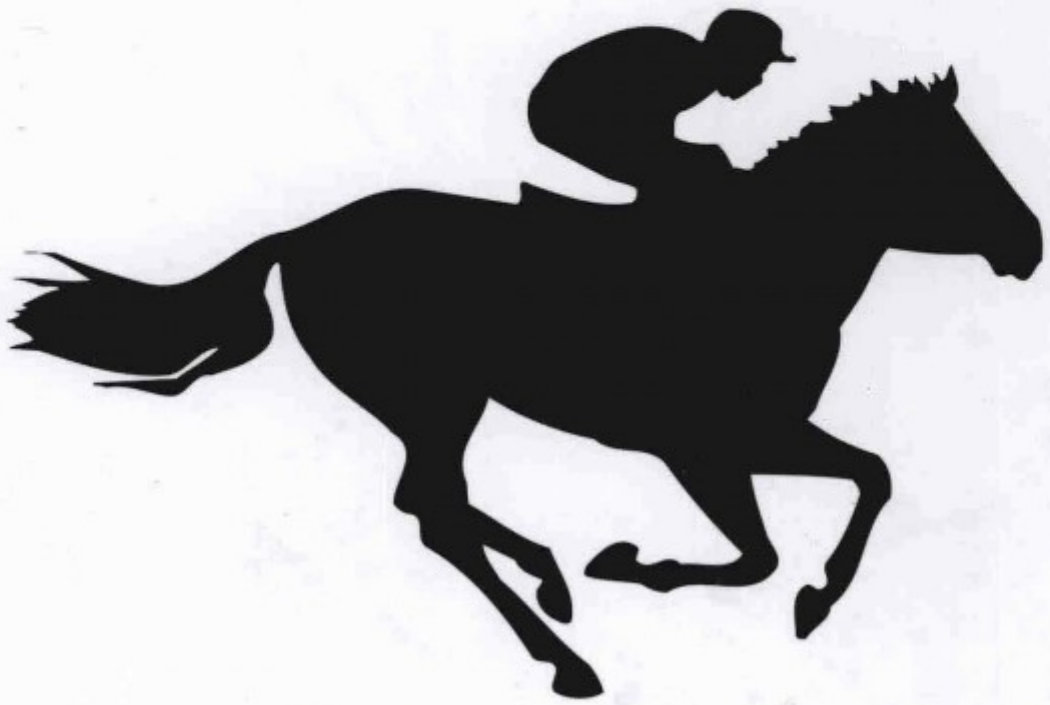
HZ BOOKS
华章科技

依据HTML 5标准的最新草案撰写，对HTML 5进行了系统、全面、透彻的讲解

106个精心设计的经典案例对各个知识点进行了补充和阐释，理论与实践完美结合

實戰



陶国荣 著

HTML 5 in Action

HTML 5实战

机械工业出版社
China Machine Press

實戰

51CTO.com 技术成就梦想　html51.com　联合策划

IT技术的世界永远都是瞬息万变的，几年前，无论是技术界、产业界，还是媒体界，都一致看好RIA技术的发展与前途。但是短短几年之后，HTML 5的出现使这一格局发生了颠覆性的变化。Adobe希望通过收购PhoneGap来延续Flash的生命并实现在HTML 5时代的完美转型，Microsoft则将Silverlight的命运寄托在了Windows Phone身上。

HTML 5的盛行是大势所趋，作为技术人员的你，如果不想被时代所抛弃，建议趁早拥抱HTML 5带来的变革。如果要学习HTML 5，在现有的同类书中，本书应该是一个绝佳的选择。它不仅讲了最新的HTML 5技术，而且内容十分全面，很适合系统地学习。除此之外，本书还有一个很大的特色，几乎每个知识点都配有一个完整的案例，每个案例包括功能描述、实现代码、效果展示和代码分析等几个部分，非常便于读者一边学习一边实践，从而提高学习效率。

——51CTO（www.51cto.com）

无论是浏览器厂商，还是大型的互联网企业，都在HTML 5上做了重要的战略部署，能否在HTML 5时代Hold住，也许会决定企业未来的命运。作为技术人员，我们需要掌握的技能永远都要随着市场的需求而变化，HTML 5必将盛行，我们应该未雨绸缪。如何才能又好又快地从宏观和微观上掌握HTML 5的最新技术？本书是一个不错的选择！它内容全面，既适合作为系统学习的教材，又适合在开发时参考。同时，书中提供的106个案例为我们动手实践提供了丰富的素材。强烈推荐！

——HTML 5用户组

作者的上一本书《jQuery权威指南》凭借优质的内容和创新性的写作方式而得到了读者的一致好评，并取得了不错的销售成绩，这在原创图书中的确不多见。在内容上，本书在理论和实践上都非常突出，系统全面，实战性强；在写作方式上，本书继承了《jQuery权威指南》的优点，而且一些细节之处还有所改进，相信将进一步提高读者的学习体验。值得期待!

——HTML51（www.html51.com）

上架指导：计算机/程序设计/Web开发
ISBN 978-7-111-35873-2
9 787111 358732
定价：59.00元

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com
网上购书：www.china-pub.com

實戰

HTML 5 in Action

HTML 5实战

陶国荣 著

机械工业出版社
China Machine Press

这是一本系统而全面的 HTML 5 教程，根据 HTML 5 标准的最新草案，系统地对 HTML 5 的所有重要知识点进行了全面的讲解。在写作方式上，本书以一种开创性的方式使理论与实践达到极好的平衡，不仅对理论知识进行了清晰而透彻的阐述，而且根据读者理解这些知识的需要，精心设计了 106 个完整（每个案例分为功能描述、实现代码、效果展示和代码分析 4 个部分）的实战案例，旨在帮助读者通过实践的方式迅速掌握这些知识。

全书共 11 章，内容涵盖了 HTML 5 的各个方面。第 1 章通过实现一个简单的 HTML 5 页面讲解了如何搭建支持 HTML 5 的浏览器环境、HTML 5 页面所具备的特征，以及如何检测浏览器对 HTML 5 的各种特性的支持情况；第 2 章介绍了 HTML 5 中常用的交互元素，包括内容交互元素、菜单交互元素和状态交互元素等几大类；第 3 章介绍了 HTML 根元素、文档元素，以及与脚本、节点、分组内容、文本层次语义、嵌入内容、公共属性相关的重要元素；第 4 章和第 5 章讲解了 HTML 5 中的表单和文件的功能特性以及常见的各种操作；第 6 章和第 7 章讲解了 HTML 5 中的音频、视频和绘图相关的知识，重点讲解了各种常见的操作和使用方法；第 8 章和第 9 章讲解了 HTML 5 中的数据存储和离线应用；第 10 章对 Web Sockets、Geolocation、Web Workers、元素的拖放等重要内容进行了全面的讲解。

本书适合所有想系统学习 HTML 5 的读者阅读。如果按照本书的顺序逐章阅读，同时亲自动手实现本书中的案例，相信一定能达到事半功倍的效果。

封底无防伪标均为盗版
版权所有，侵权必究
本书法律顾问　北京市展达律师事务所

图书在版编目（CIP）数据

HTML 5 实战 / 陶国荣著 . —北京：机械工业出版社，2011.9

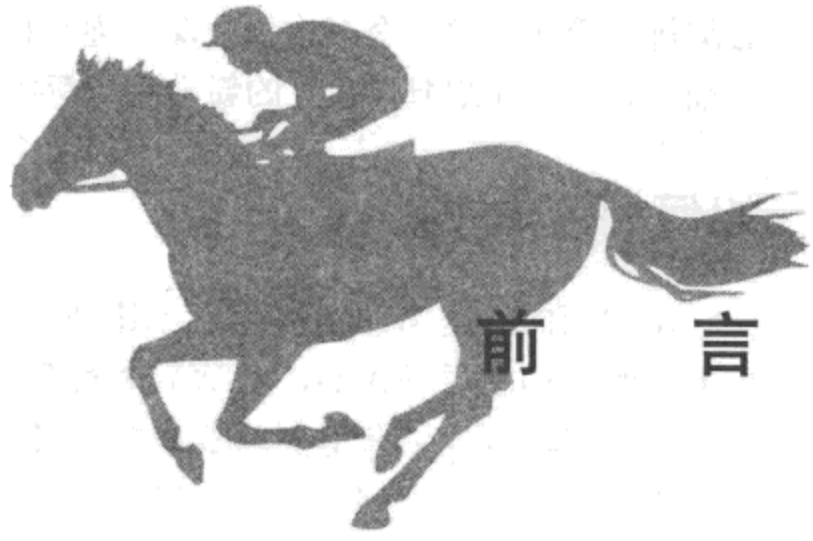
ISBN 978-7-111-35873-2

Ⅰ. H…　Ⅱ. 陶…　Ⅲ. 超文本标记语言，HTML —程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2011）第 189332 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）
责任编辑：白　宇
北京京师印务有限公司印刷
2012 年 1 月第 1 版第 1 次印刷
186mm×240mm • 20.25 印张
标准书号：ISBN 978-7-111-35873-2
定价：59.00 元

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



前　言

为什么要写这本书

当今时代，网络应用正处在不断变革中，而作为与应用密切相关的前端技术更是备受瞩目，其中，以 HTML 5 为代表的新一代技术尤为受到多方的关注，因为 HTML 5 不仅仅是一次简单的技术升级，更代表了未来 Web 开发的方向，被寄予了太多的期望与依托。曾有人预言，在不久的将来，不仅仅是 Web 服务端，就连客户端的开发也将是 HTML 5 的天下。

在 2004 年，HTML 5 的草案被提起，直到 2007 年才被 W3C 所接纳，最终于 2008 年首份草案被公布。HTML 5 的框架在原版本的基础上，废除了许多 HTML 4 中不合理的效果标记，创造性地增加了很多用于富媒体、富图形的新标记，最大限度地减少了对外部插件的依赖；同时，通过对本地离线存储方式的优化，使 HTML 5 更加有利于移动客户端的开发。

当然，HTML 5 所涉及的新特征与新功能十分广泛，远不能通过简短的几句话进行概括，许多新增加的 API 或元素属性需要借助相关的书籍来引导开发者进行学习，使其快速掌握 HTML 5。但纵观市场中已出版的书籍，绝大多数是简单的定义解析与理论灌输，没有对应的实例操作，缺乏真正的实践指导。为了使广大的 Web 开发者真正了解与全面掌握 HTML 5，笔者撰写了本书。衷心希望读者能通过本书的学习与实践演练，早日开发出更加前沿与时尚的 Web 应用。

“临渊羡鱼，不如退而结网”，每一位从事 Web 应用开发的工程师，无论是从事前端开发还是服务端的代码开发，都有理由更新自己的知识结构，掌握这一门技术。

本书特点

“学以致用”是本书的一个重要特点，全书始终体现一个“用”字，无论是理论知识的介绍，还是实例的开发，无一例外都是从实用的角度出发，每一个实例都是精心选择的，介

绍详细；为了使读者能够通过实例执行后的页面效果加深对应用的理解，每一个示意图都精心编排，简明易懂；全书由浅入深，逐步推进，以实例为主线，激发读者的阅读兴趣；全面、详细、完整地介绍 HTML 5 的新功能与新特征，又是本书的另外一个重要特点。

如何阅读本书

本书面向的是 Web 开发者，不论是前端开发者，还是后台程序员，都可以使用本书。由于本书的结构是层进式的，各章节之间有一定的关联，因此，建议读者按章节的顺序，逐章阅读；在实践时，尽量不要照搬书中的实例，要理解主要的、核心的代码，自己动手开发相似功能的应用，并逐步完善其功能，从而真正掌握代码的实质。

联系作者

写作本书耗时半年，其中积累了笔者数年心得与技术感悟。希望本书能给每位阅读过本书的人带来思路上的启发与技术上的提升，使每位读者能够有所悟、有所得。同时，也非常希望能借此机会与国内热衷于 HTML 5 技术的开发者进行交流。由于本人水平有限，其中难免存在疏漏之处，希望大家批评、指正。大家可以通过邮箱 tao_guo_rong@163.com 与笔者联系。

