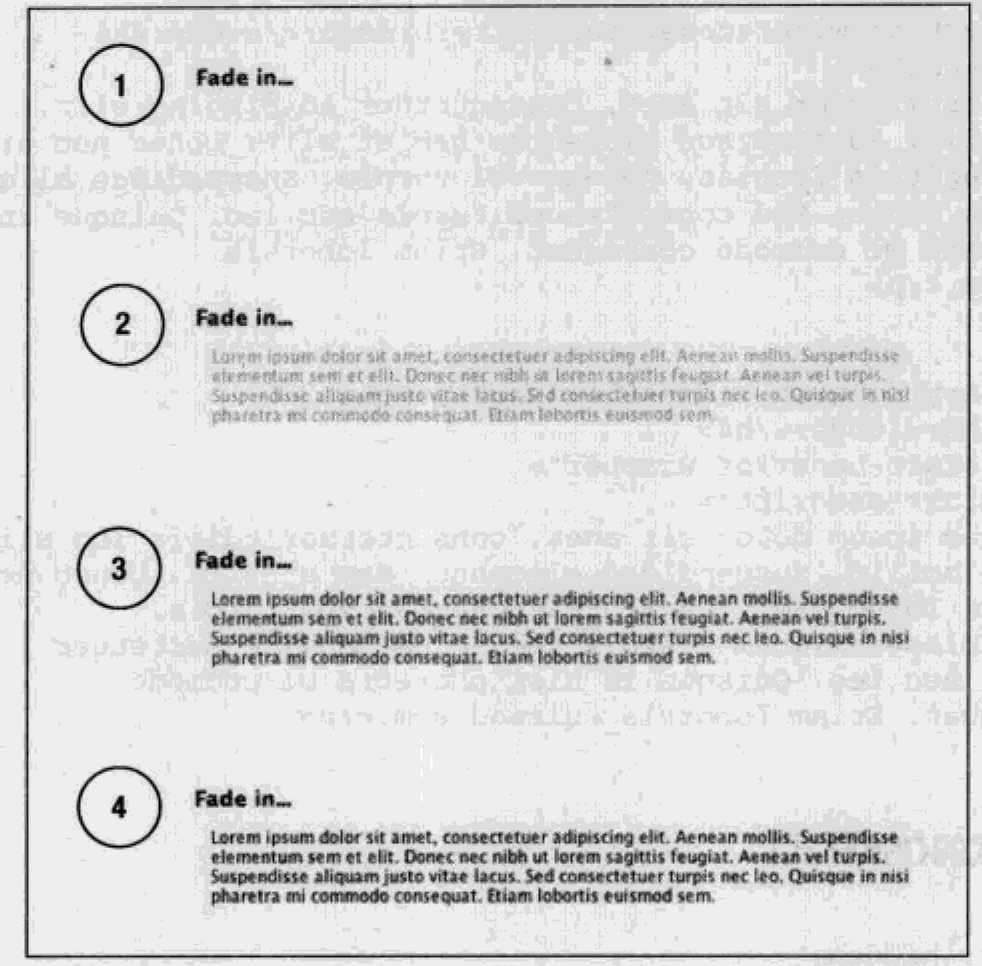



图10-1 文本段落的不透明度从0%到100%淡入过程中的四个步骤

这里面有一个大问题，你可能没有意识到。为了设置这个效果的初始状态，需要在CSS样式表中将元素的visibility设置为hidden，或者将display设置为none。但是，如果浏览器支持CSS但不支持JavaScript会怎样呢？这样的话，淡入的元素将永远不会出现，因为没有脚本可以让它显示出来。

对此，一个简单的解决方案就是通过像下面这样的load事件在一开始隐藏元素。于是，当DOM脚本无法运行时，元素依然是可见的：

```
ADS.addEvent(window, 'load', function() {
    ADS.setStyle('element-id', { 'visibility' : hidden });
});
```

虽然这个方案适用于许多情况，但有可能导致不协调的结果。比如，像我们在第4章中所讨论的，如果页面的load事件被调用得过晚，那么在页面载入期间元素也会可见。

另一个方案是使用<noscript>元素。<noscript>元素通常用于显示嵌入的JavaScript的备用内容。由于本书所有的DOM脚本都放在外部文件中，所以<noscript>是不必要的。但是，我们仍然可以在必要时利用这个元素的特性。<noscript>元素的内容会在JavaScript无效时，被解释为DOM树的组成部分，而在JavaScript有效时被隐藏起来。唯一需要注意的是，<noscript>的内容只会作为<noscript>节点的文本节点存在，而不是经过解析的DOM片段。

为了利用<noscript>元素中隐藏的内容，需要在页面载入时取得这些内容并将它们添加到文档中。对于下面的HTML代码片断（位于测试文件chapter10/noscript/noscript.html中）：

TopSage.com

www.TopSage.com

```
<h4>Fade in starting with <code>visibility:hidden;</code></h4>
<div id="start-css">
    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
    Aenean mollis. Suspendisse elementum sem et elit. Donec nec nibh
    ut lorem sagittis feugiat. Aenean vel turpis. Suspendisse aliquam
    justo vitae lacus. Sed consectetuer turpis nec leo. Quisque in
    nisi pharetra mi commodo consequat. Etiam lobortis
    euismod sem.</p>
</div>

<h4>Fade in starting with <code>&lt;noscript&gt;
&lt;/noscript&gt;</code></h4>
<noscript id="start-noscript-wrapper">
    <div id="start-noscript">
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
        Aenean mollis. Suspendisse elementum sem et elit. Donec nec
        nibh ut lorem sagittis feugiat. Aenean vel turpis.
        Suspendisse aliquam justo vitae lacus. Sed consectetuer
        turpis nec leo. Quisque in nisi pharetra mi commodo
        consequat. Etiam lobortis euismod sem.</p>
    </div>
</noscript>
```

和下面用于测试的CSS样式：

```
#start-css {
    visibility: hidden;
}
```

可以使用常规的老式DOM方法及innerHTML，或者专门的库方法（如这里展示的jQuery库的方法），取得<noscript>中的内容并将其添加为原<noscript>元素的同辈元素：

```
jQuery.noConflict();

ADS.addEvent(window,'load',function() {

    // 使用moo.fx库来实现#start-css element
    // 从透明到不透明的过渡效果
    var myFx = new Fx.Style(
        'start-css',
        'opacity',
        {duration:2000}
    ).start(0,1);

    // 首先取得<noscript>的内容，并将这些
    // 内容添加为<noscript>元素的同辈元素。
    // 可以移除<noscript>元素，但不是必需的
    var wrapper = jQuery('#start-noscript-wrapper');

    var content = wrapper.text() || wrapper.html();
    if(content) {
        jQuery('#start-noscript-wrapper').after(content);

        // 使用moo.fx库来实现#start-css element
        // 从透明到不透明的过渡效果
        var myFx = new Fx.Style(
            'start-noscript',
```

www.TopSage.com

```
            'opacity',
            {duration:2000}
        ).start(0,1);
    }

});
```

> 由于Safari不支持以DOM方式访问<noscript>元素的内容，所以这个例子当前在Safari中还无法正确运行。

这样，无论是在哪种情况下，都能获得适当的效果。如图10-2所示，在JavaScript有效时，两个段落都是可见的。

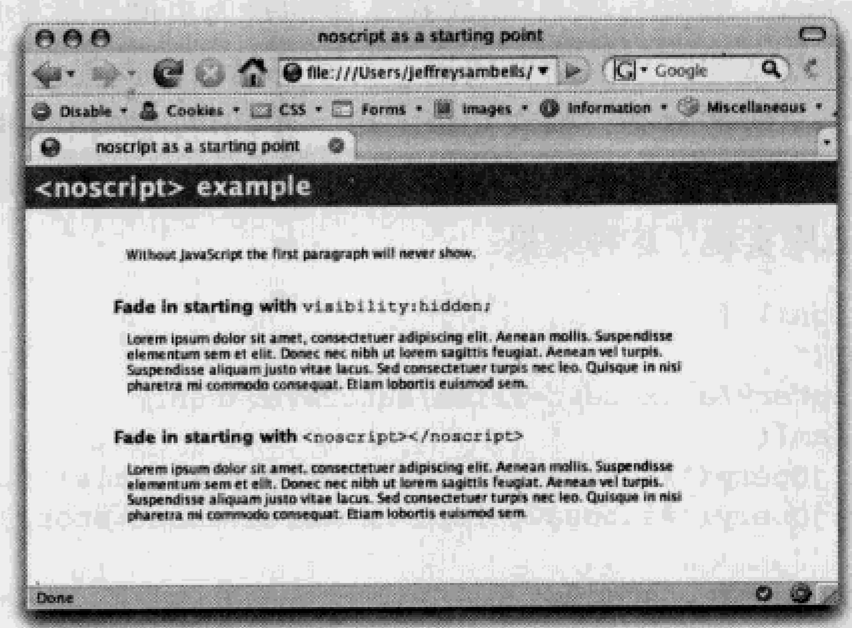



图10-2 在启用JavaScript的浏览器中观察以上示例页面的效果

但在浏览器支持CSS而禁用了JavaScript的情况下，则只有第2段文本可以访问，如图10-3所示。

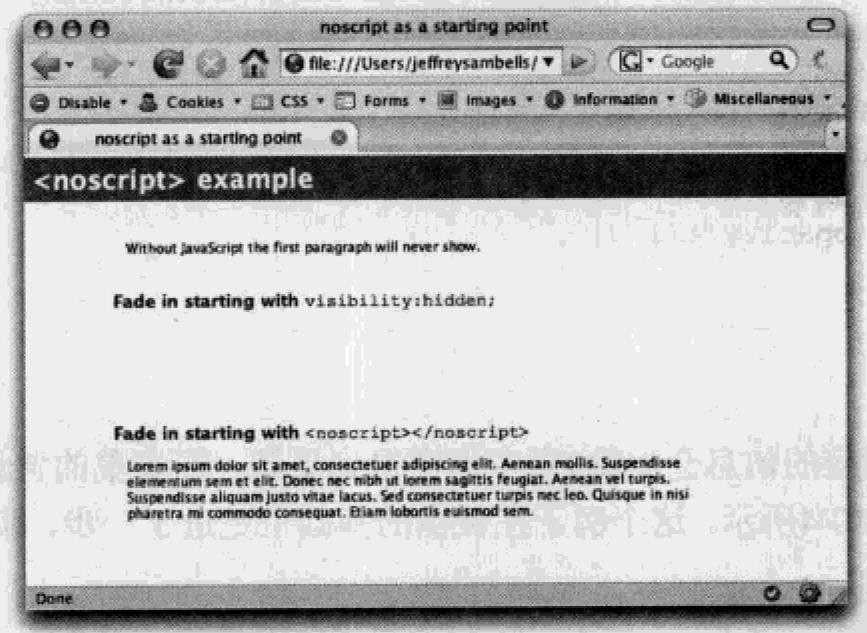



图10-3 在未启用JavaScript的浏览器中观察以上示例页面的效果

www.TopSage.com

## 10.1.2 提供反馈

精致的效果说起来是很令人引以为荣的，“嘿！看这儿！开始变了！”。而且，在使用Ajax工作流时这些效果非常有用。我们在本书第二部分讨论过，基于Web应用程序的通信模式更新用户界面是至关重要的。在传统的工作流中，用户操作与变化都与页面的重载相关。而对于Ajax来说，由于修改的只是页面中的某个局部（如果有），不会有明显变化的迹象，所以需要将这个变化标识出来。

### 1. 黄褪技术

一种比较流行的方法，是由37signals（http://37signals.com/）的旗舰产品Basecamp所展示的YFT（Yellow Fade Technique，黄褪技术）。这种技术指的是为页面中受影响部分的背景，添加细微的黄白渐变效果。这种细微却明显的渐变效果，为动态修改页面提供了必要的反馈手段。

黄褪技术虽然算不上高级或复杂，但它却很实用。我们可以把相同的逻辑也应用到第9章介绍的自动保存机制中，比如例子页面chapter10/fade-technique/fade.html，就综合使用了jQuery库进行Ajax调用，使用Moo.fx库实现了这种效果：

```
setInterval(function() {
    jQuery.getJSON(
        '../../chapter9/ajax-test-files/autosave.json',
        jQuery.param({
            title:jQuery('#autosave-form input[@name=title]').val(),
            story:jQuery('#autosave-form textarea[@name=story]').val()
        }),
        function(response, status) {
            if(status == 'success') {
                var color = '#00ff00';
            } else {
                var color = '#ff0000';
            }
            jQuery('#autosave-status').html(response.message)
            var myFx = new Fx.Styles(
                'autosave-status',
                {duration:2000}
            ).start({
                'background-color':[color,'#ffffff'],
                'opacity':[1,0]
            });
        }
    );
},5000);
```

当保存表单时，相应的信息会如第9章中那样得到更新，而表单的背景也会从一种颜色渐变为白色，大致过程如图10-4所示。这个例子在黄褪的基础上更进了一步，即以绿色表示保存成功，而以红色表示保存失败。

www.TopSage.com

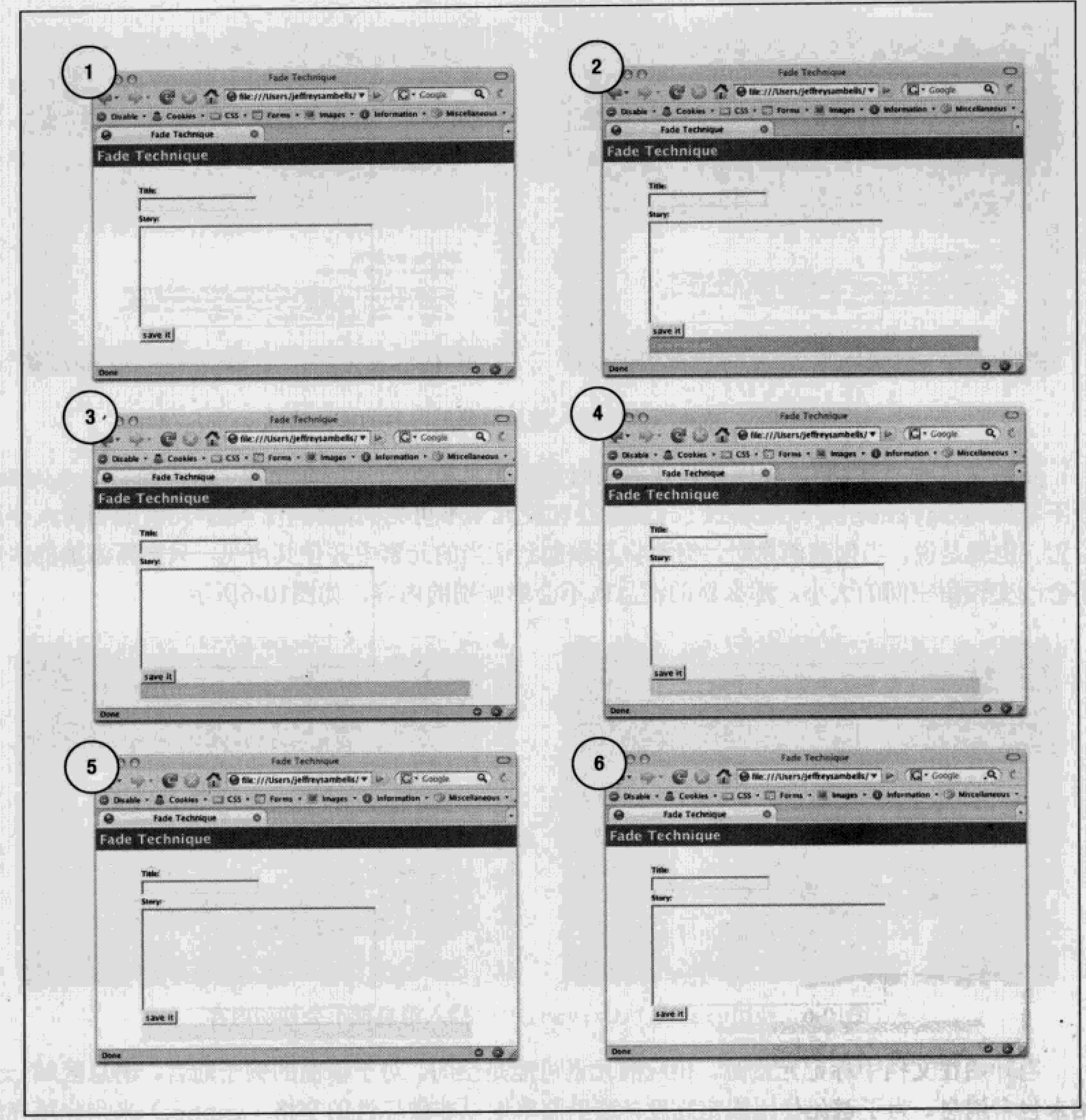
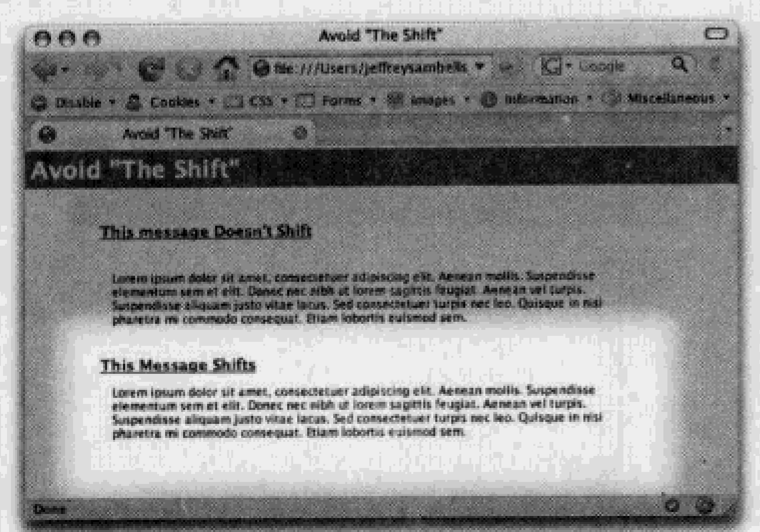



图10-4 对第9章自动保存的示例进行细微增强后的效果

## 2. 避免移动内容

在使用效果时的另一个常见问题就是移动内容。当新内容出现在文档中时，是要占用空间的。如果在开始的布局中没有考虑到为新内容预留空间，那么其余的内容将为了适应新内容的出现而移动位置。在某些情况下，移动内容可能是预期的结果。但一般来说，移动内容都会把读者的眼球从效果的初始焦点上面移开，导致分散读者的注意力。图10-5展示了例子页面chapter10/no-shift/no-shift.html在添加一条消息前后的界面效果。注意消息下面的内容是怎么被推向页面下方的。



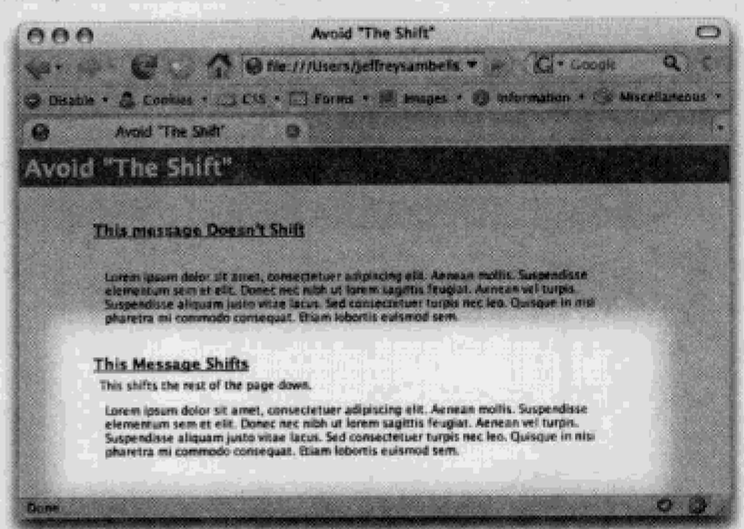
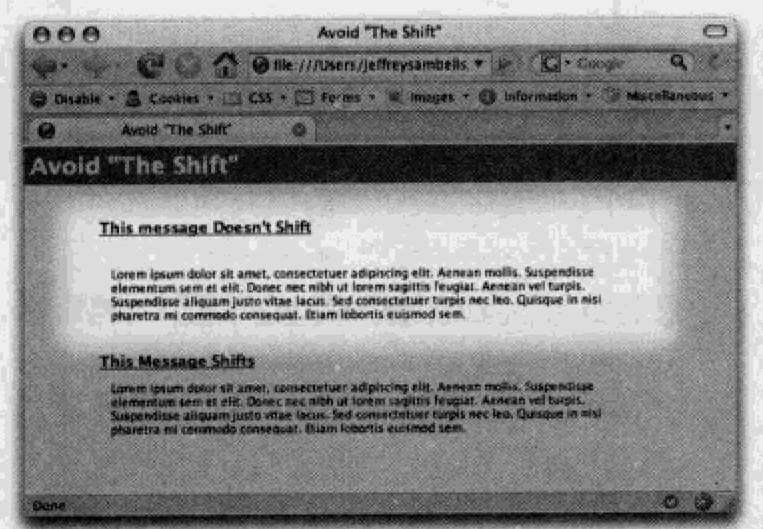




图10-5　观察页面中插入消息框前后内容的移动情况

要避免这种“移动”，可以在设计界面时预留消息框的空间，即使用`visibility:hidden`而不是`display:none`。`visibility`属性会在使元素不可见的情况下，维护元素的相对大小及位置。也就是说，当创建消息时，需要将其添加到适当的元素中并使其可见。只要新添加的内容不会改变预留空间的大小，那么新的消息就不会影响别的内容，如图10-6所示。



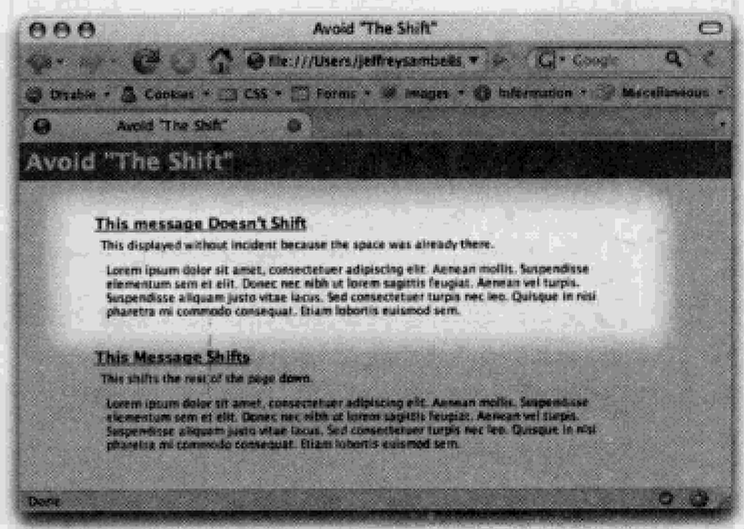




图10-6　使用`visibility:visible`[1]插入消息框不会移动内容

当开始在文档中标记元素时，相应的元素可能是空的。对于前面的例子而言，消息区域一开始未包含消息。为了避免使用固定高度，这里简单地以非破坏性的空格（` `）来预先填充消息区域，以便该区域随着文本内容的增多而扩大。如果消息包含的内容在一行显示不下，那么页面仍然会发生移动现象，因为我们假设的是新消息只占用一行的空间。

## 10.2　几个视觉效果库简介

上一章中，我们讨论的重点是生产力和更高效的DOM脚本编程。在本章中，我们要集中讨论在视觉美学及交互作用这个主题。为此，我们再介绍如下两个库：

① 原文display:visible有误。——译者注

www.TopSage.com

- Moo.fx
- Script.aculo.us

这两个库都是在Prototype的基础上构建起来的，不过Moo.fx也提供了一个作为Mootools JavaScript库组件（http://mootools.net）的版本。本章，我们会使用Moo.fx库的Mootools版，因为这个版本中包含Prototype版中没有的一些效果。

> Moo.fx库依靠`$()`和`$$()`方法来取得元素。在本章的例子源文件及代码中，`$()`和`$$()`引用的是Mootools库提供的方法。从本章的主题出发，可以认为这两个方法的操作与第9章中介绍的Prototype库的相应方法是相同的。如果你打算高度依赖Mootools版的Moo.fx库，那么就需要将Mootools库视为第9章所介绍的那些库的一个替代品，并且进一步研究这个库的文档（http://docs.mootools.net/）。

1. Moo.fx

Moof.fx（http://moofx.mad4milk.net/）自许为“一个基于`prototype.js`或`mootools`框架的，超轻量级的、极小但却包罗万象的JavaScript效果库”。总体上看，Moo.fx非常易用并且采取了低层次方法，使开发者能够识别元素并指定在某个时间段内修改哪个CSS属性。相应的修改只应用于特定的元素，而不会影响其子元素（除非子元素通过层叠机制继承相应的CSS属性）。使用这个库所提供的低层次特性，几乎能够以最小的努力创建出任何效果来。

- **命名空间**，Fx（取决于哪个版本）
- **许可**，MIT类型的许可（http://moofx.mad4milk.net/License.txt）
- **依赖关系**，作为Mootools的一个内置组件（http://mootools.net）或基于Prototype（http://prototypejs.org）的独立版本。

2. Script.aculo.us

Script.aculo.us（http://script.aculo.us）“是构建跨浏览器的用户界面的、易用的JavaScript库，它能使你的Web站点和Web应用程序飞扬起来”。Script.aculo.us采取了高层次的手段并提供五种核心效果以及它们的组合。 通过这些高层次的效果，所有给定元素的子元素也可能会受到影响。比如，当在一个段落上调用`Effect.Scale`效果时，段落中的字体大小也会随着段落中的范围元素及任何子元素的物理宽度及高度相应缩放。这些高层次的组合使得应用大型、复杂的效果变得简单，因为要做的它都帮你做完了。

- **命名空间**，因使用的效果对象不同而异
- **许可**，MIT类型的许可（http://wiki.script.aculo.us/Script.aculo.us/show/License）
- **依赖关系**，基于Prototype（http://prototypejs.org）库

## 10.3 视觉盛宴

视觉效果，在使用适当的情况下，可以为Web应用程序增色不少，而且效果也能为有些枯燥的Web体验添加几分趣味性。为了更好地理解每个库，我们先来看上述库的几个关键方法。

www.TopSage.com

### 10.3.1　Moo式的CSS属性修改

前面提到过，Moo.fx允许你修改元素的CSS属性。而这个修改过程主要是通过Fx命名空间中如下两个对象来完成的。

- Fx.Style(element, property, options)，可以用来修改任何数字形式的CSS属性，包括十六进制颜色值。
- Fx.Styles(element, options)，可以在一个效果中为同一个元素的多个属性应用不同的效果。

要创建一个新效果，可以通过创建适当的Fx方法的一个新实例来完成：

```
var myFx = new Fx.Style($('element-id'),'opacity');
```

不过，此时此刻，创建的效果还只是一个对象，并没有实际地影响页面。为了将效果应用到页面中，还需要调用另外的方法，比如set()或start()，稍后我们会介绍到。

而通过一些选项属性还可以进一步修改每个Fx样式方法，这些选项如下。

- duration，以毫秒定义的效果的时间长度。
- fps，定义动画需要达到的每秒帧数——默认值为30。
- transition，是用于改变动画中线性时间间隔的数学计算选项。
- unit，定义与数值相关的单位。默认值为px，不过在某些情况下可能是em或%。
- wait，用于表示一次变换是否需要等待元素上的其他变换结束之后再开始。默认值为true。

> 有关transition选项，我们将在稍后讨论如何使效果更真实时进行详细介绍。

除了前面的这些属性之外，还可以通过一个函数来指定如下3个回调选项。

- onStart：会在效果开始时调用。
- onComplete：会在效果结束时调用。
- onCancel：会在效果被取消或者中断时调用。

当需要在效果完成之后或者开始之前为文档添加额外的变化时，这些回调方法会非常有用。Fx.Style()方法第3个参数中的每个选项，都可以通过下面熟悉的对象语法来指定：

```
var myFx = new Fx.Style($('element-id'),'opacity' {
    duration: 1000,
    onComplete: function() {
        alert('Hello!');
    }
});
```

为了确保不透明度（opacity）效果在IE中正常应用，还必须为相应的元素设置布局（layout）。在为元素设置布局时，必须使用下列设置之一。

- 设置width或height属性（如果页面所用的DOCTYPE不标准，则只对行内元素有效）。
- 设置display属性为inline-block。

www.TopSage.com

- 设置position属性为absolute。
- 设置writingMode属性为tb-rl。
- 设置contentEditable属性为true。

> 要了解在IE中布局及该浏览器种种怪异行为的内容，请参阅http://www.satzansatz.de/cssd/onhavinglayout.html。

### 1. 一次修改一个属性

当完成了效果的实例化之后，接下来还需要调用它。

在使用Fx.Style方法时，如果你愿意的话，可以通过set()方法立即设置效果的CSS属性值，例如：

```
// 将不透明度设置50%
var myFx = new Fx.Style($('element-id'),'opacity').set(0.5);
```

但是，在多数情况下，可能都需要让属性值在两个值之间实现变换，此时就要用到start()方法，该方法的两个参数分别是初始值和期望的最终值，例如：

```
// 将不透明度从0（完全透明）淡入为1（完全不透明）
var myFx = new Fx.Style($('element-id'),'opacity' {
    duration: 1000
}).start(0,1);
```

start()方法中的初始值是可选的。也就是说，如果只为该方法指定了一个值，那么属性的当前值将会作为初始值，例如：

```
// 将不透明度从属性的当前值（或者完全不透明，或者完全透明）淡入为50%
var myFx = new Fx.Style($('element-id'),'opacity',{
    duration: 1000
}).start(0.5);
```

### 2. 一次修改多个属性

在通过Fx.Styles()对象同时改变多个属性时，虽然也要使用相同的start()方法，但该对象的工作机制会略有不同。Fx.Styles()方法只接受元素和选项两个参数，比如[①]：

```
var myFx = new Fx.Styles($('element-id'), { duration: 1000 });
```

而具体的属性和相应的起/止值则直接在start()方法中指定，即使用对象表示法和键/值对来分别表示属性及相应的值：

```
// 同时修改opacity、border-width
// 和font-size
var myFx = new Fx.Styles($('element-id'),{ duration: 1000 }).start({
    'opacity':[0,1],
    'border-width':[4],
    'font-size':[24]
});
```

① 原文代码中缺少英文逗号（,）。——译者注

www.TopSage.com

在这个例子中，opacity将会从透明（0）淡入为完全不透明（1），同时，border-width也会从当前值变为4，而font-size则会从当前值变为24。

### 3. 重用效果

可喜的是，Moo.fx对象的效果可以多次重复使用。也就是说，在实例化Fx样式对象后，可以通过不同的值多次调用set()和start()方法。比如，要为一个元素创建脉动效果，只需如下简单地以多个值依次重用一个opacity效果即可：

```
// 创建一个opacity Fx对象
var myFxOpacity = new Fx.Style(
    $('element-id'),
    'opacity',
    {wait, true}
);
// 从0淡入为0.9
myFxOpacity.start(0, 0.9);
// 淡入回0.5
myFxOpacity.start(0.5);
// 淡入回0.9
myFxOpacity.start(0.9);
// 淡入回0
myFxOpacity.start(0);
```

在前面的例子中，将wait选项设置为true是很关键的，因为只有这样每个效果才会依次发生。在默认情况下，wait的值为true，但为了说明这个选项的重要性，此处也包含了该选项。

### 4. 在多个对象上实现多个效果

Moo.fx还包含第3个方法——Fx.Elements(elements,options)，这个方法可以用来修改传递进来的元素数组。此时，唯一的不同之处在于start()方法使用了另一种语法来指定每个元素，如下所示：

```
// 为列表中的元素应用一种效果
var myFx = new Fx.Elements($$('li'),{
    duration: 1000
}).start({
    // 将第1个元素淡入为不透明
    '0': { 'opacity':[0,1] },
    // 淡入并从右侧移入第2个元素
    '1': {
            'opacity':[0,1],
            'left':[-500,300]
        }
    // 将第3个元素的背景从当前颜色淡入为绿色
    '2': { 'background-color':['#00FF00'] }
});
```

其中，start()方法参数对象中的每个属性，都与数组中元素的索引相关，而且属性的值都使用了与Fx.Styles(...).start(...)方法中同样的对象表示法来定义。

### 5. 通过Moo.fx实现滑动效果

Moo.fx中唯一专门的内置效果是Fx.Slide(element[, options])，该对象用于实现图

10-7中的滑动效果。

> 这个方法只在作为Mootools组件的Moo.fx包中有效，而在其Prototype版中是无效的。

使用这个内置对象可以在任何块级元素上实现滑动效果，但是，相应的元素不能带有任何定位属性，例如绝对定位或外边距等。如果必须要对该元素进行定位，那么可以通过定位其父元素来达到目的。

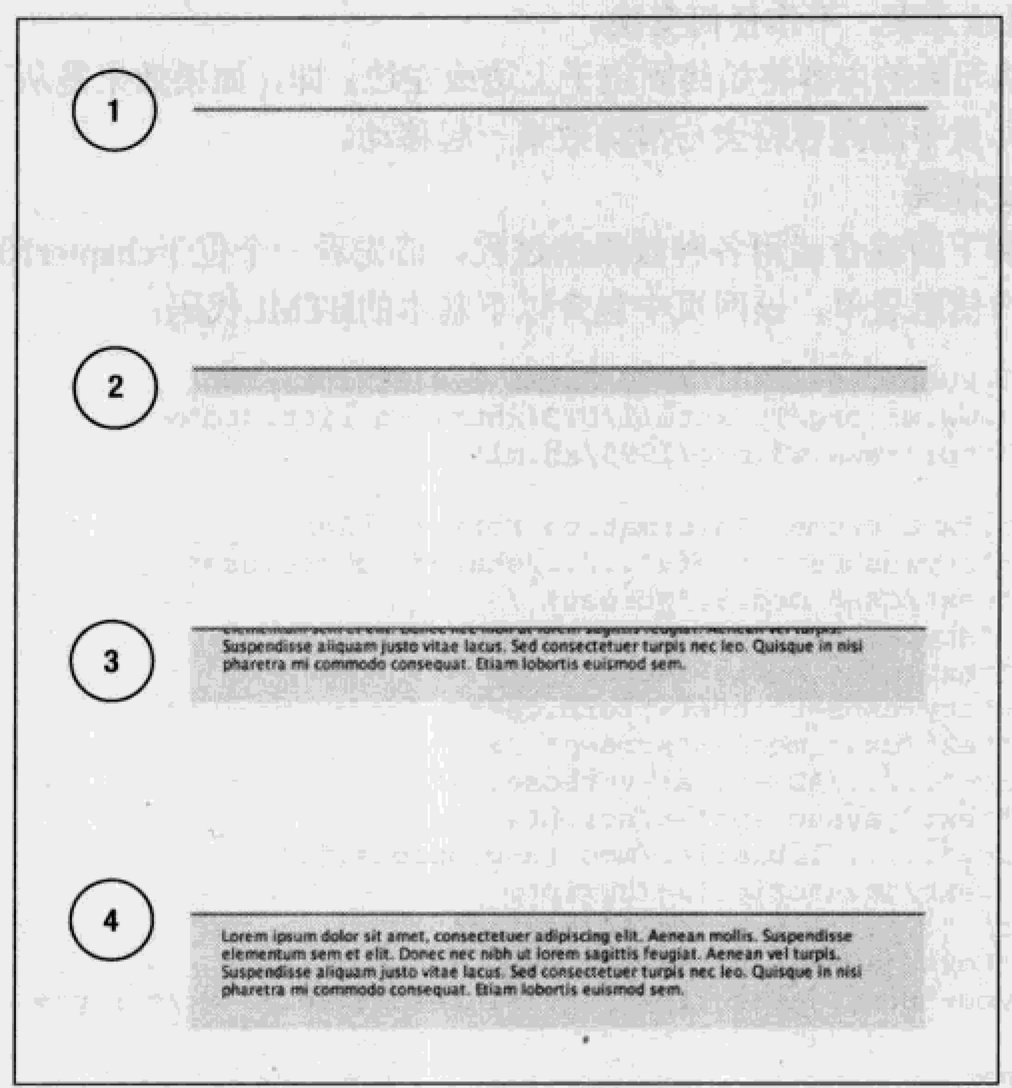



图10-7 Moo.fx实现的滑动效果

在创建新的滑动效果时，仍然采用与Fx的样式方法相同的选项来实例化对象，除了需要使用mode选项：

```
var myFxSlider = new Fx.Slide($('slide-me'), {
    duration: 500,
    mode:'horizantal'
});
```

Fx.Slider需要一个额外的选项——mode，该选项定义了滑动的方向：

- vertical，沿垂直方向滑动（默认值）。
- horizontal，沿水平方向滑动，如这里所示。

与Fx的样式方法类似，你需要根据想要采取的操作，使用下列5种方法中的一种方法来调用

www.TopSage.com

滑动效果的实例：

- slideIn()，将元素滑入，使其可见。
- slideOut()，将元素滑出，使其消失。
- toggle()，在slideIn()和slideOut()之间切换，无论执行哪个方法都将是当前状态的相反效果。
- hide()，隐藏元素，不作任何变换。
- show()，显示元素，不作任何变换。

遗憾的是，滑动元素的内容将始终固定于上边或左边。即，如果效果是从下往上滑动，或是从右侧滑出，那么元素中的内容将会与滑动效果一起移动。

**6. 让表单反馈更漂亮**

为了更好地理解下面综合运用各种效果的过程，请先看一个位于chapter10/form/form.html源文件中的简单的客户信息表单，该网页中包含以下基本的HTML代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Moo.fx Customer Information Form</title>
    <link rel="stylesheet" href="../../shared/source.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="../chapter.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="form.css"
        type="text/css" media="screen" />
    <script src="../../ADS-final-verbose.js"
        type="text/javascript"></script>
    <script src="../../libraries/moo.fx-mootools.js"
        type="text/javascript"></script>
    <script src="../../libraries/jquery.js"
        type="text/javascript"></script>
    <script type="text/javascript" src="interactive.js"></script>
</head>
<body>
<h1>Customer Information</h1>
<div id="content">
    <h2>Moo.fx Effects Example</h2>
    <div id="customer-info">
        <form id="customer-form" action="POST">
            <fieldset>
                <legend>
                    <span class="sign"> Personal Information</span>
                </legend>
                <div class="required">
                    <label for="name" accesskey="1">Name</label>
                    <input title="your name" name="name" id="name"
                        tabindex="1" type="text"/>
                    <span id="name-error" class="error"></span>
                </div>
                <div class="required">
                    <label for="street" accesskey="2">Street</label>
```

www.TopSage.com