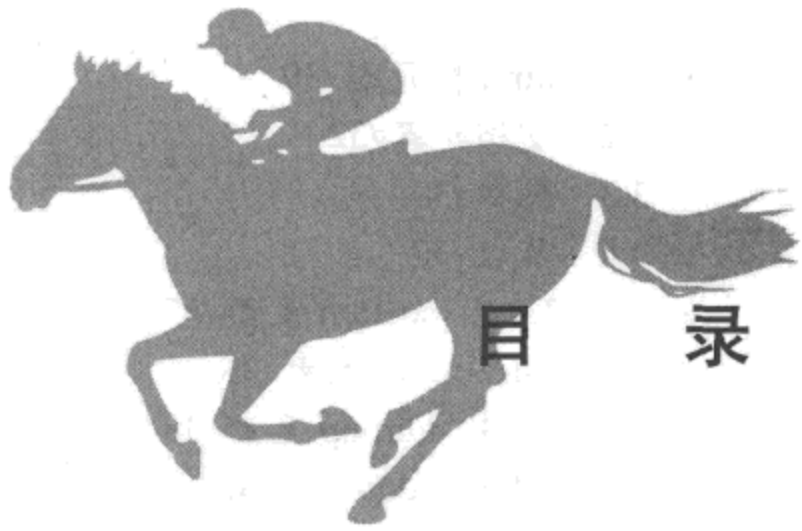
另外，参加本书编写工作的还有：刘义、李建洲、李静、裴星如、李建勤、陶红英、陈建平、孙文华、孙义、陶林英、闵慎华、孙芳、赵刚。

致谢

首先要感谢机械工业出版社华章公司的编辑们，尤其是杨福川与白宇，正是他们在写作过程中的全程指导，才使整个创作思路不断被完善，全书的框架不断被优化，确保了本书顺利完稿。另外，要感谢我的家人，正是他们的理解与默默支持，才使我能够全心写作，顺利完成本书的编写工作。

陶国荣

2011 年 8 月



目　录

前　言

第 1 章　拥抱 HTML 5 /1

1.1　一个简单的 HTML 5 页面 /2
　1.1.1　搭建支持的浏览器环境 /2
　1.1.2　检测浏览器是否支持 HTML 5 标记 /2
　1.1.3　使用 HTML 5 结构编写一个简单的 Web 页面 /4
1.2　HTML 5 页面的特征 /6
　1.2.1　应用全新的 HTML 5 特征结构化元素 /6
　1.2.2　使用 CSS 文件美化 HTML 5 新元素 /9
1.3　本章小结 /10

第 2 章　HTML 5 中常用的交互元素 /11

2.1　内容交互元素 /12
　2.1.1　details 元素 /12
　2.1.2　summary 元素 /16
2.2　菜单交互元素 /17
　2.2.1　menu 元素 /17
　2.2.2　command 元素 /20
2.3　状态交互元素 /23
　2.3.1　progress 元素 /24
　2.3.2　meter 元素 /26
2.4　本章小结 /28

第 3 章　HTML 5 中的重要元素 /29

3.1　html 根元素 /30
3.2　文档元素 /32
3.3　脚本 /34
3.4　节点 /37
　3.4.1　section 元素 /37
　3.4.2　nav 元素 /38
　3.4.3　hgroup 元素 /38
　3.4.4　address 元素 /38
3.5　分组内容 /39
　3.5.1　ul 元素 /39
　3.5.2　ol 元素 /40
　3.5.3　dl 元素 /41
3.6　文本层次语义 /42
　3.6.1　time 元素 /42
　3.6.2　mark 元素 /43
　3.6.3　cite 元素 /45
3.7　嵌入内容 /46
　3.7.1　img 元素 /46
　3.7.2　iframe 元素 /47
　3.7.3　object 元素 /48
3.8　公共属性 /48
　3.8.1　draggable 属性 /48
　3.8.2　hidden 属性 /50
　3.8.3　spellcheck 属性 /51
　3.8.4　contenteditable 属性 /53
3.9　本章小结 /55

第 4 章　HTML 5 中的表单 /57

4.1　input 元素的新增类型 /58
　4.1.1　email 邮件类型 /58
　4.1.2　url 地址类型 /60
　4.1.3　number 数字类型 /62
　4.1.4　range 数字滑动条 /64

4.1.5 date 日期类型 /66
4.1.6 search 搜索类型 /69
4.2 input 元素新增的公用属性 /71
4.2.1 autofocus 属性 /71
4.2.2 pattern 属性 /73
4.2.3 placeholder 属性 /75
4.2.4 required 属性 /76
4.3 新增表单元素 /78
4.3.1 datalist 元素 /78
4.3.2 output 元素 /80
4.3.3 keygen 元素 /81
4.4 表单新增的验证方法和属性 /83
4.4.1 checkValidity 显式验证法 /83
4.4.2 使用 setCustomValidity 方法修改提示信息 /85
4.4.3 表单的 novalidate 属性 /87
4.5 本章小结 /89

第 5 章 HTML 5 中的文件 /91

5.1 选择文件 /92
5.1.1 选择单个文件 /92
5.1.2 选择多个文件 /93
5.1.3 使用 Blob 接口获取文件的类型与大小 /95
5.1.4 通过类型过滤选择的文件 /97
5.1.5 通过 accept 属性过滤选择文件的类型 /99
5.2 使用 FileReader 接口读取文件 /101
5.2.1 FileReader 接口的方法 /101
5.2.2 使用 readAsDataURL 方法预览图片 /101
5.2.3 使用 readAsText 方法读取文本文件 /104
5.2.4 侦听 FileReader 接口中的事件 /106
5.3 使用 DataTransfer 对象拖放上传图片文件 /109
5.4 文件读取时的错误与异常 /112
5.4.1 发生错误与异常的条件 /112
5.4.2 错误代码说明 /113
5.5 本章小结 /114

第 6 章　HTML 5 中的视频和音频 /115

6.1　多媒体元素基本属性 /116
6.1.1　元素格式 /116
6.1.2　width 与 height 属性 /117
6.1.3　controls 属性 /119
6.1.4　poster 属性 /121
6.1.5　networkState 属性 /122
6.1.6　error 属性 /124
6.1.7　其他属性 /127
6.2　多媒体元素常用方法 /131
6.2.1　媒体播放时的方法 /131
6.2.2　canPlayType 方法 /133
6.3　多媒体元素重要事件 /136
6.3.1　媒体播放事件 /136
6.3.2　timeupdate 事件 /138
6.3.3　其他事件 /140
6.4　本章小结 /141

第 7 章　HTML 5 绘图基础 /143

7.1　画布的基础知识 /144
7.1.1　canvas 元素的基本用法 /144
7.1.2　绘制带边框矩形 /146
7.1.3　绘制渐变图形 /148
7.2　在画布中使用路径 /151
7.2.1　moveTo 与 lineTo 的用法 /151
7.2.2　使用 arc 方法绘制圆形 /153
7.2.3　绘制渐变圆形 /157
7.3　对画布中图形的操作 /160
7.3.1　变换图形原点坐标 /160
7.3.2　组合多个图形 /163
7.3.3　添加图形阴影 /166
7.4　处理画布中的图像 /168
7.4.1　绘制图像 /168
7.4.2　平铺图像 /171

7.4.3 切割图像 /174
7.4.4 处理像素 /176
7.5 画布的其他应用 /179
7.5.1 绘制文字 /179
7.5.2 保存、恢复及输出图形 /182
7.5.3 制作简单的动画 /185
7.6 本章小结 /188

第 8 章 HTML 5 中的数据存储 /189

8.1 Web Storage 存储简介 /190
8.1.1 sessionStorage 对象 /190
8.1.2 localStorage 对象 /192
8.2 localStorage 详解 /196
8.2.1 清空 localStorage 数据 /196
8.2.2 遍历 localStorage 数据 /199
8.2.3 使用 JSON 对象存取数据 /202
8.2.4 管理 localStorage 数据 /205
8.3 Web SQL 数据库基础 /210
8.3.1 打开与创建数据库 /210
8.3.2 执行事务 /212
8.3.3 插入数据 /215
8.3.4 数据管理 /218
8.4 本章小结 /225

第 9 章 HTML 5 中的离线应用 /227

9.1 离线应用程序 /228
9.1.1 manifest 文件简介 /228
9.1.2 配置 IIS 服务器 /229
9.1.3 离线应用的开发过程 /231
9.2 本地缓存的更新及状态检测 /233
9.2.1 updateready 事件 /234
9.2.2 update 方法 /236
9.2.3 swapCache 方法 /239
9.2.4 更新本地缓存时触发的其他事件 /241
9.3 检测在线状态 /244

9.3.1 onLine 属性 /244
9.3.2 online 与 offline 事件 /246
9.3.3 离线数据交互应用开发过程 /249
9.4 本章小结 /254

第 10 章 HTML 5 中的其他应用型 API /255

10.1 Web Sockets API /256
10.1.1 postMessage 方法 /256
10.1.2 使用 WebSocket 传送数据 /260
10.1.3 使用 WebSocket 传送 JSON 对象 /263
10.2 Geolocation API /267
10.2.1 使用 getCurrentPosition 方法获取当前地理位置 /267
10.2.2 使用 Google 地图锁定位置 /273
10.3 Web Workers API /275
10.3.1 Worker 对象处理线程 /276
10.3.2 使用线程传递 JSON 对象 /279
10.3.3 使用线程嵌套交互数据 /281
10.4 本章小结 /284

第 11 章 HTML 5 中元素的拖放 /285

11.1 拖放基础 /286
11.1.1 使用 JavaScript 代码实现拖放 /286
11.1.2 在 HTML 5 中实现拖放时触发的事件 /288
11.2 dataTransfer 对象应用详解 /291
11.2.1 使用 setData 与 getData 方法存入与读取拖放数据 /292
11.2.2 使用 setDragImage 方法设置拖放图标 /295
11.2.3 使用 effectAllowed 与 dropEffect 属性设置拖放效果 /297
11.3 拖放应用实战 /300
11.3.1 购物车的实现 /300
11.3.2 相册的管理 /304
11.4 本章小结 /307

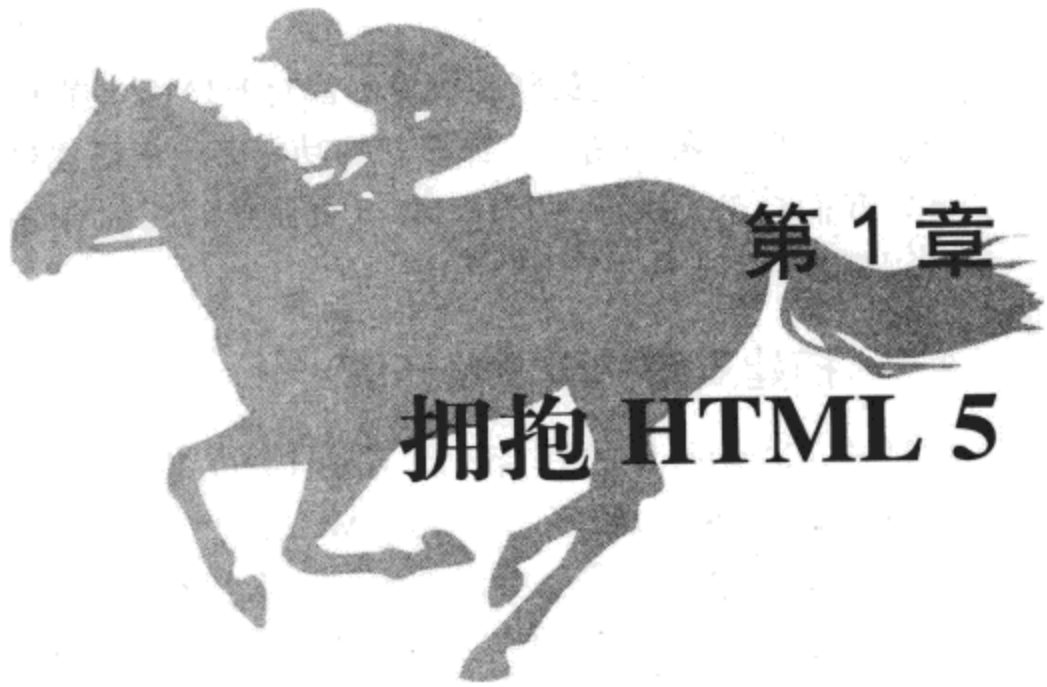
实例目录

实例 1-1　检测浏览器是否支持 HTML 5 /2
实例 1-2　Hello,World 页面的实现 /4
实例 1-3　页面分栏实现 /6
实例 1-4　样式化页面实现 /9
实例 2-1　交互元素 <details> 的使用 /13
实例 2-2　用脚本控制交互元素 <details> 的使用 /14
实例 2-3　交互元素 <summary> 与 <details> 的结合使用 /16
实例 2-4　交互元素 <menu> 的使用 /18
实例 2-5　交互元素 <command> 与 <menu> 的结合使用 /20
实例 2-6　交互元素 <progress> 的使用 /24
实例 2-7　交互元素 <meter> 的使用 /26
实例 3-1　元素 <html> 的使用 /30
实例 3-2　元素 <head> 的使用 /33
实例 3-3　元素 <script> 与 <noscript> 的使用 /35
实例 3-4　元素 <ol> 的使用 /40
实例 3-5　元素 <mark> 的使用 /43
实例 3-6　元素 <cite> 的使用 /45
实例 3-7　公共属性 draggable 的使用 /49
实例 3-8　公共属性 hidden 的使用 /50
实例 3-9　公共属性 spellcheck 的使用 /51
实例 3-10　公共属性 contenteditable 的使用 /53
实例 4-1　email 类型的 <input> 元素的使用 /59
实例 4-2　url 类型的 <input> 元素的使用 /61

实例 4-3　number 类型的 <input> 元素的使用 /62
实例 4-4　range 类型的 <input> 元素实现颜色选择器 /64
实例 4-5　分类展示不同形式的选择日期 /66
实例 4-6　search 类型的 <input> 元素的使用 /69
实例 4-7　<input> 元素中 autofocus 属性的使用 /71
实例 4-8　<input> 元素中 pattern 属性的使用 /73
实例 4-9　<input> 元素中 placeholder 属性的使用 /75
实例 4-10　<input> 元素中 required 属性的使用 /77
实例 4-11　<datalist> 元素的使用 /78
实例 4-12　<output> 元素的使用 /80
实例 4-13　<keygen> 元素的使用 /82
实例 4-14　调用表单的 checkValidity 方法 /83
实例 4-15　调用表单的 setCustomValidity 方法 /85
实例 4-16　表单中 novalidate 属性的使用 /88
实例 5-1　选择单个文件上传 /92
实例 5-2　选择多个文件上传 /94
实例 5-3　获取上传文件的类型与大小 /95
实例 5-4　通过类型过滤上传文件 /97
实例 5-5　通过 accept 属性过滤上传文件的类型 /99
实例 5-6　使用 readAsDataURL 方法预览图片 /102
实例 5-7　使用 readAsText 方法读取文本文件 /104
实例 5-8　展示文件读取时触发事件的先后顺序 /107
实例 5-9　使用 DataTransfer 对象拖放上传图片文件 /109
实例 6-1　使用多媒体元素播放文件 /116
实例 6-2　设置 <video> 元素的大小与样式 /118
实例 6-3　设置 <video> 元素的控制条工具属性 /119
实例 6-4　设置 <video> 元素的 poster 属性 /121
实例 6-5　获取 <video> 元素 networkState 属性的返回值 /122
实例 6-6　获取 <video> 元素 error 属性的返回值 /125
实例 6-7　自定义 <video> 元素控制条工具栏 /131
实例 6-8　使用 canPlayType 方法检测浏览器支持媒体类型 /133
实例 6-9　获取多媒体元素在播放事件中的不同状态 /136
实例 6-10　通过 timeupdate 事件动态显示媒体文件播放时间 /138
实例 7-1　使用 <canvas> 元素绘制正方形 /144
实例 7-2　使用 <canvas> 元素绘制带边框的矩形 /146
实例 7-3　使用 <canvas> 元素绘制有渐变色的图形 /149

实例 7-4　使用 moveTo 与 lineTo 方法绘制多条直线 /151
实例 7-5　使用 arc 方法绘制多个不同样式的圆形 /154
实例 7-6　使用 <canvas> 元素绘制径向渐变的圆形 /158
实例 7-7　使用 <canvas> 元素移动、缩放、旋转图形 /160
实例 7-8　使用 <canvas> 元素设置多图形组合显示的方式 /164
实例 7-9　使用 <canvas> 元素添加绘制图形阴影 /166
实例 7-10　使用 drawImage 方法在画布中绘制图像 /169
实例 7-11　使用 createPattern 方法在画布中平铺图像 /172
实例 7-12　使用 clip 方法在画布中切割图像 /174
实例 7-13　使用 getImageData 与 putImageData 方法处理图像像素 /177
实例 7-14　使用 fillText 与 strokeText 方法绘制文字 /180
实例 7-15　在画布中保存、恢复及输出图形 /182
实例 7-16　在画布中制作简单的动画 /186
实例 8-1　使用 sessionStorage 对象保存与读取临时数据 /191
实例 8-2　使用 localStorage 对象保存与读取登录用户名与密码 /193
实例 8-3　清空 localStorage 对象保存的全部数据 /197
实例 8-4　遍历 localStorage 对象保存的全部数据 /199
实例 8-5　使用 JSON 对象存取数据 /202
实例 8-6　管理 localStorage 数据 /206
实例 8-7　使用 openDatabase 打开与创建数据库 /211
实例 8-8　使用 transaction 方法执行事务 /213
实例 8-9　使用 executeSql 方法插入记录 /215
实例 8-10　使用 executeSql 方法管理数据记录 /218
实例 9-1　开发一个简单的离线应用 /231
实例 9-2　监测 updateready 事件触发 /234
实例 9-3　使用 update 方法更新本地缓存 /236
实例 9-4　使用 swapCache 方法更新本地缓存 /239
实例 9-5　检测离线应用在加载过程中触发的事件 /241
实例 9-6　通过 onLine 属性检测网络的当前状态 /245
实例 9-7　通过 online 与 offline 事件检测网络的当前状态 /247
实例 9-8　开发一个离线留言数据交互应用 /249
实例 10-1　使用 postMessage 方法实现跨文档传输数据 /256
实例 10-2　使用 WebSocket 对象传送数据 /261
实例 10-3　使用 WebSocket 传送 JSON 对象 /264
实例 10-4　使用 getCurrentPosition 方法获取出错数据信息 /268
实例 10-5　使用 getCurrentPosition 方法获取地理位置信息 /270

实例 10-6　使用 Google 地图锁定位置 /273
实例 10-7　使用 Worker 对象处理线程 /276
实例 10-8　使用线程传递 JSON 对象 /279
实例 10-9　使用线程嵌套交互数据 /281
实例 11-1　使用 JavaScript 代码实现元素拖放 /286
实例 11-2　元素在拖放过程中触发的事件 /289
实例 11-3　使用 setData 与 getData 方法存入与读取拖放数据 /292
实例 11-4　使用 setDragImage 方法设置拖放图标 /295
实例 11-5　使用 effectAllowed 与 dropEffect 属性设置拖放效果 /298
实例 11-6　使用拖放 API 将商品拖入购物车 /301
实例 11-7　使用拖放 API 将图片拖入回收站 /304



第 1 章

拥抱 HTML 5

本章内容

- 一个简单的 HTML 5 页面
- HTML 5 页面的特征
- 本章小结

PDG

HTML 5 的草案诞生于 2004 年，在 2007 年时被 W3C 采纳。W3C 组建了专门的 HTML 开发小组对 HTML 5 进行开发和维护。与此同时，各大浏览器厂商也给予 HTML 5 强有力的支持，如 Mozlilla、Microsoft、Google 等公司开发的最新的版本浏览器都在不同程度上支持了 HTML 5 的新功能和新特性。可以说，HTML 5 将引领 Web 发展的方向，代表 Web 开发的未来。

1.1 一个简单的 HTML 5 页面

尽管各主流厂商的最新版浏览器都对 HTML 5 提供了很好的支持，但毕竟 HTML 5 是一种全新的 HTML 标记语言，许多新的功能必须在搭建好相应的浏览环境后才可以正常浏览。为此，我们在正式执行一个 HTML 5 页面之前，必须先搭建支持 HTML 5 的浏览器环境，并检查浏览器是否支持 HTML 5 标记。

1.1.1 搭建支持的浏览器环境

目前，Microsoft 的 IE 系列（仅有 IE 9）浏览器，以及 Mozlilla 的 Firefox 与 Google 的 Chrome 浏览器都可以很好地支持 HTML 5。

本书所有的应用实例，主要执行的浏览器为 Chrome，其对应的版本号是：10.0.648.11。

如需运行本书中的实例，则要安装该版本的 Chrome 浏览器，以浏览相应的实例页面效果。

1.1.2 检测浏览器是否支持 HTML 5 标记

安装相应的浏览器以后，为了能进一步了解浏览器支持 HTML 新标签功能的情况，还可以在引入新标签前，通过编写 JavaScript 代码来检测浏览器是否支持该标签。

浏览器在加载 Web 页面时会构造一个文本对象模型（Document Object Model，DOM），然后通过该对象模型来表示页面中的各个 HTML 元素，这些元素被表示为不同的 DOM 对象。全部的 DOM 对象都共享一些公共或特殊的属性，如 HTML 5 的某些特性，如果在支持该属性的浏览器中打开页面，就可以很快检测出这些 DOM 对象是否支持这些特性。

下面以加入画布标记为例，简单地说明检测浏览器的整个过程。

实例 1-1 检测浏览器是否支持 HTML 5

1. 功能说明

在 HTML 页面中插入一段 HTML 5 画布标记，当浏览器支持该标记时，将出现一个矩形；反之，则在页面中显示“该浏览器不支持 HTML 5 的画布标记！”的提示。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 1-1.html，加入代码如代码清单 1-1 所示。

代码清单 1-1　检测浏览器是否支持 HTML 5

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
     TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<title>检测浏览器是否支持 HTML 5</title>
</head>
<body style="font-size:13px">
<canvas id="myCanvas" width="200" height="100"
  style="border:2px solid #CCC;background-color:#EEE">
  该浏览器不支持 HTML 5 的画布标记!
  </canvas>
</body>
</html>
```

3. 页面效果

将页面文件 1-1.html 在 IE 8 浏览器中执行。由于 IE 8 浏览器暂不支持 HTML 5 的画布标记，因此将显示如图 1-1 所示的页面效果。

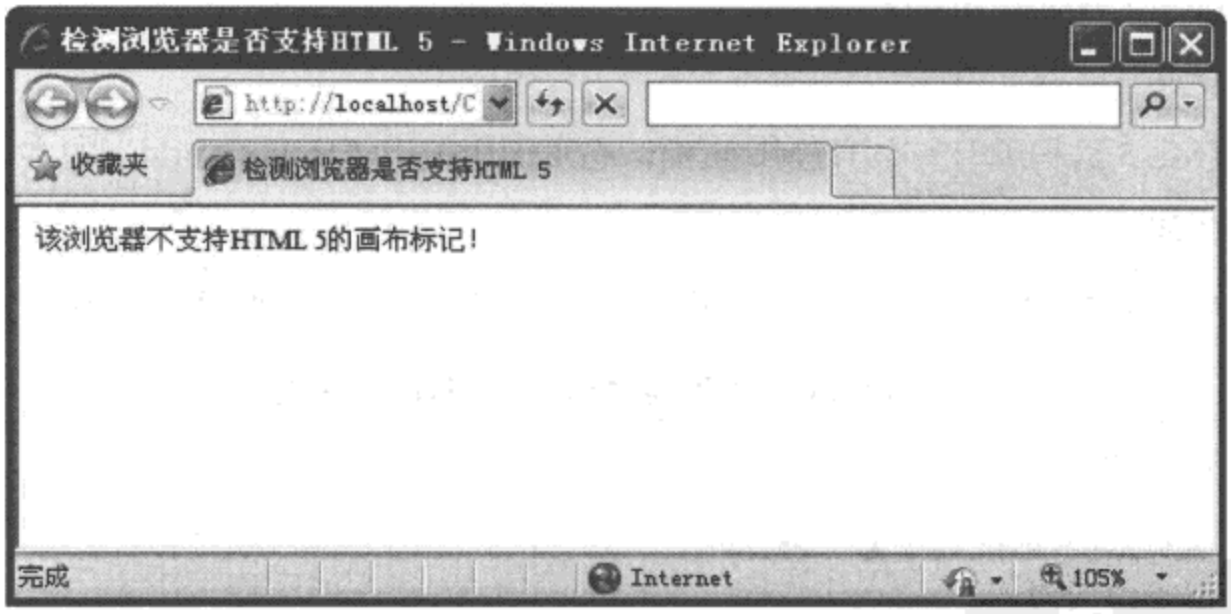


图 1-1　IE 8 浏览器不支持 HTML 5 的画布标记

将页面文件 1-1.html 在 Chrome 浏览器中执行。由于该浏览器支持 HTML 5 的画布标记，因此将显示如图 1-2 所示的页面效果。

4. 源码分析

虽然是同样一个页面，但由于不同的浏览器对 HTML 5 标记的支持情况不同，其显示的页面效果也各异。因此，在编写 HTML 5 新标记时，有必要先检测浏览器是否支持该标记。

接下来，我们利用 HTML 5 结构编写一个简单的程序，从中体会 HTML 5 的代码简洁所在。

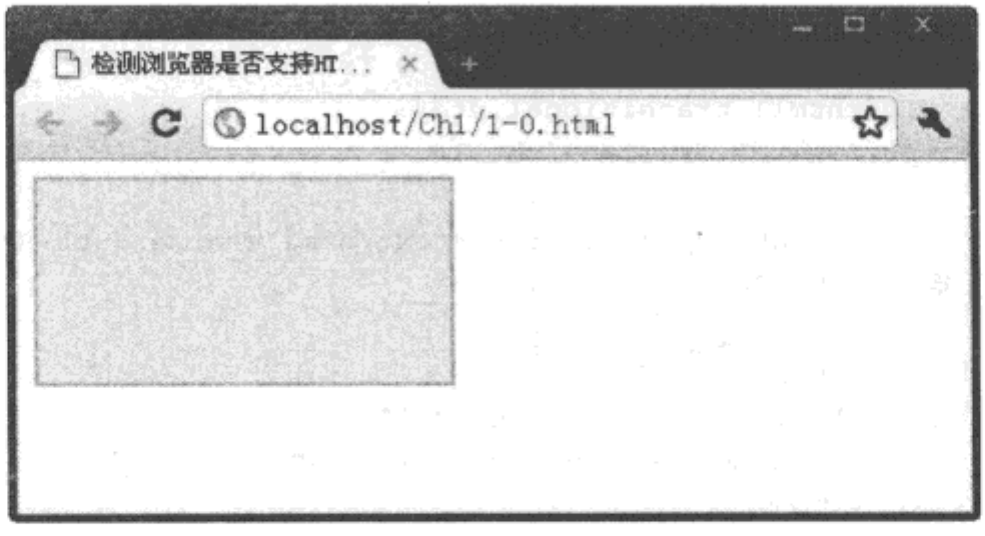


图 1-2　Chrome 浏览器支持 HTML 5 的画布标记

1.1.3　使用 HTML 5 结构编写一个简单的 Web 页面

HTML 5 中不仅增加了很多新的页面标记，而且与 HTML 4 相比，整体页面的结构也发生了根本的变化。下面使用 HTML 5 新结构来编写一个简单的页面。

实例 1-2　Hello,World 页面的实现

1. 功能说明

使用 HTML 5 结构编写一个 HTML 页面，在页面中输出“Hello,World”的字样。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 1-2.html，加入代码如代码清单 1-2 所示。

代码清单 1-2　我的第一个 HTML 5 页面