```
                    <input title="your street" name="street"
                       id="street" tabindex="2" value="" type="text"/>
                    <span id="street-error" class="error"></span>
                </div>
                <div class="required">
                    <label for="city" accesskey="3">City</label>
                    <input title="your city" name="city" id="city"
                        tabindex="3" value="" type="text"/>
                    <span id="city-error" class="error"></span>
                </div>
                <div class="required">
                    <label for="region" accesskey="4">Province</label>
                    <input title="your state / province" name="region"
                       id="region" tabindex="4" value="" type="text"/>
                    <span id="region-error" class="error"></span>
                </div>
                <div class="required">
                    <label for="postal">Postal Code</label>
                    <input title="zip or postal code" name="postal"
                        id="postal" tabindex="5" value="" type="text"/>
                    <span id="postal-error" class="error"></span>
                </div>
                <div>
                    <label for="website" accesskey="6">Website</label>
                    <input title="your website" name="website"
                        id="website" tabindex="6" type="text"/>
                    <span id="website-error" class="error"></span>
                </div>
                <div class="required">
                    <label for="email" accesskey="7">Email
                        (for Verification)</label>
                    <input title="your email" name="email"
                        id="email" tabindex="7" type="text"/>
                    <span id="email-error" class="error"></span>
                </div>
                <div class="buttons">
                    <input type="submit" value="Save Information"
                        tabindex="7"/>
                </div>
            </fieldset>
        </form>
    </div>
</div>
</body>
</html>
```

这个页面的样式表中包含如下样式：

```
/*
其余的布局样式位于
source/shared/chapter.css
source/chapter10/chapter.css
*/

/* 为显示错误消息的元素指定高度以防止移动 */
span.error {
    display:block;
```

www.TopSage.com

```
    color:red;
    height:1em;
}

/* 将滑动元素定位到表单的底部 */
#slider-wrapper {
    margin: 0;
    padding: 0;
    position:absolute;
    left: 20px;
    display:none;
    width: 436px;
}
#slider-wrapper div {
    margin: 0;
    padding: 0;
}
#slider-wrapper span {
    padding: 1em;
    display: block;
}
#error-slider {
    margin: 0;
    background:red url(images/arrow.png) no-repeat left center;
    color:white;
}
```

在没有添加`load`事件或者说JavaScript的情况下，网页中的表单也是比较中规中矩的，而且可以正常地向服务器提交表单信息。然而，为了把事情做得更好一些，我们在`interactive.js`文件中包含了一个`load`事件（使用第9章介绍的jQuery库），以便在提交表单时进行一些简单的验证：

```
// 避免冲突
jQuery.noConflict();

// load事件
jQuery(document).ready( function() {

    // 为显示错误消息的元素添加一个非破坏性的空格
    // 以保证该元素具有适当的高度
    jQuery('.error').html(' ');

    var form = jQuery('#customer-form');

    // 为.required中的输入字段
    // 添加简单的错误检查
    var inputs = jQuery('.required input',form);

    // 当获得焦点时，清除错误消息
    inputs.focus(function() {
        jQuery('#' + jQuery(this).attr('id') + '-error').html('');
    });

    // 当失去焦点时，检测字段是否已填写
    inputs.blur(function() {
        var input = jQuery(this);
        if(!input.val()) {
```

www.TopSage.com

```
            jQuery('#' + input.attr('id') + '-error').html(
                'Please fill in '
                + input.attr('title')
                + '.');
        }
    });

    // 添加submit事件，同样也进行错误检查
    form.submit(function() {

        var error = false;

        inputs.each(function(e) {
            var input = jQuery(this);
            if(!input.val()) {
                error = true;
                jQuery('#' + input.attr('id') + '-error').html(
                'Please fill in '
                + input.attr('title')
                + '.');
            }
        });

        if(error) {
            // 返回false，以在jQuery中避免默认动作
            return false;
        }
        return true;

    });

});
```

在通过jQuery添加了load事件之后，表单能够在字段下方显示消息来指出错误，如图10-8所示。

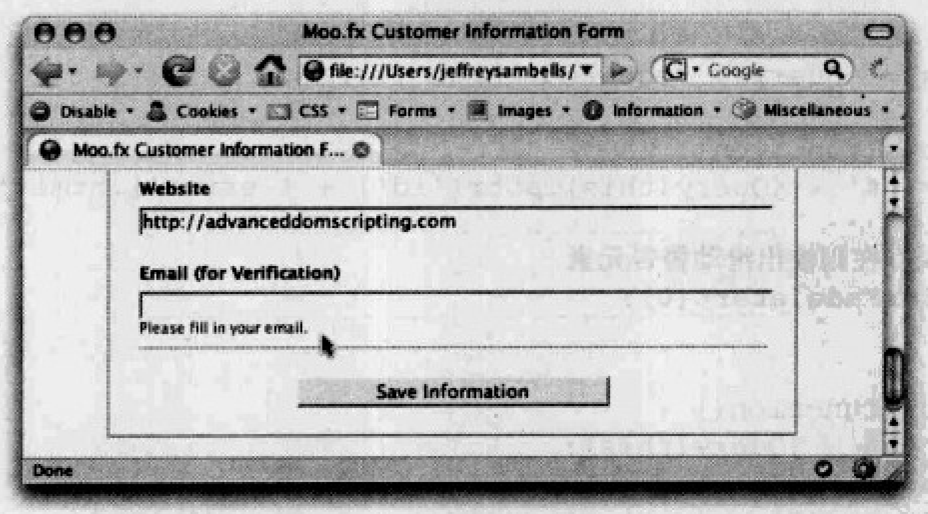



图10-8 在简单的客户信息表单中提示字段未完成

不过，这个表单仍然有待改进。假如用户只注意表单的底部，那么很可能会看不到出现在表单顶部的错误消息。虽然也可以通过添加一个alert()方法来提醒用户，但是如果使用Moo.fx的方法为错误消息添加效果，则会使表单更富有生气而且也更有魅力。比如，可以使用Fx.Slide()效果滑出一个指示框，通过覆盖住Save Information按钮来表示有错误存在。而盖住继续按钮片刻即可让用户意识到表单中存在错误。下面是为了添加上述效果而在原文件中修改后的结果：

www.TopSage.com

```
jQuery.noConflict();
jQuery(document).ready( function() {

    jQuery('.error').html(' ');

    var form = jQuery('#customer-form');

    // 添加滑动的错误元素，以便在发生错误时
    // 覆盖住提交按钮
    jQuery('#customer-form .buttons').append(
    '<div id="slider-wrapper"><div id="error-slider">
        Oops! It seems you forgot something.</div></div>');

    // 实例化滑动效果
    var mySlider = new Fx.Slide('error-slider', {
        duration: 600,
        wait:true,
    });

    // 开始时隐藏滑动效果
    mySlider.hide();

    // 在滑动元素上实例化一个淡入效果
    var mySliderFade = new Fx.Style('error-slider', 'opacity', {
        duration: 500,
        wait:true,
        onComplete: function() {
            // 当淡入完成时，隐藏滑动
            // 并将opacity重置回100%
            mySlider.hide();
            mySliderFade.set(1);
        }
    });

    var inputs = jQuery('.required input',form);

    inputs.focus(function() {
        jQuery('#' + jQuery(this).attr('id') + '-error').html('');

        // 如果存在则淡出滑动警告元素
        mySliderFade.start(0);
    });

    inputs.blur(function() {
        var input = jQuery(this);
        if(!input.val()) {
            jQuery('#' + input.attr('id') + '-error').html(
                'Please fill in '
                + input.attr('title')
                + '.'
            );
        }
    });

    form.submit(function() {
```

www.TopSage.com

```
        var error = false;

        inputs.each(function(e) {
            var input = jQuery(this);
            if(!input.val()) {
                error = true;
                jQuery('#' + input.attr('id') + '-error').html(
                    'Please fill in '
                    + input.attr('title')
                    + '.'
                );
            }
        });

        if(error) {
            // 如果存在错误，则滑出错误元素，使其可见
            jQuery('#slider-wrapper').css('display','block');
            mySlider.slideIn();
            setTimeout(function() {
                // 在3.5秒后自动淡出
                mySliderFade.start(0);
            },3500);

            return false;
        }

        return true;
    });

});
```

现在，当有错误发生时，错误提示会更加显而易见，如图10-9所示。而且，滑动的提示框在视觉上也远比alert()更令人赏心悦目。

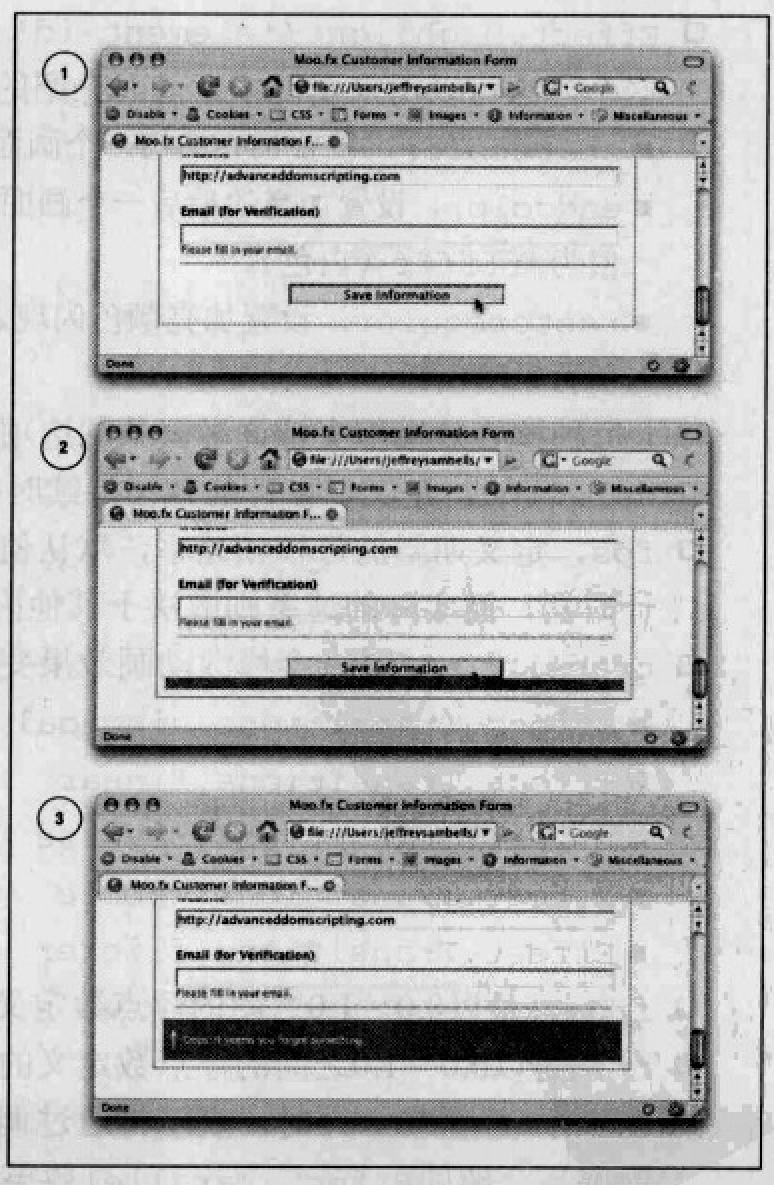


图10-9 用户信息表单例子中改进后的错误提示效果

### 10.3.2 通过 Script.aculo.us 实现视觉效果

我们要介绍的第二个视觉效果库是Script.aculo.us。前面曾经提到过了，Script.aculo.us采取的是高层次的手段，并且提供了如下4种核心效果。

- Effect.Opacity('element-id' [, options])，改变元素的不透明度。
- Effect.Scale('element-id', percent [, options])，改变元素的宽度和高度，也是以em为单位进行计算的基础，而且会影响到给定元素中的子

www.TopSage.com

元素。这个效果拥有下列自定义选项。

- scaleX，设置效果为水平缩放。默认值为true。
- scaleY，设置效果为垂直缩放。默认值为true。
- scaleContent，设置缩放元素的内容，而不仅仅是元素自身。默认值为true。
- scaleFromCenter，如果值为true，则以元素在屏幕上的中心点为准进行缩放。默认值为false。
- scaleMode，如果值为“box”，则缩放元素的可见区域；如果值为“content”，则缩放整个元素。也可以像下面样通过对象表示法将originalHeight和originalWidth变量指定给scaleMode，来控制元素按比例进行缩放：

```
scaleMode: { originalHeight: 100, originalWidth: 150 }
```

  scaleMode的默认值为box。
- scaleFrom，设置缩放的起始百分比，默认值为100.0。

- Effect.MoveBy('element-id', y, x [, options])，通过指定以像素表示的X/Y值对来移动元素。
- Effect.Highlight('element-id' [, options])，闪现某种作为元素背景的颜色。这个方法可以用来实现本章前面介绍的YFT（黄褪）效果。这种效果也拥有自定义的选项：
  - startcolor，设置加亮的第一个画面的颜色。默认值就是方便的#ffff99（亮黄色）。
  - endcolor，设置加亮的最后一个画面的颜色。通常将其设置为加亮元素的背景颜色。默认值为#ffffff（白色）。
  - restorecolor，设置加亮颜色闪现之后相应元素的背景颜色。默认值为被加亮元素的当前背景颜色。

以上每种核心效果都支持许多公共的选项，主要有如下几个。

- duration，定义以秒计的效果持续时间，而且要使用浮点数值。默认值为1.0。
- fps，定义期望的每秒帧速率，默认值为25，但不能高于100。这个选项用于调整内部的计时器，而实际的速率则取决于其他因素，如计算机或浏览器的执行速度。
- transition，是一个修改动画效果变换方式的函数。可以指定的内置变换函数包括：
  - Effect.Transitions.sinoidal（默认值）
  - Effect.Transitions.linear
  - Effect.Transitions.reverse
  - Effect.Transitions.wobble
  - Effect.Transitions.flicker
- from，是以0.0～1.0之间的浮点数定义的变换效果的起点。默认值为0.0。
- to，是以0.0～1.0之间的浮点数定义的变换效果的终点。默认值为1.0。
- sync，当值为true时，表示将通过调用render()方法来手动渲染效果。这种效果将用于稍后介绍的Effect.Parallel效果中。

www.TopSage.com

- queue，决定了队列中的位置。可以使用front或end来定义此选项，从而使当前效果在全局效果中进行排队。也可以使用{position:'front/end',scope:'scope',limit:1}形式的对象表示法来表示不同的队列。要了解有关队列对象的更多信息，请参阅http://script.aculo.us中的相关文档。
- direction，设置膨胀或收缩效果变换的方向。可能的值有top-left、top-right、bottom-left、bottom-right或center（center是默认值）。

与Moo.fx类似，在执行效果的过程中，也可以指定几个供调用的回调方法：

- beforeStart，在效果开始时调用。
- beforeUpdate，在效果中每一“帧”被绘制之前调用。
- afterUpdate，在效果中每一“帧”被绘制之后调用。
- afterFinish，当效果结束时调用。

为调用Script.aculo.us库中的核心效果，需要对Effect对象进行实例化，例如下面所示的Effect.Opacity效果：

```
// 将#element-id从透明淡入为不透明
new Effect.Opacity($('element-id'),{
    duration: 1.0,
    from: 0,
    to: 1
});
```

但与Moo.fx不同的是，以上效果会在实例化过程中立即被调用，不必再为应用效果而调用其他方法。

**并行效果**

当在Script.aculo.us库中实例化效果对象时，相应的效果会立即被调用。如果想同时组合应用多个效果，那么就需要用到第五个核心方法Effect.Parallel(effects[, options])。与其他的效果不同，Effect.Parallel本身就是一种效果，它不接受元素作为参数，只接受将要组合的子效果数组：

```
// 移动一个元素并将其不透明度从0%淡入为100%
new Effect.Parallel([
    new Effect.MoveBy(
        $('element-id'),
        400,
        400,
        { sync: true }
    ),
    new Effect.Opacity(
        $('element-id'),
        { sync: true, to: 0.0, from: 1.0 }
    )],
    {
        duration: 0.5
    }
);
```

www.TopSage.com

最终得到的效果将会把MoveBy和Opacity效果组合成一个效果。其中的sync属性用于防止个别的效果分别渲染自己的帧——所有的帧都将由Parallel()方法负责渲染。

为方便起见，Script.aculo.us也提供了很多内置的组合效果，这些组合效果都基于对一或多个核心效果的调用来实现。比如，Effect.Appear就是遵循相同思想的一个组合效果：

```
new Effect.Appear('element-id', { duration: 3.0 });
```

在此，我们不会详细讨论组合的效果，如果想了解下面这些组合效果的具体细节，请参阅Script.aculo.us库的在线文档http://wiki.script.aculo.us/scriptaculous/：

- Effect.Appear('element-id')
- Effect.Fade('element-id')
- Effect.Puff('element-id')
- Effect.BlindDown('element-id')
- Effect.BlindUp('element-id')
- Effect.SwitchOff('element-id')
- Effect.SlideDown('element-id')
- Effect.SlideUp('element-id')
- Effect.DropOut('element-id')
- Effect.Shake('element-id')
- Effect.Pulsate('element-id')
- Effect.Squish('element-id')
- Effect.Fold('element-id')
- Effect.Grow('element-id')
- Effect.Shrink('element-id')
- Effect.Highlight('element-id')
- Effect.toggle('element-id','name of effect')

### 10.3.3　通过Moo.fx实现逼真的运动效果

你有没有见过两个看上去基本相同的效果，但由于某种原因，其中一个却比另一个在感观上更逼真？如果有，那么这种感觉可能就来自于效果之间细微的运动差别。

在现实中，我们不会机械地行事，坐立行走也都非常自然。比如，当你挥手时，手在运动过程中不会以一个不变的速度从A点移动到B点。同样，当你移动手臂时，可能会经历以下几个环节：

（1）以静止的状态从一端启动。

（2）向另一端加速挥动。

（3）达到匀速运动状态。

（4）当接近终点时减速。

（5）以静止的状态在另一端停止。

www.TopSage.com

除此之外，在开始和停止挥动手臂的那一刻，也可能并非完全处于静止状态。可能会稍微摆得过了头而再回摆一些，或者会突然停止接着再次加速。而正是所有这些并不一致的运动才使得它看起来真实。假如你将自己手臂的运动过程拍成录像，并每隔数毫秒捕获一个画面，那么整个挥动的过程看起来大致会如图10-10所示。

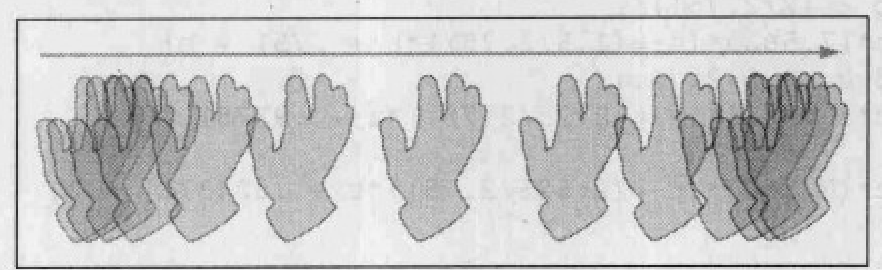


图10-10 在从A点向B点移动的一秒钟内，非线性运动的示意图

使用setTimeout()方法的Web动画，类似我们在本章前面所完成的那些运动效果——与此相比当然迥然不同。如果通过setInterval()方法像本章开始时的某个效果一样，将一个元素从A点移动到B点，那么结果就是间隔一致的线性运动效果，如图10-11所示。

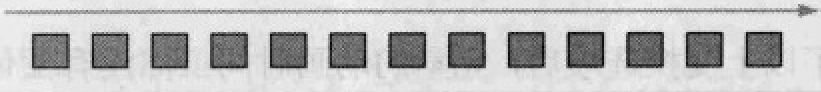


图10-11 在从A点向B点移动的一秒钟内，线性运动的示意图

通过比较图10-10和图10-11可以看出明显的差别。虽然两者都是在相同的时间内从A移动到B，而且最终的目标也相同，但在起始和结束的过程中，元素的位置却差别很大。为了实现更接近真实的效果，需要让效果更像图10-10，或者更准确地说，更不像图10-11。

令人惊喜的是，Moo.fx库中包含了大量可以实现逼真效果的Fx.Transitions对象。这些变换都是通过数学计算相对于图10-10所示的直线常规区间，来修改元素实际位置的。例如，前面所说的手臂运动的过程，就可以在Moo.fx的Fx.Transitions.sineInOut方法中，通过使用正弦数学公式来计算：

```
……省略的代码……
/* 属性：sineInOut */
sineInOut: function(t, b, c, d){
    return -c/2 * (Math.cos(Math.PI*t/d) -1) + b;
}
……省略的代码……
```

如果在应用了这个函数的情况下，将元素相对于时间的位置绘制成图形，那么就会得到图10-12所示的曲线图。

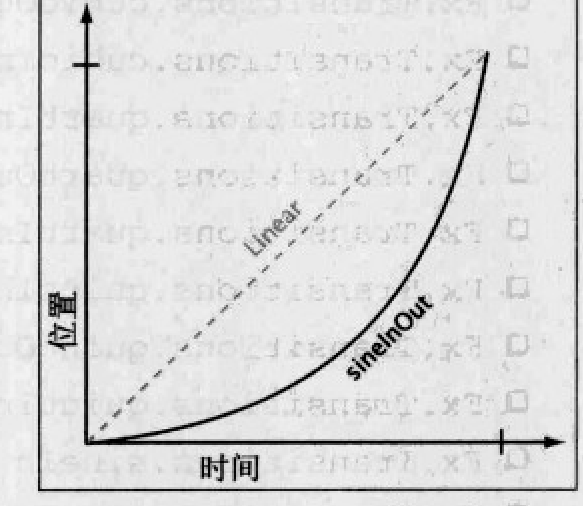



图10-12 在使用Fx.Transitions.sineInOut方法时，表示位置与时间之间函数关系的曲线图

虽然我们无意在这里深入讲解数学知识，但问题的关键是要理解元素位置随时间推移的不同变化，即不再是以等量增长的方式移动了。而且，不同的运动类型，也需要进行不同的计算。如果想让元素在运动结束时产生反弹效果，那么可以使用类似Fx.Transitions.bounceOut中

www.TopSage.com

的公式：

```
/* 属性：bounceOut */
bounceOut: function(t, b, c, d){
    if ((t/=d) < (1/2.75)){
        return c*(7.5625*t*t) + b;
    } else if (t < (2/2.75)){
        return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
    } else if (t < (2.5/2.75)){
        return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
    } else {
        return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
    }
}
```

如果你一看到这些数学公式就感到头晕，别担心。实际上，你不必记住其中任何公式。因为把一个Fx.Transitions公式应用到Moo.fx库中的某个效果上，就像下面这样简单：

```
var myFx = new Fx.Style($('element-id'),'left' {
    duration: 1000,
    transition : Fx.Transitions.sineInOut
}).start(10,400);
```

如此而已。在添加了以上变换选项后，元素的动画时间虽然没有变化，但元素的位置却会遵循更改后的非线性运动方式而变化。

Moo.fx库的Fx.Transitions对象包含下列变换方法，每个方法都建立在不同的数学计算基础之上：

- Fx.Transitions.linear
- Fx.Transitions.quadIn
- Fx.Transitions.quadOut
- Fx.Transitions.quadInOut
- Fx.Transitions.cubicIn
- Fx.Transitions.cubicOut
- Fx.Transitions.cubicInOut
- Fx.Transitions.quartIn
- Fx.Transitions.quartOut
- Fx.Transitions.quartInOut
- Fx.Transitions.quintIn
- Fx.Transitions.quintOut
- Fx.Transitions.quintInOut
- Fx.Transitions.sineIn
- Fx.Transitions.sineOut
- Fx.Transitions.sineInOut
- Fx.Transitions.expoIn

www.TopSage.com

- Fx.Transitions.expoOut
- Fx.Transitions.expoInOut
- Fx.Transitions.circIn
- Fx.Transitions.circOut
- Fx.Transitions.circInOut
- Fx.Transitions.elasticIn
- Fx.Transitions.elasticOut
- Fx.Transitions.elasticInOut
- Fx.Transitions.backIn
- Fx.Transitions.backOut
- Fx.Transitions.backInOut
- Fx.Transitions.bounceIn
- Fx.Transitions.bounceOut
- Fx.Transitions.bounceInOut

说起变换，别忘了这些方法并不仅限于创建移动元素的效果。事实上，任何变化（比如颜色和不透明度），似乎都能因非线性运动而受益匪浅：

```
var myFx = new Fx.Style($('element-id'),'opacity' {
    duration: 1000,
    transition : Fx.Transitions.sineOut
}).start(0,1);
```

> Moo.fx库中变换计算的主要部分都建立在Robert Penner的工作基础之上。如果你对这些模拟现实的公式感兴趣，或者你只想拓宽一下视野，可以参阅一下他本人在http://www.robertpenner.com/easing/上面所作的、关于渐变和缓动效果的说明。

**进一步美化用户表单**

仍以本章前面提到的用户表单为例，我们可以通过为其增加变换效果对其进一步美化：

```
// 实例化滑动效果
var mySlider = new Fx.Slide('error-slider', {
    duration: 600,
    wait:true,
    transition:Fx.Transitions.elasticOut
});

// 开始时隐藏滑动
mySlider.hide();

// 在滑动元素上实例化淡入效果
var mySliderFade = new Fx.Style('error-slider', 'opacity', {
    duration: 500,
    wait:true,
    transition:Fx.Transitions.sineIn,
```

www.TopSage.com

```
        onComplete: function() {
            // 当淡入完成时，隐藏滑动
            // 将opacity值重设为100%
            mySlider.hide();
            mySliderFade.set(1);
        }
    });
```

结果虽然相同，但却可以通过不同的变换效果影响用户的体验。一个反弹或者具有弹性的滑动效果，会给人更有趣也更具娱乐性的感觉，而简单的线性变换则会给人更机械的感觉。

可以分别试试各种不同的效果，看这些效果对界面的交互性都会产生什么影响。即使在显示相同信息的情况下，你可能也会惊异于某些效果所产生的奇妙感觉。

## 10.3.4 圆角效果

我们要介绍的最后一个效果是圆角——Web 2.0的一个精华元素。在广受认同的CSS 3规范获得充分的支持之后，我们可以使用内置的border-radiurs属性来实现圆角效果：

```
.rounded {
    border-radius: 1.6em;
}
```

或者，也可以采取设置多幅背景图像的方式：

```
.fancyRounded {
    background-image: url(top-left.gif), url(top-right.gif),url(bottom-left.gif),
url(bottom-right.gif);
    background-repeat: no-repeat, no-repeat, no-repeat, no-repeat;
    background-position: top left, top right, bottom left,bottom right;
}
```

如此轻易地实现圆角效果在今天看起来似乎仍然是一种神话。假如你不希望在标记中掺入不必要的元素，而且也不介意JavaScript无效时圆角变回方角，那么就可以利用DOM脚本来添加能将方角修圆的标记。

第9章介绍的MochiKit库中也提供了修圆方角的解决方案。这个方案就是MochiKit库的MochiKit.Visual.roundElement(element [, options])方法，该方法接受给定的元素及下列选项。

- corners，指定要修圆哪个角。可以是all或者以空格分隔的表示多个角的字符串。即tl、tr、bl、br分别表示左上角、右上角、左下角、右下角。默认值为all。
- color，指定用于填充元素的颜色。默认值为计算后的当前填充的颜色。
- bgColor，指定用于填充背景的颜色。默认值为父元素的颜色。
- blend，布尔值，当该值为true时，颜色呈混合效果。默认值为true。
- border，布尔值，指定是否包含边框。默认值为false。
- compact，布尔值，指定是否使用紧凑（较小）的圆角。默认值为false。

具体到为某个元素实现圆角效果，只需调用roundElement()方法即可：

www.TopSage.com

```
MochiKit.Visual.roundElement('element-id',{
    corners: 'bl tr'
});
```

这样会得到如图10-13所示的修圆了右上角和左下角的盒子。

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Curabitur cursus, nunc in tempus bibendum, arcu ligula semper ligula, volutpat mattis odio nunc sed lectus. Vestibulum fermentum tortor nec felis rhoncus congue. Vestibulum tempor, augue sed congue mollis, risus nisl dictum massa, ac scelerisque felis nunc vel dui. In est. Vivamus nec massa. Vestibulum pellentesque.

图10-13 对右上角和左下角应用了修圆效果的元素

为了实现圆角效果，roundElement()方法会对相应的元素进行适当地格式化。以下面简单的div元素为例：

```
<div id="round-me">
    <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
</div>
```

相应圆角效果将会由一些脚本添加的1×1像素的元素构成：

```
<div id="round-me">
<div>
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 2px 0px 0px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 1px;
    font-size: 1px; margin-right: 3px; margin-left: 0px;">
    </span>
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 1px 0px 0px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 1px;
    font-size: 1px; margin-right: 2px; margin-left: 0px;">
    </span>
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 1px 0px 0px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 1px;
    font-size: 1px; margin-right: 1px; margin-left: 0px;">
    </span>
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 1px 0px 0px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 2px;
    font-size: 1px; margin-right: 0px; margin-left: 0px;">
    </span>
</div>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</p>
<div style="background-color: rgb(255, 255, 255);">
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 0px 0px 1px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 2px;
    font-size: 1px; margin-left: 0px; margin-right: 0px;">
    </span>
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 0px 0px 1px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 1px;
    font-size: 1px; margin-left: 1px; margin-right: 0px;">
    </span>
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 0px 0px 1px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 1px;
    font-size: 1px; margin-left: 2px; margin-right: 0px;">
    </span>
```

www.TopSage.com

```
    <span style="border-style:solid; border-color:rgb(230, 230, 230);
    border-width: 0px 0px 0px 2px; overflow: hidden;
    background-color: rgb(204, 204, 204); display: block; height: 1px;
    font-size: 1px; margin-left: 3px; margin-right: 0px;">
    </span>
</div>
</div>
```

### 10.3.5 其他库

到现在为止，我们只介绍了Moo.fx和Script.aculo.us库中的效果。但像MochiKit、jQuery和YUI等这些库中，同样也包含许多效果和生成动画的方法，我同样也建议你探索一下这些效果。在本章剩余的内容中，我们会关注一些对用户界面整体的增强，例如拖放效果等。这些整体增强的效果，需要组合本章前面讨论过的许多效果。而且，这次我们会从与用户交互的角度来应用效果，而不仅仅是为了让界面看起来更花哨。

## 10.4 行为增强

当视觉效果与用户交互相结合时，就会获得一些良好的行为增强功能，比如拖放。

### 通过 Script.aculo.us 实现拖放

在第6章中，我们已经就使用适当的鼠标事件这个主题，涉及到了拖动的概念。拖放对象也是许多库的主打功能。比如MochiKit和Mootools都有其内置的拖放功能，而且实现原理也类似。但在这里，我们只讨论如何通过Script.aculo.us来实现拖放。拖放功能的一般特征就是可以将某个界面元素拖动到任何地方，拖放的范围或方向会受限制，而且可以将元素放在其他目标元素之上。

#### 1. 随处拖动

通过Script.aculo.us来创建可拖放的对象是一件极其简单的事，而且通常只需使用一行代码，例如：

```
// 让某个元素可拖动
var draggable = new Draggable('element-id');
```

在新的`Draggable(element[, options])`对象实例化完成后，相应的元素马上就能够被拖动到页面上的任何位置。要限制运动或影响元素可以设置以下选项。

- `handle`，定义调用拖动功能的对象。默认值为完整的基本元素，不过也可以使用这个选项来指定一个子元素，例如窗口上的标题栏。如果通过对象引用或者类选择符指定了拖动手柄，那么拖动作为子元素的这个手柄就可以在整个元素上调用拖动功能。

```
Draggable('element-id',{ handle: 'title-bar' });
```

- `revert`，可以是布尔值，如果为true，则将元素恢复到其原始位置。

```
Draggable('element-id',{ revert: true });
```

www.TopSage.com

或者，也可以为该选项指定一个回调函数，这个回调函数会在元素被释放时调用：

```
Draggable('element-id',{ revert: function() {
  alert('Drag over');
}});
```

该选项的默认值为false。

- snap，修改元素的位置并将元素吸附到某个网格上，该选项可以通过如下3种方式来指定。
  - 如果使用一个值来定义该选项，那么这个值则会用来对每个坐标进行取模操作。例如，如果snap的值为15，那么元素会被吸附到x和y坐标都能被15整除的点(x, y)：

    ```
    Draggable('element-id',{ snap: 15 });
    ```

  - 如果使用一个数组来定义该选项，那么数组中的值则会用来对每个坐标(x, y)进行取模操作。

    ```
    Draggable('element-id',{ snap: [10,30] });
    ```

  - 如果使用一个回调函数来定义该选项，那么这个函数会接收元素当前位置的x和y坐标，并且应该返回一个包含两个元素的数组，数组的两个元素表示元素实际上应该吸附到的位置。

    ```
    Draggable('element-id',{ snap: function(x,y) {
      // 代码
      return [newX,newY];
    }});
    ```

- zindex，是可拖放项的CSS z-index属性值。默认值为1000。
- constraint，可以是horizontal或vertical，用来约束元素只能沿水平或垂直方向拖动。
- ghosting，创建并拖动一个元素的半透明副本。这个副本会在鼠标释放时消失。
- starteffect，指定拖动开始时为元素应用的效果。默认值为Effect.Opacity效果。
- reverteffect，指定当revert选项值为true，而且元素恢复到原始位置时为元素应用的效果。默认值为Effect.Move效果。
- endeffect，指定拖放结束时为元素应用的效果。默认值为Effect.Opacity效果。

此外，还可以通过change选项指定一个回调函数，在元素被拖放过程中，元素的重新定位会触发对这个回调函数的调用。这个选项有点类似于mousemove事件。

通过指定这些选项，可以实现对被拖动元素的各种控制，如约束其可达区域、将其吸附到特定的点，或者仅仅拖动一个副本等。这些选项的具体用法，会在后面介绍一个简单的拖放购物车时再进行讨论。

### 2. 可以放置的目标对象：可放置对象

在屏幕上随处拖动元素还只是完整过程的一半。多数情况下，拖动某个元素都会带有一定的目的性，至少是想将元素拖到某个目标对象上。此时我们就将放置元素的目标对象称为是可放置的。当在一个可放置对象上面释放一个可拖动对象时，可放置对象就会以某种指定的方式与其进行交互。

在使用Script.aculo.us库时，无需实例化可放置对象。但是，需要使用Droppables（复数形

www.TopSage.com

式）对象的add()方法来创建一个新的可放置对象：

```
Droppables.add('element-id', {
    hoverclass:'drop-it'
});
```

Script.aculo.us库的可放置对象中也可以包含如下一些选项，通过这些选项可以为可放置对象添加更多的属性。

- accept，一个表示被允许的可拖动对象CSS类选择符的字符串。任何放在这个可放置对象上的可拖动对象都必须拥有该类。如果想指定多个类，需要使用一个包含类选择符的数组。
- containment，将可放置对象限定为指定元素的子元素。该选项可以是表示单个元素的字符串，也可以是表示多个元素的字符串数组。
- hoverclass，当一个被允许的可拖动对象位于可放置对象之上时，为可放置对象应用的类名。设置这个选项的目的是为了表明在这个元素上可以释放鼠标。
- overlap，如果将这个选项设置为horizontal或vertical，那么可放置对象与可拖动对象只有在沿指定方向重合50%以上时才会进行交互。这个选项主要用于对象排序。
- greedy，当该选项为true时，可以防止定位在这个可放置对象之下的可放置对象与可拖动对象进行交互。默认值为true。

此外，可放置对象中也包括下列回调函数选项。

- onHover，当被允许的可拖动对象移动到可放置对象上方时调用，该回调函数会取得如下3个参数。
  - 可拖动的元素
  - 可放置的元素
  - 在overlap选项所指定方向上重合的百分比
- onDrop，当被允许的可拖动对象在可放置对象的上方释放时调用，该回调函数会取得如下3个参数。
  - 可拖动的元素
  - 可放置的元素
  - 事件

当与可拖动元素进行交互时，通过onDrop回调函数还可以继续对页面进行操作，这一点在下面购物车的例子中将会涉及。

### 3. 通过Script.aculo.us构建可拖放的购物车

对于大多数的商品销售网站来说，购物车都是最基本的构件。下面我们来看一看位于**chapter10/cart/cart.html**中，作为简单的购物车例子起点的HTML代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Scriptaculous Shopping Cart Example</title>
```

www.TopSage.com

```
        <link rel="stylesheet" href="../../shared/source.css" type="text/css"
media="screen" />
        <link rel="stylesheet" href="../chapter.css" type="text/css" media="screen" />
        <link rel="stylesheet" href="cart.css" type="text/css" media="screen" />

        <!-- Include the prototype and script.aculo.us libraries -->
        <script src="../../libraries/prototype.js"
            type="text/javascript"></script>

        <script src="../../libraries/scriptaculous/src/scriptaculous.js"
            type="text/javascript"></script>
        <script type="text/javascript" src="interactive.js"></script>
    </head>
    <body>
        <h1>A Simple Shopping Cart</h1>
        <div id="content">
            <h2>Buy Some Books!</h2>
            <div id="product-wrapper">
                <ul id="products">
                    <li class="product-item" id="pid-1">
                        <img src="books/1590593812.jpg" alt="Web Standards Solutions: The
Markup and Style Handbook"/>
                        <a href="server.json?id=pid-1"
                            title="Add to cart">Add</a>
                    </li>
                    <li class="product-item" id="pid-2">
                        <img src="books/1590595335.jpg" alt="DOM Scripting: Web Design
with JavaScript and the Document Object Model"/>
                        <a href="server.json?id=pid-1" title="Add to cart">Add</a>
                    </li>
                    <li class="product-item" id="pid-3">
                        <img src="books/1590596145.jpg" alt="CSS Mastery: Advanced Web
Standards Solutions"/>
                        <a href="server.json?id=pid-1" title="Add to cart">Add</a>
                    </li>
                    <li class="product-item" id="pid-4">
                        <img src="books/1590596382.jpg" alt="Web Accessibility: Web
Standards and Regulatory Compliance"/>
                        <a href="server.json?id=pid-1" title="Add to cart">Add</a>
                    </li>
                    <li class="product-item" id="pid-5">
                        <img src="books/1590597656.jpg" alt="HTML Mastery: Semantics,
Standards, and Styling"/>
                        <a href="server.json?id=pid-1"
                            title="Add to cart">Add</a>
                    </li>
                    <li class="product-item" id="pid-5">
                        <img src="books/1590598032.jpg" alt="Web Standards Creativity:
Innovations in Web Design with XHTML, CSS, and DOM Scripting"/>
                        <a href="server.json?id=pid-1"
                            title="Add to cart">Add</a>
                    </li>
                    <li class="product-item" id="pid-5">
                        <img src="books/1590594304.jpg" alt="Web Designer's Reference"/>
                        <a href="server.json?id=pid-1"
                            title="Add to cart">Add</a>
                    </li>
                </ul>
                <div class="clear"></div>
```

www.TopSage.com

```
        </div>
        <h3>In Your Cart...</h3>
        <p id="message"> </p>
        <div id="cart-wrapper">
            <ul id="cart">
                <li class="cart-item">
                    <img src="books/1590598032.jpg" alt="Web Standards Creativity:
Innovations in Web Design with XHTML, CSS, and DOM Scripting"/>
                    <a href="server.json?id=pid-5"
                        title="Remove from cart">Remove</a>
                </li>
                <li class="cart-item">
                    <img src="books/1590596145.jpg" alt="CSS Mastery: Advanced Web
Standards Solutions"/>
                    <a href="server.json?id=pid-3"
                        title="Remove from cart">Remove</a>
                </li>
            </ul>
            <div class="clear"></div>
        </div>
    </div>
</body>
</html>
```