```
<!DOCTYPE HTML>
<META charset="utf-8">
<TITLE>我的第一个 HTML 5 页面</TITLE>
<P>Hello,World</P>
```

3. 页面效果

该页面在 Chrome 浏览器下执行后的页面效果如图 1-3 所示。

4. 源码分析

通过短短几行代码就完成了一个页面的开发，这充分说明了 HTML 5 语法的简洁；同时，HTML 5 不是一种 XML 语言，其语法也很随意，下面从这两方面进行逐句分析。

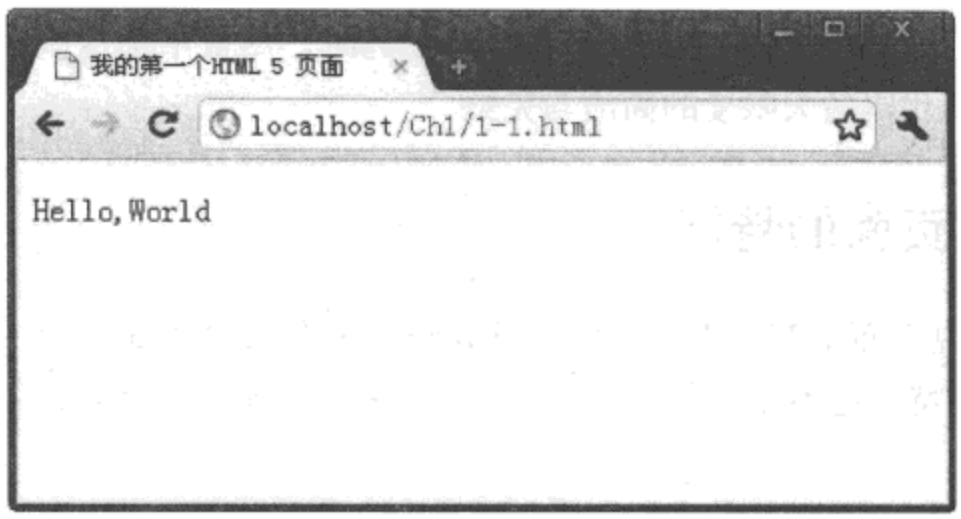


图 1-3　Hello,World 页面实现效果

第一行代码如下：

```
<!DOCTYPE HTML>
```

短短几个字符，甚至不包括版本号，就能告诉浏览器需要一个 doctype 来触发标准模式，可谓简明扼要。接下来，我们需要说明文档的字符编码，否则将出现浏览器不能正确解析，导致安全隐患的后果，为此我们加入如下一行代码：

```
<META charset="utf-8">
```

同样也是几个字符，便说明了该文档的字符编码；同时，HTML 5 不区分字母大小写、标记结束符及属性是否加引号，即下列代码是等效的：

```
<meta charset="utf-8">
<META charset="utf-8" />
<META charset=utf-8>
```

在主体中，可以省略 <html> 与 <body> 标记，直接编写需要显示的内容，代码如下：

```
<P>Hello,World</P>
```

虽然我们在编写代码时省略了 <html> 与 <body> 标记，但在浏览器进行解析时，将会自动进行添加，如图 1-4 所示。

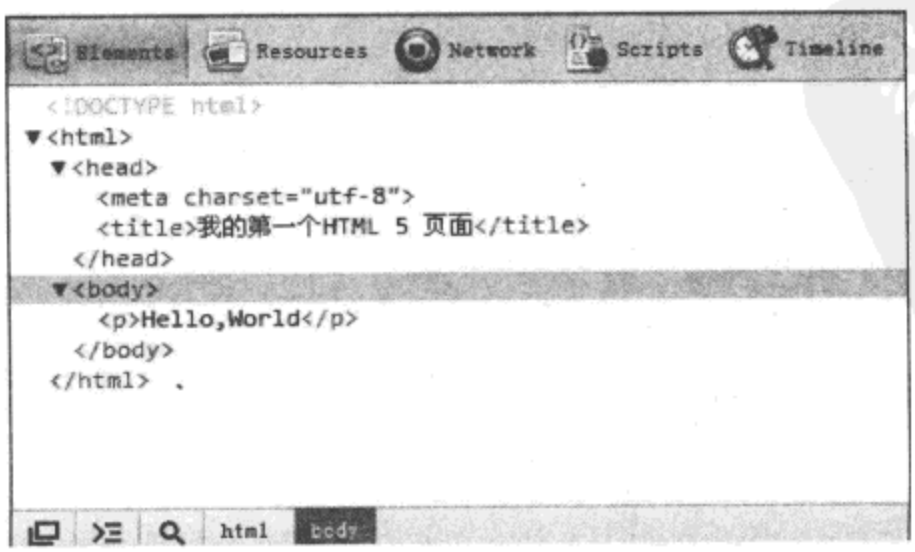


图 1-4　自动添加 <html> 与 <body> 标记后的源文件

因此，考虑到代码的可维护性，我们在编写代码时，尽量增加这些在 HTML 5 中可选的元素，从而实现页面代码最大限度的简洁与完整。

1.2 HTML 5 页面的特征

通过上节的实现页面 1-2.html，我们简单介绍了一个 HTML 5 页面的创建过程。而实际上，HTML 5 页面的特征远不止这些，下面通过一个较为完整的实例页面，来介绍 HTML 5 页面特征。

1.2.1 应用全新的 HTML 5 特征结构化元素

通过研究 Web 页面我们发现，如果使用一些带有语义性的标记，可以加速浏览器解释页面中元素的速度，如早期的 <samp>、<var> 元素；HTML 5 继承了这些元素，并根据用户使用最为频繁的类名称和 ID 号不断开发新的标记，因为这些标记才能真正体现开发者真实意图所在。

下面通过实例 1-3 说明 HTML 5 是如何使用这些全新的 HTML 5 特征来结构化元素的。

实例 1-3 页面分栏实现

1. 功能说明

将页面分成上、中、下三部分。上部用于显示导航；中部又分成两个部分，左边设置菜单，右边显示文本内容；下部显示页面版权信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 1-3.html，加入代码如代码清单 1-3 所示。

代码清单 1-3 设计页面 1-3.html 的结构

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
     TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>页面结构</title>
<style type="text/css">
  #header,#siderLeft,#siderRight,#footer
  {border:solid 1px #666;padding:5px}
  #header{width:500px}
  #siderLeft{float:left;width:60px;height:100px}
  #siderRight{float:left;width:428px;height:100px}
  #footer{clear:both;width:500px}
  </style>
  </head>
```

```
<body>
<div id="header">导航</div>
<div id="siderLeft">菜单</div>
<div id="siderRight">内容</div>
<div id="footer">底部说明</div>
</body>
</html>
```

尽管上述代码不存在任何错误，还可以在 HTML 5 环境中很好地工作，但该页面结构的很多部分对于浏览器来说都是未知的，这是因为浏览器是通过 ID 号定位元素的。因此，只要开发者不同，那么就允许元素的 ID 号各异，这对浏览器来说，不能很好地表明元素在页面中的位置，必然影响页面解析的速度。幸好，HTML 5 中新增的元素可以很快地定位某个标记，明确地表示页面中的位置。将上述代码修改成 HTML 5 支持的页面代码，如下所示：

```
<!DOCTYPE html>
<head>
<meta charset=utf-8>
<title>页面结构</title>
<style type="text/css">
 header,nav,article,footer
 {border:solid 1px #666;padding:5px}
 header{width:500px}
 nav{float:left;width:60px;height:100px}
 article{float:left;width:428px;height:100px}
 footer{clear:both;width:500px}
</style>
</head>
<body>
<header>导航</header>
<nav>菜单</nav>
<article>内容</article>
<footer>底部说明</footer>
</body>
</html>
```

3. 页面效果

虽然两段代码不一样，但在 Chrome 浏览器下实现的页面效果相同，如图 1-5 所示。

4. 源码分析

从上述两段代码来看，使用 HTML 5 新增元素创建的页面代码更加简单和高效。下面通过上述实例，详细分析两组代码的差异。

例如，原来的代码为：

```
<div id="header">导航</div>
```

修改后的代码为：

```
<header>导航</header>
```

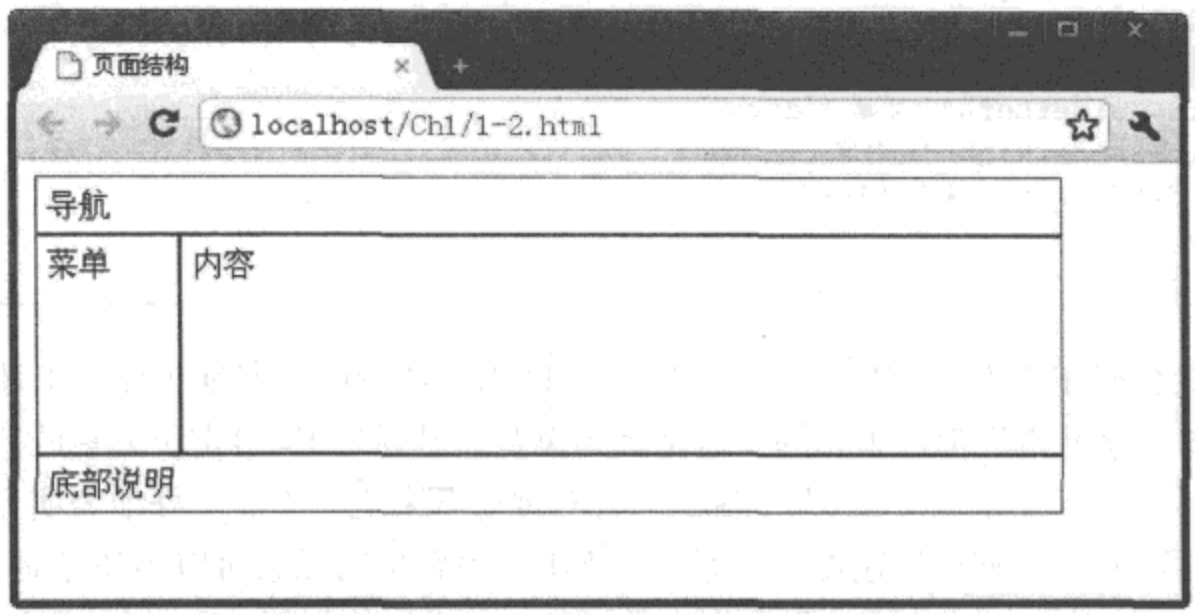


图 1-5　应用 HTML 5 元素构造页面结构

可以很容易地看出，使用 、 、 和 这些标记元素没有任何实现的意义，即浏览器不能根据标记的 ID 号属性来推断这个标记的真正含义；因为 ID 号是可以变化的，不利于寻找。