与前面介绍的客户信息表单的例子类似，上面购物车例子的HTML页面也是在不考虑JavaScript的前提下构建的，其中每件商品图片都有自己的Add按钮，同时还有一些CSS样式：

```
/* 针对购物车及商品的样式 */
#products, #cart {
    list-style: none;
    clear: both;
    margin: 0;
    padding: 0;
}
#products li, #cart li {
    display: block;
    height: 40px;
    width: 33px;
    float: left;
}
#products li img, #cart li img {
    height: 40px;
    width: 33px;
}
#products li a, #cart li a {

}
#product-wrapper {
    background: #efefef;
    padding: 15px;
    margin: 1em;
    width: 430px;
}
#cart-wrapper {
    background: #fff9ea;
    padding: 15px;
    margin: 1em;
    width: 430px;
}
```

www.TopSage.com

```
#cart-wrapper.drop-it {
    background: #eaffea;
}
#message {
    font-size:1em;
    height: 1em;
}
.product-item *, .cart-item * {
    display: block;
    clear: none;
}
```

此时购物车的页面外观类似图10-14所示。

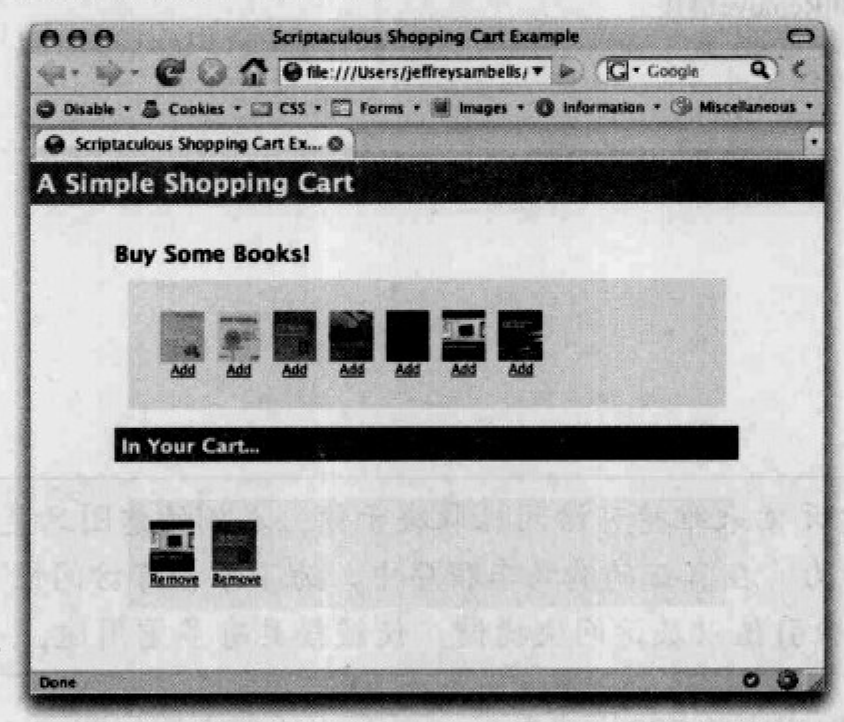



图10-14 一个典型的购物车

同样，为了保证可访问性，我们要以一个能够正常运行的、不依赖JavaScript的Web应用程序为起点（假设已经编写完成了相应的服务器端代码），通过添加一些拖放性的交互功能来使其更有个性。

开始，我们先打开chapter10/cart/interactive.js文件，并看一看其中的注释：

```
// 使用Prototype库的load事件
Event.observe(window, 'load', function(event) {

    // 隐藏所有Add和Remove链接

    // 使一个特殊队列中的全部商品都可以拖动
    // 以便后面为其应用更多的效果

    // 使一个特殊队列中的全部现有的购物车中的项都可以拖动
    // 以便后面可以通过应用更多的效果来移除这些项

    // 创建一个表示某项是否
    // 应该被移除的布尔值

    // 使购物车可放置
```

www.TopSage.com

```
    // 创建观察程序，用于移除
    // 拖动到购物车之外的项

});
```

因为我们的起点是一个不具有拖放界面功能的能够正常运行的Web应用程序，所以标记中带有一些附加的元素，例如Add及Remove链接。所以，首先需要将这些附加的元素隐藏起来：

```
// 使用Prototype库的load事件
Event.observe(window, 'load', function(event) {

……省略的代码……

    // 隐藏所有Add和Remove链接
    $$('li.product-item a','li.cart-item a').each(function(e) {
        e.setStyle({
            position:'absolute',
            top:0,
            left:'-10000px'
        });
    });

……省略的代码……

});
```

> 至于通过添加什么元素来维持可访问性取决于你。本例的意图只是为了示范Script.aculos.us库提供拖放效果的能力。在真正的购物车程序中，为了维护可访问性，还必须为每个Add和Remove链接添加tab键索引值以及访问快捷键，使链接具有多重用途。

在配合以上改变添加相应的CSS样式之后，页面的外观应该如图10-15所示。

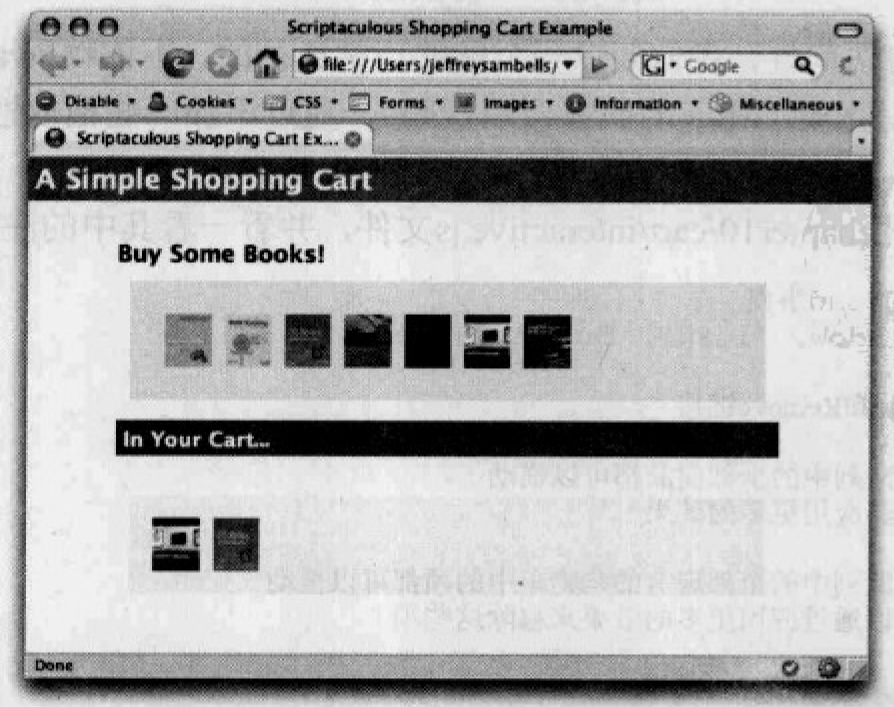



图10-15　修改后的购物车页面外观

www.TopSage.com

接着，要做的就是将所有商品及购物车中已有的项都转换为可拖动的对象，并设置一个值为false的变量keepMe：

```
// 使用Prototype库的load事件
Event.observe(window, 'load', function(event) {

……省略的代码……

    // 使一个特殊队列中的全部商品都可以拖动
    // 以便后面为其应用更多的效果
    $$('li.product-item').each(function(e) {
        new Draggable(e,{
            revert:true,
            queue:'cart_draggables'
        });
    });

    // 使一个特殊队列中的全部现有的购物车中的项都可以拖动
    // 以便后面可以通过应用更多的效果来移除这些项
    $$('li.cart-item').each(function(e) {
        new Draggable(e,{
            revert:true,
            queue:'cart_draggables'
        });
    });

    // 创建一个表示某项是否
    // 应该被移除的布尔值
    var keepMe = false;

……省略的代码……

});
```

位于load事件作用域中的keepMe变量，将在后面用于跟踪应该从购物车中移除哪个项。在默认情况下，只有当某项在购物车中被释放时才会被移除。

然后，将#cart-wrapper创建为可放置对象，而且只允许带有.product-item和.cart-item类的元素可拖动：

```
// 使用Prototype库的load事件
Event.observe(window, 'load', function(event) {

……省略的代码……

    // 使购物车可放置
    Droppables.add($('cart'),{

        // 允许商品及购物车中的项可拖动
        accept:['product-item','cart-item'],

        // 改变购物车的类以表示它是目标对象
        hoverclass:'drop-it',

        // 按需要为购物车添加项
        onDrop:function(draggable, droppable, event) {

            // 在购物车上放下的项不会被删除
```

www.TopSage.com

```
                // 这样可以避免在购物车中放下项
                // 时而移除该项
                keepMe = true;

                // 只添加.product-item类
                if(draggable.className == 'product-item') {

                    $('message').innerHTML = 'Contacting server...';

                    // 将增加的项保存到服务器
                    new Ajax.Request(
                        draggable.getElementsByTagName('A')[0].getAttribute('href'),
                        {
                            method:'get',
                            onSuccess: function(response) {
                                $('message').innerHTML = response.responseText;

                                // 为构建列表而创建新的DOM节点
                                var newItem = document.createElement('LI');
                                newItem.className = 'cart-item';
                                var newThumb = document.createElement('IMG');
                                var oldImage = draggable.getElementsByTagName('IMG')[0];
                                newThumb.src = oldImage.src;
                                newThumb.alt = oldImage.alt;
                                newItem.appendChild(newThumb);
                                var newAnchor = document.createElement('A');
                                newAnchor.setAttribute(
                                    'href',
                                    'server.json?id=' + draggable.id
                                );
                                newAnchor.setAttribute(
                                    'title',
                                    'Remove from cart'
                                );
                                newAnchor.style.display = 'none';
                                newItem.appendChild(newAnchor);

                                // 使新项也可以拖动
                                new Draggable(newItem,{
                                    revert:true,
                                    queue:'cart_draggables'
                                });

                                // 为新项添加淡入效果
                                new Effect.Opacity(newItem,{from:0,to:1});

                                $('cart').appendChild(newItem);
                            },
                            onFailure: function() {
                                $('message').innerHTML = 'Could not add';
                            }
                        }
                    );
                }
            }
        });

    ……省略的代码……
    });
```

www.TopSage.com

在可放置对象的选项中，onDrop回调函数负责完成购物车的大部分处理事务。首先，向购物车中添加的项，还会使用与Add链接中相同的URL以Ajax方式传送回服务器。其次，如果添加成功——这里实际上都会成功，因为没有真正的服务器端程序，那么会将带移除锚链接的商品图像的副本添加到购物车中。以上两步完成了向购物车中添加商品的过程，不过别忘了用户也可能会从购物车中移除项目。

> 本例没有加入任何键盘或非鼠标设备导航的功能。但在真正的应用中，还应该添加各种与按键相关的事件，或者为购物车中的新增项添加更多的属性，以便获得更好的可访问性。

要从购物车中移除项，只需将另外一个对象设置为可放置的"垃圾桶"，并允许购物车中的项可以拖动到它上面即可。因此，这个过程与向购物车中添加项的过程是相同的，例外之处就是onDrop事件需要负责从服务器和购物车中移除项。此外，我们这里允许用户只要将项拖出购物车之外，就可以将该项移除。这需要像前面一样，使购物车中的新增项可以拖动：

```
// 使新项也可以拖动
new Draggable(newItem,{
    revert:true,
    queue:'cart_draggables'
});
```

而且，还需要访问可拖动对象自身的属性及方法，以便创建观察程序来与所有可拖动的元素相互协调。

#### 4. 通过观察程序与可拖动元素交互

多数情况下，移动可拖动对象并将它们放到可放置对象上面可能就足够了。但是，Script.aculo.us库还提供了通过Droppables对象来访问可拖动及可放置对象的全局列表的能力。这里我们只详细介绍所有相关方法中的一个——观察程序（observer），即通过Droppables对象可以添加另一个元素，作为拖动过程的观察程序。

所谓观察程序，就是一个带有element属性及一或多个下列回调方法的对象。

- onStart，当开始在页面上移动任何可拖动对象时调用。
- onDrag，在拖动期间每次鼠标移动时调用。
- onEnd，当拖动完成时调用。

以上每个回调方法都可以通过两个参数来调用：eventName及与当前拖动操作关联的可拖动元素的实例。

具体到购物车这个例子而言，由于被拖出购物车的对象没有可放置的目标对象，因此需要使用观察程序来监视这些实例，并在适当时候将它们移除：

```
// 使用Prototype库的load事件
Event.observe(window, 'load', function(event) {

……省略的代码……

    // 创建观察程序，用于移除拖动到购物车之外的项
```

www.TopSage.com

```
Draggables.addObserver({
element: null,
onEnd:function(eventName, draggable) {

        // 取得可拖动对象的当前位置，以便
        // 服务器请求失败时将其移回原位置
        var delta = draggable.currentDelta();

        if(!keepMe && draggable.element.className == 'cart-item') {
            // 这个项应该移除

            // 基于原始的起点位置
            // 计算偏移位置
            var topoffset = delta[1]-draggable.delta[1];
            var leftoffset = delta[0]-draggable.delta[0];

            // 防止服务器端
            // 还原可拖放对象
            draggable.options.revert = false;

            new Ajax.Request(
                draggable.element.getElementsByTagName('A')[0].getAttribute('href'),
                {
                    method:'get',
                    onSuccess: function(response) {

                        // 服务器端的移除操作成功，因此
                        // 销毁可拖动对象并设置淡出效果
                        $('message').innerHTML = 'Remove: '
                            + response.responseText;
                        draggable.destroy();

                        // 向可拖动队列的最后添加褪化效果
                        new Effect.Fade(draggable.element, {
                            duration:0.2,
                            queue: {
                                scope:'cart_draggables',
                                position:'end'
                            },
                            afterFinish: function(){
                                // 一旦完成则移除可拖动元素
                                draggable.element.remove();
                            }
                        });

                    },
                    onFailure:function() {
                        // 移除失败，故将可拖动对象还原
                        // 到其在购物车中的原始位置
                        $('message').innerHTML = 'Could not be removed';

                        var dur = Math.sqrt(Math.abs(topoffset^2)
                            +Math.abs(leftoffset^2))*0.02;
                        new Effect.Move(draggable.element, {
                            x: -leftoffset,
                            y: -topoffset,
                            duration: dur,
```

```
                        queue: {
                            scope:'cart_draggables',
                            position:'end'
                        },
                        afterFinish: function() {
                            // 将当前增量重置回
                            // 新位置并启用还原
                            draggable.delta = draggable.currentDelta()
                            draggable.options.revert = true;
                        }
                    });
                }
            }
        );
    }

    // 将keepMe标记重置为false
    keepMe = false;
  }
 });
……省略的代码……

});
```

每次拖动操作的最后都会调用这个观察程序。在这个例子中，没有涉及类，观察程序只是检查被拖动的元素是否应该移除。

位于购物车中的已购项目都关联了一个不同的类名，以便同常规的商品项区别开来。观察程序首先要检查类名以确保被拖动的元素带有`.cart-item`类，同时还要检查`keepMe`是否为`false`。由于可放置对象`#cart`的`onDrop`回调方法会在有项放置其上时，将`keepMe`设置为`true`，所以`keepMe`变量为`false`的唯一可能就是被拖动对象位于购物车之外。如果这两个条件都满足，那么相应的元素就会从购物车中被移除；否则，则会被还原到初始位置。

> 这个购物车的例子与完整的产品化程序之间存在着非常大的差距，不过它却为我们提供了一个典型的拖放应用的实例。要了解Droppables对象及其他属性的更多信息，请参阅http://wiki.script.aculo.us/Script.aculo.us/show/Draggables文档网站。

**5. 更丰富的拖放功能**

拖放式界面为交互性Web应用程序提供了全新的可能性，但无论如何都要紧记可访问性。如果你希望非鼠标设备能够访问自己的网站，就要保证提供备用的方法，或者至少要做到在JavaScript完全无效的情况下，也能提供无需鼠标而仍然一致的用户体验。

为了再刺激一下你的胃口，我向你介绍一种新效果。在Script.aculo.us库中，另外一种便利的效果是可排序列表。比如，对于下面的列表：

```
<ul id="categories">
    <li>Markup</li>
    <li>Style</li>
    <li>Behavior</li>
```

www.TopSage.com

```
    <li>Server Scripts</li>
    <li>Browsers</li>
</ul>
```

只需一行代码：

```
Sortable.create('categories');
```

就可以使列表能够通过拖放方式进行排序。是不是很酷？而且，可排序对象也有许多选项，包括在列表间排序、使用嵌套的列表以及在多个列表中拖放等。要了解所有选项，请参阅http://wiki.script.aculo.us/scriptaculous/show/Sortable.create中的文档。此外，通过将重新排序列表项与Ajax请求结合起来，就能够提供一种比在文本框中填写数字来排序更具亲和力的替代方案。

## 10.5 小结

在本章中，我们介绍了通过JavaScript中的setTimeout()方法创建自己效果的基本知识，也介绍了能够为平凡的Web应用程序注入活力的两个效果库。其中，我们讨论了几种简单的效果，如Yellow Fade Technique（黄褪技术）。而且，你也知道了通过向Web应用程序中添加适当的效果，不仅能够改变交互方式，而且还会影响用户体验到的网站整体感及品味。

在实现效果时，要时时紧记可访问性。虽然华丽而有趣的网站能给人留下难忘的印象，但如果花哨的效果阻碍了用户访问必要的信息，那么问题也将不期而至。

在下一章，我们会介绍几个API，通过它们可以为你的Web应用程序添加更多的功能（比如地图和搜索），但同样不需要付出太多的努力。

# 第 11 章 丰富的Mashup!运用API添加地图、搜索及更多功能

所谓Mashup，从其字面意思上理解，就是将两个或多个应用程序混合为一个应用程序。最常见的Mashup的例子，就是使用Google Maps在地图上面标注某些信息。本章后面也会介绍这种应用。地图Mashup的应用包括“Top 10 beaches with WiFi access（10大可以通过WiFi上网的海滩）”（http://www.geekabout.com/2007-01-18-78/top-10-beaches-with-wifi-internet-access.html）、实时火车位置（http://www.mackers.com/projects/dartmaps/）以及全球突发事件地点指引（http://www.globalincidentmap.com/）等。

不论将Mashup用于什么目的，基本上都会涉及到将自己收集的数据，或者从某个公共数据源取得的数据，通过API混入到一种公共服务当中。而混合的结果最终将由你自己的网站显示或者利用，从而以最小的工作量提供某种强大的服务。

Web服务所提供的API使得Mashup应用成为可能。这些API是一组预定义的开放性的通信接口的集合，通过它们能够以编程方式与相应服务实现查询和交互。每种服务提供的API都包含一组带说明的方法和一定的使用条件，同第9章和第10章中针对DOM脚本编程的API一样。而且，不同Web服务提供的API还具有各自的风格，有的可以通过服务器端协议实现，而有的则可以在浏览器中通过JavaScript或其他插件（如Adobe Flash）直接调用。

最常见的API都使用某种服务器端语言（如PHP）取得并提交信息。例如，通过下面几行简单的PHP代码，可以创建最近发表在http://flick.com中的照片列表：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Recent Flickr Photos!</title>
</head>
<body>
<?php
    $url = 'http://www.flickr.com/services/feeds/photos_public.gne';
    foreach(simplexml_load_file($url)->entry as $item) {
        echo $item->content;
    }
?>
</html>
```

www.TopSage.com

同样，使用某些API也可以如同往标记中添加几个HTML元素那样简单。比如，要在自己的网站中包含一个视频文件，可以先将这个视频文件上传到http://youtube.com，然后再在网站中放入适当的链接：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>My YouTube Video!</title>
</head>
<body>
<object width="425" height="350">
    <param name="movie" value= ➥
"http://www.youtube.com/v/hFFH8DaOHQg"></param>
    <param name="wmode" value="transparent"></param>
    <embed src="http://www.youtube.com/v/your_video_id"
        type="application/x-shockwave-flash"
        wmode="transparent" width="425" height="350"></embed>
</object>
</body>
</html>
```

与针对DOM脚本编程的各种库的API不同，像这样的服务API都是由服务本身所提供的。也就是说，我们不能下载到Flickr API，更不可能在自己的机器中为YouTube网站供应视频文件。这样，服务自身可以随时进行更新或者功能升级，而我们则可以在不必升级本地软件的条件下，始终访问其最新版本。使用服务API的唯一要求，就是遵照API文档生成适当的请求，进而取得适当的响应。

Flickr和YouTube的这两个例子都是最最简单的情形。事实上，Flickr API不仅提供了取得最近照片的功能（本章后面我们会介绍到），而且嵌入一个来源于YouTube的Flash文件，同嵌入其他Flash文件也没有多大的不同。

此外，要与服务器端的API交互也可能会涉及实现下列通信协议。

- SOAP（Simple Object Access Protocol，简单对象访问协议）
- REST（Representational State Transfer，表述性状态转移）
- XML-RPC（XML remote procedure calls，XML远程过程调用）

这些协议都非常有用，不过讨论它们的细节超出了本书的范围。因此，对这些协议我们不会深入介绍。

在本章中，我们会学习通过API实现Mashup应用的基本知识。首先从一些强大的客户端JavaScript API入手，尝试与地图和搜索结果进行交互。在本章第二部分，将会讨论通过借助代理脚本的服务器端API整合DOM脚本的问题。

## 11.1　API密钥

本章中，为了实现与各种服务之间的交互，必须注册不同的API密钥。API密钥是服务用来辨别哪个用户在访问和请求信息的唯一标识符。每种服务处理各自密钥的方式都有所不同，或者说需要使用不同类型的验证方法，我们会在介绍相应服务时指出这一点。不过，问题的关键是，

www.TopSage.com

本章的每个例子都要使用你自己的密钥。如果你没有注册并获得相应的密钥，那么将无法运行本章中的许多例子。

> 在注册自己的API密钥及账户时，要注意阅读每种API的服务条款。很多情况下，为了保证访问API必须遵守API的要求和限定条件。

## 11.2　客户端 API：离不开 JavaScript

客户端JavaScript API虽然能提供诸多强大的功能，但也存在一个潜在的弱点——依赖JavaScript。本书各处也都在不断强调，高度依赖JavaScript的网站很可能会导致可访问性问题，为此必须根据情况提供适当的替代方案。在本节中，我们要介绍两个JavaScript API：Google Maps API和Google Ajax Search API，还会介绍组合这两种API的Mashup应用。本章所提供的例子，也将一如既往地以保证可访问性为基准，或者会提供替代方案，或者使用渐进增强技术。

### 11.2.1　地图中的 Mashup 应用

Mashup这个概念以及Ajax，都是随着Google发布其公共的Google Maps API而日益流行起来的。通过使用Google Maps API，可以将类似http://local.google.com的全功能地图，嵌入到个人网站中。而且，开发者可以在地图上任意添加标注和数据，不仅开启了全新的世界（一语双关），也使得各种基于地理位置的数据，能够在交互式的地图上以图形化的元素显示出来——完全通过JavaScript来操控。

要在个人网站中集成Google Map是非常简单的一件事。基本上只需要到http://www.google.com/ apis/maps/signup.html上面注册申请一个API 密钥。为了知道什么该做，什么不该做，一定要认真看一遍服务条款中的主要内容，其中可能会包含某些限制。然后，输入你网站的URL并单击Generate API Key按钮，如图11-1所示。

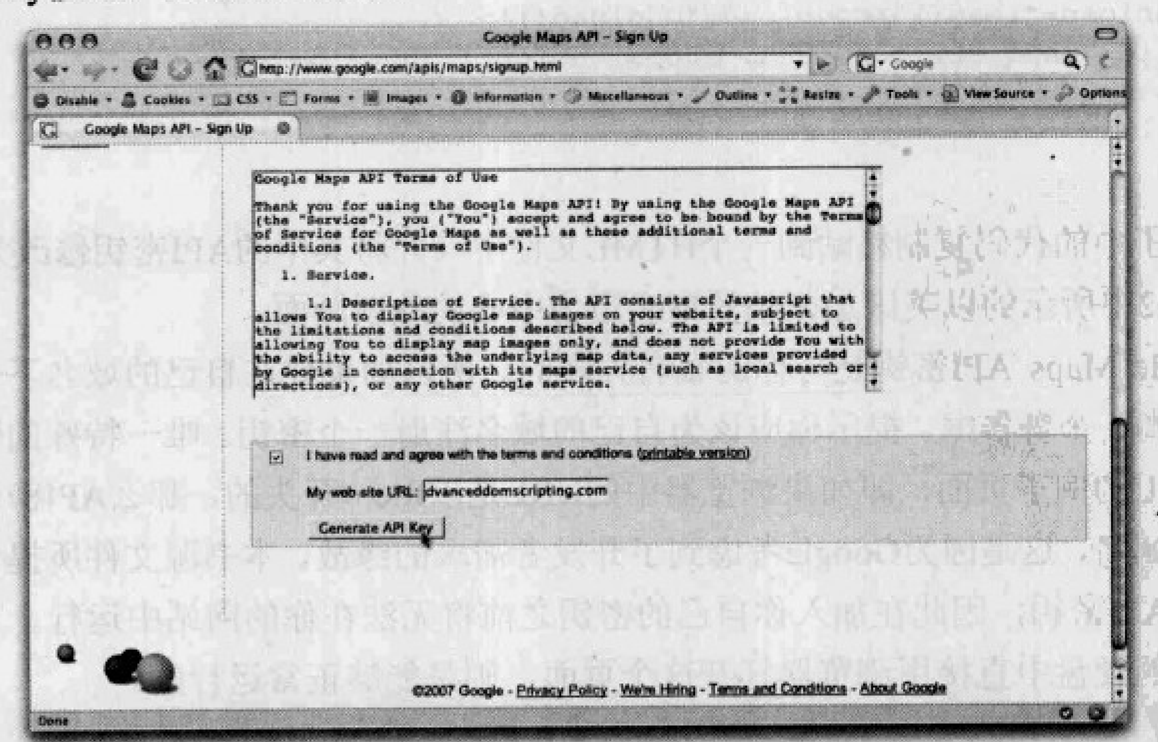



图11-1　生成Google Maps API密钥

www.TopSage.com

在结果页面上，会出现你申请的API密钥，这个密钥是由许多字符随机组成的，比如：

```
ABQIAAAAFOOBBfcVGVmeJR5FXDm_1BR2mdOh-K__WeD915i10Jagt9QYmBRA7MOsv8cdXJDNdSN1EozosI
ssBA
```

此外，你还会看到一个简单的“Hello World”式的例子：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src="http://maps.google.com/maps?file=api&amp;v=2 ➥
&amp;key=ABQIAAAAFOOBBfcVGVmeJR5FXDm_1BR2mdOh-K__WeD915i10Jag ➥
t9QYmBRA7MOsv8cdXJDNdSN1EozosIssBA"
      type="text/javascript"></script>
    <script type="text/javascript">

    //<![CDATA[

    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
      }
    }

    //]]>
    </script>
  </head>
  <body onload="load()" onunload="GUnload()">
    <div id="map" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

将这个例子中的代码复制粘贴到一个HTML文件中（并将其中的API密钥修改为你自己的），则会看到图11-2中所示的以美国加州帕罗奥多市为中心的地图画面。

由于Google Maps API密钥是与你的域名关联的，所以如果你在自己的域名下使用了错误的密钥，则会看到一个警告框，提示你应该为自己的域名注册一个密钥。唯一特殊的情况就是打开位于本地硬盘上的例子页面，即如果浏览器中的URL是以file://开头的，那么API密钥将不会受限于某个特定的域名，这是因为Google考虑到了开发者需求的缘故。本书源文件所提供的例子页面中没有适当的API密钥，因此在加入你自己的密钥之前将无法在你的网站中运行。不过，如果你在源文件所在的硬盘中直接用浏览器打开这个页面，则是能够正常运行的。

www.TopSage.com

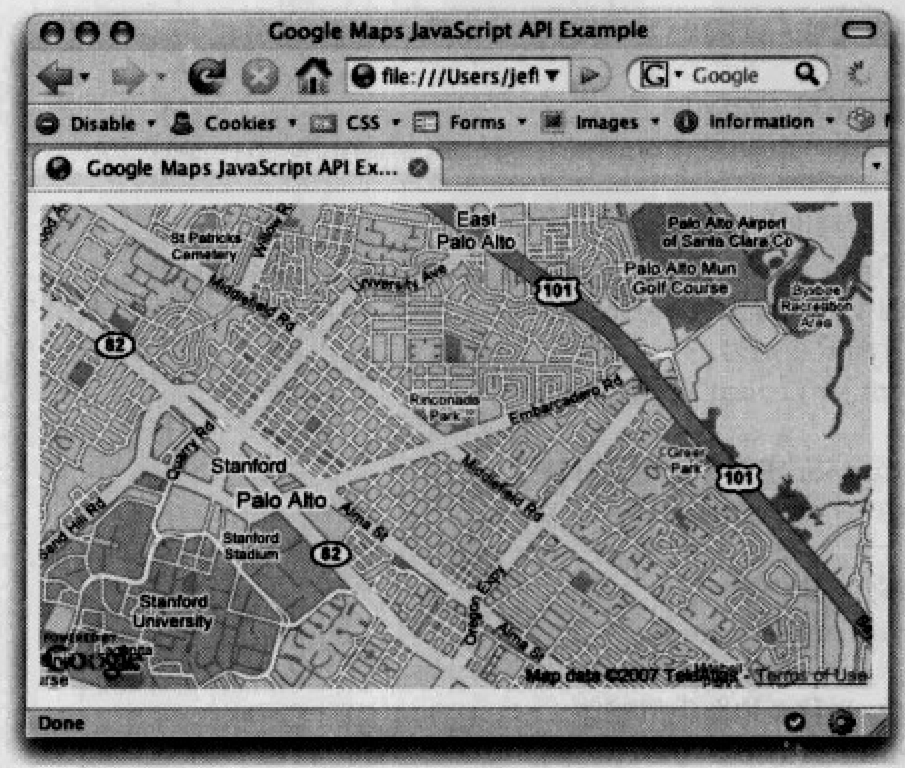



图11-2　以美国加州帕罗奥多市为中心的“Hello World”式的Google地图

Google提供的例子代码并没有将JavaScript与HTML标记分开（这与本书其他部分的例子形成了反差），我建议你花点时间把这两者分开。而且，在本书源文件chapter11/map/google.html中，也提供了一个分离的版本，其中还包含了几个CSS文件：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type"
        content="text/html; charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <link rel="stylesheet" href="../../shared/source.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="../chapter.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="map.css"
        type="text/css" media="screen" />
    <script src="../../ADS-final-verbose.js"
        type="text/javascript"></script>

    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp; ➡
key=YOUR_API_KEY"type="text/javascript"></script>
    <script src="map.js" type="text/javascript"></script>
  </head>
  <body>
    <div id="map"></div>
  </body>
</html>
```

在这个页面包含的脚本文件chapter11/map/map.js中，添加了下面的ADS `load`事件侦听器：

```
ADS.addEvent(window,'load',function() {
```

```
    if (GBrowserIsCompatible()) {

        // 实例化地图API（v2版）
        var map = new GMap2(document.getElementById("map"));

        // 实例化新的纬度和经度坐标
        var location = new GLatLng(37.4419, -122.1419);
        // 将地图的中心设置到location
        // 同时设置缩放级别为13
        map.setCenter(location, 13);

        // 使用location实例化一个新标注
        var marker = new GMarker(location);

        // 将标注添加到地图中
        map.addOverlay(marker);

        // 添加缩放/平移及地图类型控件
        map.addControl(new GLargeMapControl());
        map.addControl(new GMapTypeControl());
    }
});

ADS.addEvent(window,'unload',GUnload);
```

通过前面的例子或以上map.js文件，我们会注意到Maps API的代码被包含在了以GBrowserIsCompatible()方法为检测条件的if语句中。GBrowserIsCompatible()方法与第1章介绍的ADS.isCompatible()方法作用类似，是为了确保浏览器能够与Google Maps API的各个方面都能兼容。而且，只需短短3行代码就调用了这种高级地图的所有标记和交互功能。第一行代码在#map元素中创建了地图的实例：

```
// 实例化地图API（v2版）
var map = new GMap2(document.getElementById("map"));
```

接下来的两行代码基于地理上的纬度和经度数据设置了地图的中心点：

```
// 实例化新的纬度和经度坐标
var location = new GLatLng(37.4419, -122.1419);
// 将地图的中心设置到location
// 同时设置缩放级别为13
map.setCenter(location, 13);
```

其中的GLatLng对象表示的是地球表面上纬度和经度的交点，用于定位要放在地图中的所有对象。地球表面上某一点的纬度，指的是以赤道为界向北（+）或向南（–）的度数，取值范围从+90度（北级）到–90度（南级）。而经度则是指本初子午线向东（+）或向西（–）的度数，取值范围从+180度到–180度，恰好覆盖了地球圆周的360度。纬度和经度如图11-3所示。

www.TopSage.com

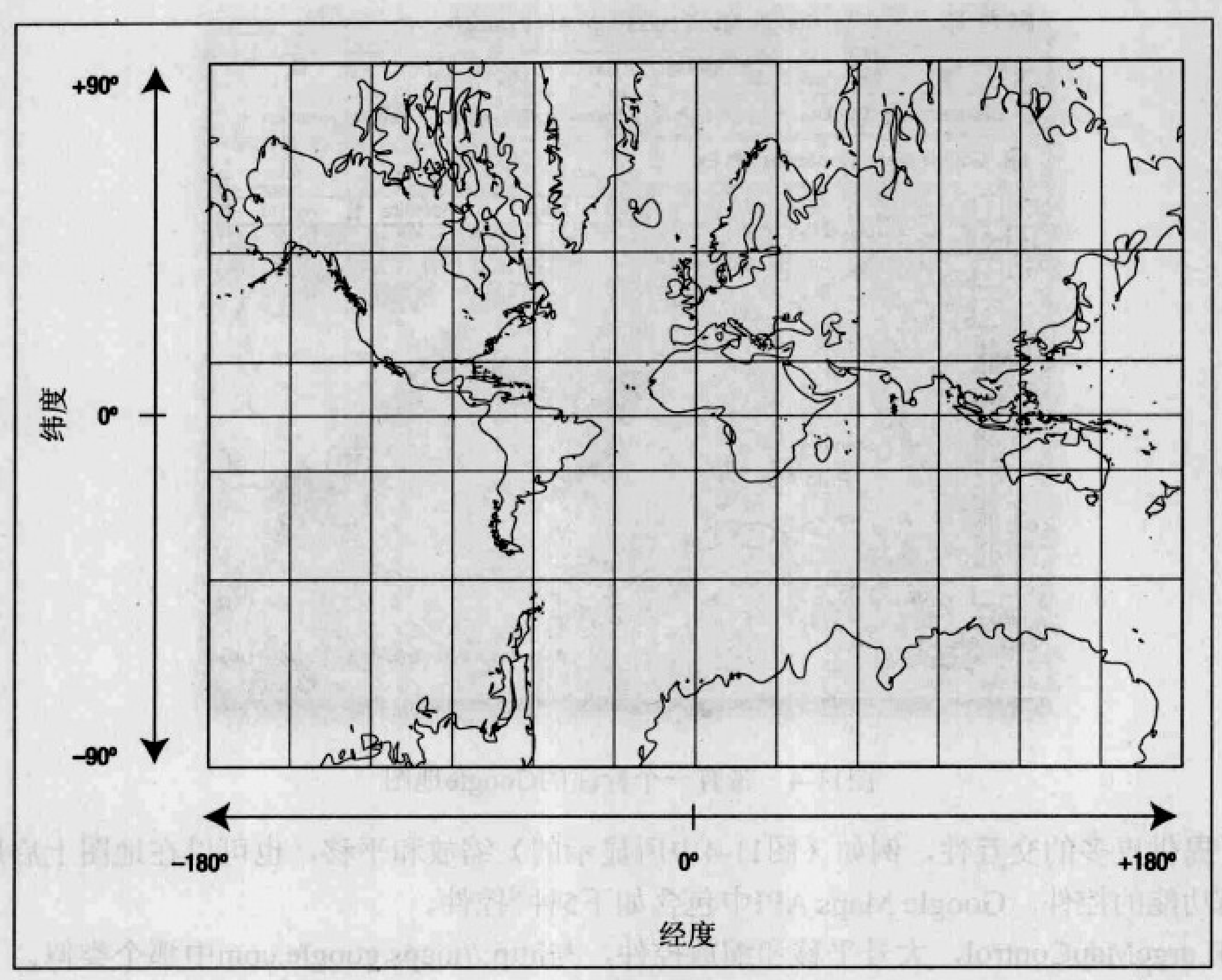



图11-3 地球表面的经度和纬度

> 如果不知道要显示位置的纬度和经度，也不必担心。稍后我们会讨论地理译码（geocoding）（将地址转换为纬度和经度坐标的程序处理）这个主题。

由于这个例子中有了纬度和经度，所以可以使用`GMarker()`对象的一个实例，在地图中创建标注覆盖物。`GMarker`对象会在地图上创建一个Google Maps中常见的倒置水滴状的标注，创建该对象需要以`GLatLng`对象来指定标注的位置：

```
// 使用location实例化一个新标注
var marker = new GMarker(location);
```

在实例化之后，还需要通过地图对象的`addOverlay()`方法将它添加到地图中：

```
// 将标注添加到地图中
map.addOverlay(marker);
```

结果是一幅在指定位置带有好看的标注和一些控件的地图，如图11-4所示。

www.TopSage.com

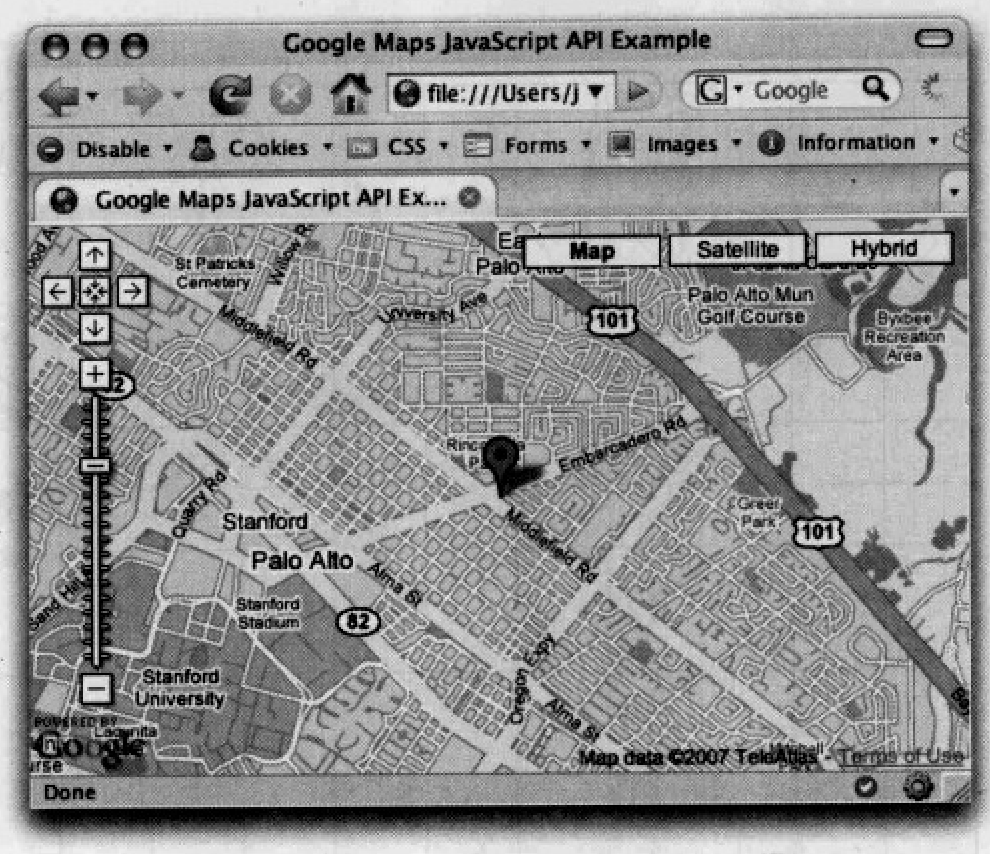



图11-4　带有一个标注的Google地图

为了提供更多的交互性，例如（图11-4中所显示的）缩放和平移，也可以在地图上启用一些具有不同功能的控件。Google Maps API中包含如下5种①控件。

- GLargeMapControl，大号平移和缩放控件，与http://maps.google.com中那个类似。
- GSmallMapControl，小号平移和缩放控件。
- GScaleControl，地图比例尺控件。
- GSmallZoomControl，小号缩放控件。
- GMapTypeControl，有效的地图类型控件，例如路面图或卫星图。

要启用以上控件，可以使用地图对象的addControl()方法，以其中一种控件的实例作为参数。例如，可以像下面这样为地图添加GLargeMapControl和GMapTypeControl：

```
// 添加缩放/平移及地图类型控件
map.addControl(new GLargeMapControl());
map.addControl(new GMapTypeControl());
```

由此可见，通过Google Maps API创建全功能的交互式地图有多么简单，只需几行代码而已。而且，可以实现的功能还远不止这些。比如，可以

- 为每个标注添加信息弹出窗口。
- 修改标注的图标样式。
- 创建自定义的对象和图像覆盖物。
- 整合你自己的地图类型及拼贴层。

① Google Maps API 2中还有第6种控件，即GOverviewMapControl，用于添加缩览图。——译者注

www.TopSage.com

- 添加从一个地点到另一地点的行车路线。

要了解API的全部特性，请查阅Google Maps API文档（http://www.google.com/apis/maps/documentation/）。

> *如果你对创建具有更强大特性的Google Maps应用非常感兴趣，请参考我的另外两本书beginning Google Maps Applications with PHP and Ajax和beginning Google Maps Applications with Rails and Ajax*（Apress，ISBN-13：978-1-59059-707-1和978-1-59059-787-3），这两本书的网站是http://googlemapsbook.com。

### 1. 取得纬度和经度

要在地图上放置标注，必须知道绘制点的纬度和经度。所幸，Google Maps API在其`GClientGeocoder`对象中包含了地理译码器（geocoder），可以通过它来取得人们熟知的地点、城市及邮政地址的纬度和经度。所得信息的精确度取决于数据的有效性。这个译码器通常可以用于下列国家。

- 美国
- 加拿大
- 法国
- 德国
- 日本
- 意大利
- 西班牙
- 澳大利亚
- 新西兰

使用`GClientGeocoder`对象的方式与Ajax调用非常类似，只不过需要传递的是地址和在载入完成后执行的回调函数：

```
var geocoder = new GClientGeocoder();
var address = '1600 Amphitheatre Parkway Mountain View, CA';
geocoder.getLatLng(
    address,
    function(point) {
        if (!point) {
            alert(address + " not found");
        } else {
            // 基于地址重新居中地图并创建标注
            map.setCenter(point, 12);
            var marker = new GMarker(point);
            map.addOverlay(marker);
        }
    }
);
```

www.TopSage.com

如果成功的话，回调函数的参数中将会包含一个表示相应位置纬度和经度的GLatLng对象，可以使用这个对象来定义新的标注。如果出于其他原因，你只想取得与地址对应的纬度和经度，那么可以使用point.lat()方法和point.lng()方法分别得到纬度和经度的值。

与Ajax请求类似，地理译码器的请求也是异步请求。因此，你可能有必要参阅第8章中有关异步请求复杂性的讨论。

> 在服务器端，也可以通过调用地理译码器取得同样的信息。因此，我们强烈建议不要在每次重载页面时，都使用JavaScript地理译码器频繁地请求同一个位置。最好是创建相应的服务器端脚本，取得打算显示的每一点的纬度和经度信息，并将它们存储到数据库中。这样会使页面中的地图显示得更快，而且即使你的网站流行到炙手可热的程度，也不会由于地理译码器的原因而遭遇瓶颈问题。

**2. 通过微格式来维护可访问性**

能以交互式的地图展示所有数据当然再好不过了，但如果地图无法访问又该如何呢？作为地图的一种替代，我们通过整合微格式的应用，提供了一种在DOM脚本无效的情况下，也能作为位置点数据库来使用的解决方案。

所谓微格式（http://microformat.org）就是一种标记数据的方式，以便人类和计算机都能轻松而一致地解释相应的数据。微格式的种类很多，但我们这里感兴趣的则是geo格式（http://microformats.org/wiki/geo），它是hcard格式（http://microformats.org/wiki/hcard）的一个子集，而且可以用它像下面这样来表示地理上的纬度和经度信息：

```
<div class="geo">
    <abbr class="latitude" title="45.27">
        45&#176; 16&#39; 12&#34; N
    </abbr>
    <abbr class="longitude" title="-75.42">
        75&#176; 25&#39; 12&#34; E
    </abbr>
</div>
```

为了示范如何使用微格式，我们来看一看对chapter11/map-accessible/cities.html中的内容删节后的代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>An Accessible Google Map</title>
    <link rel="stylesheet" href="../../shared/source.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="../chapter.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="map.css"
        type="text/css" media="screen" />
```

www.TopSage.com

```
    <script src="../../ADS-final-verbose.js"
        type="text/javascript"></script>

    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp; ➥
    key=YOUR_API_KEY" type="text/javascript"></script>
    <script src="map.js" type="text/javascript"></script>
</head>
<body>
  <h1>An Accessible Google Map</h1>
  <div id="content">
    <h2>Capital Cities of the World</h2>
    <div id="map"></div>
        <ul id="cities">
            <li class="vcard">
                <div class="adr">
                    <div class="country-name">
                        Heard Island and McDonald Islands
                    </div>
                </div>
                <div class="geo">
                    <abbr class="latitude" title="-53">53&#176; ➥
00&#39; 0&#34; S</abbr>
                    <abbr class="longitude" title="74">74&#176; ➥
00&#39; 0&#34; E</abbr>
                </div>
            </li>

            ……省略的代码……

            <li class="vcard">
                <div class="adr">
                    <span class="locality">Reykjavik</span>,
                    <div class="country-name">
                        Iceland
                    </div>
                </div>
                <div class="geo">
                    <abbr class="latitude" title="64.1"> ➥
64&#176; 05&#39; 60&#34; N</abbr>
                    <abbr class="longitude" title="-21.57"> ➥
21&#176; 34&#39; 12&#34; W</abbr>
                </div>
            </li>
        </ul>
    </div>
  </body>
</html>
```

图11-5所示的页面中，包含了全世界的所有首都和首府城市的一个简单的无序列表，但每个列表中都有一些特殊的地方：

- 使用geo微格式指定了城市的纬度和经度。
- 使用addr微格式指定了城市和国家（将用于生成标注的信息窗口）。

www.TopSage.com

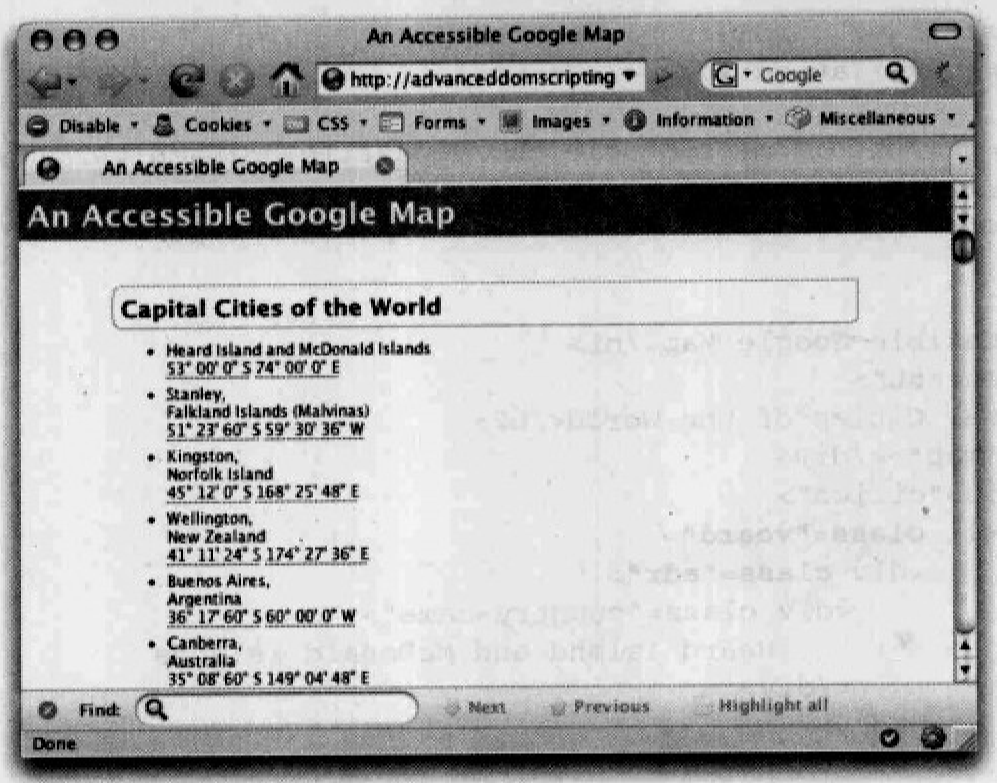



图11-5　一个整合了以geo微格式表示各城市位置的无序列表

为了基于这个列表来创建地图，可以在调用Google Map API之前，使用在本书中用过的各种DOM方法解析并获得必要的信息。例如，可以使用下面这个load事件：

```
ADS.addEvent(window,'load',function() {

    if (GBrowserIsCompatible()) {

      // 修改样式以显示地图
      ADS.setStyle('map',{
          width:'300px',
          height:'300px',
          float:'left'
      });

      ADS.setStyle('cities',{
          width:'180px',
          height:'300px',
          overflow:'auto',
          float:'right',
          "list-style":'none',
          margin:0,
          padding:0
      });

      // 实例化地图API
      var map = new GMap2(document.getElementById("map"));

      // 实例化新的纬度和经度坐标
      var location = new GLatLng(0, 0);

      //将地图中心点设置为缩放级别为13时的位置
      map.setCenter(location, 2);
```

www.TopSage.com

```
        // 添加缩放/平移及地图类型控件
        map.addControl(new GLargeMapControl());
        map.addControl(new GMapTypeControl());

        // 取得所有城市的<li> vcard元素
        var cities = ADS.$('cities').getElementsByTagName('li');

        // 通过一个函数来为info窗口信息
        // 维护适当的变量作用域
        function makeInfoWindow(marker,city) {
            var node = ADS.getElementsByClassName(
                'adr',
                'div',
                city
            )[0].cloneNode(true);
            GEvent.addListener(marker,'click',function() {
                marker.openInfoWindow(node);
            });
            ADS.addEvent(city,'click',function() {
                GEvent.trigger(marker,'click');
            });
            ADS.addEvent(city,'mouseover',function() {
                ADS.addClassName(city,'hover');
            });
            ADS.addEvent(city,'mouseout',function() {
                ADS.removeClassName(city,'hover');
            });

        }

        // 循环遍历每个城市并从微格式中
        // 取得相应的纬度和经度值
        for(i=0 ; (city = cities[i]) ; i++ ) {

            // 这里假设所有元素都存在
            // 因而未进行任何错误检查
            var latitude = ADS.getElementsByClassName(
                'latitude',
                'abbr',
                city
            )[0].getAttribute('title');
            var longitude = ADS.getElementsByClassName(
                'longitude',
                'abbr',
                city
            )[0].getAttribute('title');

            // 创建并向地图中添加标注
            var marker = new GMarker(
                new GLatLng(latitude, longitude)
            );
            makeInfoWindow(marker,city);
            map.addOverlay(marker);
        }
    }
});

ADS.addEvent(window,'unload',GUnload);
```

www.TopSage.com

在有了这个列表作为替代的数据源之后，其中的信息在常规情况下会如图11-6所示。而当地图不可见或者无法使用时，仍然能够保持其可访问性。同样的思路也可以应用到邮政地址目录的数据上，这种情况下，需要整合GClientGeocoder对地址进行编码以生成地图。

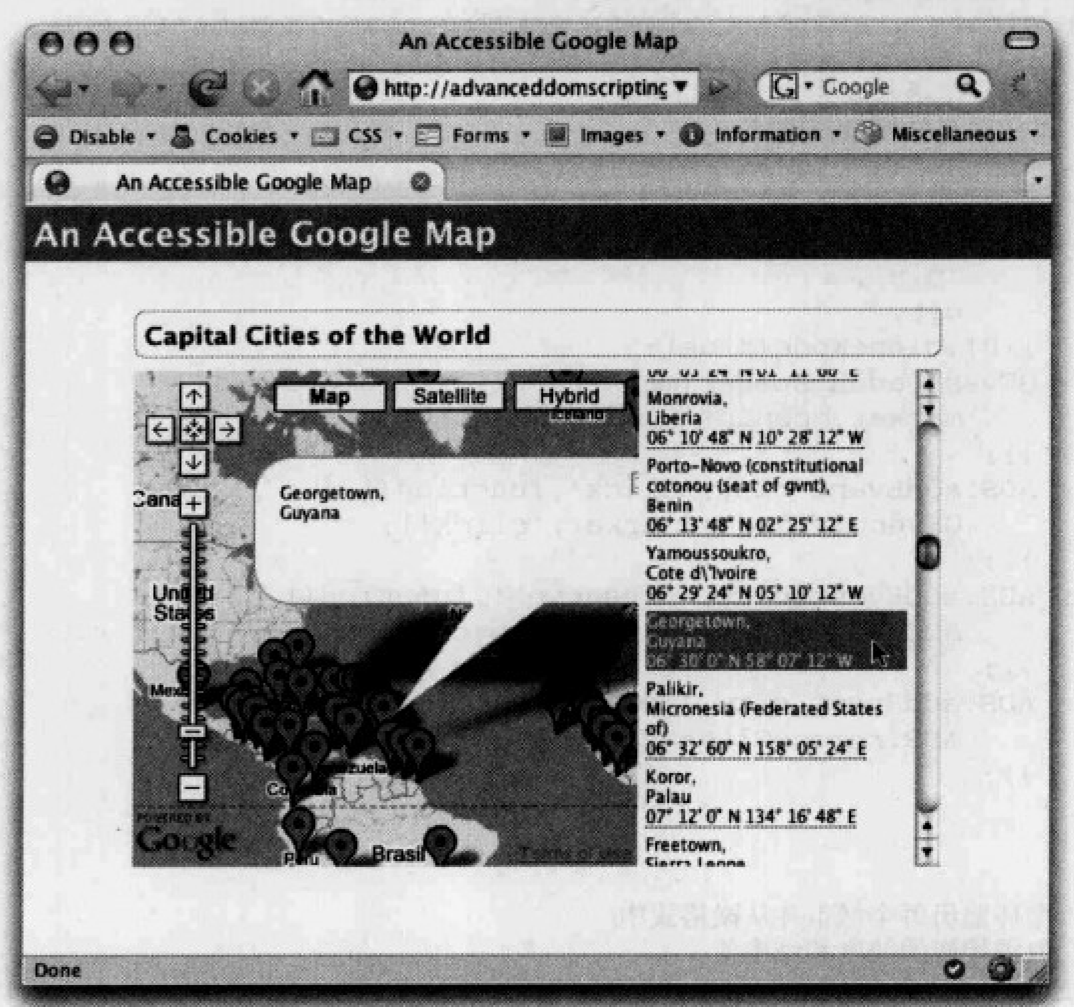



图11-6　将geo微格式无序列表转换为地图后的效果

## 11.2.2　Ajax搜索请求

Google Ajax Search API（http://code.google.com/apis/ajaxsearch/）是另外一种时髦的JavaScript API，它提供了一种直接在网站中整合自定义Google搜索结果的解决方案。

与Google Maps API类似，要使用Google Ajax Search API也需要注册一个API密钥（http://code.google.com/apis/ajaxsearch/signup.html）。在获得这个密钥之后，Google同样也提供了一个可以作为起点的例子：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html;➡
charset=utf-8"/>
    <title>My Google AJAX Search API Application</title>
    <link href="http://www.google.com/uds/css/gsearch.css"
        type="text/css" rel="stylesheet"/>
```

www.TopSage.com

```
    <script src="http://www.google.com/uds/api?file=uds.js ➥
&amp;v=1.0&amp;key=YOUR_API_KEY" type="text/javascript"></script>
    <script language="Javascript" type="text/javascript">
    //<![CDATA[

    function OnLoad() {
      // 创建搜索控件
      var searchControl = new GSearchControl();

      // 添加整套搜索部件
      var localSearch = new GlocalSearch();
      searchControl.addSearcher(localSearch);
      searchControl.addSearcher(new GwebSearch());
      searchControl.addSearcher(new GvideoSearch());
      searchControl.addSearcher(new GblogSearch());

      // 设置本地搜索中心点
      localSearch.setCenterPoint("New York, NY");

      // 绘制并添加搜索控件
      searchControl.draw(document.getElementById("searchcontrol"));

      // 执行初始搜索
      searchControl.execute("Google");
    }
    GSearch.setOnLoadCallback(OnLoad);

    //]]>
    </script>
  </head>
  <body>
    <div id="searchcontrol"/>
  </body>
</html>
```

这个例子中展示了下列API对象。

- GlocalSearch
- GwebSearch
- GvideoSearch
- GblogSearch

没有包含在这个例子中的对象如下[①]。

- GnewsSearch
- GbookSearch

可以在http://code.google.com/apis/ajaxsearch/documentation/中查看每个对象的文档，以了解它们的详细特性。

根据实现的选项不同，生成的结果中可能会包含可扩展的结果、过滤器及其他选项，如图11-7所示。

① 在翻译本书时，Google Ajax Search API中添加了一个新对象GimageSearch。——译者注

www.TopSage.com

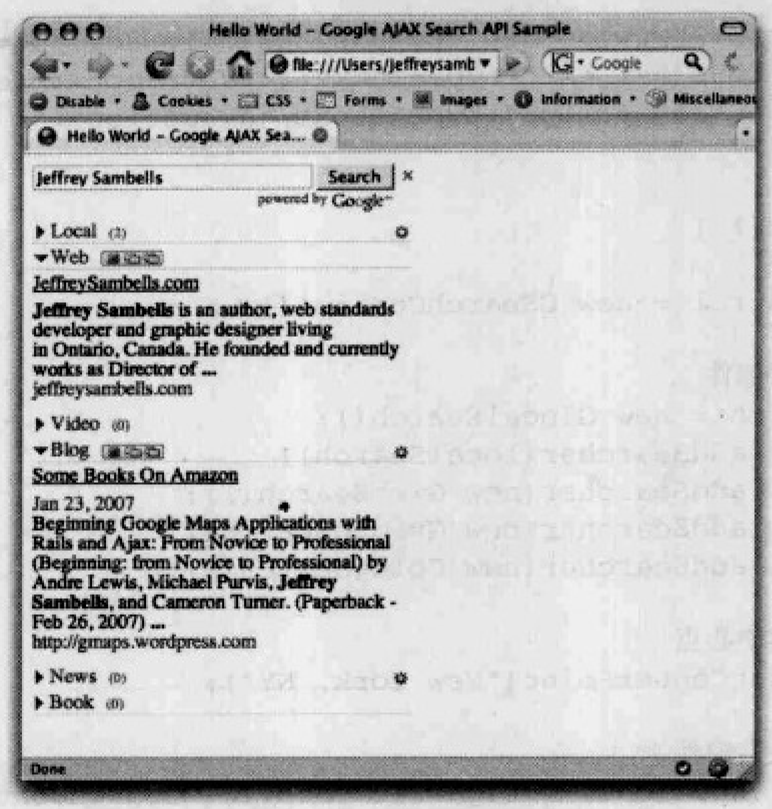



图11-7　例子页面中显示的搜索结果

在默认情况下，结果中不会包含任何嵌入的样式。但由于在文档的<head>部分也包含了Google的`gsearch.css`样式表，所以这个例子和图11-7所示的界面都体现了Google搜索结果的典型配色方案：

```
<link href="http://www.google.com/uds/css/gsearch.css"
    type="text/css" rel="stylesheet"/>
```

如果没有这个CSS文件，那么同一个页面将如图11-8所示。

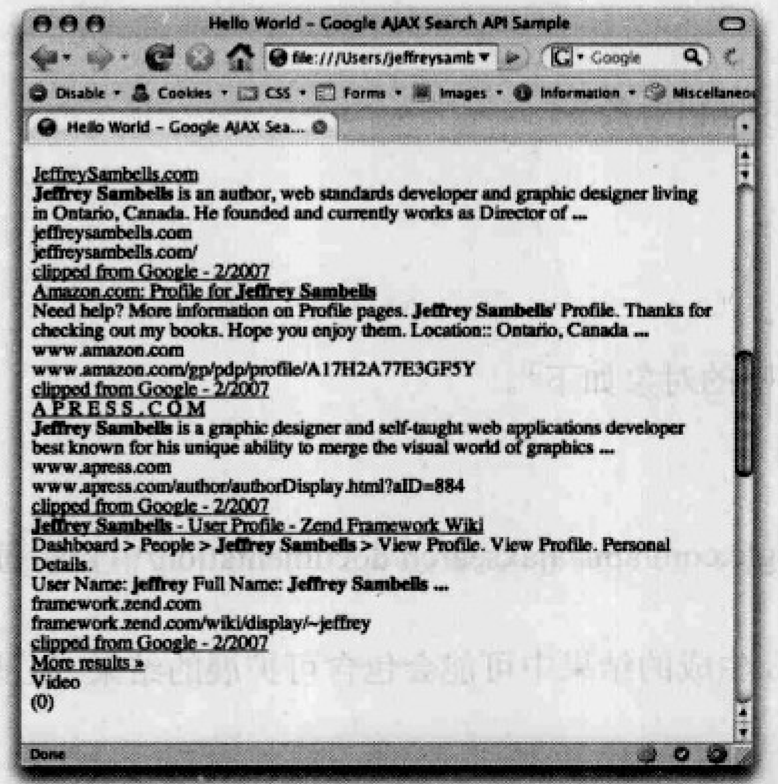



图11-8　在没有应用CSS样式的情况下结果页面的外观

www.TopSage.com

在生成的搜索结果标记中，包含了许多用于标识结果中不同部分的类，比如下面由一次GwebSearch搜索请求返回的结果标记。在下面的标记中，为了说明不同的部分，我已经添加了注释——但注释并非生成的标记中原有的：

```
<div id="search-related" class="gsc-resultsRoot gsc-tabData ➥
gsc-tabdActive gsc-resultsbox-visible">
    <!-- 一个用于扩展控件及结果数目的表格 -->
    <table cellspacing="0" cellpadding="0" class="gsc-resultsHeader">
        <tbody>
            <tr>
                <td class="gsc-twiddleRegionCell gsc-twiddle-opened">
                    <div class="gsc-twiddle">
                        <div class="gsc-title">
                            This Site
                        </div>
                    </div>
                    <div class="gsc-stats">
                        (4)
                    </div>
                    <div class="gsc-results-selector gsc-more-results-active">
                        <div class="gsc-result-selector ➥
gsc-one-result" title="show one result">
                             
                        </div>
                        <div class="gsc-result-selector ➥
gsc-more-results" title="show more results">
                             
                        </div>
                        <div class="gsc-result-selector ➥
gsc-all-results" title="show all results">
                             
                        </div>
                    </div>
                </td>
                <td class="gsc-configLabelCell">
            </tr>
        </tbody>
    </table>
    <div class="gsc-results gsc-webResult" style="display: block;">
        <div class="gsc-webResult gsc-result">
        <!-- 第一条结果包含在这个元素中 -->
            <div class="gs-webResult gs-result">
                <div class="gs-title">
                    <!--结果的标题 -->
                    <a href="http://playground.jeffreysambells.com ➥
/category/lipsum/" class="gs-title" target="_blank"></a>
                    <div class="gs-title">
                        <!--匹配的关键词包含在<b>标签中 -->
                        <strong>Lipsum</strong> archive at ➥
JeffreySambells.com - Play
                    </div>
                </div>
                <div class="gs-snippet">
```

www.TopSage.com

```
                <!—与匹配项相关的站点内容摘要 -->
                Published September 8th, 2006 in another,
                testing and <strong>Lipsum</strong>. 0
                Comments.<br>
                Lorem ipsum dolor sit amet, consectetuer
                adipiscing elit. Donec viverra mauris nec
                <strong>...</strong>
            </div>
            <div class="gs-visibleUrl gs-visibleUrl-short">
                <!—找到结果的域名-->
                <a href="#" class="gs-visibleUrl"></a>
                <div class="gs-visibleUrl">
                playground.jeffreysambells.com
            </div>
        </div>
        <div class="gs-visibleUrl gs-visibleUrl-long">
            <!—找到结果的完整URL -->
            <a href="#" class="gs-visibleUrl"></a>
            <div class="gs-visibleUrl">
              playground.jeffreysambells.com/category/lipsum/
            </div>
        </div>
        <div class="gs-watermark">
            <!-- 关于返回结果时间的参考 -->
            <a href="http://code.google.com/apis/ajaxsearch ➥
/faq.html" class="gs-watermark" target="_blank"></a>
                <div class="gs-watermark">
                    clipped from Google - 2/2007
                </div>
            </div>
        </div>
    </div>
    <div class="gsc-expansionArea">
        <!--
            除了第一条结果之外，其余结果项都
            包含在扩展区域元素中。但每条结果
            都与第一条结果遵循相同的编码模式
            -->
        <div class="gsc-webResult gsc-result">
            <div class="gs-webResult gs-result">
                <div class="gs-title">
                    <a href="http://playground.jeffreysambells ➥
.com/archives/" class="gs-title" target="_blank"></a>
                    <div class="gs-title">
                        Archives at JeffreySambells.com - Play
                    </div>
                </div>
                <div class="gs-snippet">
                    Browse by Category. Uncategorized; another;
                    testing; <strong>Lipsum</strong>
                </div>
                <div class="gs-visibleUrl gs-visibleUrl-short">
                    <a href="#" class="gs-visibleUrl"></a>
                    <div class="gs-visibleUrl">
```

```
                    playground.jeffreysambells.com
                  </div>
                </div>
                <div class="gs-visibleUrl gs-visibleUrl-long">
                  <a href="#" class="gs-visibleUrl"></a>
                  <div class="gs-visibleUrl">
                    playground.jeffreysambells.com/archives/
                  </div>
                </div>
                <div class="gs-watermark">
                  <a href="http://code.google.com/apis/ ➥
ajaxsearch/faq.html" class="gs-watermark" target="_blank"></a>
                  <div class="gs-watermark">
                    clipped from Google - 2/2007
                  </div>
                </div>
              </div>
            </div>
          </div>
          <div class="gsc-trailing-more-results">
            <a href="http://www.google.com/search?hl=en ➥
&amp;source=uds&amp;q=Lipsum%20site%3Ajeffreysambells.com" ➥
class="gsc-trailing-more-results" target="_blank"></a>
            <div class="gsc-trailing-more-results">
              More results »
            </div>
          </div>
        </div>
      </div>
```

不同的搜索对象会根据查到的结果生成其他的类名，因此为了找到特定的类就必须像上面这样剖析结果的源代码。如果想修改结果页面的外观，则要先看看服务条款，因为Google规定了要使用这项服务，必须要保持某些与品牌相关元素的一致性。

### 1. 针对具体网站的搜索结果

要使用Google搜索引擎只针对某个特定的网站查找结果，可以使用GwebSearch对象。通过GwebSearch对象可以对搜索进行某些限制，例如搜索哪个网站或者使用哪个自定义的引擎（http://google.com/coop/cse/）。以搜索http://advanceddomscripting.com这个网站为例，需要用到GwebSearch对象的setSiteRestrictions()方法：

```
// 创建搜索控件
var searchControl = new GSearchControl();

// 创建新的Web搜索对象
var siteSearch = new GwebSearch();
// 将搜索范围限制在advanceddomscripting.com中
siteSearch.setSiteRestriction("advanceddomscripting.com");

// 为控件中添加Web搜索对象
searchControl.addSearcher(siteSearch);

// 绘制并添加搜索控件
```

www.TopSage.com

```
searchControl.draw(document.getElementById("searchcontrol"));
```

这样就会将搜索范围限制为advanceddomscripting.com域，而搜索结果则可以直接整合到你的网站中。

在默认情况下，搜索结果会添加到作为参数传递给搜索控件的draw()方法的元素中。如果想把搜索结果放在页面中其他地方，可以使用搜索控件的setRoot()方法指定一个替代的结果容器：

```
searchControl.setRoot(document.getElementById('search-results'));
```

我们将在下一节讨论有关自定义结果的内容。

在使用Google Ajax Search时仅有的两个注意事项是：它依赖于JavaScript和必须要等到Google收录你的网站之后，才能从中搜索到结果（当然了）。如果你的网站刚刚启用，那么最好过一段时间再添加针对网站的搜索功能，以保证Google能够将你的网站添加到索引中。

> 为了加快Google收录网站的速度，我建议你注册Google Webmaster工具（http://www.google.com/webmasters）。最好再创建一个sitemap.xml文件（http://www.sitemaps.org/）以通知Google及其他搜索引擎如何索引你的网站。

### 2. 相关链接

高产的博客站长可能会发现，在自己的博客中添加与博客文章相关的搜索结果会非常有用。如果只想包含来自同一个网站的相关资料，那么可以使用前面的例子来限制搜索范围。不过，同样也可以在相关链接中包含Web上的任何内容。无论采取哪种方案，增加的唯一工作量，就是使用搜索控件的execute()方法及搜索字符串，在博客文章载入时自动执行搜索：

```
var searchControl = new GSearchControl();
var siteSearch = new GwebSearch();
searchControl.addSearcher(siteSearch);
searchControl.draw(document.getElementById("searchcontrol"));

// 自动取得与关键字中的字符串相关的结果
searchControl.execute(keywords);
```

这里的keywords可以是博客文章的标题，也可以是与文章相关的一组关键词。

如果沿着这个思路再发挥一下，我们先看看chapter11/related/post.html中的HTML源代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html;
        charset=utf-8"/>
    <title>Related Blog Posts</title>
    <link rel="stylesheet" href="../../shared/source.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="../chapter.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="related.css"
        type="text/css" media="screen" />
```

www.TopSage.com

```
    <script src="../../ADS-final-verbose.js"
        type="text/javascript"></script>

    <script src="http://www.google.com/uds/api?file=uds.js&amp; ➥
v=1.0&amp;key=YOUR_API_KEY" type="text/javascript"></script>
    <script src="related.js" type="text/javascript"></script>
  </head>
  <body>
    <div id="content">
        <div id="post">
            <p>Lorem ipsum dolor sit amet, consectetuer adipiscing
            elit. Cras leo. Vestibulum fermentum nisl vel lectus.
            Nam at quam. Nulla vulputate lacus eu enim. Sed a sapien
            at arcu tristique tempus.
            <p id="keywords">Lorem ipsum dolor sit amet</p>
        </div>
        <div id="related">
            <h4>Related Links</h4>
          <div id="search-controls"></div>
          <div id="search-related"></div>
          <div id="branding"></div>
        </div>
    </div>
  </body>
</html>
```

在添加下面的load事件后，以上页面就会从文章的关键字列表中取得相应的keywords，并使用GblogSearch对象从其他博客文章中动态地创建相关链接：

```
ADS.addEvent(window,'load',function() {

    GSearch.getBranding(ADS.$("branding"));

    var siteSearch = new GwebSearch();
    siteSearch.setUserDefinedLabel("This Site");

    var blogSearch = new GblogSearch();
    blogSearch.setUserDefinedLabel("blogosphere");

    searchOptions = new GsearcherOptions();
    searchOptions.setExpandMode(GSearchControl.EXPAND_MODE_OPEN);
    searchOptions.setRoot(ADS.$("search-related"));
    searchControl = new GSearchControl();
    searchControl.addSearcher(siteSearch,searchOptions);

    var options = new GdrawOptions();
    options.setDrawMode(GSearchControl.DRAW_MODE_TABBED);
    searchControl.draw(ADS.$("search-controls"), options);

    searchControl.setResultSetSize(GSearch.SMALL_RESULTSET);
    searchControl.execute(ADS.$('keywords').innerHTML);

});
```