而 HTML 5 中的新增元素 <header>，明确地告诉浏览器此处是页头，<nav> 标记用于构建页面的导航，< article> 标记用于构建页面内容的一部分，< footer> 表明页面已到页脚或根元素部分，并且这些标记都可以重复使用，这样极大地提高了开发者的工作效率。

此外，有些新增的 HTML 5 元素还可以单独成为一个区域，如下列代码所示：

```
<header>
  <article>
    <h1> 内容 1</h1>
  </article>
</header>
...
<header>
  <article>
    <h2> 内容 2</h2>
  </article>
</header>
```

在 HTML 5 中，一个 <article> 可以创建一个新的节点，并且每个节点都可以有自己的单独元素，如 <h1> 或 <h2>，这样不仅使内容区域各自分段，便于维护，而且代码简单，局部修改方便。

在 HTML 5 中，同样可以使用 CSS 样式，美化新增的元素，下面通过一个简单实例加以说明。

1.2.2　使用 CSS 文件美化 HTML 5 新元素

在支持 HTML 5 新元素的浏览器中，样式化各新增元素变得十分简单，我们可以对任意一个元素应用 CSS（包括直接设置或引入 CSS 文件）。需要说明的是，在默认情况下，CSS 假设元素的“display”属性是“inline”的，因此，为了更加正确地显示设置的页面效果，我们需要将元素的“display”属性设置为“block”。下面通过一个简单实例来说明这一点。

实例 1-4　样式化页面实现

1. 功能说明

在页面中设置相关样式，显示一段文章的内容。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 1-4.html，加入代码如代码清单 1-4 所示。

代码清单 1-4　样式化 HTML 5 新元素

```
<!DOCTYPE html >
<meta charset="utf-8" />
<title>设置新元素的样式</title>
<style type="text/css">
  article{display::block}
  article header p{font-size:13px}
  article header h1{font-size:16px}
  .p-date{font-size:11px}
</style>
<article>
  <header>
    <p class="p-date">日期：2011-03-01</p>
    <h1>
      <a>
      今天的天气不错啊
      </a>
    </h1>
    <p>蓝蓝的天空中飘着几朵白云 .</p>
  </header>
</article>
```

3. 页面效果

该页面在 Chrome 浏览器下执行后的页面效果如图 1-6 所示。

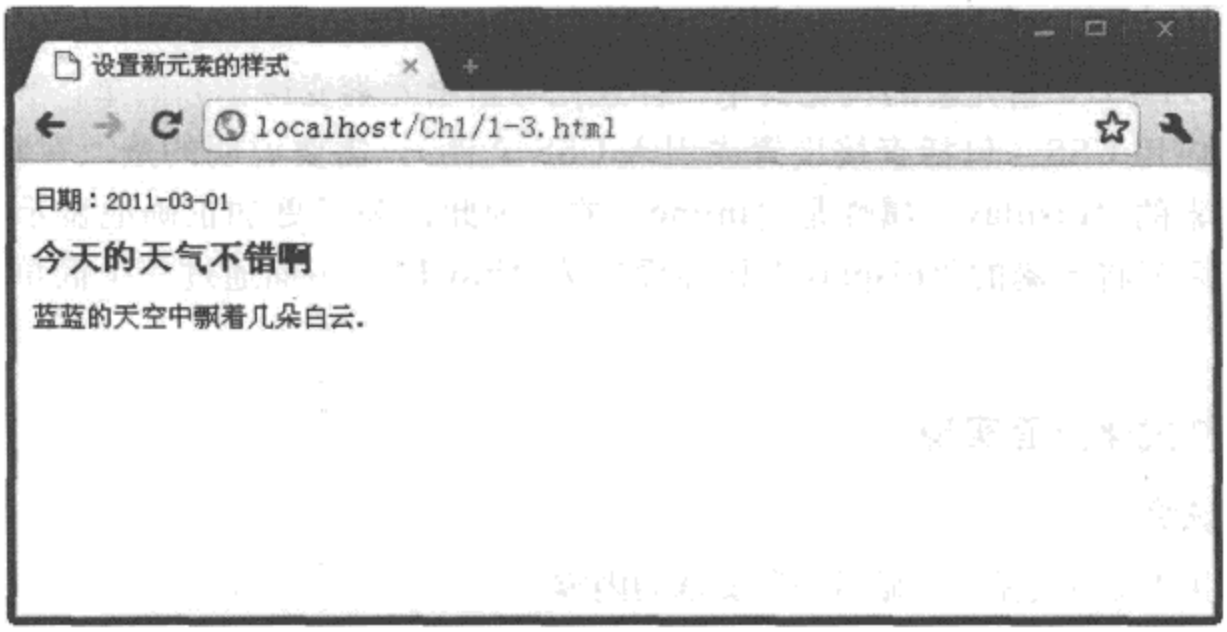


图 1-6　使用 CSS 美化 HTML 5 新元素

4. 源码分析

由于有些浏览器并不支持 HTML 5 中的新增元素，如 IE 8 或更早版本，其 CSS 只应用 IE 支持的那些元素。因此，为了能使新增的 HTML 5 元素应用到样式，可以在头部标记 <head> 中加入如下 JavaScript 代码，这样就可以应用样式了。

```
<script type="text/javascript">
 document.createElement('article');
 document.createElement('header');
</script>
```

考虑到各浏览器的兼容性不一样，可以对上述的 JavaScript 代码进行优化，即使用条件语句包含该 JavaScript 代码，使浏览器只在不支持 HTML 5 的情况下才执行这段脚本。

1.3　本章小结

本章先从检测浏览器是否支持 HTML 5 新元素开始，通过几个简单实例简要介绍了 HTML 5 新增元素在 Chrome 浏览器中的使用方法，并介绍了新增元素应用 CSS 的效果。通过本章的学习，可以对 HTLM 5 有一个大致的了解，并为后续章节的深入学习奠定基础。

页面的交互是用户操作的一个重要环节，在第 2 章中，将详细介绍每一个 HTML 5 新增交互元素。

第2章

HTML 5中常用的交互元素

本章内容

- 内容交互元素
- 菜单交互元素
- 状态交互元素
- 本章小结

HTML 5 是一些独立特性的集合，它不仅新增了许多 Web 页面特征，而且本身也是一个应用程序（Web Applications）。对于应用程序而言，表现最为突出的就是交互操作。HTML 5 为操作新增了对应的交互体验元素，如 details、summary、menu、command、key、meter、progress 等，这些元素可以在不请求服务器任何资源的情况下，改变用户选择的内容与展现状态。

在 HTML 5 中，交互元素分为三类，分别为：

- 内容交互元素
- 菜单交互元素
- 状态交互元素

通过这些新增（或改良后的）交互元素，极大地改善了 HTML 5 页面的用户体验和功能。本章将通过实例介绍几个重要的交互元素的定义、属性与使用方法。

2.1 内容交互元素

在 HTML 5 中，<details> 与 < summary> 元素属于新增的内容交互元素，主要用于文档的标题、细节、内容的交互显示。

2.1.1 details 元素

<details> 是 HTML 5 中新增的一个标记，用于说明文档或某个细节信息的作用，常与 <summary> 元素配合使用。在默认情况下，标记中的内容是不显示的，当与 <summary> 标记配合使用时，单击 <summary> 标记后，才显示 <details> 元素中设置的内容。

<details> 元素的常用属性及描述如表 2-1 所示。

表 2-1 <details> 元素的属性及描述

属 性	值	描 述
open	open	用于控制 <details> 元素是否显示，默认值为不可见
subject	sub_id	用于设置元素所对应的项目 ID 号
draggable	true/false	设置是否可以拖动元素，默认值为 false

下面通过两个实例分别介绍使用 CSS 样式和 JavaScript 代码控制 <details> 元素的“open”属性。

在 HTML 5 中，使用 CSS 样式控制页面元素的方法十分频繁，接下来先通过实例 2-1 介绍如何使用 CSS 样式代码，调用元素属性的方法。

实例 2-1　交互元素 <details> 的使用

1. 功能描述

在新创建的页面中加入一个 <details> 元素，通过不设置该元素的 open 属性值与设置该属性值为“open”进行比较，并将结果展示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 2-1.html，加入代码如代码清单 2-1 所示。

代码清单 2-1　交互元素 <details> 的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title> 交互元素 <details> 的使用 </title>
<style type="text/css">
  body{
       font-size:12px
  }
  span{
    font-weight:bold
  }
  details {
    overflow: hidden;
    height: 0;
    padding-left:200px;
    position: relative;
    display: block;
  }
  details[open] {
    height:auto;
  }
</style>
</head>
<body>
  <span> 隐藏脚注 </span>
  <details>
    本页面生成于 2011-03-17
  </details>

  <!--<span> 显示脚注 </span>
  <details open="open">
    本页面生成于 2011-03-17
    </details>-->
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 2-1 所示。

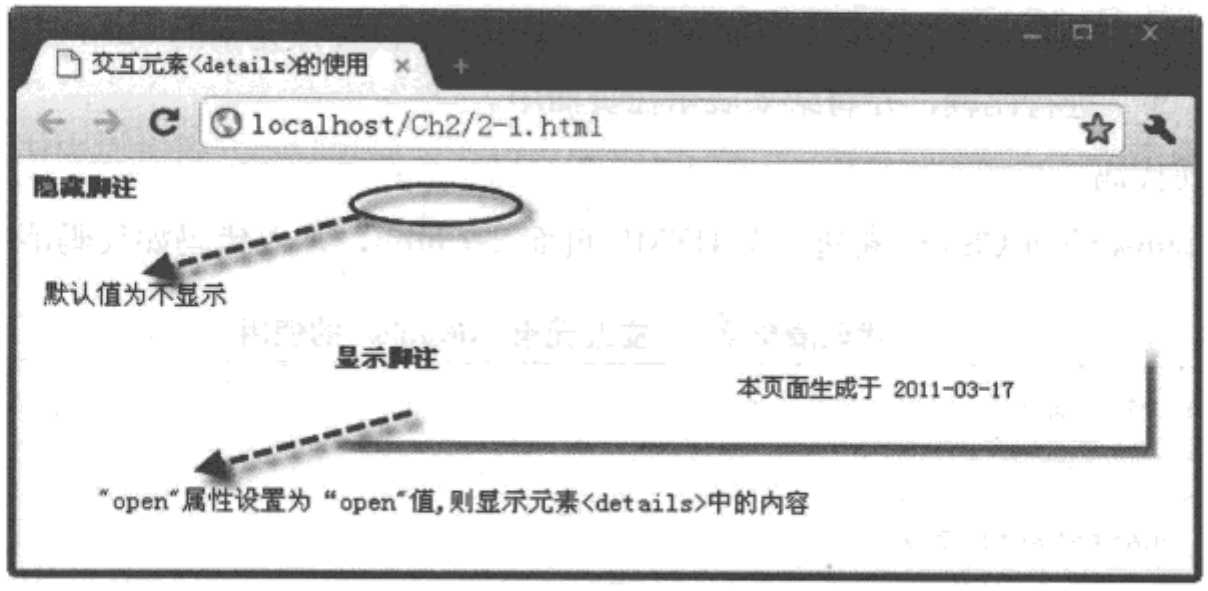


图 2-1　使用交互元素 <details> 的页面效果

4. 源码分析

为了能更好地验证 <details> 元素的“open”属性，在页面的样式中分别定义了该元素的默认样式和显示状态的样式，代码如上面代码中的加粗部分所示。

除了可以使用 CSS 样式控制 <details> 元素的属性外，还可以通过 JavaScript 代码来获取 <details> 元素的“open”属性，可通过设置该属性的值来控制元素内容显示的状态。