通过以上load事件创建的相关链接如图11-9所示。

www.TopSage.com

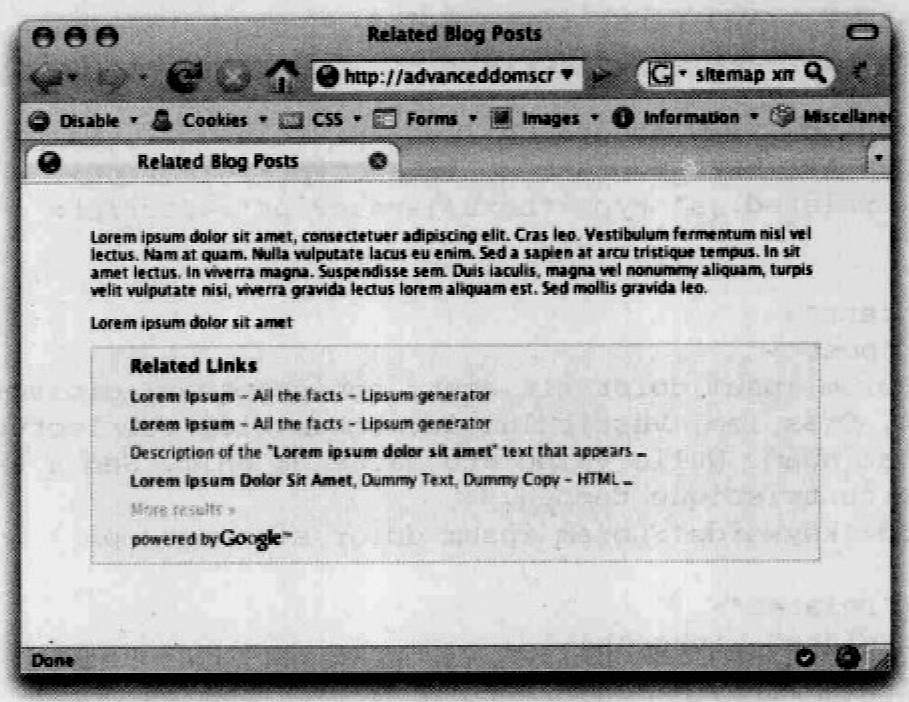



图11-9　自定义的相关链接列表

图11-9中的页面同样也整合了一个自定义的结果区域，并且重新设置了样式，也通过一些CSS规则隐藏了不必要的元素并重新定位了Google的标志，以便结果与博客的风格保持一致：

```
#search-controls {
    display:none;
}
#related {
    background: #fefde0;
    border: 1px solid #e0e0e0;
    padding: 0.5em;
}
#related h4 {
    padding-left: 20px;
    margin: 0;
    font-size: 1.2em;
}

#search-related a {
    padding-left: 20px;
}

#search-related a:link {
    text-decoration: none;
    color: #6b6b5f;
}
#search-related a:visited {
}
#search-related a:hover {
    text-decoration: underline;
}
#search-related a:active {
    color: red;
}
#search-related table,
```

www.TopSage.com

```
#search-related .gs-snippet,
#search-related .gs-visibleUrl,
#search-related .gs-watermark
{
    display:none;
}
#search-related .gs-result {
    margin: 0.7em 0;
}
#search-related a.gsc-trailing-more-results {
    coLor: #d1d2ba;
}
#search-related a.gsc-trailing-more-results:hover {
    color: #6b6b5f;
}
.gsc-branding {
    text-align: right;
    margin-top: 4px;
    width: 120px;
    padding-left: 8px;
}
.gsc-branding table {
    margin-left: auto;
    width: 100%;
}
.gsc-branding-text {
    display: block;
    font-size: 1em;
    color: #676767;
    padding-right: 0.3em;
    margin-left: auto;
    text-align: right;
    width: 100%;
}
```

我们注意到，实际的搜索控件通过下面的CSS规则从视图中被隐藏了起来：

```
#search-controls {
    display:none;
}
```

Google Ajax Search API中有一条规定，即任何人都必须维持"Powered by Google"标志及文本的可见性。因此这个例子中，通过使用`GSearch.getBranding()`方法将以上标志及文本放到了底部的另一个<div>元素中：

```
GSearch.getBranding(document.getElementById("branding"));
```

至于自定义搜索结果的方式，可谓五花八门，不一而足。要得到更多的启发，请参阅Google在Google Ajax Search文档（http://code.google.com/apis/ajaxsearch/）中提供的例子。

### 11.2.3 地图与搜索的 Mashup 应用

本章要介绍的最后一种客户端Mashup应用，就是包含在Google Ajax Search API中的，整合了Google Maps API和Google Ajax Search API的GSmapSearchControl。通过使用这个控件，只需很少

www.TopSage.com

量的工作就能创建出将搜索结果显示在地图中心位置上的搜索工具。想找到你所在位置附近的商业实体吗？试一试chapter11/map-search/google.html中包含的例子，看看能在加拿大安大略省多伦多市的地图中找到些什么。搜索地图的结果如图11-10所示。

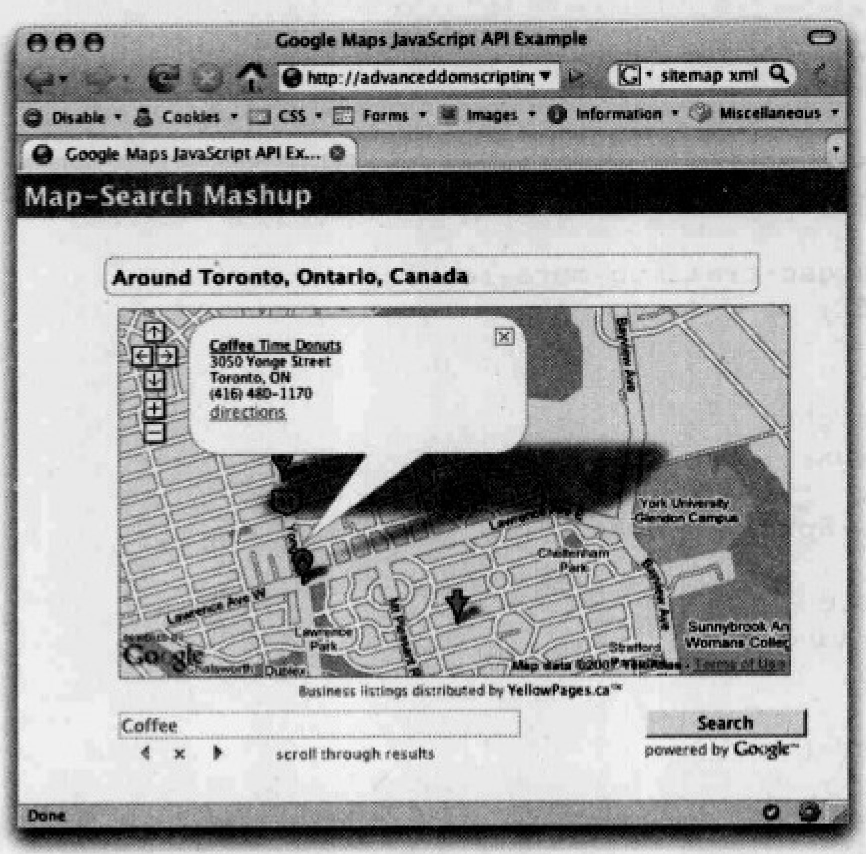



图11-10　加拿大安大略省多伦多市的咖啡店

这个例子返回的是在多伦多市地图上面绘制的搜索结果。你所要做的，就是创建一个简单的HTML文件，链接到Google Maps API、Google Ajax Search API，用于执行地图搜索的Mashup脚本及适当API密钥：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="content-type" content="text/html;
        charset=utf-8"/>
    <title>Google Maps JavaScript API Example</title>
    <link rel="stylesheet" href="../../shared/source.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="../chapter.css"
        type="text/css" media="screen" />
    <link rel="stylesheet" href="mapsearch.css"
        type="text/css" media="screen" />
    <script src="../../ADS-final-verbose.js"
        type="text/javascript"></script>

    <script src="http://maps.google.com/maps?file=api&v=2 ➥
&key=YOUR_API_KEY" type="text/javascript"></script>
    <script src="http://www.google.com/uds/api?file=uds.js ➥
&v=1.0&source=uds-msw&key=YOUR_API_KEY"
```

```
        type="text/javascript"></script>
    <style type="text/css">
    @import url("http://www.google.com/uds/css/gsearch.css");
    </style>

    <!-- 地图搜索控件及样式表 -->
    <script type="text/javascript">
    window._uds_msw_donotrepair = true;
    </script>
    <script src="http://www.google.com/uds/solutions/mapsearch ➥
/gsmapsearch.js?mode=new" type="text/javascript"></script>
    <style type="text/css">
    @import url("http://www.google.com/uds/solutions/mapsearch ➥
/gsmapsearch.css");
    </style>
    <script src="mapsearch.js" type="text/javascript"></script>
</head>
<body>
    <h1>Map-Search Mashup</h1>
    <h2>Around Toronto, Ontario, Canada</h2>
    <div id="mapsearch"></div>
</body>
</html>
```

然后，再使用几行CSS样式规则：

```
#mapsearch {
    width : 365px;
    height : 350px;
    margin: 10px;
    padding: 4px;
}
.gsmsc-mapDiv {
    height : 275px;
}

.gsmsc-idleMapDiv {
    height : 275px;
}
```

和必需的load事件：

```
ADS.addEvent(window,'load',function() {

    new GSmapSearchControl(
        document.getElementById("mapsearch"),
        "Toronto, Ontario, Canada",
        {
            zoomControl : GSmapSearchControl.ZOOM_CONTROL_ENABLE_ALL,
            idleMapZoom : GSmapSearchControl.ACTIVE_MAP_ZOOM,
            activeMapZoom : GSmapSearchControl.ACTIVE_MAP_ZOOM
        }
    );

});

ADS.addEvent(window,'unload',GUnload);
```

www.TopSage.com

事实上，最后的load事件也非常普通，它只是简单地基于一个地址字符串和几个用于控制地图的选项（例如控件和缩放级别等），指定了地图的中心位置。

## 11.3　服务器端API：需要代理脚本

客户端JavaScript API固然强大，但更有数百种要求进行服务器端交互的其他API同样可以利用。下面仅列举提供这种API的几种服务。

- Amazon product information and management（http://aws.amazon.com）
- Basecamp project management（http://www.basecamphq.com/api/）
- eBay auction services（http://developer.ebay.com/common/api）
- FedEx shipping services（http://www.fedex.com/us/solutions/wis/）
- Flickr photo sharing（http://www.flickr.com/services/api/）
- Google services for everything from Calendar to AdWords（http://code.google.com/apis/）
- YouTube video sharing（http://www.youtube.com/dev）

这些服务只提供对服务器端有效的API，主要是因为：

- 访问及交互需要更高级的技术，如SOAP、REST；或者需要专有的方法。
- 大数据量传输。这种情况下如果通过客户端方法来实现则会降低效率，造成过多重复访问。
- 访问限制。比如浏览器的同源安全策略会阻塞对服务的访问，或者导致实现起来过于复杂。
- 服务中加入了更安全的访问方法。例如私人密钥或者用户名/密码认证，而这是无法通过客户端API实现的。

为此，我们面临的问题，就是如何使用DOM脚本访问这些强大而出色的API。解决方案就是创建服务器端代理脚本。

如果想在浏览器中，通过DOM脚本直接访问这些仅对服务器端有效的API，就必须在应用程序中创建相应的服务器代理程序，以便让Ajax调用从你的服务器上取得必要的信息，而你的服务器再从目标API中取得信息，整个过程如图11-11所示。

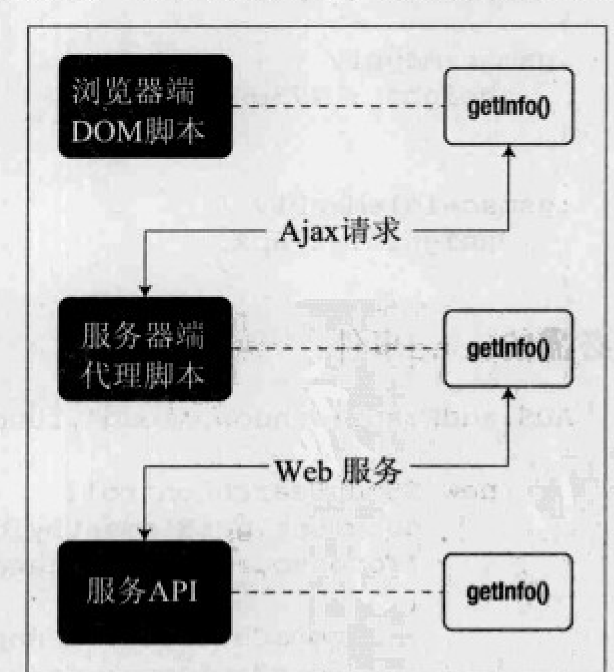



图11-11　服务器端代理脚本的工作流程图

服务器端代理脚本可以是只包含像下面这样一行PHP代码的文件，它只负责按照给定的URL取得相应的XML文档：

```
<?php
echo @simplexml_load_file($_GET['url'])->asXML();
?>
```

或者是其他使用API提供的自定义方法的更复杂的脚本。可以将前面的PHP代理脚本与常见

www.TopSage.com

的Ajax请求结合起来，取得一个RSS源：

```
ADS.ajaxRequest(
    '/path/to/proxy.php?url=http://advanceddomscripting.com/rss',
    onComplete : function() {
        // 解析源并将其中的内容添加到文档
    }
);
```

不过，过于简单的代理脚本也存在一些问题。由于只需要传递一个URL，也就意味着任何人可以传递任何URL，于是很可能导致有人冒充你的服务器来请求文件。而某些动机不纯的人则可能会通过随机的URL请求随机的文件，最终将导致你的服务器无法响应。如果你打算使用前面的代理脚本，我建议你再添加一些用于安全保护的代码，如检查URL来源和使用关键字指定URL：

```
<?php
if(strpos(
    strtolower($_SERVER['HTTP_REFERER']),
    'http://advanceddomscripting.com'
) !== 0 ) {
    die('Not Allowed');
}
switch($_GET['do']) {
    case 'advanceddomscripting';
        $url = 'http://advanceddomscripting.com/rss';
        break;
    case 'jeffreysambells';
        $url = 'http://jeffreysambells.com/rss';
        break;
    default:
        die();
}

header('Content-type:application/xml');
echo simplexml_load_file($url)->asXML();
?>
```

这样，服务器只会允许来自http://advanceddomscripting.com的请求，并且也不再随意接受URL，而是只对下面的URL作出响应：

```
/path/to/proxy.php?do=advanceddomscripting
/path/to/proxy.php?do=jeffreysambells
```

> 检查URL来源也不能真正提高安全性，因为恶意用户很容易伪造来源并发送代理脚本允许的URL。但是，检测来源可以防止其他网站在不必要的情况下意外地链接到你的代理脚本。

对于取得RSS源的应用而言，最好能够加入某种类型的服务器端缓存机制，以免每次页面载入都会请求该源，从而保证应用程序响应得更快。

www.TopSage.com

在准备好接受请求的代理脚本之后，客户端DOM脚本可以向这个代理自由地发送调用，以取得相应的信息。比如，要将http://advanceddomscripting.com中的最新RSS项嵌入到你的个人网站中，可以使用前面的代理脚本（在修改来源URL之后）取得RSS源的XML文件，并将其中的内容解析到DOM文档中：

```
ADS.addEvent(window,'load',function() {
    // 创建取得RSS源的Ajax请求
    ADS.ajaxRequest('proxy.php?do=advanceddomscripting',{
        completeListener:function() {
            // 只有DOM2核心方法有效（DOM2 HTML方法无效）
            // 解释RSS源
            var doc = this.responseXML;
            var posts = doc.getElementsByTagName('item');
            // 循环遍历每篇文章
            for(i=0; (post = posts[i]) ; i++) {
                var title = post.getElementsByTagName('title')[0];
                var description = post.getElementsByTagName( ➥
'description')[0];
                var link = post.getElementsByTagName('link')[0];
                // 创建新的列表项
                var li = document.createElement('li');

                // 检查并添加标题
                if(title && title.firstChild) {
                    var h4 = document.createElement('h4');
                    if(link && link.firstChild) {
                        var a = document.createElement('a');
                        a.setAttribute('href',link.firstChild.nodeValue);
                        a.setAttribute('title','Read more about: '
                            + title.firstChild.nodeValue + '');
                        a.appendChild(title.firstChild)
                        h4.appendChild(a);
                    } else {
                        h4.appendChild(title.firstChild);
                    }
                    li.appendChild(h4)
                }
                // 检查并添加简介
                if(description && description.firstChild) {
                    var p = document.createElement('p');
                    p.appendChild(description.firstChild);
                    li.appendChild(p)
                }
                // 将项添加到列表
                document.getElementById('rss-feed').appendChild(li);
            }
        }
    });
});
```

通过在Web应用程序中整合代理脚本，就能够访问任何Web服务了。

接下来，我们要介绍两个比简单地请求XML文件更复杂一些的例子。第1个例子是与

Basecamp项目管理API进行交互，以便取得并向Basecamp的To-Do列表（用于Web项目开发非常合适！）中添加项。第2个例子是通过用户注册表单中的电子邮件地址，取得Flickr中的个性头像图标。

> 下面的例子将全部使用PHP来编写服务器端代理脚本。类似的技术和方法也适用于大多数其他服务器端语言。从本质上来说，它们都是通过某种方法发送API规定的适当请求。

## 11.3.1 通过 Basecamp 构建集成的 To-Do 列表

Basecamp（http://basecamphq.com）是由37Signals提供的一种小型的、基于Web的、出色的项目管理应用程序。在提供众多强大的项目管理功能的同时，该应用程序也提供了一组API，从而使我们能够通过带有特定头部信息的常规HTTP POST请求，来管理自己的Basecampe账户。在这个例子中，我们将示范如何使用这组API与你在Basecamp中的账户（允许免费注册）进行交互，同时为你的网站创建一个To-Do列表，如图11-12所示。

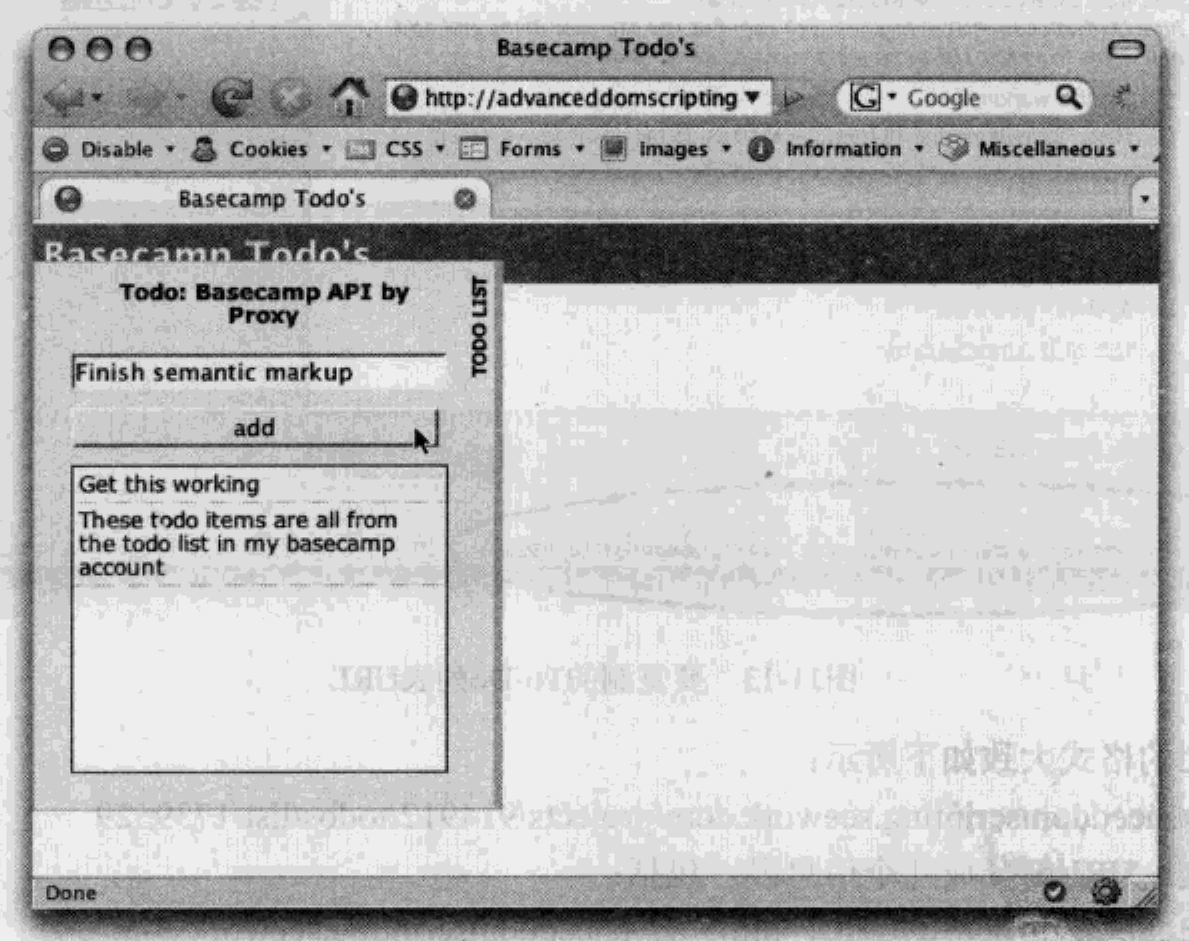



图11-12 使用Basecamp API的一个完整的To-Do列表

### 1. 你的Basecamp账户信息

首先，你需要注册一个Basecamp账户，如果还没有，那就登录http://basecamphq.com注册一个。为了让用户体验它的功能，Basecamp提供了一个免费的、单项目的账户方案，恰好适合我们这个例子使用。然后，按照下面的步骤进行：

www.TopSage.com

(1) 创建自己的新项目，并随便为它起个名字。

(2) 在你的项目中创建To-Do列表，也随便为它起个名字。

在To-Do页面中，将鼠标悬停在列表名称上面，复制列表的URL，见图11-13。

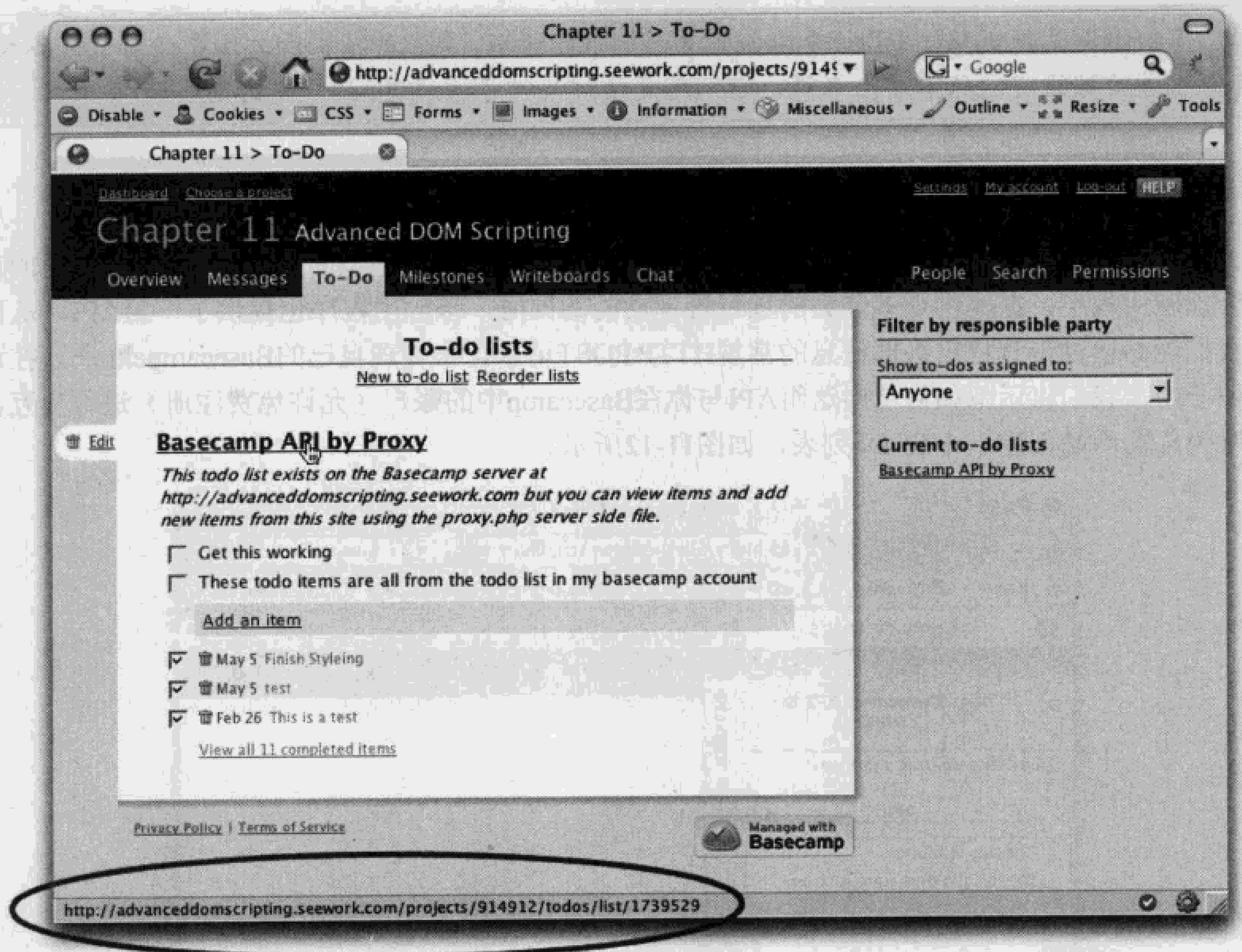



图11-13　要复制的To-Do列表URL

这个URL的格式大致如下所示：

http://advanceddomscripting.seework.com/projects/914912/todos/list/1739529

可以将这个URL细分成几个信息段，包括：

- **你的Basecamp账户的URL**，http://advanceddomscripting.seework.com
- **项目ID号**，914912
- **To-Do列表ID号**，1739529

当与API进行交互时，需要用到这3段信息及用于验证身份的用户名和密码。

(3) 在使用API之前要做的最后一件事，就是为你的Basecamp账户启用API访问功能。转到Dashboard，然后选择Account导航标签，结果将如图11-14所示。

www.TopSage.com

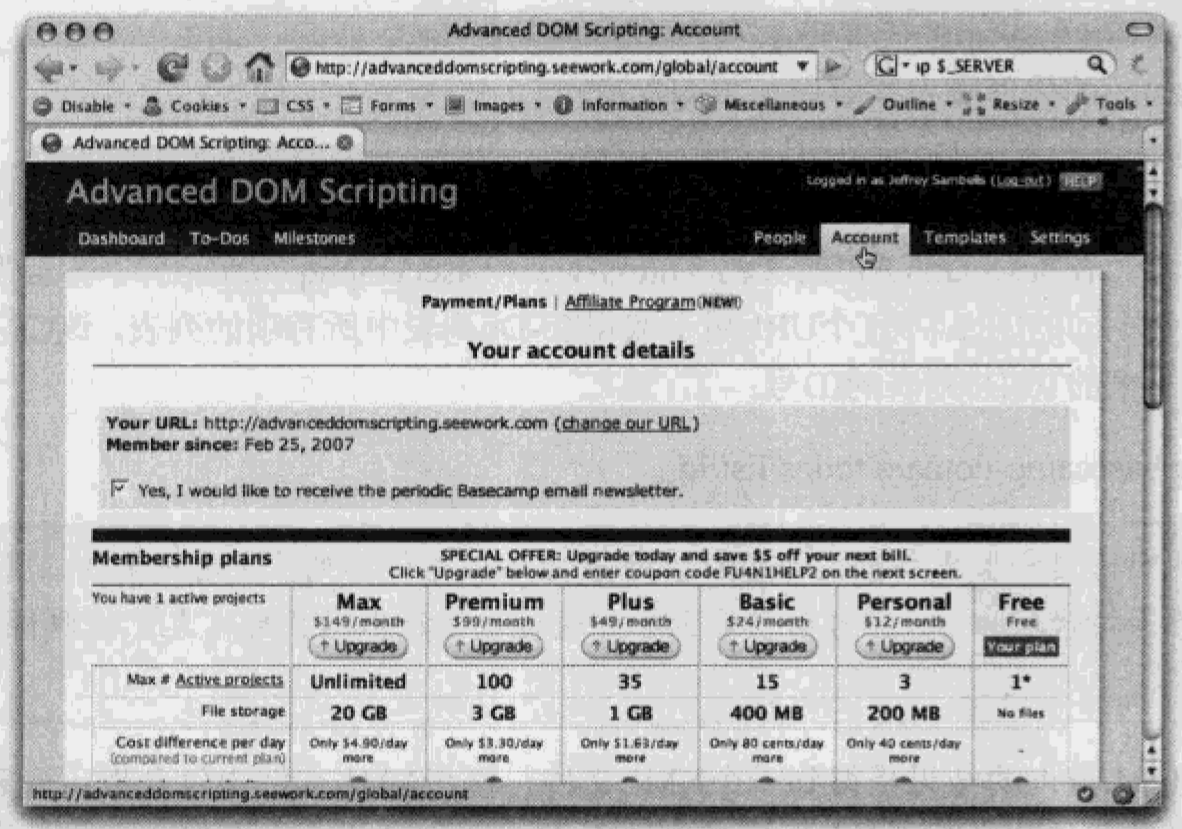



图11-14 Basecamp中的Account页面

(4) 在阅读服务条款之后启用Basecamp API，如图11-15所示。这样，就可以通过Basecamp API的来管理你自己的账户了。

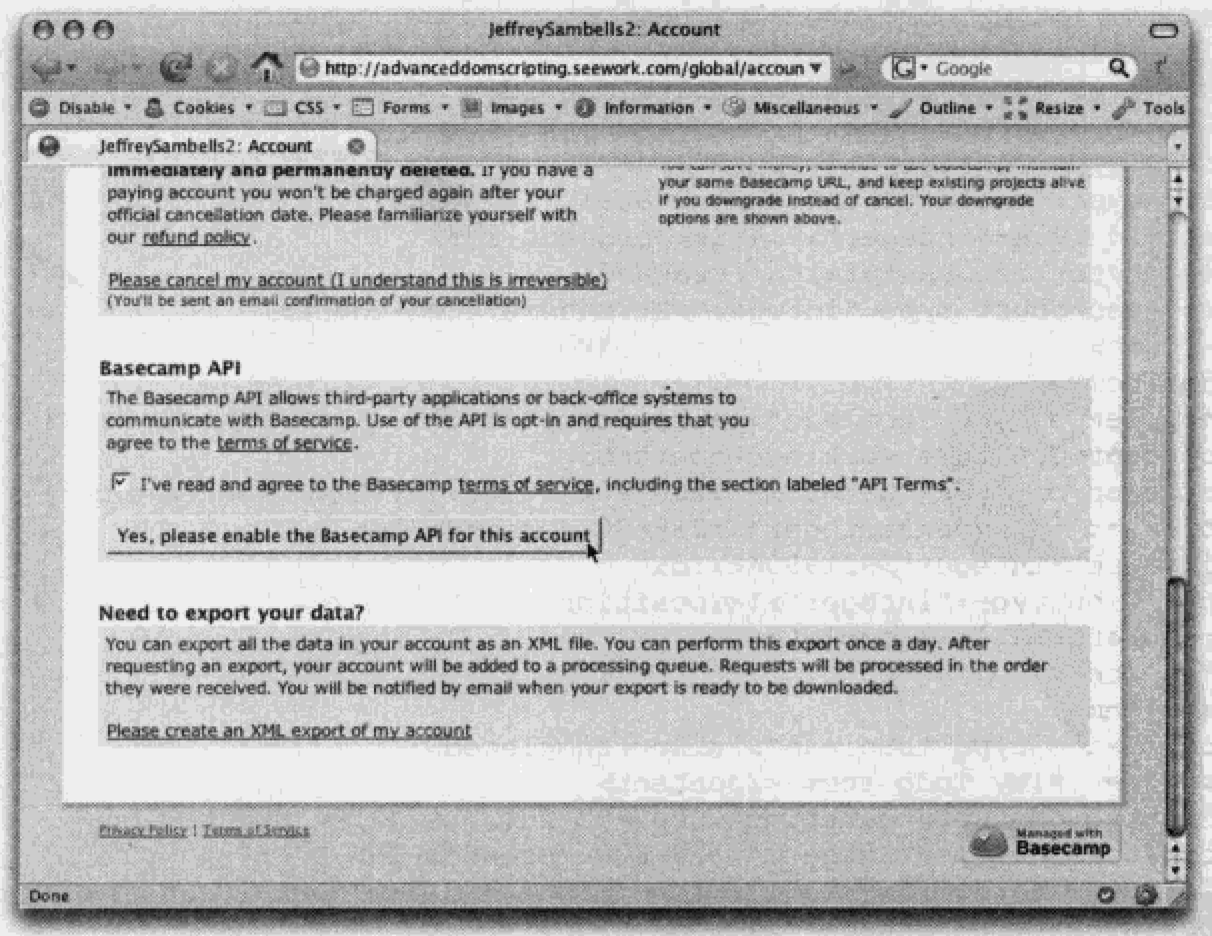



图11-15 为你自己的账户启用Basecamp API管理功能

### 2. 构建针对Basecamp的代理脚本

针对Basecamp的代理脚本需要做如下两件事。

- 从你刚刚创建的To-Do列表中取得项列表。
- 允许向同一个To-Do列表中添加新项。

如果大致看一看Basecamp API的文档（http://www.basecamphq.com/api/），你会发现这两个任务需要访问两个不同的URL。第1个URL用于取得To-Do列表中所有项的列表，该URL中必须包含你在前面复制下来的To-Do列表的ID号：

http://your-basecamp-domain/todos/list/**id**

> 如果你此时通过浏览器访问这个URL，结果将什么也得不到。这是正常的，因为我们马上要创建的服务器端代理脚本，会自动登录服务并通过发送特殊的头部信息才能取得正确的XML输出。

使用这个URL查询的结果是得到一个XML文件，这个XML文件中包含指定列表的所有元数据和项，下面是从我的位于http://advanceddomscripting.seework.com上面的Basecamp账户中得到的列表示例：

```
<todo-list>
  <completed-count type="integer">0</completed-count>
  <description>This list will be editable via proxy from ➥
advanceddomscripting.com</description>
  <id type="integer">1739529</id>
  <milestone-id type="integer"></milestone-id>
  <name>Proxy Updater</name>
  <position type="integer">1</position>
  <private type="boolean">false</private>
  <project-id type="integer">914912</project-id>
  <tracked type="boolean">false</tracked>
  <uncompleted-count type="integer">2</uncompleted-count>
  <todo-items>
    <todo-item>
      <completed type="boolean">false</completed>
      <content>Get this working</content>
      <created-on type="datetime">2007-02-25T20:11:35Z</created-on>
      <creator-id type="integer">1259600</creator-id>
      <id type="integer">9219994</id>
      <position type="integer">1</position>
      <complete>false</complete>
    </todo-item>
    <todo-item>
      <completed type="boolean">false</completed>
      <content>#{My Todo item}</content>
      <created-on type="datetime">2007-02-25T21:08:21Z</created-on>
      <creator-id type="integer">1259600</creator-id>
      <id type="integer">9220511</id>
      <position type="integer">2</position>
      <complete>false</complete>
    </todo-item>
```

www.TopSage.com

```
  </todo-items>
  <complete>false</complete>
</todo-list>
```

完成第2个任务需要使用的URL格式如下：

http://your-basecamp-domain/todos/create_item/**id**

通过在这个URL中指定相同的To-Do列表ID号可以创建一个新的To-Do项，但此时，API也需要你在POST请求的主体中包含定义新To-Do项的XML数据：

```
<request>
    <content>The new todo item</content>
<request>
```

> URL中的create_item请求也可以用来指定To-Do列表项的其他属性，例如分配到这个任务的个人等。出于示范的需要，本例只向列表中添加不带任何额外属性的项。将来，你可以在Basecamp中编辑这些项，并将它们指定给特定的人。当然，也可以在代理脚本中添加更多的功能，以便在创建新任务时能够选择相应的人员。

除了适当的URL和请求的格式之外，Basecamp API还要求在请求中包含如下两个特殊的头部信息。

- 值为application/xml的Content-Type。
- 值为application/xml的Accept。

通过这两个头部信息，Basecamp系统可以识别出请求是API请求，而非常规的浏览器请求。如果没有这两个特殊的头部信息，请求就得不到适当的响应。

将前面讲到的所有内容综合起来，最终的PHP代理脚本（使用CURL程序发送请求及特殊的头部信息并取得响应）大致如下：

```
<?php

// 目标URL：
// http://advanceddomscripting.seework.com/projects/914912/todos/list/1739529

// 使用CURL通过命令行
// 实际发送请求的函数
function makeRequest($url,$data=null) {
    $url = 'http://advanceddomscripting.seework.com'.$url;
    $authent = 'username:password';
    if($data) {
        $data = ' -d '.escapeshellarg($data);
    }
    $command = "curl -H 'Accept: application/xml' -H ➥
'Content-Type: application/xml' -u {$authent} $data $url";
    $output = null;

    exec(
        $command,
        $output
    );
```

www.TopSage.com

```
        if($output) {
            $xml = new SimpleXMLElement(join($output));
            return $xml;
        }

        return false;
    }

    // 根据GET请求中的do=var字符串来切换操作
    switch ($_GET['do']) {
        case 'create' :
            // 创建新的To-Do项
            if(!$_GET['todo']) die('');
            $request = <<<REQUEST
    <request>
      <content>{$_GET['todo']}</content>
    <request>
    REQUEST;

        if(($xml = makeRequest(
            '/todos/create_item/1739529',
            $request
        ))!==false) {
            echo json_encode($xml);
        } else {
            echo 'false';
        }

        break;
    case 'list':
        // 取得To-Do项列表
        if(($xml = makeRequest('/todos/list/1739529'))!==false) {
            echo json_encode($xml);
        } else {
            echo 'false';
        }
        break;

    }
    ?>
```

以上代理脚本接受两种类型的URL，一种URL通过`do=list`表示要取得项列表，如：

`/path/to/proxy.php?`**`do=list`**

另一种URL则通过`do=create`和经过URL编码的`todo`参数表示要创建新的To-Do项，如：

`/path/to/proxy.php?`**`do=create&todo=A%20New%20Item`**

无论是哪种情况，代理脚本都会返回一个JSON对象，以便在DOM脚本中用它来创建HTML列表。

> 这里的代理脚本中没有加入任何安全检查代码。也就是说，任何人都可以通过它来查看To-Do列表中的项，或者向列表中添加新项。为此，当在开发环境下使用这个脚本时，还需要添加额外的验证代码。

www.TopSage.com

### 3. Basecamp例子中的DOM脚本

本例最后一部分内容就是与服务器代理交互的DOM脚本。在这里，DOM脚本将负责创建与To-Do列表有关的所有标记，因此将会适用于任何HTML文件。其中，`load`事件会创建一个绝对定位的元素，当鼠标在其可见部分上面悬停时，整个元素会从屏幕边缘突然出现。而且，其中也包含了创建新To-Do项的`<form>`元素：

```
var todoContainer;
var todoList;

function createTodoBar(data) {
    // 创建保存To-Do列表元素的容器
    todoContainer = document.createElement('DIV');

    // 为容器添加样式
    ADS.setStyle(todoContainer,{
        'width': '250px',
        'overflow': 'hidden',
        'border': '1px solid #ccc',
        'border-right-width' : '5px',
        'background': '#fed url(images/todo.gif) no-repeat right top',
        'color': '#000',
        'font':' 12px/13px verdana,tahoma,sans',
        'text-align':'left'
    });

    // 设置起点以便在窗口
    // 左侧只剩下一条区域
    var basePosition= {
        'position' : 'absolute',
        'opacity' : '.55',
        'z-index' : '2000',
        'top' : '20px',
        'left' : '-235px'
    }
    ADS.setStyle(todoContainer,basePosition);

    // 当鼠标移出时将样式设置默认定位
    ADS.addEvent(todoContainer,'mouseout',function() {
        ADS.setStyle(todoContainer,basePosition);
    });

    // 设置悬停定位以便
    // 窗口从边缘弹出
    var hoverPosition= {
        'opacity' : '1',
        'left' : '0px' 、
    }
    // 当鼠标悬停时设置hover定位
    ADS.addEvent(todoContainer,'mouseover',function() {
        ADS.setStyle(todoContainer,hoverPosition);
    });

    // 添加头部信息
    var h4 = document.createElement('h4');
```

www.TopSage.com

```
ADS.setStyle(h4,{
    'color': '#000',
    'padding': '5px 20px',
    'margin': '5px',
    'font':'bold 12px/13px verdana,tahoma,sans',
    'text-align':'center'
});
h4.appendChild(document.createTextNode('Todo: ' + data.name));
todoContainer.appendChild(h4);

// 添加一个用于向列表中增加新项的表单
var form = document.createElement('form');
ADS.setStyle(form,{
    'display': 'inline',
    'border': '0',
    'padding': '0'
});

var text = document.createElement('input');
text.setAttribute('type','text');
ADS.setStyle(text,{
    'width':'200px',
    'margin':'5px 20px',
    'margin-right': '25px'
});

var submit = document.createElement('input');
submit.setAttribute('type','submit');
submit.setAttribute('value','add');
ADS.setStyle(submit,{
    'width':'200px',
    'margin':'5px 20px',
    'margin-right': '25px'
});

form.appendChild(text);
form.appendChild(submit);
todoContainer.appendChild(form);

// 根据To-Do项创建列表
todoList = document.createElement('UL');
ADS.setStyle(todoList,{
    'height':'170px',
    'border':'1px solid black',
    'padding': '0',
    'margin': '5px 25px 20px 20px',
    'list-style':'none',
    'overflow': 'auto',
    'background':'white'
});

todoContainer.appendChild(todoList);
document.body.appendChild(todoContainer);

// 当提交表单时，联系服务器代理脚本以添加新项
ADS.addEvent(form,'submit',function(W3CEvent) {
```

www.TopSage.com

```
        if(text.value) {
        // 如果存在一个值则只发送请求
        ADS.ajaxRequest('proxy.php?do=create&todo='
            + escape(text.value),{
            completeListener:function() {

                // 在请求完成时重新取得
                // 应该在列表中的全部项
                ADS.ajaxRequest('proxy.php?do=list',{
                    completeListener:function() {
                        text.value='';
                        eval('addTodoItems('
                            + this.responseText + ')');
                    }
                });
            }
        });
        }
    var event = ADS.getEventObject(W3CEvent);
    ADS.preventDefault(event);

    });
}

function addTodoItems(data) {
    // 以新项重新填充列表
    ADS.removeChildren(todoList);
    for (var i in data['todo-items']['todo-item']) {
        var item = data['todo-items']['todo-item'][i];
        if(item.completed == 'false') {
            var li = document.createElement('li');
            li.appendChild(document.createTextNode(item.content));
            ADS.setStyle(li,{
                'border': '0',
                'padding': '0.25em',
                'margin': '0',
                'border-bottom': '1px dotted #ccc'
            });
            todoList.appendChild(li);
        }
    }
}

// 当页面载入完成后创建并生成To-Do列表
ADS.addEvent(window,'load',function() {
    ADS.ajaxRequest('proxy.php?do=list',{
        completeListener:function() {
            eval('createTodoBar(' + this.responseText + ')');
            eval('addTodoItems(' + this.responseText + ')');
        }
    });
});
```

这个例子的结果是前面图11-12中所示的一个整合的To-Do列表。如果在你自己的网站中实现了这个应用，那么它将成为项目评审过程中的一个得力工具——你的客户在发现bug时，随时可以添加必要的To-Do项。Basecamp也提供了能够改进这一应用的更多特性，因此还能够在其中集成消息收发及其他项目管理功能。

www.TopSage.com

> 在这个例子中，如果没有JavaScript是无法访问To-Do项的。因为你必须考虑这个例子的设计意图就是作为一个开发者工具，而作为开发者应该能够控制访问这个应用的设备和访问系统的浏览器类型。如果可访问性对你而言是个问题，那么当页面初次载入时，列表很有可能无法自动填充，进而让列表出现在原始的标记中。

## 11.3.2　通过Flickr取得个性头像

当用户在你的网站上创建账户时，你可能会为他们提供一个输入URL的选项，该URL指向能够代表注册者的图片（个人形象）。如果你也要求用户在表单中填写他们的电子邮件地址，那么可以把收集电子邮件的字段放在前面，然后与某个服务（如Flickr）联系而自动取得代表该用户的静态个性头像，如图11-16所示。

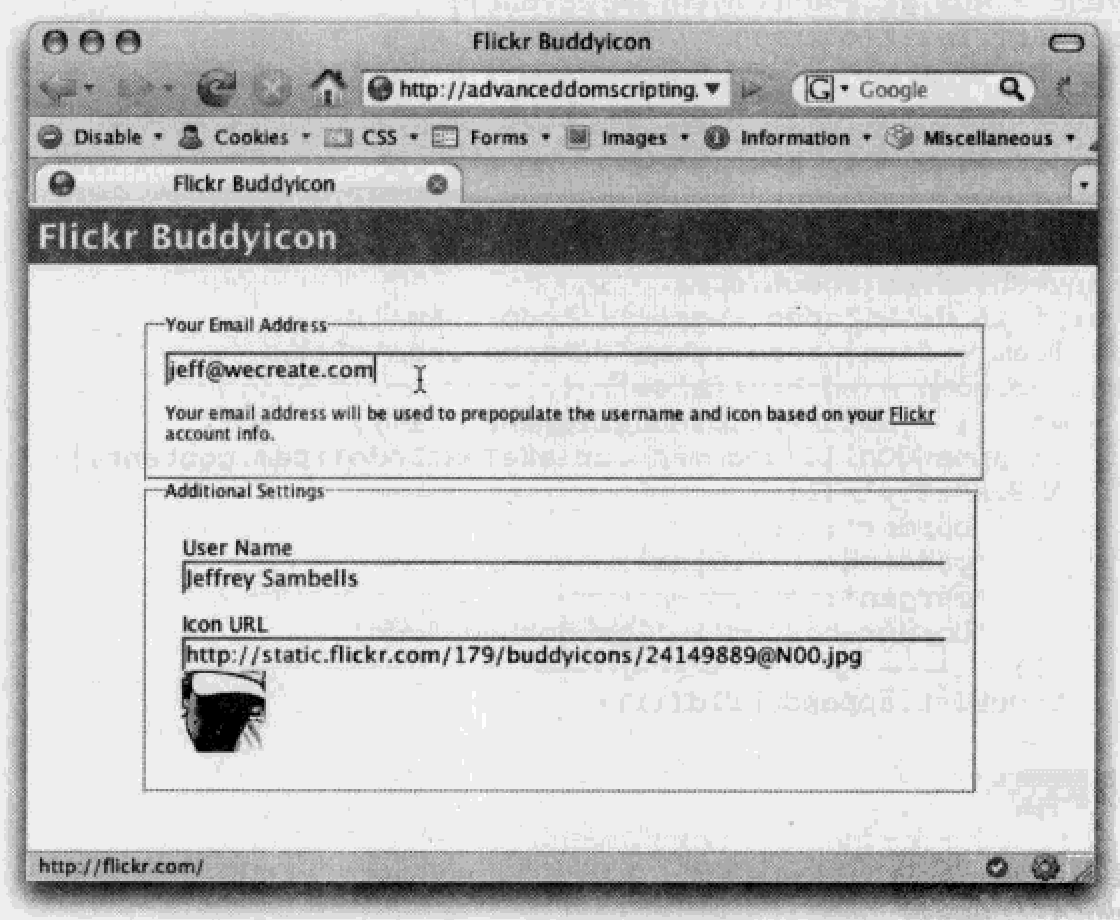



图11-16　基于电子邮件地址取得Flickr的个性头像及用户名

### 1. Flickr API密钥

为了访问Flickr API，首先要注册并获得API密钥（http://www.flickr.com/services/api/keys）。Flickr API的密钥与本章前面提到的Google API密钥的形式类似，也是由随机数字和字母组成的字符串。在此我就不公布自己的密钥了，否则无论谁都会以我的身份来访问Flickr API。不过，只要获得了自己的密钥之后，你就可以使用http://www.flickr.com/services/api中载明的所有特性。

### 2. 构建Flickr访问代理

取得与一个Flickr账户关联的自定义个性头像（如果存在的话），需要了解与用户账户相关的

www.TopSage.com

一些信息。首先，引用保存在Flickr特殊服务器上面的自定义JPG个性头像的URL格式如下：

```
http://static.flickr.com/icon-server/buddyicons/nsid.jpg
```

其中nsid是用户的Flickr ID，而icon-server则是保存个性头像的服务器。这是两个需要通过API取得的重要信息。

为取得ID，可以使用Flickr提供的flickr.people.findByEmail方法，该访法不仅能够返回nsid信息，还能够返回与给定电子邮件地址关联的用户名。例如，要取得我的Flickr ID（nsid），可以在包含你自己的Flickr API密钥之后，调用下面的REST服务URL：

```
http://api.flickr.com/services/rest/?method=flickr.people. ➥
findByEmail&api_key=YOUR_FLICKR_API_KEY&find_email= ➥
jeff%40advanceddomscripting.com
```

> 注意，URL中的电子邮件地址经过了转义处理，因此@符号被转换为了%40。

如果调用成功的话，你应该能够看到下列XML结果：

```
<rsp stat="ok">
    <user id="24149889@N00" nsid="24149889@N00">
        <username>Jeffrey Sambells</username>
    </user>
</rsp>
```

这样你就知道了我的nsid，即24149889@N00，同时也知道了我的用户名。

在取得了我的nsid之后，可以进一步使用其他的API方法来取得有关我的账户的更多信息。而为了取得我的个性头像，则还需要找到与我的账户关联的图标服务器。这个服务器信息可以使用flickr.people.getInfo方法来获得，而且同样也要在包含你自己的Flickr API密钥的情况下调用下面的REST服务URL：

```
http://api.flickr.com/services/rest/?method=flickr.people. ➥
getInfo&api_key=YOUR_FLICKR_API_KEY&user_id=24149889%40N00
```

结果会返回与我的账户有关的很多信息，其中也包括iconserver和我的账户下的照片数量：

```
<rsp stat="ok">
    <person id="24149889@N00" nsid="24149889@N00" isadmin="0" ➥
ispro="0" iconserver="179" iconfarm="1">
        <username>Jeffrey Sambells</username>
        <realname/>
        <mbox_sha1sum>48185cbcf809b112eb63690fb552ea5716b865b2</mbox_sha1sum>
        <location/>
        <photosurl>http://www.flickr.com/photos/jeffreysambells/</photosurl>
        <profileurl>http://www.flickr.com/people/jeffreysambells/</profileurl>
        <mobileurl>
            http://www.flickr.com/mob/photostream.gne?id=7084143
        </mobileurl>
        <photos>
            <firstdatetaken>2006-09-08 22:25:14</firstdatetaken>
            <firstdate>1172458623</firstdate>
            <count>9</count>
```

www.TopSage.com

```
        </photos>
    </person>
</rsp>
```

> 如果结果中的iconserver值为0，则说明与nsid关联的用户还没有设置它们默认的自定义个性头像。在这种情况下，将会使用默认图片http://www.flickr.com/images/buddyicon.jpg来代替。

现在，你已经知道了所有必要的信息，可以着手构建获取我的个性头像的URL了（如下所示），而我的个性头像如图11-17所示。

```
http://static.flickr.com/179/buddyicons/24149889@N00.jpg
```

图11-17　我的Flickr个性头像

可以把以上几个步骤全部整合到服务器端的代理脚本中，这个过程与使用Basecampe API时差不多。唯一的不同之处在于，这里可以使用PHP5的simplexml_load_file()方法来取得API的响应：

```
<?php
$apiKey = 'your api key';

switch ($_GET['do']) {

    case 'getFlickrInfo' :
        if(!$_GET['email']) die('');
        $email = urlencode($_GET['email']);
        if(!$xml1 = simplexml_load_file("http://api.flickr.com/ ➥
services/rest/?method=flickr.people.findByEmail&api_key={$apiKey}& ➥
find_email={$email}")) die('false');

$nsid = $xml1->user[0]['nsid'];

        if(!$xml2 = simplexml_load_file("http://api.flickr.com/ ➥
services/rest/?method=flickr.people.getInfo&api_key={$apiKey}& ➥
user_id={$nsid}")) die('false');

        if( $xml2->person['iconserver'] > 0) {
            $icon = "http://static.flickr.com/{$xml2-> ➥
person['iconserver']}/buddyicons/{$nsid}.jpg";
        } else {
            $icon='';
        }
            echo <<<JSON
{
        'username':'{$xml1->user->username}',
        'icon':'{$icon}'
    }
    JSON;

        break;
    }

    ?>
```

这个代理脚本只接受指定相应操作的特殊URL及要查询的电子邮件地址：

```
proxy.php?do=getFlickrInfo&email=jeff%40advanceddomscripting.com
```

www.TopSage.com

然后，脚本会以一个包含相关信息（如果没有找到结果则包含null）的JSON对象作为响应：

```
{
    'username':'Jeffrey Sambells',
    'icon':'http://static.flickr.com/179/buddyicons/24149889@N00.jpg'
}
```

### 3. Flickr例子中的DOM脚本

本例中的DOM脚本部分，只负责取得信息并在注册表单中的User Name及Icon URL字段尚未填写的情况下，预先填充这两个字段：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Flickr Buddyicon</title>
    <link rel="stylesheet" type="text/css"
        href="../../shared/source.css" />
    <link rel="stylesheet" type="text/css"
        href="../chapter.css" />
    <script type="text/javascript"
        src="../../ADS-final-verbose.js"></script>
    <script type="text/javascript" src="flickr.js"></script>
</head>
<body>
<h1>Test Page</h1>
<div id="content">
<form>
    <fieldset>
        <legend>Your Email Address</legend>
        <input type="text" id="email" name="email">
        <p>Your email address will be used to prepopulate the
        username and icon based on your <a href="http://flickr.com">
        Flickr</a> account info.</p>
    </fieldset>
    <fieldset>
        <legend>Additional Settings</legend>
        <div>
            <label for="username">User Name</label>
            <input type="text" id="username" name="username">
        </div>
        <div>
            <label for="icon">Icon URL</label>
            <input type="text" id="icon" name="icon">
        </div>
    </fieldset>
</form>
</div>
</body>
</html>
```

而脚本只需要在Your Email Address字段调用change事件侦听器时，将用户提供的电子邮件地址发送给服务器端代理，并取得服务器端代理返回的响应信息：

```
ADS.addEvent(window,'load',function() {

    var iconInput = ADS.$('icon');
    var preview = document.createElement('img');
```

www.TopSage.com

```
    preview.alt = 'Icon preview';
    preview.src = iconInput.value;
    ADS.setStyle(preview,{
        width:'48px',
        height:'48px'
    });
    ADS.insertAfter(preview,iconInput);

    function updateInfo(data) {
        var username = ADS.$('username');
        if(data.username && !username.value) {
            username.value = data.username;
        }
        var icon = ADS.$('icon');
        if(data.icon && !iconInput.value) {
            iconInput.value = data.icon;
        }
        preview.src = iconInput.value;
    }

    ADS.addEvent(ADS.$('email'),'change',function(W3CEvent) {
        if(this.value) {
            ADS.ajaxRequest('proxy.php?do=getFlickrInfo&email='
                + escape(this.value),{
                completeListener:function() {
                    eval('updateInfo(' + this.responseText + ')');
                }
            });
        }
    });

    ADS.addEvent(iconInput,'change',function() {
        preview.src = this.value;
    });

});
```

最终，我们得到的表单虽然表面上看很普通，但实际上却有点非同寻常，这一点通过图11-16已经有所体现。如果将本例与第4章中介绍的查询邮政编码的例子结合起来，那么仅通过邮政编码及电子邮件地址，就可以做到预先填写大部分表单字段了。

## 11.4　小结

学习完本章之后，我们了解到通过使用种类繁多的API，可以轻松地增强自己的网站。许多API都提供了辅助显示、取得及操纵信息的强大工具，而更有一些API还提供了管理解决方案。当API只对服务器端有效时，也没有理由不去创建一个服务器端的代理脚本，以便让DOM脚本能够访问这些服务器端API。正如本章例子所展示的，这样一来又为我们提供了更加广泛的可能性。

在使用API时，要时刻紧记以本书其他章节都在强调的同样的渐进增强原则为指导，在可能的时候集成使用标准化的标记，如微格式，来确保网站的可访问性。

现在，将你在本书中所学到的有关最佳实践、JavaScript、事件、样式、库等所有的新知识都熔于一炉，创建一个不仅外观光彩照人，而且在任何情形都能进退自如、运转良好的网站吧。

下一章，将由Aaron Gustafson介绍一个案例学习，涉及以典型的表单选择元素为对象，综合运用CSS、DOM脚本及Prototype框架，实现更高级的功能增强。

www.TopSage.com

# 第12章 案例研究：用DOM设计选择列表

作者：Aaron Gustafson

在过去的几年中，我有点病态地痴迷着select元素。也许“病态”这个词有点过于强烈（会让你想歪了），但我只是想表达自己对选择列表元素进行反复思考的状态，并告诉你如果让我说了算，结果会有什么不同。

我记不清这一切是从什么时候开始的了。也许是从我第一次看到Mac平台上的IE 5（IE5/Mac），在处理包含optgroup元素的select时那种真正优雅的方式（图12-1左图）时开始的。也许是从我深入研究CSS并意识到无法改变其难看的默认边框样式（在多数浏览器中都以嵌入形式存在，如图12-1中图），或者从我厌烦了Safari及Camino（见图12-1右图）中带出来的OS X的本地用户界面（UI）控件时开始的（不论看起来有多漂亮）。

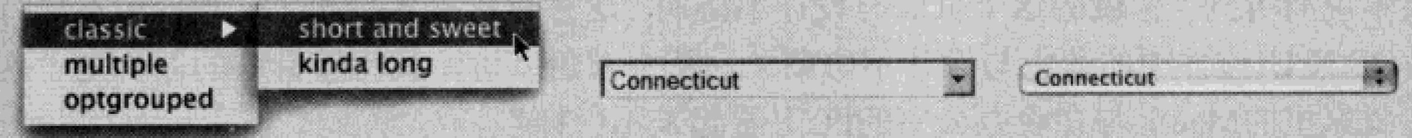



图12-1　从左往右：OS X Panther平台上的IE 5.2中展开的通过optgroup组织的选择列表，Windows XP（经典主题）平台上的IE 6中典型的选择列表和OS X Panther平台上的Safari 1.3中基于控件的本地UI选择列表

但无论如何，我的痴迷病已经发作了。因此在过去的几年中，我不断搜索处理select的最佳方案，并按照自己的（也可能是你的）意愿对各种方案进行改造。本案例学习中出现的脚本，展示了我最新的研究成果，而且也会为你提供根据自己的设想提升select可操作性的一种思路。

## 12.1 经典的感觉

从图12-1可以看出，其中展示的经典的select元素确确实实相当令人讨厌。另外，其表单外观在不同浏览器间也很不一致，尤其是在基于Mac平台的各种浏览器中，这种不一致性更加突出。图12-2汇总了在各种浏览器中标准的select元素的外观（包括展开和闭合两种状态）。

www.TopSage.com

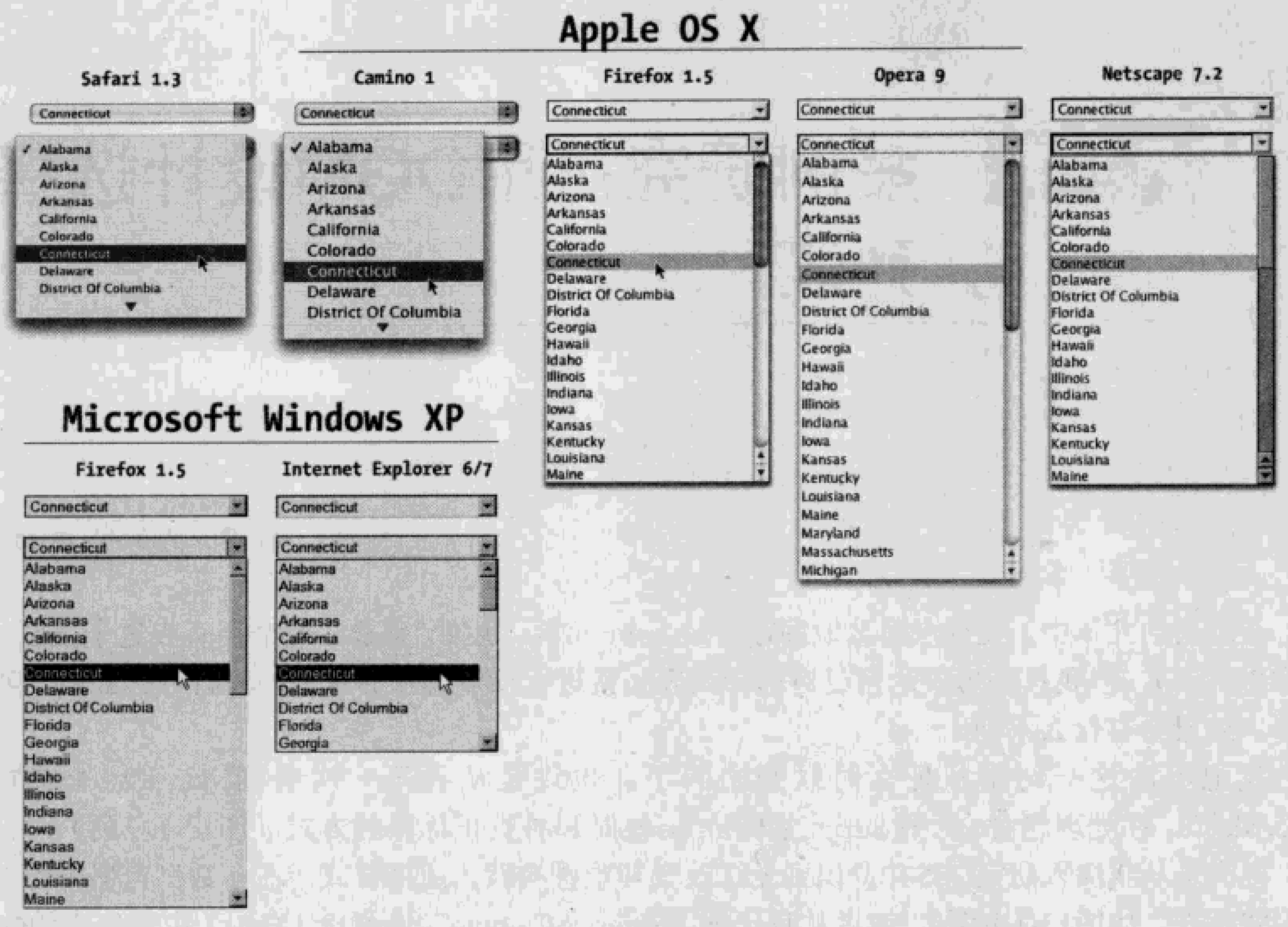



图12-2　Mac和Windows平台的浏览器中默认的选择列表元素的外观

很难说清听到“你做的这个页面与设计不协调”这种评价有多少次了。我也记不清有多少次被指责OS X UI控件中的蓝色与设计不协调了。从个人角度讲，我比较喜欢的评价是“这个设计方案在我家里的Mac上与在我办公室的PC上看起来不一样”。

某些人之所以信奉系统为表单提供的UI控件，是因为用户对它们熟悉。在某种层次上，我同意，但也必须承认拥有统一的、可样式化的选择列表控件的想法会给人一种温馨的感觉。而且，希望摆脱前述那些情景对话的想法也有利我更加理智地思考问题。

## 12.2　构建更好的选择列表

那要怎么做呢？我们要创建一个select的代用品，使其在可应用样式的条件下，其他方面与真正的select元素一样。换句话说，我们需要一个人造的select。

我可以说构建一个人造的select并不难，但究竟又是什么值得我们如此努力呢？说实话，虽然整个过程相当复杂，但将任务分成容易处理的几块内容，会使其变得容易一些。首先，我们分析一下经典select的标记构成，以下面选择美国州的列表为例：

```
<select name="state">
  <option value="AL">Alabama</option>
  <option value="AK">Alaska</option>
```

www.TopSage.com

```
    <option value="AZ">Arizona</option>
    <option value="AR">Arkansas</option>
    <option value="CA">California</option>
    <option value="CO">Colorado</option>
    <option value="CT" selected="selected">Connecticut</option>
    ...
</select>
```

这些代码很简单。但现在我们要问，这个选择列表的本质特征是什么？

- 它包含一些option元素。
- 当列表闭合时
  - 它会显示选中的或者说默认的option。
  - 单击显示的option可以展开列表。
- 当列表展开时
  - 会显示全部（如果列表很长，也可能是部分）option。
  - 每个option都拥有鼠标悬停状态。
  - 会以某种方式标出当前选中的option。
  - 单击某个option会将相应的值设置为select的值并闭合select。
  - 单击下拉列表的外部会导致列表闭合。
  - 单击显示的option（即通过单击它打开列表的那个）也会导致列表闭合。

还不错，并没有想象中那样可怕。我们来看一看要重新创建其中某些特征都需要做些什么，就从第一个任务开始。

包含一些option元素？嗯，听起来很有点要构建列表的味道。好吧，创建人造select的第一个任务就是构建列表。可能会有人提出异议说有序列表更好一些，但这里使用的却是一个无序列表，因为无序列表会给我们更多自由发挥的空间，况且有没有数字都无所谓：

```
<ul class="faux-select">
    <li>Alabama</li>
    <li>Alaska</li>
    <li>Arizona</li>
    <li>Arkansas</li>
    <li>California</li>
    <li>Colorado</li>
    <li>Connecticut</li>
    ……省略的代码……
</ul>
```

好，到现在为止仍然很简单。接下来呢？

当列表闭合时，它会显示选中的或者默认的option？要显示一个option。可以通过为其中一个<li>元素指定“selected”类来实现这一点：

```
<li class="selected">Connecticut</li>
```

但是，如果在单击展开人造select时它也必须同时出现在下拉列表中，那么这样做就不方便（到现在为止，还没有让一个元素同时出现在两个位置上的可靠方式）。更好主意是将选中的

www.TopSage.com

option（我们只把它看成列表的值）放在它自己的元素（例如p）中，然后将这个元素与整个ul放到同一个容器元素中，比如一个div：

```
<div class="faux-container">
  <p class="faux-value">Connecticut</p>
  <ul class="faux-select">
    <li>Alabama</li>
    <li>Alaska</li>
    <li>Arizona</li>
    <li>Arkansas</li>
    <li>California</li>
    <li>Colorado</li>
    <li>Connecticut</li>
    ……省略的代码……
  </ul>
</div>
```

这样，我们人造的select元素就具备了适当的语义化标记结构，那我们就继续编写JavaScript代码将它构建起来。

## 12.3　策略？我们不需要臭哄哄的策略……

等等，也许需要。

为了让脚本更具灵活性（也为了给我们的朋友留下好印象），我们要使用第2章中讨论的面向对象编程的方法，并利用第9章介绍的某些JavaScript库来构建这个人造select。但要事为先：首先要明确脚本如何运行。

以下是要经过的几个步骤的概述：

(1) 确定脚本是否应该运行。

(2) 搜索页面中的所有select元素。

(3) 循环遍历这些select元素，依次为它们构建人造的select：

　a. 收集所有option。

　b. 将每个option都指定为一个列表项。

　c. 将选中的option（默认情况下是第一个）指定为列表的值。

### 12.3.1　相关的文件

为了方便起见，本书chapter12/faux-select-start源文件夹中提供了下列起点文件。

- /test.html
- /main.css
- /faux-select.css
- /faux-select.js
- /prototype.js
- /mod-moo.fx.js

其中的测试文件（test.html）是我们要操纵的HTML文件。由于脚本会修改select元素，

www.TopSage.com

所以该文件中也包含了一个美国的州选择列表及一些form标记。

打开这个文件，你会注意到其中使用了两个库：

```
<script type="text/javascript" src="prototype.js"></script>
<script type="text/javascript" src="mod-moo.fx.js"></script>
```

使用Prototype库（http://prototypejs.org）是为了让创建类、操纵属性和管理事件更轻松一些。而为了添加效果，这里使用的是基于Prototype的Moo.fx库（http://moofx.mad4milk.net）的轻微修改版。

> 要了解与Prototype和Moo.fx库有关的更多内容，请参见第9章和第10章。

页面中包含的main.css文件提供了页面布局（排版、背景等）的一些基本样式。它的唯一目的就是在对人造的select应用样式时，提供一种环境。稍后我们会向这个文件中添加一些样式，以便对人造select元素的外观进行修饰，但用于驱动人造select的样式将由faux-select.css负责提供。

如果现在打开test.html文件，将会看到如图12-3所示的结果。

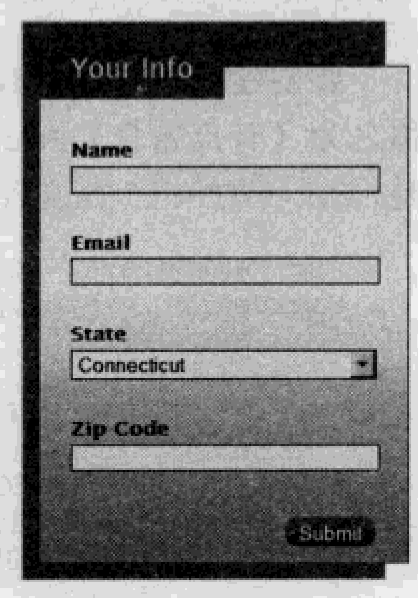



图12-3 在Firefox中查看test.html页面的结果

我们后续的开发工作量将主要集中在两个文件：faux-select.js和faux-select.css中。

## 12.3.2 FauxSelect 对象

为了维护基本的秩序（也为了获得规模效益），需要在faux-select.js中创建两个对象。第一个对象作为乐队的指挥（Conductor）——给它这么个头衔是因为它要像乐队指挥一样，负责协调对select元素的替换。这个对象负责维护页面上人造select的列表（即数组），并通过其initialize()方法来触发构建人造select的过程。FauxSelectConductor对象同时也是与所有人造select有关的几个属性的保管人，它要替我们保管一些DOM节点，以便在需要时复制相应的节点（毕竟，复制节点要比每次都创建新节点来得更经济）。知道了这些之后，就容易开展工作了。下面就是faux-select.js文件中包含的FauxSelectConductor对象的结构，以及有关下一步该做什么的注释：

```
var FauxSelectConductor = {
  // 保存所有人造SELECT的列表
  list: [],
  // 针对人造SELECT的CSS样式表
  cssFile: 'faux-select.css',
  // 人造SELECT所在的层次
  zindex: 10000,
  // 下拉列表默认的最大长度
  maxHeight: 300,
  // BODY的高度
```

www.TopSage.com

```
    bodyHeight: 0,
    // 用于复制的“模型”元素
    elements: { li: document.createElement( 'li' ),
                div: document.createElement( 'div' ),
                p: document.createElement( 'p' ),
                ul: document.createElement( 'ul' ) },
    initialize: function( params ){
      // 收集所有params

      // 设置主体高度

      // 添加标准的CSS文件

      // 收集SELECT
        // 循环遍历集合
        // 如果SELECT没有ID则为其添加一个
        // 将id传递给FauxSelect
    }
  };
```

要创建的第2个对象，是与页面中每个人造select关联的对象（维护各自的事件等）。我们称之为（有点露骨）FauxSelect，并且通过Prototype库的Class对象来完成对象创建。下面就是faux-select.js文件中这个对象的基本结构：

```
  var FauxSelect = Class.create();
  FauxSelect.prototype = {
    // SELECT的ID
    id: false,
      // 原始的SELECT元素
      select: false,
      // 包含人造SELECT的DIV
      container: false,
      // 人造SELECT的UL
      faux: false,
      // 人造SELECT的P（作为被选中的值）
      value: false,
      // 人造SELECT的closer DIV
      closer: false,
      // SELECT的类型（standard、multiple或optgrouped）
      type: 'standard',
      // 保持人造SELECT闭合吗？修复一个模糊的bug
      preventClose: false,

      initialize: function( id ){
        // 构建人造SELECT并设置事件处理程序
      },

      // --- DOM构建方法
      makeFake: function( node ){
        // 通过它来生成OPTION，但最终的功能更多
      },

      // --- 人造SELECT操作
      open: function(){
        // 展开人造SELECT
      },
```

www.TopSage.com

```
  close: function(){
    // 闭合人造SELECT
  },
  flip: function(){
    // 调换类并执行其他内部处理
  },

  // --- 人造SELECT相关的事件
  clickValue: function(){
    // 值的click事件处理程序
  },
  clickUL: function() {
    /* 当在UL内部单击时会怎么样；
       用于解决浏览器的bug */
  },
  clickLI: function(){
    // 人造OPTION的click事件处理程序
  },
  mouseoverLI: function(){
    // 人造OPTION的mouseover事件处理程序
  },
  mouseoutLI: function(){
    // 人造OPTION的mouseout事件处理程序
  },

  // --- 人造SELECT的内部事务处理
  selectLI: function(){
    // 更新真正的SELECT并选择一个人造OPTION
  },
  deselectLI: function(){
    // 删除一个人造OPTION
  },

  // --- 真正的SELECT事件
  focus: function(){
    // SELECT的focus事件处理程序
  },
  blur: function(){
    // SELECT的blur事件处理程序
  },
  updateFaux: function(){
    /* 根据真正的SELECT更新人造的SELECT
      （用于键盘事件） */
  }
};
```

由此可见，FauxSelect对象会对某些信息保持跟踪，比如要替换的select的id（保存在FauxSelect.id中），select自身的DOM引用（FauxSelect.select）及要创建的人造select的DOM引用（FauxSelect.faux）等。该对象还包含了很多用于在人造select中负责处理交互的方法，后面我们会逐个介绍。我们先来看看FauxSelectConductor对象。

### 12.3.3 开始创建人造 select 元素

在建立了两个对象的框架之后，就可以编码了。首先，需要基于页面的载入事件完成初始化。

www.TopSage.com

对对象而言，将使用Prototype的`Event.observe()`方法。在`faux-select.js`文件中，添加下列代码：

```
// 如果Prototype、select及必要的DOM方法有效
if( typeof( Prototype ) != 'undefined' &&
    typeof( Fx ) != 'undefined' &&
    document.getElementById &&
    document.getElementsByTagName &&
    document.createElement &&
    document.getElementsByTagName( 'select' ) ){
    Event.observe( window, 'load', function(){
      FauxSelectConductor.initialize();
    }, false );
  }
```

以上代码没有将检查方法的操作放到`FauxSelectConductor`对象的`initialize()`方法中，只是将这个方法包装在了对`Event.observe()`方法的调用中。这样做的原因是，如果必要的方法无效，就不必再运行`initialize()`方法了。而且，在`if`语句的条件中也包含了对Prototype（`typeof(Prototype) != 'undefined'`）和Moo.fx（`typeof(Fx) != 'undefined'`）的对象检测代码。因为这两个库中任何一个库有问题，都会带来严重的问题，所以如果它们不存在干脆早点（默默地）结束运行。

这里的`Event.observe()`与第1章的`ADS.addEvent()`方法非常相似。这里，通过`Event.observe()`设置了在页面载入后要运行`FauxSelectConductor.initialize()`方法。

此外，我们是在一个匿名函数中完成了对`FauxSelectConductor`对象的初始化。这样做与JavaScript对`this`的处理方式有关。原因是：如果我们将`FauxSelectConductor.initialize`方法像下面这样指定给`window`的`load`事件：

```
Event.observe( window, 'load', FauxSelectConductor.initialize, false );
```

那么，`FauxSelectConductor.initialize()`中的`this`关键字将引用`window`对象，而不再是`FauxSelectConductor`。

为了让`this`引用正确的对象，通过把匿名函数指定为`window`对象的方法，保证了`initialize()`方法仍然处于`FauxSelectConductor`对象的作用域之内。这样，在`FauxSelectConductor.initialize()`方法内部，每当要引用`FauxSelectConductor`对象时就可以通过`this`来代替了——当然，也有一些限制，这一点稍后会介绍。

> 实际上，稍微激进一些的脚本都会考虑到在DOM有效后立即触发`FauxSelectConductor.initalize()`，而不是等到所有追加的项（如嵌入在页面中的图像）都载入到浏览器中之后再行触发。Firefox和Opera都支持非标准的`DOMContentLoaded`事件，但IE等其他浏览器则不支持这个事件。不过可以通过一些技巧来实现这一功能。jQuery（http://jquery.com）及其他一些库都已经支持这一功能，但在本文写作时，Prototype已发布的版本中还不包含这个功能。如果你坚持一客不烦二主，希望继续使用Prototype，那么Dan Webb的LowPro（http://svn.danwebb.net/external/lowpro/tags/rel-0.4/dist/）恰好可以弥补这一不足。

www.TopSage.com

这样，我们就拥有了一个当页面载入完成后随时可以运行的，名为FauxSelectConductor的对象字面量及它的initialize()方法。

### 12.3.4 查找 select 元素

根据前面的To-Do表，接下来需要收集页面上的所有select元素。而使用document.getElementsByTagName()方法就能轻而易举地做到这一点：

```
var FauxSelectConductor = {
  ……省略的代码……
  initialize: function(){
    ……省略的代码……
    // 收集SELECT
    var selects = $A( document.getElementsByTagName( 'select' ) );
    ……省略的代码……
  }
};
```