下面通过一个完整的实例 2-2 来介绍如何使用 JavaScript 代码来控制 <details> 元素的内容显示。

实例 2-2　用脚本控制交互元素 <details> 的使用

1. 功能描述

在新建的页面中加入一个 与 <details> 元素，单击 标记时，将根据 <details> 元素的属性值显示元素内容。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 2-2.html，加入代码如代码清单 2-2 所示。

代码清单 2-2　用脚本控制交互元素 <details> 的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title>脚本控制交互元素 <details></title>
...省略页面样式内容
```

```
</head>
<body>
  <span onClick="span1_click();">脚注</span>
  <details id="details1">
    本页面生成于 2011-03-17
  </details>
  <script type="text/javascript">
//根据属性控制内容是否显示
function span1_click(){
  var objD=document.getElementById("details1");
  var attD=objD.getAttribute("open");

  if(attD!="open"){
      objD.setAttribute("open","open");
    }else{
      objD.removeAttribute("open");
  }
}
</script>
</body>
</html>
```

① ②

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 2-2 所示。

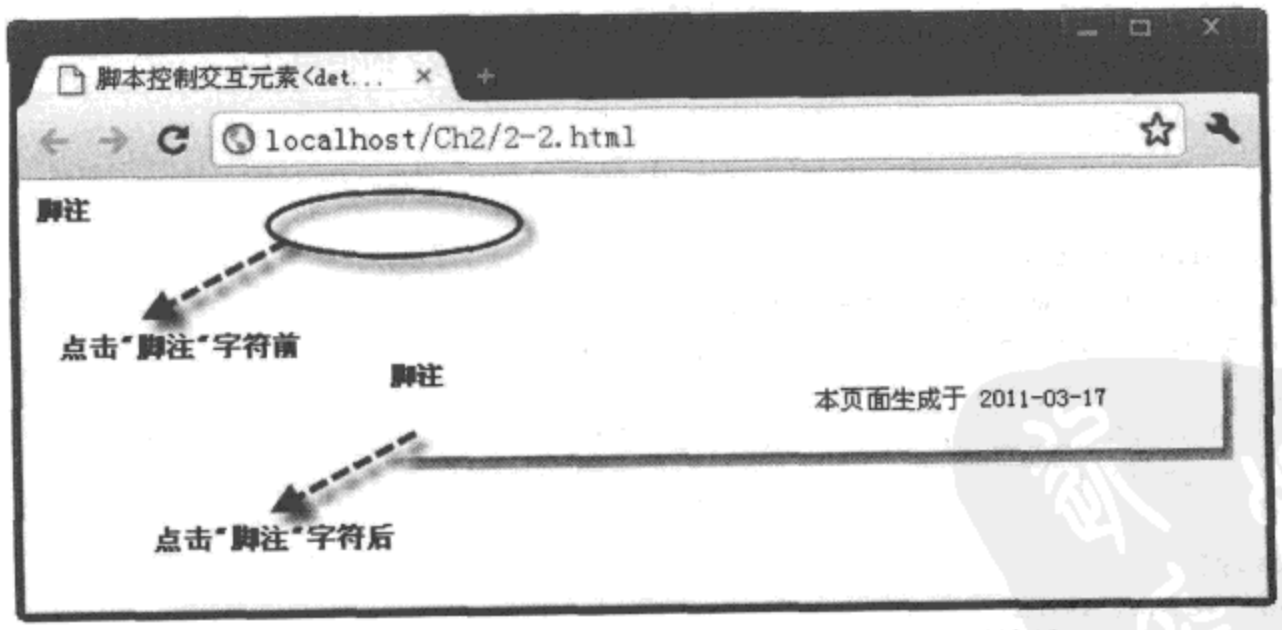


图 2-2　使用脚本控制交互元素 <details> 的页面效果

4. 源码分析

在实例 2-2 的 JavaScript 代码中，首先，利用 getAttribute() 方法获取 <details> 元素的“open”属性，实现的代码如上面的代码片段①所示。

然后，判断获取的元素属性值，当该值为“open”时，利用 removeAttribute() 方法删除

<details> 元素的“open”属性；反之，使用 setAttribute() 方法增加该属性，其实现的代码如代码片段②所示。

从实例 2-1 与实例 2-2 中我们不难看出，在 HTML 5 中，交互元素 <details> 的属性可以很方便地通过 CSS 或 JavaScript 代码进行访问或设置。

2.1.2 summary 元素

<summary> 是 HTML 5 中新增的一个标记，常包含于 < details> 元素中，配合 < details> 元素使用。在两者结合起来使用的代码中，<summary> 元素说明文档的标题，< details> 元素用于说明文档的详细信息。<summary> 元素是 < details> 元素中的第一个子元素，二者经常同时出现在页面中。

下面通过一个完整的实例 2-3 介绍 <summary> 元素与 <details> 元素结合使用的方法。

实例 2-3 交互元素 <summary> 与 <details> 的结合使用

1. 功能描述

在新建的页面中分别加入一个 <details> 元素和一个 <summary> 元素，当显示 <details> 元素内容时，其子元素 <summary> 以字体加粗的形式展示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 2-3.html，加入代码如代码清单 2-3 所示。

代码清单 2-3 交互元素 <summary> 的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title> 交互元素 <summary> 的使用 </title>
<style type="text/css">
  body{
    padding:5px;
    font-size:12px
  }
  summary{
    font-weight:bold;
  }
  ... 省略部分实例重复样式代码
</style>
</head>
<body>
  <details open="open">
    <summary> 页面说明 </summary>
    本页面生成于 2011-03-17
```

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

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 2-3 所示。

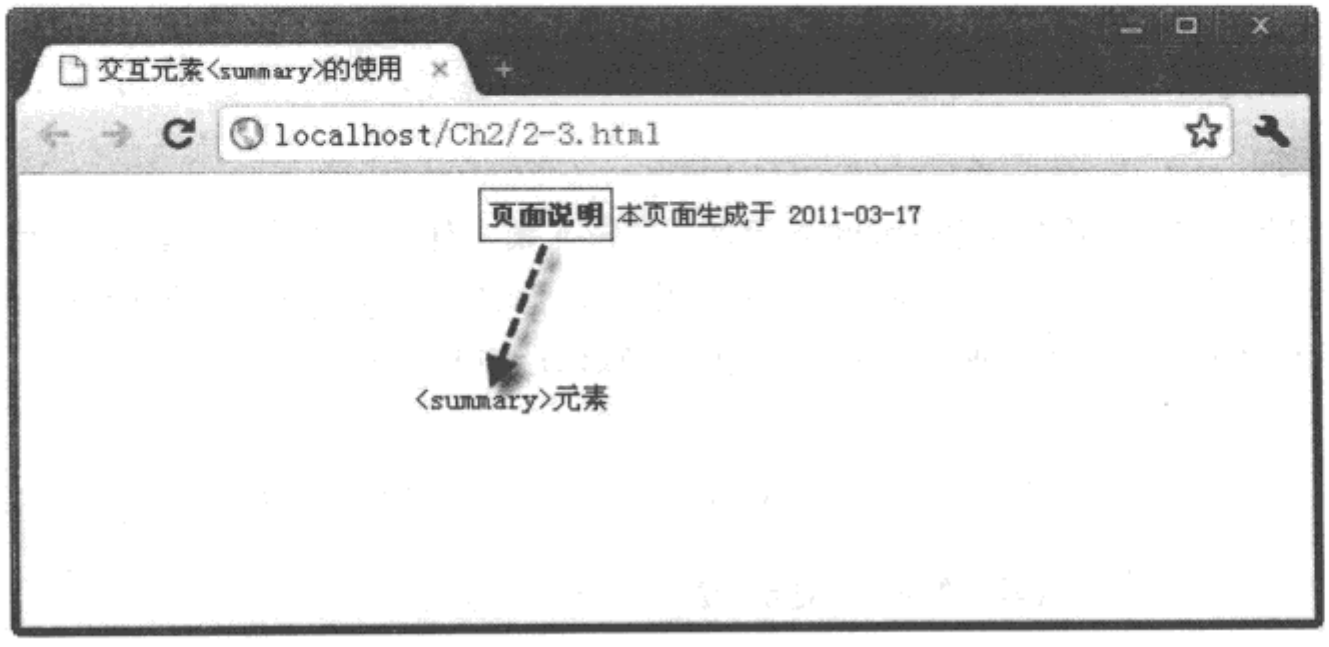


图 2-3　使用交互元素 <summary> 的页面效果

4. 源码分析

在页面样式文件中，为了突出显示 <summary> 元素，增加了一个加粗的字体效果，如上面的加粗代码所示。从代码的结构中不难看出，<summary> 元素包含在 <details> 元素中，是 <details> 元素的子元素，在摆放位置时尽量放在第一个。

2.2　菜单交互元素

在 HTML 5 的交互元素中，除内容交互元素外，使用较为频繁的是菜单交互元素，其中以 <menu> 与 <command> 两个元素为代表。接下来详细介绍这两个交互元素的使用方法。

2.2.1　menu 元素

<menu> 是 HTML 5 中重新启用的一个旧标记。<menu> 元素在 HTML 2 时就已经存在，但是在 HTML 4 时曾一度被废弃，而在 HTML 5 中重新恢复使用，并赋予了新的功能含义。该元素常与 <li> 列表元素结合使用，用来定义一个列表式的菜单。其属性及描述如表 2-2 所示。

表 2-2 <menu> 元素的属性及描述

属 性	值	描 述
autosubmit	true 或 false	如果属性值为 true，当表单中的元素发生变化时会自动提交
label	任意字符	为菜单定义一个可见的标注
type	context toolbar list	定义菜单显示的类型，默认值为 list，即列表显示菜单中的选项

下面通过一个完整的实例 2-4 介绍 <menu> 元素的基本使用方法。

实例 2-4 交互元素 <menu> 的使用

1. 功能描述

新建一个 html 页面，首先添加 <menu> 元素，在该元素中加入 <li> 列表元素；然后，在列表元素中分别放置一个 <img> 与 元素，用于展示图片与标题；最后，通过样式代码，当用户将鼠标移至某个 <li> 元素时，展示菜单中某选项被选中的效果。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 2-4.html，加入代码如代码清单 2-4 所示。

代码清单 2-4 交互元素 < menu> 的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title> 交互元素 <menu> 的使用 </title>
<style type="text/css">
  body{
    padding:5px;
    font-size:12px
  }
  menu{
    padding:0;
    margin:0;
    display:block;
    border: solid 1px #365167;
    width:222px
  }
menu li{
    list-style-type:none;
    padding:5px;
    margin:5px;
    height:28px;
    width:200px
  }
  menu li:hover{
```

```
    border: solid 1px #7DA2CE;
    background-color:#CFE3FD
  }
  menu li img{
    clear:both;
    float:left;
    padding-right:8px;
    margin-top:-2px
  }
  menu li span{
    padding-top:5px;
    float:left;
    font-size:13px
  }
</style>
</head>
<body>
  <menu>
    <li>
      <img src="Images/firefox-32.png"></img>
      <span>Mozilla Firefox</span>
    </li>
    <li>
      <img src="Images/chrome.png"></img>
      <span>Google Chrome</span>
    </li>
    <li>
      <img src="Images/safari.png"></img>
      <span>Safari</span>
    </li>
  </menu>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 2-4 所示。

4. 源码分析

在使用 <menu> 定义菜单列表时，通常使用 <menu> 元素来定义菜单的框架，框架中的内容使用 <li> 元素来进行构造，以形成列表形状；如页面中加粗代码所示。另外，为了美化列表的展示效果，需要借助 CSS 样式，例如样式代码中的加粗部分，表示通过 CSS 样式控制鼠标在移出与移入元素时的不同展示效果。注意，菜单还可以嵌套在别的菜单中，形成带层次的菜单结构。

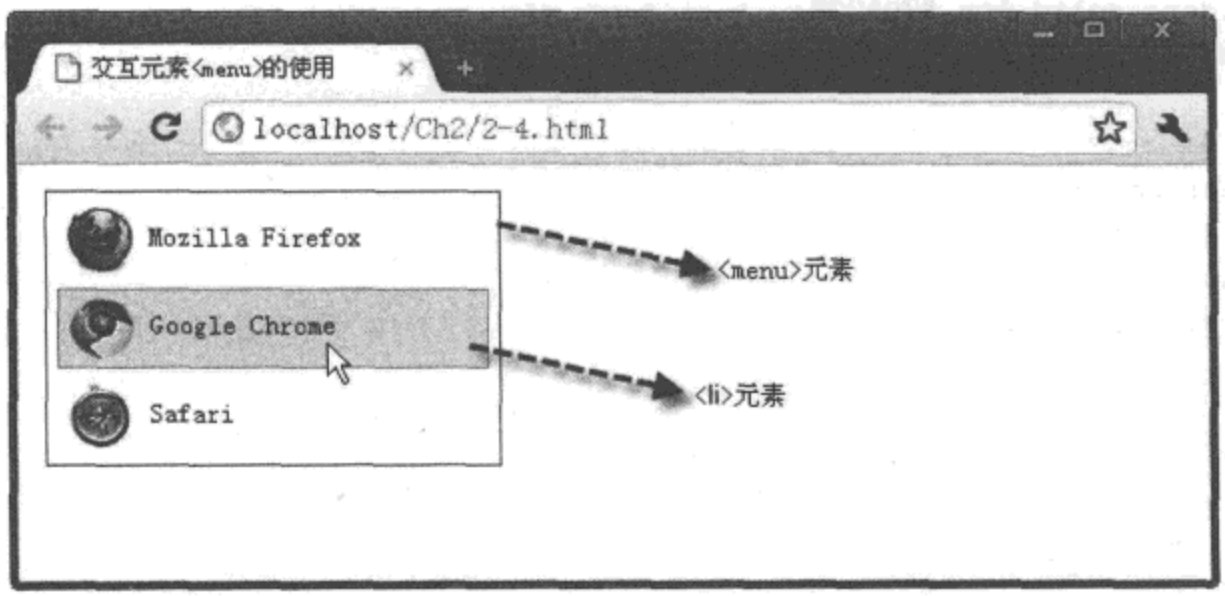


图 2-4　使用交互元素 <menu> 的页面效果

2.2.2　command 元素

<command> 元素是 HTML 5 新增的标记，用于定义各种类型的命令按钮。利用该标记的“url”属性可以添加图片，并且实现图片按钮效果；另外，改变标记中的“type”属性值，还可以定义复选框或单选框按钮。<command> 元素的属性及描述如表 2-3 所示。

表 2-3　<command> 元素的属性及描述

属　性	值	描　述
checked	checked	如果类型值定义为 checkbox 或 radio，则该属性表示是否被选中
disabled	disabled	设置按钮是否可用
icon	url	设置按钮中图片地址
label	command name	定义可显示于页面中按钮的名称
radiogroup	radiogroup name	如果类型定义为 radio 时，可以通该属性设置所属群组
type	checkbox command radio	设置按钮的类型，默认值为 command

<command> 是 HTML 5 中新增加的一个重要元素，用于定义各种类型的按钮（包括命令按钮、单选择按钮、图片按钮）及复选框。如果该元素与 <menu> 元素结合使用，可以实现弹出式的下拉菜单，当单击菜单中的某个选项时，将执行相应的操作。下面通过一个完整的实例 2-5 进行详细的说明。

实例 2-5　交互元素 <command> 与 <menu> 的结合使用

1. 功能描述

在新建的页面中分别添加一个 <menu> 和两个 <command> 元素，并将 <command> 元素包含于 <menu> 中；当单击其中一个 <command> 元素时，弹出一个对话框，并且显示对应操

作的内容。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 2-5.html，加入代码如代码清单 2-5 所示。

代码清单 2-5　交互元素 <command> 与 <menu> 的结合使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8 />
<title> 交互元素 <menu> 与 <command> 的结合使用 </title>
<style type="text/css">
  body{
    padding:5px;
    font-size:12px
  }
  menu{
    padding-left:10px;
    padding-bottom:10px;
    display:block;
    border: solid 1px #365167;
    width:40px;
    height:50px
  }
command{
  float:left;
  margin:5px;
  width:30px;
  cursor:hand;
}
#dialog{
  display:none;
  position:absolute;
  left:25%;
  top:9%;
  font-size:13px;
  width:320px;
  height:150px;
  border:#666 solid 3px
}
#dialog .title{
  padding:5px;
  background-color:#eee;
  height:21px;
  line-height:21px
}
#dialog .title .fleft{
```

```
    float:left
  }
  #dialog .title .fright{
    float:right
  }
  #dialog .content{
    padding:50px
  }
</style>
</head>
<body>
  <menu>
    <command onClick="command_click('文件')">文件</command>
    <command onClick="command_click('打开')">打开</command>
  </menu>
  <div id="dialog">
    <div class="title">
      <div class="fleft">提示</div>
      <div class="fright">关闭</div>
    </div>
    <div class="content">
      <div id="divTip"></div>
    </div>
  </div>
  <script type="text/javascript">
  function command_click(strS){
    document.getElementById("dialog").style.display="block";
    var strContent="正在操作<font color=red> "+strS+" </font>选项";
    document.getElementById("divTip").innerHTML=strContent;
  }
  </script>
</body>
</html>
```

3. 页面效果

由于 Chrome 10 浏览器不能很完美地展示 <command> 元素中的属性与方法，此处利用 Firefox（版本号为 3.6.16）浏览器进行展示，执行页面效果如图 2-5 所示。

4. 源码分析

在页面中，<command> 元素包含于 <menu> 元素中，同时，为了使元素显示手状的被单击效果，加入了如样式中粗体所示的代码；另外，<command> 元素被单击时，弹出对话框，显示操作内容，详细如 JavaScript 代码中的加粗部分所示。

<command> 元素除了可以触发 onClick 事件外，还可以通过 icon 属性设置按钮图片，代码如下所示：

```
<command icon="Images/chrome.png" label="带图片的按钮"></command>
```

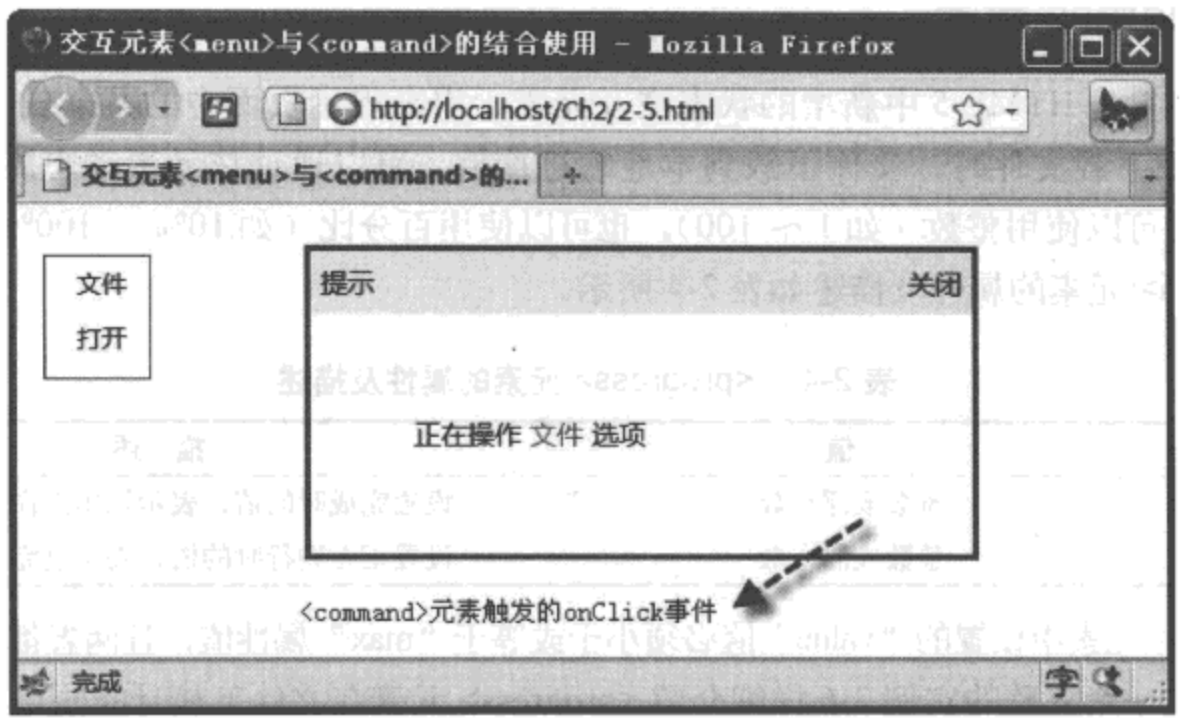


图 2-5　交互元素 <menu> 与 <command> 结合使用的页面效果

上述代码创建了一个带图片的 <command> 元素，并且指定了元素的名称是“带图片的按钮”；此外，还可以通过 JavaScript 代码控制 <command> 元素的“disabled”属性，代码如下所示：