如果只想通过前面的脚本找到几个预定义的select元素，也可以将相应的代码改为：

```
var selects = $$( 'select.faux-me' );
```

其中使用的Prototype的$$()方法，用于通过CSS选择符来查找元素。这样，任何带有faux-me类的select都将纳入被选择之列。

> Prototype的$$()方法对于某些简单的操作（例如替代document.getElementsByTagName()）而言有点过份了，事实上原生的JavaScript方法始终都要快一些，因为原生的方法……毕竟是原生的。然而，当没有可用的原生方法时，使用$$()就有必要了。例如通过CSS选择符或类名选择元素，而这两种操作都没有原生的JavaScript方法。

现在，需要循环遍历select元素的列表（此时只有一个），并为每个select元素构建相应的人造select。为此，需要为收集到的每个select元素实例化一个新的FauxSelect对象，同时传递select的id值作为参数。接着，通过FauxSelect对象的initialize()方法，为人造select元素构建相应的标记（别忘了Prototype创建的类在实例化时都会自动调用initialize()方法）。这些说起来好像很乱，但通过在FauxSelectConductor.initialize()方法中添加如下代码之后，一切都会清晰起来：

```
var FauxSelectConductor = {
  ……省略的代码……
  initialize: function(){
    ……省略的代码……
    // 收集SELECT
    var selects = $A( document.getElementsByTagName( 'select' ) );
    // 循环遍历集合
    selects.each( function( item, i ){
      // 取得id
      var id = item.getAttribute( 'id' );
      // 如果SELECT没有ID则为其添加一个
```

www.TopSage.com

```
        if( id === false ){
          id = 'auto-ided-select-' + i;
          item.setAttribute( 'id', id );
        }
        // 将id传递给FauxSelect
        this.list[id] = new FauxSelect( id );
      }.bind( this ) );
      ……省略的代码……
    }
  };
```

> 在编程领域，任何问题都可以通过多种方式来解决。此处使用Prototype的枚举方法each()来完成了简单的循环操作。这只是Prototype可以令代码更易读的诸多例子之一。但是，如果你觉得使用传统的方法更舒服，大可尊从自己的意愿。

这里使用了each()方法对select元素的集合进行了循环遍历（很新奇的方式），以便对其中每个select元素进行操作。然后，检查了每个select元素是否有id属性（如果在HTML代码中将标签元素与其进行了显式的关联，就应该有），如果没有，则为其添加一个不会与页面中其他任何元素冲突的id（即auto-ided-select-i，其中i为循环中该select元素的对应顺序号）。

> 为了保持this的正确作用域（以便在each()循环内部可以引用this.list），此处使用了Prototype的bind()方法。

**一点内部处理**

在探索FauxSelect对象之前，我们再简单看一看要添加到FauxSelectConductor对象中的其他一些代码。首先，注意一下有关params的注释：

```
var FauxSelectConductor = {
  ……省略的代码……
  initialize: function(){
    // 收集所有params
    ……省略的代码……
  }
};
```

为了保持脚本的灵活性，需要为用户提供相应的手段，以便他们能够根据需要本配置FauxSelectConductor对象。也许你想为此添加更多的配置选项，但在此我只展示一个简单的例子——将下面粗体代码添加到文件中：

```
var FauxSelectConductor = {
  ……省略的代码……
  initialize: function( params ){
    // 收集所有params
    params = params || {};
    if( typeof( params.maxHeight ) != 'undefined' )
      this.maxHeight = params.maxHeight;
    ……省略的代码……
  }
};
```

www.TopSage.com

这样，当调用initialize()方法时，就具备了向对象中传递一或多个参数的能力。而这里还只是让用户能够调整人造select元素展开时的最大高度。如果像下面这样来调用initialize()方法：

```
FauxSelectConductor.initialize( { maxHeight: 200 } );
```

就会使FauxSelectConductor.maxHeight等于200像素，而非默认的300像素。同理，也可以在此处加入更多的代码，以便用户能够灵活地配置FauxSelectConductor的其他默认属性，如zindex和cssFile（后面即将用到）。

最后的内部处理工作涉及到bodyHeight属性。虽然稍后才会用到这个属性，但现在可以将它设置为body元素的高度——现在设置：

```
var FauxSelectConductor = {
  ……省略的代码……
  initialize: function( params ){
    ……省略的代码……
    // 设置主体高度
    this.bodyHeight = $(
      document.getElementsByTagName( 'body' )[0]
    ).getHeight() + 'px';
    ……省略的代码……
  }
};
```

这里，使用Prototype的$()函数先将body元素转换为Prototype对象，然后再使用Prototype添加到Element对象中的getHeight()方法取得它的高度值。

### 12.3.5 构建 DOM 元素

大概你还记得在FauxSelectConductor对象的框架中，我们创建了一个名叫elements的属性。这个属性的值是一个对象，该对象包含了4个可重用的DOM节点：一个列表项（li），一个分区元素(div)，一个段落(p)和一个无序列表(ul)。这几个DOM节点将在实例化FauxSelect对象以构建相应的标记时反复用到。将下列代码添加到文件中：

```
var FauxSelectConductor = {
  ……省略的代码……
  // 用于复制的“模型”元素
  elements: { li: document.createElement( 'li' ),
              div: document.createElement( 'div' ),
              p: document.createElement( 'p' ),
              ul: document.createElement( 'ul' ) },
  ……省略的代码……
};
```

如果在每次调用FauxSelect对象的initialize()方法时，都使用document.createElement()，就会导致过多的资源占用。同时，也会使脚本的运行速度减慢。而通过cloneNode()来复制每个元素的样本（或模型），则会减少对系统资源的消耗。

www.TopSage.com

既然说到了这里，我们就跳到FauxSelect对象中，展示一下这种做法的优势。在保存了几个属性之后，就可以构建人造select中的元素了。将下列代码添加到文件中：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    // 保存ID
    this.id = id;

    // 保存SELECT节点
    this.select = $( id );

    // -- 构建人造SELECT -- //
    //--- 创建人造SELECT的UL
    this.faux = $( FauxSelectConductor.elements.ul.cloneNode( true ) );
    this.faux.addClassName('faux-select');

    //--- 创建作为容器的DIV
    this.container = $( FauxSelectConductor.elements.div.cloneNode( true ) );
    this.container.addClassName( 'faux-container' );
    this.container.setAttribute( 'id', 'replaces_'+id );

    //--- 创建保存值的P
    this.value = $( FauxSelectConductor.elements.p.cloneNode( true ) );
    this.value.addClassName( 'faux-value' );

    // 将复制的元素添加文档中
    this.container.appendChild( this.value );
    this.container.appendChild( this.faux );
    this.select.parentNode.appendChild( this.container );
  }
};
```

为了提高访问人造select中每个关键元素的效率，分别将每个关键元素指定给了FauxSelect对象的属性：container（容器div）、value（段落）和faux（ul元素）。有了这些引用之后，在FauxSelect对象的其他方法中，就可以通过this.faux这样的语法访问这些元素，而不必再找来找去了。而且，在从FauxSelectConductor中复制元素时，通过$()函数将它们都转换成了Prototype对象。因此，这些引用也就可以调用Prototype的方法了。

此外，还为容器元素指定了一个唯一的id，这使它与原始的select建立了关联：

```
this.fauxcontainer.setAttribute( 'id', 'replaces_' + id );
```

对于本案例中的测试页面而言，相应的id就是replaces_join-state。通过使用这样的命名结构，可以使人造select元素在DOM中保持独立，避免与其他元素发生冲突。而且，还可以在CSS规则或者尚未获得该对象引用的脚本中使用这个id。最后，在调试交付的源代码时，这种命名方式也有助于识别元素。

在第9章介绍Prototype的$()函数时，我们知道它是DOM脚本中频繁调用的document.getElementById()方法的别名函数。在Prototype 1.5中，$()函数也向元素的prototype属性中添加一些方法，以支持方法连缀。此外，前面用到的addClassName()方法是一组方便地

www.TopSage.com

为元素添加类（addClassName()）、移除类（removeClassName()）和测试类的存在性（hasClassName()）的方法。这些方法与你在第5章添加到ADS库中的样式方法类似，只不过它们作为一个方法直接应用到了元素上。因此，在使用这些方法时，只需在参数中指定类名即可。

**1. 创建人造value**

如果现在刷新测试页面，不会发现任何不同，因为前面只不过是添加了一些空标签而已。事实上，前面已经在一个div元素中创建了一个空段落（最终会包含人造select的值）和一个空的无序列表（将包含全部现有的值）。为了填充这些元素，还必须对DOM再进行一些操作。

下面就从最简单的操作开始——获得表示值的文本。之所以说这个操作"简单"，是因为select元素有一个名叫selectedIndex的属性，它可以告诉我们选中了哪个option（如果有）。下面为FauxSelect.initialize()方法添加几行（相对）简单的代码，这个方法就算是完成了：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    //--- 创建保存值的P
    this.value = $( FauxSelectConductor.elements.p.cloneNode( true ) );
    this.value.addClassName( 'faux-value' );
    // 将默认选中选项的文本作为值
    this.value.appendChild( document.createTextNode(
      this.select.getElementsByTagName(
        'option'
      )[this.select.selectedIndex].firstChild.nodeValue
    ) );
    ……省略的代码……
  },
……省略的代码……
};
```

这些代码看起来可没那么简单，下面我们来分析一下。

- this.value.appendChild()用于向value（p）中添加一个子元素。
- document.createTextNode()的作用是根据提供的内容创建一个文本节点。
- this.select.getElementsByTagName('option')是取得select元素中所有的option，不过接着又使用位于方括号中的selectedIndex属性取得了被选择的option。
- firstChild.nodeValue是从option中收集到的内容（例如，option标签之间的文本），也是提供给document.create TextNode()方法的参数。

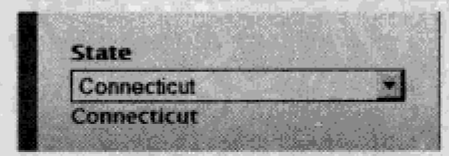


图12-4　将select的值设置成了新人造select的值

到目前为止，一切都还顺利。当再次刷新页面时，你就会发现select元素正下方显示出了一个值，如图12-4所示。

**2. 创建人造option**

现在，需要从原始的select元素中收集所有option，并分别创建对应的人造option。我们是这样做的：

www.TopSage.com

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 收集子元素并使它们可枚举
    var children = $A( this.select.childNodes );
    children.each( function( item, i ){
      if( item.nodeName.toUpperCase() == 'OPTION' ){
        // 构建人造OPTION
        var el = this.makeFake( item );
        // 将其添加到人造SELECT
        this.faux.appendChild( el );
      }
    }.bind( this ) );
    ……省略的代码……
  },
  ……省略的代码……
};
```

我们注意到，这里再次用到了bind()方法，这同样是为了确保this关键字引用FauxSelect对象的正确实例。

为了保证代码更易读，这里还使用了FauxSelect对象的makeFake()方法，并给它传递了需要替换为人造option的option元素：

```
FauxSelect.prototype = {
  ……省略的代码……
  makeFake: function( node ){
    // 复制模型LI
    var el = $( FauxSelectConductor.elements.li.cloneNode( true ) );
    // 保存人造OPTION的值
    el.val = node.getAttribute( 'value' );
    // 设置人造OPTION的文本值
    el.appendChild( document.createTextNode( node.firstChild. ➥
      nodeValue ) );
    // 检查是否被选中
    if( el.val == this.select.value ) el.addClassName( 'selected' );
    return el;①
  },
  ……省略的代码……
};
```

你可能会觉得将这个方法命名为makeFake，而不是更具描述性的createOption有点令人费解。答案很简单：在这个脚本的一个更加高级的版本（我会在本案例学习最后给出一个链接）中，这个方法担负有双重责任，它既要创建人造option，还要创建人造optgroup。为此，需要给它起一个更普通的名字，而makeFake听起来比较合适。

这些代码完成了下列操作：

(1) 复制FauxSelectConductor对象中的列表项模型（li）。

① 原文代码中遗漏了此行。——译者注

www.TopSage.com

(2) 在列表项上创建名为val的属性，并将option的value属性值赋给这个属性。

(3) 为列表项添加与option中相同的文本内容。

(4) 通过测试检查是否应该为这个人造option添加selected类。

(5) 返回新创建的列表项，以便将它添加到人造select中。

当在浏览器中运行以上脚本时，你会发现在真实的select下方，显示出了处于段落中的默认option值和一长串列表项，其中包含着所有州的名称，如图12-5所示。

人造option的每个li元素都有一个val属性，该属性与人造option模拟的真实option的值相同。后面的代码将使用这个属性来更新真实select中的值（以确保选中的值能够随表单一起提交）。

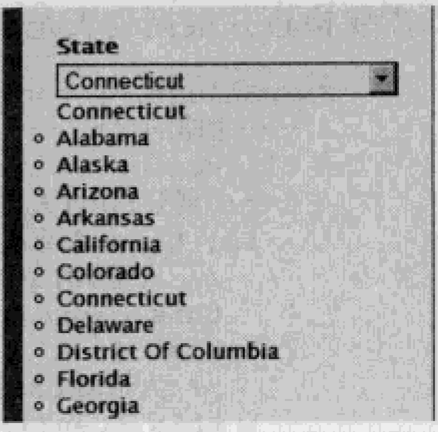



图12-5 州（state）列表中的值都出现了

## 12.4 添加事件——为人造 select 赋予生命

既然已经为人造select生成了相应的元素，并已经将它们添加到文档中，现在该让它们活动起来了。

### 展开、闭合及单击 select

需要模仿3种select的基本事件：展开列表、闭合列表和选择值。但是，与列表交互还存在一些技巧。

- 当单击标准select元素显示的值时，列表会展开；如果再次单击同一个值，列表会闭合。
- 当select展开时单击其中一个option，会触发如下两个事件。
  - 更新select的值。
  - 闭合select。
- 如果在select处于展开状态时，单击其外部任何区域，select会闭合。

听起来好像很有意思，那下面就让我们来实现这些事件！不过，在真正让人造select活动起来之前，还需要经过几个步骤。

首先，要调用FauxSelect的open()和close()方法。由于这两个方法会在FauxSelect的clickValue()方法中被调用，所以必须先在initialize()方法中将clickValue注册为一个观察员（observer）：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // this将作为展开人造SELECT的触发器
    Event.observe( this.value, 'click',
                   this.clickValue.bind( this ), false );
    ……省略的代码……
  }
  ……省略的代码……
};
```

www.TopSage.com

这样，当单击value时，将由clickValue()方法负责处理展开人造select的操作。为此，需要检查并根据列表是否已经处于展开状态，闭合或展开列表，即需要进行如下操作：

```
FauxSelect.prototype = {
  ……省略的代码……
  clickValue: function(){
    // 展开/闭合人造select
    if( this.faux.hasClassName( this.type + '-open' ) ){
      // 人造SELECT是展开的
      this.close();
    } else {
      // 人造SELECT是闭合的
      this.open();
    }
  },
  ……省略的代码……
};
```

嗯，这里都做了些什么？首先，基于人造select是否展开来调整value元素的行为，即如果是打开的则将其闭合，如果是闭合的则将其展开。也就是说，需要跟踪人造select的状态。由于人造select的展开和闭合状态直接关系到它的视觉外观，因而，为了知晓它的外观如何，我们要在它闭合时为它添加close类，而在它展开时为它添加open类。

> 实际上，类名也不是open，而是type-open。其中type属性的默认值是standard（对于标准的select而言）。对于目前的测试页面来说，这个类名就是standard-open。在本章后面，我们会修改type属性以表示在人造select中可以滚动，届时也是通过类名进行跟踪。同样，在更高级的版本中，这个类名也可能包含optgroup或multiple。由于使用类名作为标识，也使得添加样式更加方便。

然后，编写FauxSelect的open()和close()方法，并对类名进行修改：

```
FauxSelect.prototype = {
  ……省略的代码……
  open: function(){
    /* 通过移除closed类并添加opening类
       来表示人造select的开合状态 */
    this.faux.removeClassName( 'closed' );
    this.faux.addClassName( 'opening' );
  },
  close: function(){
    /* 通过移除open类并添加closing类
       来表示人造select的开合状态 */
    this.faux.removeClassName( this.type + '-open' );
    this.faux.addClassName( 'closing' );
  },
  ……省略的代码……
};
```

这些类名的变化是随着FauxSelect.clickValue()方法（及其他方法）的调用而发生的，该方法会基于类名调用FauxSelect.open()或FauxSelect.close()方法。

www.TopSage.com

检查一下已经完成的步骤，我们会发现整个过程还有所欠缺。虽然在open()及close()方法中已经添加了opening和closing类，但还没有添加open和close类。不过，这已经是最后关头了，而且还有opening和closing类名可以作为提示。

为了让人造select更加特别一些，我们打算以滑动效果展开和闭合列表，而这对于Moo.fx库来说则是小菜一碟。然后，在效果结束时，还要调用FauxSelect的flip()方法，该方法将负责完成状态的变化并设置其余的类名。然而，问题在于现在没来得及对人造select进行任何外观的美化，所以马上添加效果也不会太理想。因此，稍后我们就开始添加样式。不过现在，先让我们把flip()方法写出来，并直接在open()和close()方法中调用它：

```
FauxSelect.prototype = {
  ……省略的代码……
  open: function(){
    /* 通过移除closed类并添加opening类
       来表示人造select的开合状态 */
    this.faux.removeClassName( 'closed' );
    this.faux.addClassName( 'opening' );
    this.flip( this.faux );
  },
  close: function(){
    /* 通过移除open类并添加closing类
       来表示人造select的开合状态 */
    this.faux.removeClassName( this.type + '-open' );
    this.faux.addClassName( 'closing' );
    this.flip( this.faux );
  },
  ……省略的代码……
  flip: function(){
    // 如果是展开则闭合，否则展开
    if( this.faux.hasClassName( 'opening' ) ){
      // 展开
      this.faux.removeClassName( 'opening' );
      this.faux.addClassName( this.type + '-open' );
    } else {
      // 闭合
      this.faux.removeClassName( 'closing' );
      this.faux.addClassName( 'closed' );
    }
  },
  ……省略的代码……
};
```

当在后面添加效果时，我们还将重新部署对this.flip()方法的调用。不过，下面我们先为人造option添加一些事件处理程序。

**选择一个人造option**

在任何一个人造option上都需要处理click事件（为此，我们创建了FauxSelect.clickLI()方法），不过大多数select的实现还会在鼠标经过某个option时，提供悬停效果。你可能会想到通过CSS来模仿鼠标在option上的悬停效果，但IE 6不支持除锚元素（a）之外其他任何元素的:hover伪类选择符。因此，还需要为mouseover和mouseout事件添加事件处理程序。下面再跳回到FauxSelect.makeFake()方法中，加入下列3个事件：

```
FauxSelect.prototype = {
  ……省略的代码……
  makeFake: function( node ){
    ……省略的代码……
    // 设置事件处理程序
    Event.observe( el, 'click', this.clickLI.bind( this ), false );
    // click
    Event.observe( el, 'mouseover', this.mouseoverLI, false );
    // mouseover
    Event.observe( el, 'mouseout', this.mouseoutLI, false );
    // mouseout
    return el;
  },
  ……省略的代码……
};
```

接下来，我们要在mouseoverLI()和mouseoutLI()方法中添加悬停效果，这个过程超级简单：

```
FauxSelect.prototype = {
  ……省略的代码……
  mouseoverLI: function( e ){
    Event.element(e).addClassName( 'hover' );
  },
  mouseoutLI: function( e ){
    Event.element(e).removeClassName( 'hover' );
  },
  ……省略的代码……
};
```

这里使用了Prototype的类操作方法，为目标元素添加或移除hover类，而取得目标元素的方法是Prototype的Event.element()——一个简明扼要的方法。现在，再看看FauxSelect.clickLI()方法，恰巧，它也使用了Event.element()。

我们知道，当clickLI()被调用时，需要取消对当前选定的人造option的选择，而选定被单击的人造option，同时还要触发人造select闭合。要完成这一系列操作，要下面这样，分别调用3个不同的方法deselectLI()、selectLI()和close()：

```
FauxSelect.prototype = {
  ……省略的代码……
  clickLI: function( e ){
    var el = Event.element( e );
    // 正常的SELECT行为
    var fOpts = $$( '#replaces_' + this.id + ' li' );
        this.deselectLI( fOpts[this.select.selectedIndex] );
    this.selectLI( el );
    this.close();
  },
  ……省略的代码……
};
```

FauxSelect.clickLI()方法中的所有代码都非常直观，只有fOpts稍微费解。在该方法的第3行，使用了$$()来收集所有以replaces_something（其中something是真实select的id）作为标识的元素中包含的所有列表项。由于所得的结果是人造option的集合，所以将它命名为

www.TopSage.com

fOpts。然后，该方法取消了与真实select的selectedIndex对应的人造option的选定。虽然乍一看有点吓人，但分析过来却很简单。

下面，只需向selectLI()和deselectLI()方法中填充一些必要的逻辑就可以了：

```
FauxSelect.prototype = {
  ……省略的代码……
  selectLI: function( el ){
    // “选定” 人造OPTION
    $( el ).addClassName( 'selected' );
    // 设置真实SELECT的value
    this.select.value = el.val;
    // 设置人造SELECT的“value”
    this.value.firstChild.nodeValue = el.firstChild.nodeValue;
  },
  deselectLI: function( el ){
    el.removeClassName( 'selected' );
  },
  ……省略的代码……
};
```

selectLI()方法为被单击的li元素添加了selected类，将value中显示的文本（this.valule.firstChild.nodeValue）替换为被单击的li元素的文本（el.firstChild.nodeValue)，又将真实select的值设置为被单击的li元素的val属性值。deselectLI()更简单，它只负责移除selected类。

在为人造option添加了click事件之后，可以来看看屏幕中仿真的交互性效果了。刷新浏览器中的页面，单击人造select中的列表项，等待着人造select的值和真实select的值都发生变化的美妙时刻（噢……哈……）

在真正为自己而感到高兴之前，别忘了还要对人造的select添加样式。毕竟，这不正是本章的核心所在吗？

## 12.5 让表单绽放光彩

下面我们就着手为人造select添加样式。为了保持其灵活性，我们要用一个样式表（faux-select.css）来保存支持人造select运行所必需的核心样式。然后，再向主样式表（恰当的命名为了main.css）中添加一些样式，以便人造select与表单环境及页面风格融为一体。

最先要决定的一件事，就是所有环节如何协调共存。从本质上看，我们要操纵的元素有3个：div.faux-container、p.faux-value和ul.faux-select。为了模仿真实select的外观，需要将段落浮动到无序列表之上，可缩放的无序列表也不能影响页面上的其他元素。为此，需要使用绝对定位技术，而要实现绝对定位，则需要建立一个包含块。在开始编写样式之前，我们先来看看图12-6，以便明确人造select应该如何布局。

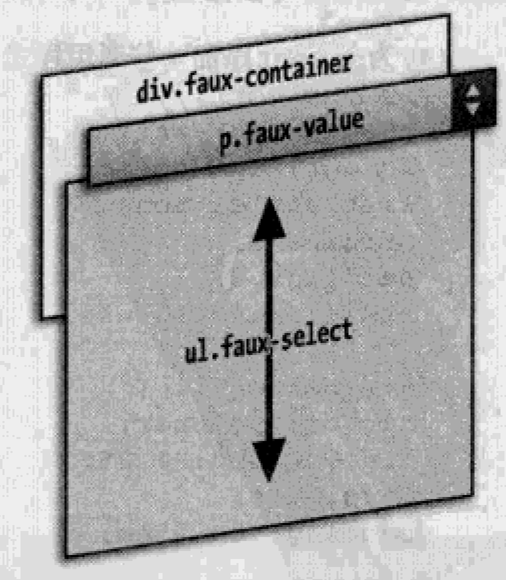



图12-6 人造select元素的布局

好，首先要建立一个包含块，以确保人造select中的组件

www.TopSage.com

在页面上飘浮起来[①]。在faux-select.css文件中，为容器添加一条规则：

```
/* 定位容器 */
.faux-container {
  position: relative;
}
```

然后，对容器中的段落及无序列表采取绝对定位：

```
/* 定位容器中的元素 */
.faux-select {
    position: absolute;
}
.faux-value {
    cursor: pointer;
    position: absolute;
    top: 0;
}
```

注意，这里没有为任何一个元素设置左(left)或右(right)偏移属性，对于ul.faux-select而言，甚至没有设置上（top）或下（bottom）偏移。而这正是构建布局的关键。

> 细心的读者也一定会注意到cursor:pointer这条规则，有了它，当鼠标悬停在人造value元素上时，就会出现人们熟悉的小手图标了。

将元素设置为position:absolute，会将它从文档的正常流中完全移出（也就是说它不会再占用页面空间），但如果不给这个元素设置任何偏移属性，那么这个元素就会像没有对它定位一样处于默认的位置上——这样就可以将绝对定位的元素保持在静态定位条件下的同一位置上。虽说这只是使用CSS布局过程中经常被忽略的技巧之一，但在目前这种情况下恰好需要这个技巧。

下面，我们再调整其他CSS属性，以便让人造select具有默认的样式。虽然稍后会以main.css中的自定义样式覆盖下面这些样式，但像这样为每个组件都设置一些基础的样式总是一件好事。这样一来就为别人定义自己的样式提供了起点和基础。将下列新增的样式规则添加到faux-select.css文件中：

```
/* 为select添加一些通用样式 */
.faux-select {
  background: #fff;
  border: 1px solid;
  list-style: none;
  margin: 0;
  padding: 0;
  position: absolute;
}
.faux-select li {
  cursor: pointer;
  line-height: 1.25;
  margin: 0;
  padding: 0 .25em;
```

① 即脱离正常页面元素流。——译者注

www.TopSage.com

```
    white-space: nowrap;
  }
  .faux-value {
    background: #fff;
    border: 1px solid;
    cursor: pointer;
    position: absolute;
    top: 0;
  }
```

这样，就有了对人造select元素添加样式的基点。不过，由于尚未把样式表链接到页面中，目前还看不到添加样式后的效果。此时，与其在使用脚本的每个页面中都添加对faux-select.css的链接，不如通过FauxSelectConductor对象的初始化方法，将其添加到页面中。为此，需要创建一个link元素，在将它添加到页面的头部区域中之前，先对相应的细节进行设置。为了保证每项操作都有条不紊（而且易于更新），我们在FauxSelectConductor的cssFile属性中使用了路径：

```
var FauxSelectConductor = {
  ……省略的代码……
  // 针对人造SELECT的CSS样式
  cssFile: 'faux-select.css',
  ……省略的代码……
  initialize: function( params ){
    ……省略的代码……
    // 添加标准的CSS文件
    var css = document.createElement('link');
        css.setAttribute('rel', 'stylesheet');
        css.setAttribute('type', 'text/css');
        css.setAttribute('href', this.cssFile);
    document.getElementsByTagName( 'head' )[0].appendChild( css );
    ……省略的代码……
  },
  ……省略的代码……
};
```

> 在定义指向CSS文件的URL时要格外注意。如果在一个页面URL层次多变的动态网站中，最好使用以反斜杠（/）开头的绝对URL，例如/path/to/faux-select.css。或者，通过服务器端的重写规则来捕获对文件的请求，并对请求路径进行适当的重定向。

当CSS文件在页面中载入后，再刷新浏览器就应该看到如图12-7所示的结果。

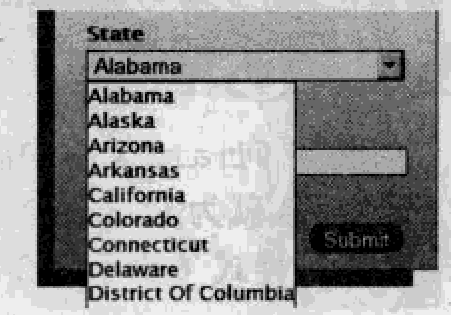


图12-7 在Windows XP平台上的Firefox 2中看到的表单布局美化效果

接下来需要面对的一个问题是下拉列表的宽度，毕竟我们不希望看到一个可变宽度的。但它应该是多宽呢？将它设置为与被替换的select一样宽是个不错的主意。就这么做吧。为了将人造select的宽度设置为与原始的select一样宽，可以通过脚本使用Prototype的getWidth()方法（该方法部分依赖于元素的offsetWidth属性）取得的值自动设置——不过像下面这样也可

www.TopSage.com

能存在问题：

```
this.faux.style.width = this.select.getWidth() + 'px';
```

> 当操纵DOM时，要确保在通过元素的固有属性（例如offsetWidth）或Prototype的getWidth()方法访问其尺寸值时，先把该元素添加到文档中。因为在将元素移交给页面之前，它还不具有尺寸。具体到这里来说，我们要取得的是已经移交给页面的select的尺寸，而这些尺寸都已经存在了。

通过getWidth()方法取得的宽度值中也包含了元素内边距和边框的宽度。为此，如果想为人造select定义自己的内边距和边框，则必须通过更新FauxSelect对象从宽度值中去除这些部分：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 创建人造SELECT的UL
    this.faux = $( FauxSelectConductor.elements.ul.cloneNode( true ) );
    this.faux.addClassName('faux-select');
    /* 基于现有的SELECT设置width
        （减掉边框及内边距宽度） */
    this.faux.style.width = (
      parseInt( this.select.getWidth() )
      // 减掉边框
      -parseInt( this.select.getStyle( 'border-left-width' ) )
      -parseInt( this.select.getStyle( 'border-right-width' ) )
      // 减掉内边距
      -parseInt( this.select.getStyle( 'padding-left' ) )
      -parseInt( this.select.getStyle( 'padding-right' ) )
    ) + 'px';
    ……省略的代码……
  },
  ……省略的代码……
};
```

> Prototype的getStyle()方法返回元素的计算样式，而不仅仅是内置的样式，这一点在第5章也曾讨论过。由于返回的值中包含单位（例如px），所以需要使用JavaScript的parseInt()方法取得其中的数字值。此时唯一要注意的是，如果不小心混合搭配了不同的单位，例如对内边距使用em而对边框使用px，会造成意想不到的结果。

现在，再向main.css中添加一些样式，以便使人造select中的组件能够与整个页面外观更加协调。下面就为p.faux-value、ul.faux-select和添加了.selected或.hover类的人造option添加样式（参考图12-8）：

```
/* =人造SELECT组件的样式 */
p.faux-value {
  background: #eee;
  border: 1px solid;
```

www.TopSage.com

```
    line-height: 18px;
    width: 175px;
    padding: 0 20px 0 3px;
  }
  ul.faux-select {
    background: #c4c4c0;
  }
  ul.faux-select .hover,
  ul.faux-select .selected {
    background: #006;
    color: #fff;
  }
```

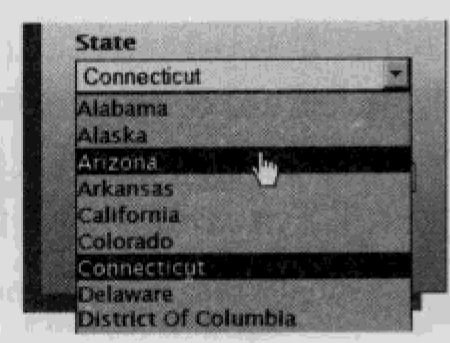



图12-8 人造select中应用了一些CSS，但却出现了重叠的问题

看一看图12-8就会发现我们的工作已经接近尾声了。不过还有一个问题，那就是列表的value被人造select给挡住了。问题的原因是这两个元素都应用了绝对定位，而且列表在标记中位于段落后面，因而在堆叠次序中拥有更高的位置。要解决这个问题，可以在main.css中将ul.faux-select的margin-top设置为等于p.faux-value的line-height，把value提到前面来：

```
  ul.faux-select {
      background: #c4c4c0;
      margin-top: 18px;
  }
  .faux-value {
    background: #fff;
    border: 1px solid;
    cursor: pointer;
    white-space: nowrap;
    position: absolute;
    top: 0;
    z-index: 100;
  }
```

由于faux-select.css是在运行时由脚本添加到文档中的，而且它在文档头部位于main.css之后，所以会在层叠过程中获得更高的权重。为了使main.css中的规则优先适用，需要提高选择符的针对性。而通过为类选择符（.faux-select）添加元素选择符（ul），便可获得足够的针对性，从而使margin-top:18px;胜过在faux-select.css中设置的margin:0;——针对性还真是可爱。

通过图12-9可以看出，目前差不多已经完成了对人造select展开状态的美化（至少完成了基本要素）。

下面，该回过头来为ul.faux-select添加扩展和折叠功能了。

没错，我们也可以自己编写生成效果的脚本，但在这里我们要简单借用Valerio Proietti在Moo.fx库（moofx.mad4milk.net）中已经完成的漂亮的工作成果。

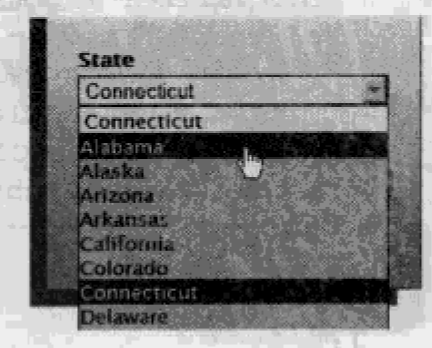



图12-9 现在value元素出现在了人造select元素的上方

虽然要向FauxSelect对象的各个部分中都添加一些代码，但

www.TopSage.com

我们还是从initialize()中的基本设置开始。首先，将Moo.fx的fx.Style()方法创建的对象赋给人造select的一个属性（heightFX），以备后面使用（通过fx.Style()可以轻易实现从一种样式值到另一种样式值之间的渐变效果，而我们要做的仅仅是包含基本的Moo.fx文件）。为使效果平滑流畅，把动画的持续时间设置为350毫秒：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 为人造SELECT设置heightFX效果，使其能够展开
    this.faux.heightFX = new fx.Style(
      this.faux, 'height',
      { duration: 350 }
    );
    ……省略的代码……
  },
  ……省略的代码……
};
```

然后，取得ul.faux-select完整的原始高度。并将其保存为人造select的一个属性，便于之后引用（为了方便，也把折叠后的尺寸（0像素）保存为人造select的一个属性）：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 取得UL打开时应有高度值
    this.faux.openHeight = this.faux.getHeight();
    this.faux.closedHeight = 0;
    ……省略的代码……
  },
  ……省略的代码……
};
```

最后，通过Moo.fx的set()方法将人造select的高度减低为0，从而将其折叠（或闭合）起来。再为人造select添加一个表示其初始状态的类——closed：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 为人造SELECT添加“closed”类
    this.faux.addClassName( 'closed' );
    // 闭合人造SELECT
    this.faux.heightFX.set( this.faux.closedHeight );
    ……省略的代码……
  },
  ……省略的代码……
};
```

取决于前面两处修改在代码中的位置，this.faux可能还没有被添加到文档中，因此不会存在高度。为避免这个问题，通过添加下列代码先使其暂时不可见：

www.TopSage.com

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // --构建人造SELECT --//
    // 创建人造SELECT的UL
    this.faux = $( FauxSelectConductor.elements.ul.cloneNode( true ) );
    this.faux.addClassName('faux-select');
    ……省略的代码……

    // 收集子元素并使它们可枚举
    var children = $A( this.select.childNodes );
    children.each( function( item, i ){
      ……省略的代码……
    }.bind( this ) );

    /* 将人造SELECT添加到SELECT的父元素中，
       但使其不可见，因为目前还不需要让它出现。
       由于需要取得它的高度（后面要用到），
       所以必须把它添加到文档中 */
    this.faux.style.visibility = 'hidden';
    this.select.parentNode.appendChild( this.faux );

    ……省略的代码……

    // 为人造SELECT添加“closed”类
    this.faux.addClassName( 'closed' );
    // 为人造SELECT设置heightFX效果，使其能够展开
    this.faux.heightFX = new fx.Style(
      ……省略的代码……
    );

    // 取得UL打开时应有高度值
    this.faux.openHeight = this.faux.getHeight();
    this.faux.closedHeight = 0;
    // 闭合人造SELECT
    this.faux.heightFX.set( this.faux.closedHeight );

    // 移除人造SELECT并重新添加到容器中
    // this.faux.parentNode.removeChild( this.faux );
    this.container.appendChild( this.faux );
    this.select.parentNode.appendChild( this.container );

    // 显示ul
    this.faux.style.visibility = ''①;
    ……省略的代码……
  },
  ……省略的代码……
};
```

通过将列表的visibility设置为hidden并将其添加文档中，可以在用户看不到该元素的情况下取得它的尺寸信息。为此，只需在该元素有效时将其从文档中移除，然后再将其添加到容器中即可。然后，通过重新设置visibility属性，一切又恢复正常。不过，这样也会带来一个微

① 原文代码中this.faux.style.visibility = null;是错误的。——译者注

www.TopSage.com

妙的问题——列表在折叠之前会闪一下。

哈，看来现在我们还不能全身而退。简单地刷新一下浏览器中的页面，会发现列表是折叠了，但所有人造option仍然可见（尽管没有背景）。为解决这个问题，还需要向faux-select.css中再添加一条规则（粗体部分）：

```
.faux-select {
  background: #fff;
  border: 1px solid;
  list-style: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  position: absolute;
}
```

这样，我们就得到了如图12-10所示的结果。

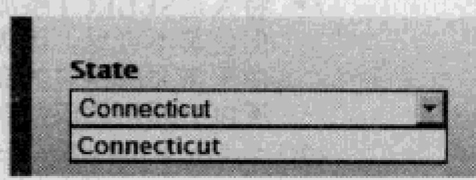



图12-10　人造select的初始状态完成之后的效果

现在再刷新浏览器，就会看到折叠后的人造select了。但是，现在还不能将其展开。为此，要修改open()方法。先删除前面对this.flip()的引用，然后添加下列代码：

```
FauxSelect.prototype = {
  ……省略的代码……
  open: function(){
    ……省略的代码……
    this.flip( this.faux );

    // 如果中途结束先停止效果
    this.faux.heightFX.stop();

    // 通过调用效果展开fauxSelect
    this.faux.heightFX.custom(
      this.faux.getHeight(),
      this.faux.openHeight
    );
  },
  ……省略的代码……
};
```

首先，如果列表正在活动则停止效果（如用户虽然关闭了人造select，但在完全关闭之前又改变了主意）。然后，让效果从当前高度扩展到原始高度。

> stop()不是Moo.fx库的核心方法（至少在本文编写时不是），而这也正是把文件命名为mod-moo.fx.js的原因。其他由本书作者添加的方法可以在mod-moo.fx.js的源代码中找到。

还需要更新close()方法以再次折叠人造select。从本质上来说，这同前面所做的事情一样，只是方向相反：

```
FauxSelect.prototype = {
  ……省略的代码……
  close: function(){
```

www.TopSage.com

```
        ……省略的代码……
        this.flip( this.faux );

        // 如果中途结束先停止效果
        this.faux.heightFX.stop();

        // 通过调用效果闭合fauxSelect
        this.faux.heightFX.custom(
          this.faux.getHeight(),
          this.faux.closedHeight
        );
      },
      ……省略的代码……
    };
```

现在所剩的就是调用flip()方法了。调用这个方法的时机是在效果结束而人造select完全闭合或者完全展开的时候。凑巧的是，Moo.fx在它的效果中提供了一个onComplete选项恰好满足这个需要。因此，只要在FauxSelect.initialize()方法中添加这个选项即可：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 为人造SELECT设置heightFX效果，使其能够展开
    this.faux.heightFX = new fx.Style(
      this.faux, 'height',
      { duration: 350,
        onComplete: this.flip.bind( this ) }
    );
    ……省略的代码……
  },
  ……省略的代码……
};
```

> 别忘了在duration属性后面加上逗号！

现在，刷新浏览器并单击value元素，会看到列表扩展的效果。再次单击，会使列表折叠起来。类似地，由于我们为人造option添加了click事件（在clickLI()中），当它们被单击时也会调用close()方法，所以通过value展开人造select后再单击人造option，不仅会改变人造select及真实select的值，而且还会闭合人造select（太棒了）。

当然，人造select以全高尺寸展开有点大了。还记得吗，FauxSelectConductor有一个名叫maxHeight的属性。可以利用这个属性来指定人造select展开的最大高度，以避免列表展开过长的问题：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 取得UL打开时应有高度值
    this.faux.openHeight = this.faux.getHeight();
    /* 如果人造SELECT中有很多OPTION
```

www.TopSage.com

```
        可能会导致列表展开后过长，因而需要
        基于在FauxSelectConductor中的设置，
        将它缩减到适当高度 */
    if( this.faux.openHeight > FauxSelectConductor.maxHeight ){
      // 将其标注为溢出的人造SELECT
      this.type = 'overflowing';
      // 重设高度值
      this.faux.openHeight = FauxSelectConductor.maxHeight;
    }
    this.faux.closedHeight = 0;
  },
  ……省略的代码……
};
```

此处，又为列表增加了一种新类型——overflowing。所以，可以向faux-select.css文件中添加一些针对该类型的样式：

```
ul.faux-select.overflowing-open {
  overflow: auto;
  overflow-x: hidden;
}
```

通过将overflow属性设置为auto，确保了所有用户都能看到列表旁边的滚动条。而设置CSS 3规范中的属性overflow-x，则可以对现代的浏览器给予更多限制，以便强制它们只显示垂直滚动条（不显示x轴的滚动条）。

> PC平台上的IE（6.0及以上版本）、Firefox和Camino都能很好地支持overflow-x和overflow-y属性。在本文编写时，Safari开发团队的Dave Hyatt已经为该浏览器的Nightly Build提交了支持这两个CSS属性的补丁程序，也就是说这两个属性（很快）就会在该浏览器的下一个官方升级版中得到支持。不过，Opera 9似乎还没有支持这两个属性的迹象。

由于CSS样式添加得还不够，所以现在还需要回到main.css文件中，将人造select打扮得更时髦一些：

```
p.faux-value {
  background: #eee;
  background: url(value.png) top left no-repeat;
  ……省略的代码……
}
```

与此同时，再给人造value元素添加一种焦点的样式，以便它看起来与实际的焦点没有什么两样（与真实的select相比——参见图12-11）。现在立刻动手：

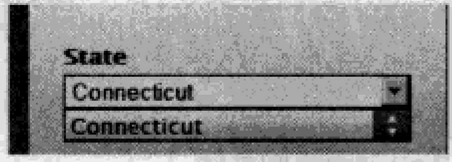



图12-11　人造的select已经完全就绪了

```
    p.faux-value.focused {
      background-image: url(value-focused.png);
}
```

> 本例所用图像都位于项目源文件所在的文件夹中。

在最终的调整之后，现在可以对这个小家伙进行包装了。

www.TopSage.com

## 12.6 行为修正

虽然已经有了功能健全的人造select，但还必须对它进行一番装扮，才能使其真正光彩照人并与真实的select难辨真伪。

**闭合人造select**

我们面临的第一个挑战，是模仿在列表外部单击时闭合select元素的行为。当在常规的select列表之外单击时，列表会闭合，但是如果单击的是其他可单击的元素（例如某个链接），那么这次单击不会激活任何别的事件。这种现象叫做操作闭合，也就是说必须再次单击才能启动光标下方元素的单击事件。

为了在单击列表外部时调用FauxSelect的close()方法，需要为可能被单击的元素指定事件处理程序。然而，为页面上所有元素都指定事件是不可行的，否则事件管理就会变成难以想象的恶梦。

> 如果IE支持事件流的捕获阶段，那这个问题解决起来还容易一些。因为我们只要在文档事件流的捕获阶段注册侦听器，并取消其他事件就可以了。但是，唉，事实完全不是那么回事，所以必须要费点周折。

既能捕获所有鼠标事件，又能阻止其他操作的最简单方式，就是在鼠标指针与文档之间放置一堵很大的屏蔽墙。在这堵墙上，注册一个单击闭合人造select的事件即可。可以在对象初始化时创建这个元素，并将它一直藏到时机成熟——调用FauxSelect.open()方法时——再把它添加到文档并放在所有元素之上。权且就把这个元素称为closer吧。

事实上，FauxSelect中已经预定义了这个元素（FauxSelect.closer），因此我们需要生成这个元素并为它添加事件。然后，当FauxSelect展开时将它添加到文档中，当FauxSelect闭合时再把它删除：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // -- 构建CLOSER -- //
    this.closer = $( FauxSelectConductor.elements.div.cloneNode( true ) );
    this.closer.addClassName( 'closer' );
    /* 我们要使用这个隐藏的DIV，来确保
       在列表外部单击时闭合人造SELECT */
    Event.observe( this.closer, 'click', this.close.➥
      bind( this ), false );
    ……省略的代码……
  },
  ……省略的代码……
  open: function(){
    ……省略的代码……
    // 将closer添加文档中
    document.getElementsByTagName( 'body' )[0].appendChild( this.closer );
  },
```

www.TopSage.com

```
  close: function(){
    ……省略的代码……
    // 移除closer
    this.closer.parentNode.removeChild(this.closer);
  },
  ……省略的代码……
};
```

接着，在faux-select.css中添加一些样式规则，以便closer能将整个文档主体部分覆盖过来：

```
.closer {
  position: absolute;
  top: 0;
  right: 0;
  left: 0;
}
```

在理想情况下，本应使用固定定位并将top、right、bottom和left的值都设置为0，但是，使用固定定位会导致Mac平台上的Firefox和Camino行为古怪（滚动条无缘无故地消失）。而且，IE 6也不支持固定定位。为避免这两种情况，还要使用绝对定位。

让IE 6表现正常并非难事，因为它支持CSS表达式。虽然CSS表达式不是什么标准，但如果仅对IE 6而言（在我们的例子中，使用Tan hack），我认为还是情有可原的。CSS表达式的工作原理几乎与JavaScript相同，因而要将closer的宽度和高度设置为与页面大小一致，可以这样来实现：

```
* html .closer {
  width: expression( document.body.scrollWidth );
  height: expression( document.body.scrollHeight );
}
```

这两个CSS表达式分别将closer元素的width和height值，设置为body元素的scrollWidth和scrollHeight属性值。接下来，需要考虑其他浏览器了。我们还记得，在FauxSelectConductor中还有一个名为bodyHeight的属性，该属性可以在对象初始化时进行设置：

```
var FauxSelectConductor = {
  ……省略的代码……
  initialize: function(){
    // 收集params
    params = params || {};
    if( typeof( params.maxHeight ) != 'undefined' ) this.maxHeight = ➥
      params.maxHeight;
    // 设置BODY的高度
    this.bodyHeight = $( document.getElementsByTagName( 'body' )[0] ). ➥
      getHeight() + 'px';
  ……省略的代码……
  }
};
```

之后，就可以在FauxSelect对象中用该值来设置closer元素的高度了：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
```

www.TopSage.com

```
    // -- 构建CLOSER -- //
    this.closer = $( FauxSelectConductor.elements.div.cloneNode( true ) );
    this.closer.addClassName( 'closer' );
    this.closer.style.height = FauxSelectConductor.bodyHeight;
    ……省略的代码……
  },
  ……省略的代码……
};
```

但是，现在还没有彻底结束。如果为closer的CSS样式中添加一种半透明的背景色，例如粉红色

```
/* 用于测试 */
.closer {
    background: pink;
    opacity: 0.6;
    filter: alpha(opacity=60);
}
```

并测试页面，会发现目前的效果不是我们所期望的。这个元素盖住了一切——也包括人造select（见图12-12）。

图12-12 覆盖所有页面元素是有问题的

处理closer问题的关键，在于确保它位于页面中所有元素的上方，但唯独位于人造select元素下方。听起来好像要把我们熟悉的z-index属性派上用场了。

## 12.6.1 z-index 来救急

巧合的是，在处理这个层次问题的过程中，还可以解决另一个难题——堆叠次序：当在同一个页面中使用多个人造select时，某个下拉列表可能会盖住其他人造select。而且，前面我们

www.TopSage.com

也讨论过，默认的堆叠次序是在标记中居后的定位元素处于居前的定位元素之上。换句话说，若不经过调整，就有可能导致图12-13所示的页面效果。

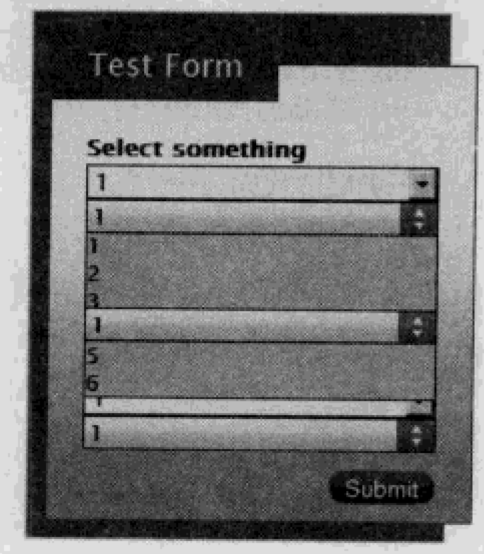



图12-13　这些人造select是默认堆叠次序的受害者。注意，其中第1个人造select的下拉列表被第2个人造select的value搞得不明不白（噢，不公平）

幸好，默认的次序可以通过z-index属性进行调整（即沿z轴方向移动元素）。全面解释与z-index相关的内容（也的确非常之多）超出了本章的范围，但如果你对这个话题很感兴趣，我建议你看看Aleksandar Vaci写的详尽地介绍这个主题的文章http://www.aplus.co.yu/css/z-pos/。

如果你对此感到无所谓，或者只想看到结果，那么我们的解决方案是：首先，设置一个非常高的默认z-index值（巧的是，在FauxSelectConductor中恰好也有这么个属性），以便应用该属性的任何元素都能居于页面上其他元素上方。然后，当展开人造select时，就将它的z-index设置为该值，而将closer的z-index设置为比该值小1的值。

> 将z-index的默认值设置为10000只是随便给出的一个数，只要它能够确保closer最终位于同样设置了z-index属性的其他元素之上即可。如果简单地把这个值设置为0，那么其他脚本就有可能把别的元素定位在closer之上，这样一来，问题就会变得复杂化了。即便如此，其他脚本也可能使用一个更大的值，如果遇到这种问题，你可以再把默认值加大一些。谁的z-index值最大谁会笑到最后。
>
> 如果你是个喜欢冒险的人，那么也可以把这个属性像maxHeight一样，作为一个灵活的属性来处理，即让它在FauxSelectConductor初始化时再获得值。

为应用z-index，我们需要再对open()和close()方法进行一番修改：

```
FauxSelect.prototype = {
  ……省略的代码……
  open: function(){
    ……省略的代码……
    // 设置人造SELECT容器的z-index
    this.container.style.zIndex = FauxSelectConductor.zindex;
    // 为closer设置一个比人造SELECT容器的z-index
    // 小1的z-index值
    this.closer.style.zIndex = FauxSelectConductor.zindex -1;
    // 将closer元素添加到文档中
    document.getElementsByTagName( 'body' )[0].appendChild( this. ➥
      closer );
  },

  close: function(){
    ……省略的代码……
    // 移除closer
    this.closer.parentNode.removeChild(this.closer);
```

www.TopSage.com

```
      // 重置z-index值
      this.container.style.zIndex = this.closer.style.zIndex = null;
    },
    ……省略的代码……
};
```

再刷新浏览器，就会看到如图12-14所示的结果。

下一个挑战，开始吧！

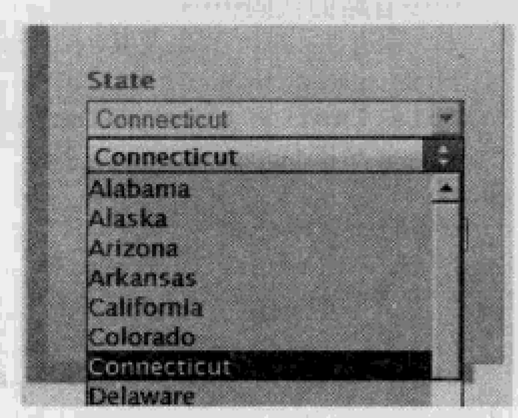



图12-14 由于使用了反置的z-index值，closer元素巧妙地栖身于人造select元素之下，现在一切正常了

### 12.6.2 键盘控制及其他细节

既然人造select与鼠标能够合作愉快，现在该是键盘登场的时候了。事实上，有很多Web用户更偏爱键盘控制，尤其是在填写表单时。例如，我更喜欢在选择了state（州）列表之后，按3次C键来填写Connecticut，而不是展开并滚动表单控件去选择它。而且，多数浏览器都支持select元素的输入查找界面，也就是说填写“CON”就可以选中Connecticut。因此，让用户在操作人造select时也拥有这种选择是非常有意义的。

每种浏览器都提供了不同的处理select元素的键盘控制方式。因此，为了简化代码的编写，最好简单地让浏览器照常运行而不受干扰。如果让真实的select存在于页面中（只是从视图中隐藏起来）我们就可以与它保持沟通，从而不仅让人造select能够更新真实的select，而且也让它可以被真实的select更新。为此，可以在select获得焦点时开始监听键盘事件，然后在真实select元素的值发生变化时，同步更新人造select——小菜一碟。

#### 1. 选择option

对发生在select上面的键盘事件的保持反应，涉及通过FauxSelect.updateFaux()方法处理大部分操作，而为有所区别，我们会在initialize()方法中使用Prototype的bindAsEventListener()方法来指定事件侦听器：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // -- 处理真实SELECT上面的事件 --//
    // 敲击键盘
    this.select.onkeyup = this.updateFaux.bindAsEventListener( this );
  },
  ……省略的代码……
};
```

然后，使用FauxSelect.last属性保持对真实select元素中的selectedIndex进行跟踪，从而确保在发生键盘事件时取消对默认值的选定。所以，首先要在脚本初始化时取得该值，然后当调用selectLI()方法时更新该值：

```
FauxSelect.prototype = {
```

```
  ……省略的代码……
  // 最后选定值的索引
  last: false,
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 保存SELECT节点及其selectedIndex
    this.select = $( id );
    this.last = this.select.selectedIndex;
    ……省略的代码……
  },
  ……省略的代码……
  selectLI: function( el ){
    ……省略的代码……
    // 更新this.last
    this.last = this.select.selectedIndex;
  },
  ……省略的代码……
  updateFaux: function( e ){
    var el = Event.element( e );
    var fOpts = $$( '#replaces_' + this.id + ' li' );
    this.deselectLI( fOpts[this.last] );
    this.selectLI( fOpts[el.selectedIndex] );
  }
};
```

通过FauxSelect.last保存真实select的前一个selectedIndex值，可以省去循环遍历所有人造option寻找需要取消选定哪一个选项的麻烦。

2. 维护焦点

此时此刻，还需要用到很早之前就已经创建的focused类。为此，需要监听select元素上面的focus和blur事件，并相应地触发focus()和blur()方法。这两个方法只负责修改value的类名，以便应用focused类的样式：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // --HANDLE EVENTS ON THE REAL SELECT -- //
    // 获得焦点
    Event.observe( this.select, 'focus', this.focus.bind( this ), false );
    // 失去焦点
    Event.observe( this.select, 'blur', this.blur.bind.( this ), false );
    // 敲击键盘
    this.select.onkeyup = this.updateFaux.bindAsEventListener( this );
  },
  ……省略的代码……
  focus: function(){
    this.value.addClassName( 'focused' );
  },
  blur: function(){
    this.value.removeClassName( 'focused' );
  },
  ……省略的代码……
};
```

www.TopSage.com

图12-15显示了应用focused类之后的外观。

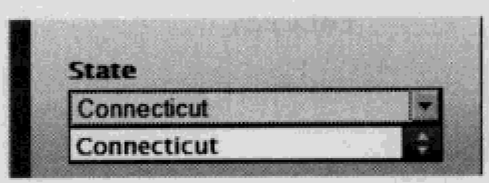



图12-15 人造select能够对发生在真实select上的focus事件进行反应。而为了从视觉上显示这一点，为value元素应用了不同的背景图像（白色）

在将下列代码添加到open()方法后，当人造select的值被单击时，焦点也会切换到真实的select元素上面，从而实现了对键盘和鼠标事件的同步跟踪：

```
FauxSelect.prototype = {
  ……省略的代码……
  open: function(){
    ……省略的代码……
    // 触发SELECT元素的focus事件
    this.select.focus();
  },
  ……省略的代码……
};
```

### 3. 闭合人造select

对于select元素而言，还有另外一些按键组合可以完成特殊的功能。比如，按回车（Enter）或者退出键（Esc）可以闭合下拉列表，因此需要在updateFaux()方法中添加对这两个键码（分别是吉祥数字13和27）的检查：

```
FauxSelect.prototype = {
  ……省略的代码……
  updateFaux: function( e ){
    ……省略的代码……
    // 针对按回车（Enter）或退出（Esc）键退出人造select的情况
    if( this.faux.hasClassName( this.type + '-open' ) &&
        ( e.keyCode == '13' ||
          e.keyCode == '27' ) ){
      this.close();
    }
    ……省略的代码……
  }
  ……省略的代码……
};
```

此外，还需要检查换档键（Alt）与上、下方向键（键码分别为38和40）的组合。在多数浏览器中，这种组合键都会触发获得焦点的select元素展开，或者如果已经展开，将其闭合。为模仿这种行为，还需要向updateFaux()方法中再添加一些代码，以便当换档键（Alt）与上、下方向键同时按下时，调用clickValue()方法（该方法实际上也会完成同样的展开和闭合操作）：

```
FauxSelect.prototype = {
  ……省略的代码……
  updateFaux: function( e ){
    ……省略的代码……
    /* 特殊组合Alt+up与Alt+Down会使人造SELECT
       展开而不会因为每个按键更新value。要确认值
       的改变，必须按下回车（Enter）键 */
    if( e.altKey &&
        ( e.keyCode == '38' ||
          e.keyCode == '40' ) ){
      this.clickValue();
```

www.TopSage.com

```
        return;
      }
      ……省略的代码……
    }
  };
```

虽然看起来为使用键盘的用户做了很多，但实际上使用这些功能只须片刻。不过，仍然有一些交互性问题亟待解决。第一个问题就是当人造select展开时，用户多次按下同一个方向键时怎么办。我们知道，当select展开时，按方向键并不会改变selectIndex，而为了与键盘保持同步，需要在这种情况下更新人造select。所以，向updateFaux()中添加如下代码：

```
FauxSelect.prototype = {
  ……省略的代码……
  updateFaux: function( e ){
    ……省略的代码……
    if( this.faux.hasClassName( this.type + '-open' ) ){
      var fOpt = false;
      // 后退（左或上方向键）。
      if( e.keyCode == '37' ||
          e.keyCode == '38' ){
        if( this.select.selectedIndex > 0 )
          fOpt = fOpts[this.select.selectedIndex - 1];
      }
      // 前进（右或下方向键）
      if( e.keyCode == '39' ||
          e.keyCode == '40' ){
        if( this.select.selectedIndex < fOpts.length )
          fOpt = fOpts[this.select.selectedIndex + 1];
      }
      // 上端（上页或起始键）
      if( e.keyCode == '33' ||
          e.keyCode == '36' ){
        fOpt = fOpts[0];
      }
      // 下端（下页或结束键）
      if( e.keyCode == '34' ||
          e.keyCode == '35' ){
        fOpt = fOpts[fOpts.length-1];
      }
      if( fOpt ){
        this.deselectLI( fOpts[this.select.selectedIndex] );
        this.selectLI( fOpt );
      }
    } else {
      this.deselectLI( fOpts[this.last] );
      this.selectLI( fOpts[el.selectedIndex] );
    }
    ……省略的代码……
  }
};
```

我们注意到，代码在条件检查中排除了非展开状态，因此不会在人造select闭合[①]时触发这

① 原文so it isn't triggered if the faux select is open 疑有误。——译者注

www.TopSage.com

些行为。现在，几乎已经处理了所有键盘事件。最后一个需要解决的交互性问题就是，当用户按制表键（Tab）从select上面移出焦点时，触发相应的blur事件。如果人造select处于展开状态，则需要将其闭合：

```
FauxSelect.prototype = {
  ……省略的代码……
  blur: function(){
    this.value.removeClassName( 'focused' );
    if( this.faux.hasClassName( this.type + '-open' ) ) this.close();
  }
};
```

当然，这也意味着会多次调用close()：当单击显示的值以打开人造select时，焦点会传递给真实的select，而当在展开的下拉列表中单击某个人造option时（调用clickLI()），大多数浏览器都会把焦点从真实select上面移开，这时clickLI()和blur()都会调用close()方法。为避免这个问题，需要在close()方法运行之前作一次检查，以确保不会闭合已经闭合的人造select：

```
FauxSelect.prototype = {
  ……省略的代码……
  close: function(){
    // 检查列表是否展开，如是不是则返回
    if(!this.faux.hasClassName(this.type + '-open')) return;
    ……省略的代码……
  },
  ……省略的代码……
};
```

由此而引发的另外一个问题是，IE会在用户滚动overflowing类型的人造select时，从真实select上面移开焦点，进而导致列表提前折叠。为防止这种现象发生，可以使用FauxSelect对象上的另一个属性——preventClose。下面我们就在FauxSelect对象中的几处地方添加这个属性：

```
FauxSelect.prototype = {
  // 要防止人造SELECT提前闭合？修复失去焦点的bug
  preventClose: false,
  initialize: function( id ){
    ……省略的代码……
    /* 为人造select绑定mousedown事件，以说明用户
       何时单击了列表。由于某些浏览器会在用户滚动
       列表时触发blur事件故而要防止列表因此闭合 */
    Event.observe( this.faux, 'mousedown', this. ➡
      clickUL.bind(this), false );
  },
  ……省略的代码……
  clickUL: function() {
    // 防止在滚动条中单击时闭合列表
    this.preventClose = true;
  },
  ……省略的代码……
  blur: function(){
```

www.TopSage.com

```
    ……省略的代码……
    if( this.faux.hasClassName( this.type + '-open' ) &&
        !this.preventClose ) this.close();
    /* 如果preventClose为true，则我们不希望调用blur
    但下一次调用时需要，因此将preventClose设置回false */
    this.preventClose = false;
  },
  ……省略的代码……
};
```

需要加以重视的变化发生在blur()方法中。通过调整代码中的测试条件，确保了this.preventClose在实际闭合列表之前是false。这样，如果用户单击展开了人造select，焦点就会传递到真实的select上面。当用户单击ul（要注意此处是ul，而非列表项或者人造option）滚动列表时，会调用clickUL()方法，进而会在blur()方法有机会闭合人造select之前，将preventClose设置为true。不过，blur()方法却会重置preventClose的值，因而在下一次再调用该方法时，就会如期闭合人造select了。

### 12.6.3 select 太大了吗

在本章前面，我们简单介绍了怎样处理人造select过长的问题。但是，设置maxHeight并不能真正给用户带来最佳的体验。如果用户通过键盘输入选项（例如，通过输入CON带出的Connecticut），那么仍然需要使滚动条与用户的键盘移动和操作保持同步。为解决这个问题，我们通过应用几个数学公式来验证突出显示的人造option能否进入当前可见的范围内。如果答案是否，则更新滚动条的位置以确保其可见：

```
FauxSelect.prototype = {
  ……省略的代码……
  updateFaux: function( e ){
    ……省略的代码……
    // 如果存在则调整滚动条
    if( this.faux.className.indexOf( 'overflowing' ) != -1 &&
        this.faux.hasClassName( this.type + '-open' ) ){
          var ulHeight = this.faux.getHeight();
      var liHeight = $( this.faux.firstChild ).getHeight();
      if( // 向下
        ( ( el.selectedIndex+1 ) * liHeight >
          this.faux.scrollTop + ulHeight ) ||
          // 向上
          ( ( el.selectedIndex * liHeight ) < this.faux.scrollTop ) ){
        this.faux.scrollTop = el.selectedIndex * liHeight;
      }
    }
  },
  ……省略的代码……
};
```

这一小段代码完成的任务是：首先，测试FauxSelect是否溢出且展开。然后进行一些侦查，收集人造select的高度信息和一个人造option的高度信息。接着，将人造select的scrollTop属性（即滚动条的位置）与当前选择的人造option的垂直位置比较，以确定该人造option是否

www.TopSage.com

超出了可见范围（上或下）。如果是，则更新人造select的scrollTop属性。

为照顾到各个方面，也可以将这个“检查器”的微调版添加到flip()方法中，以保证当前选择的人造option始终位于可见的范围内：

```
FauxSelect.prototype = {
  ……省略的代码……
  flip: function(){
    // 如果展开则闭合，否则展开
    if( this.faux.hasClassName( 'opening' ) ){
      ……省略的代码……
      // 如果是overflow类型的select，则滚动到选择的LI
      var heightLI = $( this.faux.firstChild ).getHeight();
      var top = heightLI * this.select.selectedIndex;
      if( this.type == 'overflowing' && (
          this.faux.scrollTop > top || this.faux.scrollTop
          + FauxSelectConductor.maxHeight < top + heightLI ) ) {
        this.faux.scrollTop = top;
      }
    } else {
      ……省略的代码……
    }
  },
  ……省略的代码……
};
```

此时，还必须面对另外一个小问题——有时候，人造option上的悬停状态在人造select闭合时仍然会保持（也就是说，永远不会触发blur事件）。要解决这个问题，可以向flip()方法中再添加如下代码：

```
FauxSelect.prototype = {
  ……省略的代码……
  flip: function(){
    // 如果展开则闭合，否则展开
    if( this.faux.hasClassName( 'opening' ) ){
      ……省略的代码……
    } else {
      ……省略的代码……
      // 确保所有li元素的hover类都已经移除
      // 之所以仍然会存在，是因为我们是以
      // 手动方式闭合的人造select
      $A( this.faux.childNodes ).each( function( child ){
        if( child.hasClassName( 'hover' ) ) child. ➥
          removeClassName( 'hover' );
      } );
    }
  },
  ……省略的代码……
};
```

## 12.7 最后的细节

感觉到了吗？胜利已经在望了。再处理一些细节上的代码，就可以大功告成了。你可能会问

www.TopSage.com

“我们最终摆脱了原始的select了吗？”怎么会这么问呢，当然是的。通过应用整个脚本中大概是最简单的调整，就可以把select从页面上抹去。为验证这一魔术式的过程，首先添加下列代码：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 最终隐藏原始的select元素
    this.select.addClassName('replaced');
  },
  ……省略的代码……
};
```

然后，在faux-select.css文件中，将原始的select向左移动较长的距离，这样既保证了键盘导航的有效性，同时也将其从视图中隐藏了起来：

```
select.replaced {
  position: relative;
  left: -999em;
}
```

当然，这样一来原始select所在的位置就空了出来。不过，通过为人造select的容器应用负的上外边距，就可以轻松解决这个问题：

```
FauxSelect.prototype = {
  ……省略的代码……
  initialize: function( id ){
    ……省略的代码……
    // 但是，别忘了把人造select向上移一点
    this.container.style.marginTop =
        '-' + this.select.getHeight() +'px';
    ……省略的代码……
  },
  ……省略的代码……
};
```

最后，如图12-16所示，结果出来了。

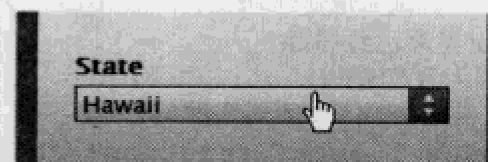



图12-16　看，没有select

## 12.8　继续替换select的冒险

如果你真有胆量，可以考虑进一步增强这个脚本，使其能够支持多个select以及基于optgroup的select。如果你想走捷径，在chapter12/advanced[1]中可以找到现成的方案。要找新版本，访问http://code.google.com/p/easy-designs/wiki/FauxSelect试试看。图12-17展示了在使用这

① 原文faux-select-advanced有误。——译者注

www.TopSage.com

一技术之后表单的精美外观。祝你好运！

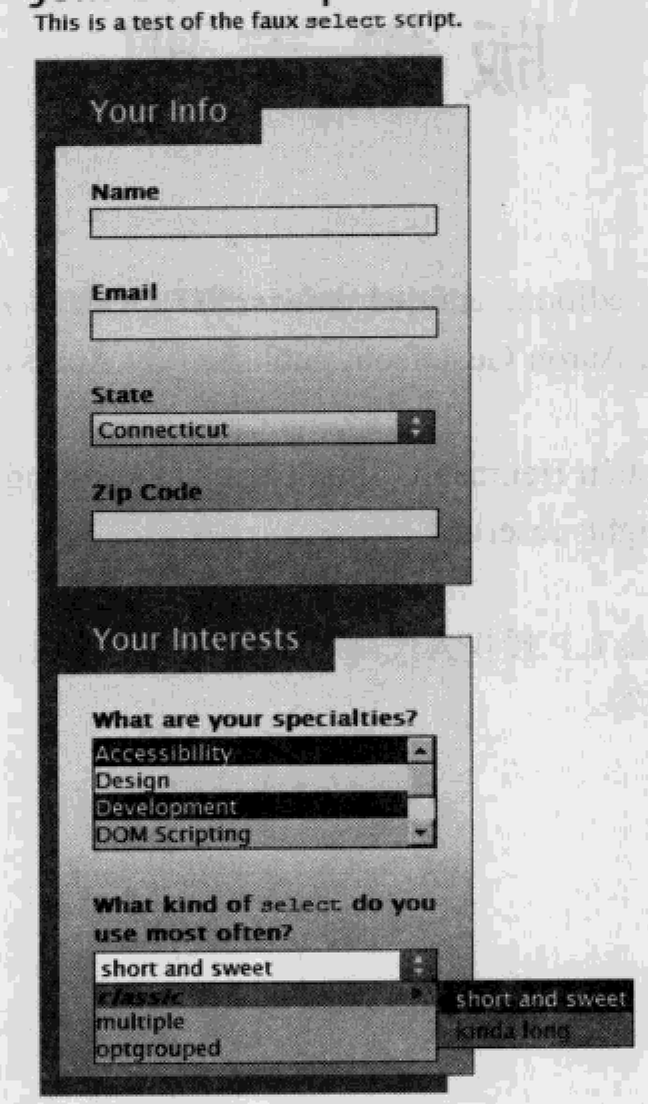



图12-17 使用FauxSelect技术创建的精美表单

## 12.9 小结

除了提供一个非常有用的脚本（特别是在你和我一样具有挑剔的视觉倾向的情况下）之外，本章的案例学习也为怎样使用、为什么使用以及何时使用JavaScript库提供了一个真实的例子。而且，通过这个例子也证明了，即使像重新构建select表单控件这样难以置信的任务，也并非不可逾越的。问题在于，能否将艰巨的任务分解为一个个的简单步骤——积跬步可以致千里嘛。正如Ovid所言："Gutta cavat lapidem（滴水可以穿石）"。

现在，合上这本书，去创造奇迹吧！

www.TopSage.com

# 计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

**Java** 一览无余：**Java** 视频教程 | **Java SE** | **Java EE**

**.Net** 技术精品资料下载汇总：**ASP.NET** 篇

**.Net** 技术精品资料下载汇总：**C#**语言篇

**.Net** 技术精品资料下载汇总：**VB.NET** 篇

撼世出击：**C/C++**编程语言学习资料尽收眼底 电子书+视频教程

**Visual C++(VC/MFC)**学习电子书及开发工具下载

**Perl/CGI** 脚本语言编程学习资源下载地址大全

**Python** 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 **Ruby**、**Ruby on Rails** 精品电子书等学习资料下载

数据库精品学习资源汇总：**MySQL** 篇 | **SQL Server** 篇 | **Oracle** 篇

最强 **HTML/xHTML**、**CSS** 精品学习资料下载汇总

最新 **JavaScript**、**Ajax** 典藏级学习资料下载分类汇总

网络最强 **PHP** 开发工具+电子书+视频教程等资料下载汇总

**UML** 学习电子资下载汇总 软件设计与开发人员必备

经典 **LinuxCBT** 视频教程系列 **Linux** 快速学习视频教程一帖通

天罗地网：精品 **Linux** 学习资料大收集(电子书+视频教程) **Linux** 参考资源大系

**Linux** 系统管理员必备参考资料下载汇总

**Linux shell**、内核及系统编程精品资料下载汇总

**UNIX** 操作系统精品学习资料<电子书+视频>分类总汇

**FreeBSD/OpenBSD/NetBSD** 精品学习资源索引 含书籍+视频

**Solaris/OpenSolaris** 电子书、视频等精华资料下载索引

“本书是一本全景式的、沟通历史和未来的Web开发经典好书，是对现有JavaScript DOM程序开发最佳实践的一次大检阅和大放送，是推动Web标准化和向下一代Web开发挺进的里程碑式著作。”

——本书译者

“如果你是一位中级JavaScript开发人员，还想更上一层楼，那么这将是使你梦想成真的绝妙好书。”

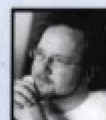
——DOMAssistant库的作者Robert Nyman

# AdvancED DOM Scripting Dynamic Web Design Techniques
# JavaScript DOM高级程序设计

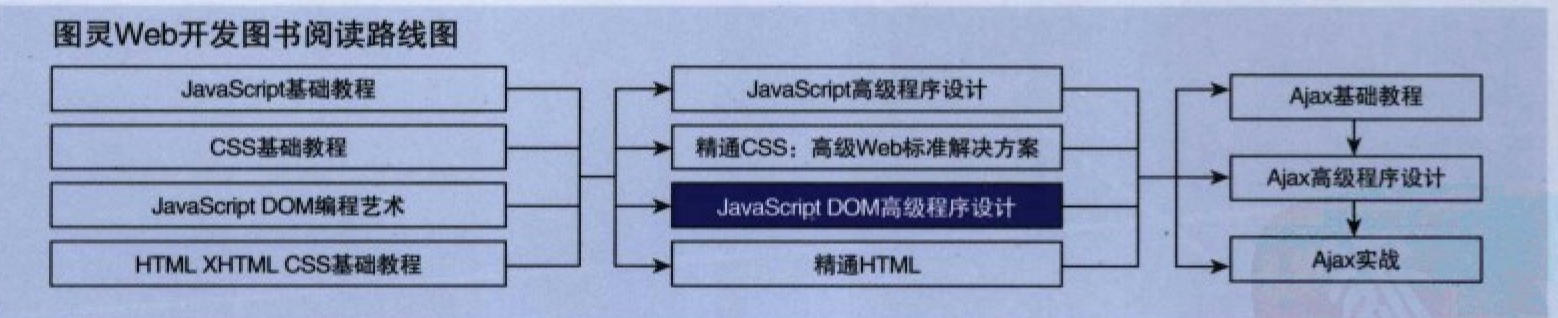
本书深入浅出地讲述了作为一名专业的Web开发人员（或者真正的高手）所必须了解和掌握的高级知识，是Web编程领域名副其实的扛鼎之作。书中对核心JavaScript原理的总结和概括、对最佳实践的倡导和践行、对DOM规范讲解的提纲挈领、对浏览器外部通信（Ajax）的反思与解决之道、对Web 2.0内容整合（Mashup）的分类与讲说等，无一不折射出这本书是作者博观约取、厚积薄发的心血力作。

与此同时，如果你也醉心于Prototype、jQuery、YUI、Ext等优秀的JavaScript库，想见微知著地真正理解这些库背后的工作原理，甚至希望创建自己的库，那么这本书恰好适合你。

**Jeffrey Sambells** 资深Web设计师和程序员，We-Create公司创始人之一暨研发总监。除本书外，他还与人合写了*Beginning Google Maps Applications with PHP and Ajax*等著作。



**Aaron Gustafson** 世界顶尖的Web工程师，创建了Web咨询公司Easy! Designs LLC.。Aaron是WaSP（Web标准项目）和GAWDS（可访问性Web设计师协会）的成员。他还是A List Apart网站的技术编辑，*Digital Web Magazine*和*MSDN*等著名杂志的撰稿人。



Apress®

本书相关信息请访问：图灵网站 http://www.turingbook.com
读者/作者热线：(010)88593802
反馈/投稿/推荐信箱：contact@turingbook.com

**上架建议** 计算机/网络开发/程序设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-18109-1/TP

定价：59.00 元


www.TopSage.com