```
<script type="text/javascript">
  ...省略其他代码
  document.getElementsByName("command").disabled="disabled";
  ...省略其他代码
</script>
```

上述 JavaScript 代码的功能是全部的 <command> 元素都将禁止单击，即不可用，这段代码常常被放置在单击 <command> 元素操作某项功能后，防止用户反复单击或提示用户按钮已经单击成功。

说明　虽然各浏览器对 HTML 5 兼容性都进行了很好的支持，但毕竟不可能照顾到每个元素的全部属性，例如 <command> 元素就有许多属性不能被浏览器支持，因此，我们所提到的功能也只是 HTML 5 元素所具有的功能，暂时还不能真正执行，但随着各大浏览器厂商对 HTML 5 的兼容性力度的加强，这种暂时不兼容的功能终将解决。

2.3　状态交互元素

在 HTML 5 中，交互元素还包含状态交互元素。所谓状态交互元素，表示页面在与用户进行数据交互时，为了增强用户的 UI 体验，显示在页面中的各种进度状态。状态交互元素包括 <progress> 与 <meter> 元素，下面分别进行详细的介绍。

2.3.1 progress 元素

<progress> 是 HTML 5 中新增的状态交互元素，用来表示页面中的某个任务完成的进度（进程）。例如下载文件时，文件下载到本地的进度值，可以通过该元素动态展示在页面中，展示的方式既可以使用整数（如 1 ～ 100），也可以使用百分比（如 10% ～ 100%）。

<progress> 元素的属性及描述如表 2-4 所示。

表 2-4　<progress> 元素的属性及描述

属　性	值	描　述
max	整数或浮点数	设置完成时的值，表示总体工作量
value	整数或浮点数	设置正在进行时的值，表示已完成的工作量

<progress> 元素中设置的“value”值必须小于或等于“max”属性值，且两者都必须大于 0。

下面通过一个完整的实例 2-6 详细介绍 <progress> 元素在文件下载时的使用。

实例 2-6　交互元素 <progress> 的使用

1. 功能描述

在页面中创建一个 <progress> 元素和一个“下载”按钮。当单击按钮时，元素 <progress> 动态展示下载进度状态和百分比信息；下载结束时，提示“下载完成！”。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 2-6.html，加入代码如代码清单 2-6 所示。

代码清单 2-6　交互元素 <progress> 在下载中的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>progress 元素在下载中的使用 </title>
<style type="text/css">
body {
    font-size:13px
}
p {
    padding:0px;
    margin:0px
}
.inputbtn {
    border:solid 1px #ccc;
    background-color:#eee;
    line-height:18px;
    font-size:12px
}
```

```
</style>
</head>
<body>
 <p id="pTip">开始下载</p>
 <progress value="0" max="100" id="proDownFile"></progress>
 <input type="button" value="下载"
        class="inputbtn" onClick="Btn_Click();">
<script type="text/javascript">
  var intValue = 0;
  var intTimer;
  var objPro = document.getElementById('proDownFile');
  var objTip = document.getElementById('pTip');
  // 定时事件
  function Interval_handler() {
    intValue++;
    objPro.value = intValue;
    if (intValue >= objPro.max) {
        clearInterval(intTimer);
        objTip.innerHTML = "下载完成!";
    } else {
        objTip.innerHTML = "正在下载" + intValue + "%";
    }
  }
  // 下载按钮单击事件
  function Btn_Click(){
    intTimer = setInterval(Interval_handler, 100);
  }
</script>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 2-6 所示。

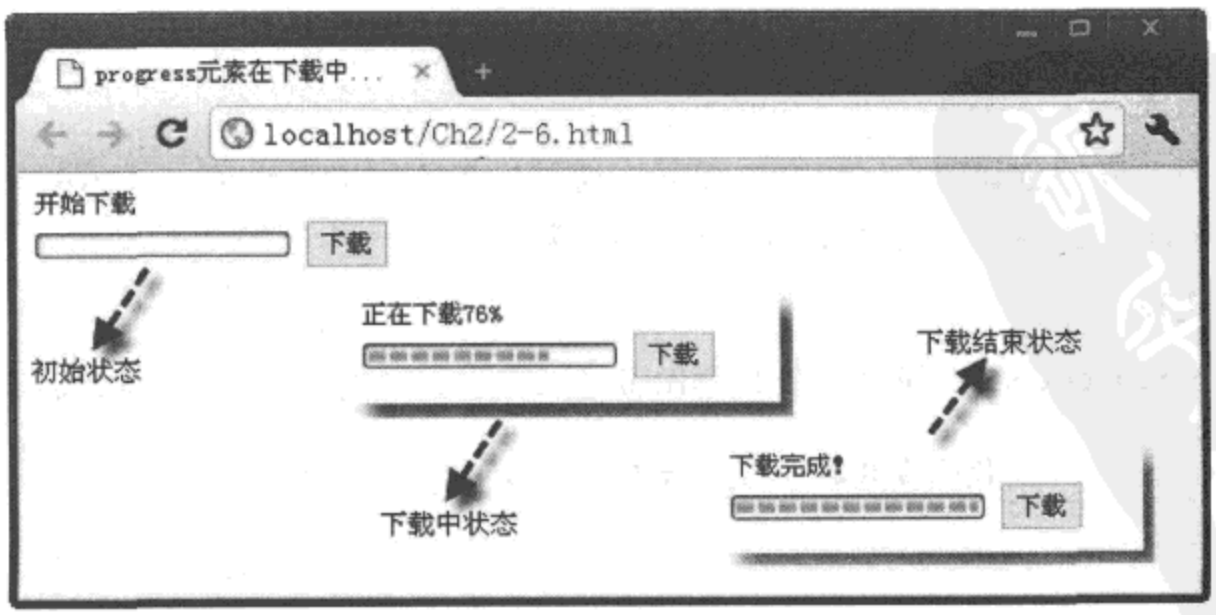


图 2-6　交互元素 <progress> 在下载中使用的页面效果

4. 源码分析

在实例 2-6 中，为了使 <progress> 元素能动态展示下载进度，需要通过 JavaScript 代码编写一个定时事件。在该事件中，累加变量值，并将该值设置为 <progress> 元素的“value”属性值；当这个属性值大于或等于 <progress> 元素的“max”属性值时，则停止累加，并显示“下载完成！”的字样；否则，动态显示正在累加的百分比数，详细源码见 JavaScript 代码中的加粗部分。

2.3.2 meter 元素

<meter> 是 HTML 5 中新增的标记，用于表示在一定数量范围中的值，如投票中，候选人各占比例情况及考试分数等。<meter> 元素仅是帮助浏览器识别 HTML 中的数量，而不对该数量做任何的格式修饰，元素中的 6 个属性会根据浏览器的特征，以最好的方式展示这个数量。

<meter> 元素的属性及描述如表 2-5 所示 。

表 2-5 <meter> 元素的属性及描述

属 性	值	描 述
value	数量	定义元素展示的实际值，可以为浮点数，默认值为 0
min	数量	定义元素展示的最小值，默认值为 0
max	数量	定义元素展示的最大值，默认值为 1
low	数量	定义元素展示的最低值，该值必须小于或等于 min
high	数量	定义元素展示的最高值，该值必须小于或等于 max
optimum	数量	定义元素展示的最优值，该值必须在 min 与 max 值之间

注意，<meter> 元素中“optimum”的属性值，表示的是最佳数量值。如果该值比属性“high”值大，表示实际值越高越好；如果该值比属性“low”值小，则表示实际值越低越好。

下面通过一个完整的实例 2-7 详细介绍 <meter> 元素在展示投票结果时的使用。

实例 2-7 交互元素 <meter> 的使用

1. 功能描述

在页面中，分别创建两个 <meter> 元素，通过设置不同的属性值，展示两个候选人的占票数比例，同时，用 元素说明比例的百分数。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 2-7.html，加入代码如代码清单 2-7 所示。

代码清单 2-7 交互元素 <meter> 在投票中的使用

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="utf-8" />
<title>meter 元素在投票中的使用 </title>
<style type="text/css">
body {
    font-size:13px
}
</style>
</head>
<body>
  <p> 共有 120 人参与投票，明细如下：</p>
  <p> 张三：
  <meter value="0.30" optimum="1"
         high="0.9" low="1" max="1" min="0">
  </meter>
  <span> 30% </span></p>
  <p> 李四：
  <meter value="70" optimum="100"
         high="90" low="10" max="100" min="0">
  </meter>
  <span> 70% </span></p>
</body>
</html>
```

①

②

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 2-7 所示。

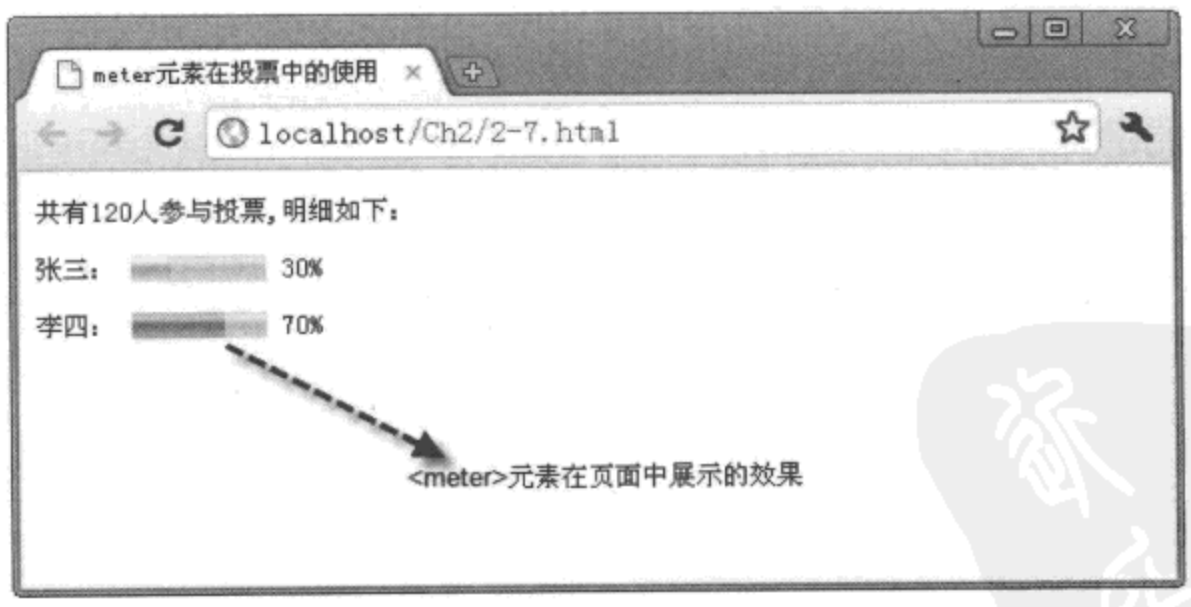


图 2-7　交互元素 <meter> 在投票中使用的页面效果

4. 源码分析

在 HTML 源码中的第二处加粗代码中，候选人“李四”所占的比例是百分制中的 70，最低比例可能为 0，但实际最低为 10；最高比例可能为 100，但实际最高为 90。<meter> 元素中

的数量也可以使用浮点数表示，如第一处加粗代码所示。为了展示这些比例值，可以引入其他的元素，例如本实例中使用了 元素展示这些数值。

2.4 本章小结

页面中的交互操作是 HTML 5 新增的一项重要功能。本章通过介绍 HTML 5 的几个重要的交互元素的使用方法，可以使读者进一步加深对 HTML 5 元素的了解。

当然，HTML 5 中新增加的元素远不止这些交互元素，在后续章节中将着重介绍一些常用的功能性很强的重要元素。

第3章 HTML 5中的重要元素

本章内容

- html根元素
- 文档元素
- 脚本
- 节点
- 分组内容
- 文本层次语义
- 嵌入内容
- 公共属性
- 本章小结

PDG

在第 2 章中，介绍了 HTML 5 中的交互元素。实际上，交互元素是 HTML 5 新增元素的一部分。在 HTML 5 中，新增或改良的元素还有许多，主要分为以下几大类：文档元素、脚本、节点元素、文本元素、嵌入元素等。下面以实例的形式，分别介绍各元素在 HTML 5 中的使用方法。

3.1 html 根元素

在 HTML 文档中，元素 <html> 代表了文档的根，其他所有的元素都是在该元素的基础上进行延伸或拓展的。该元素也是 HTML 文档的最外层元素，因此也称为**根元素**。

<html> 元素在浏览器执行页面时，告知它执行的是一个 HTML 文档。在 HTML 5 与 HTML 4.0.1 中，该元素的差异不大，主要区别在于 xmlns 属性。在 HTML 4.0.1 中，该属性是必需的，因为它将在 HTML 转成 xmlns 的过程中发挥作用；而在 HTML 5 中，可以不必增加这个属性。另外，HTML 5 中新增加了一个属性“manifest”，用于指向文档缓存信息 URL。

<html> 元素的常用属性及描述如表 3-1 所示。

表 3-1 < html > 元素的属性及描述

属 性	值	描 述
manifest	URL	该 URL 指向描述文档缓存信息的地址
xmlns	http://www.w3.org/1999/xhtml	设置 XML namespace 的属性

下面通过简单的实例 3-1 介绍 <html> 元素的详细使用过程。

实例 3-1 元素 <html> 的使用

1. 功能描述

在新建的页面中显示“内容部分…”几个字符。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-1.html，加入代码如代码清单 3-1 所示。

代码清单 3-1 元素 <html> 的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>主题</title>
<style type="text/css">
   body{
      font-size:10pt
   }
```

```
</style>
</head>
<body>
内容部分 ...
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-1 所示。

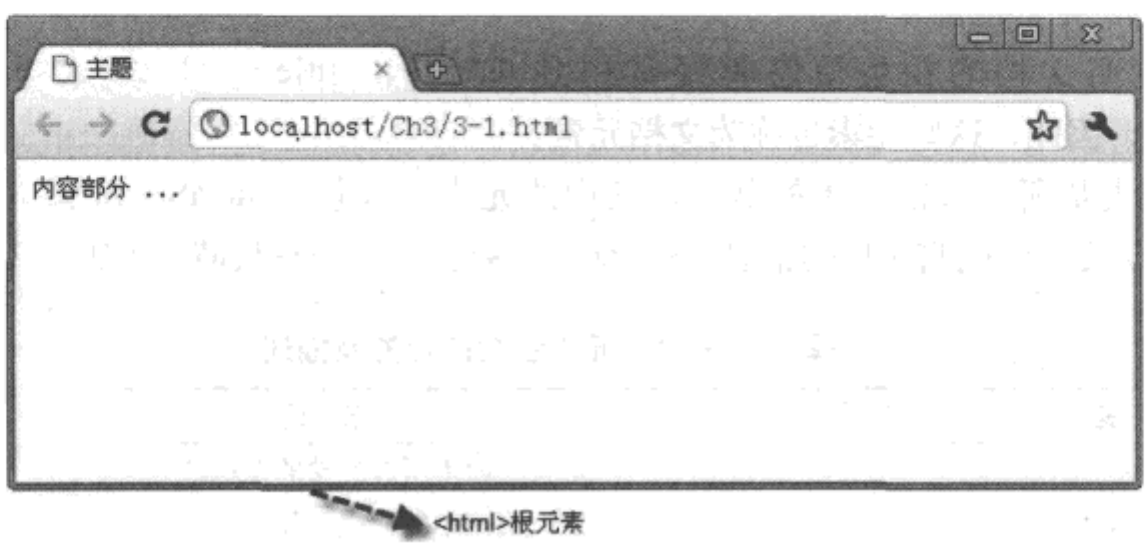


图 3-1　使用 <html> 元素的页面效果

4. 源码分析

在代码清单 3-1 中，<html> 元素是最外层元素，它包括两个主要的部分——头部 <head> 与主体 <body>。头部又包含三个元素，其中 <meta> 元素的“charset”属性描述页面的字符集格式为“utf-8”类型，<title> 元素说明页面的标题内容，<style> 元素中设置了页面中 <body> 元素的字体样式。在主体 <body> 部分中直接输入页面中需要显示的文字。其页面结构如图 3-2 所示。

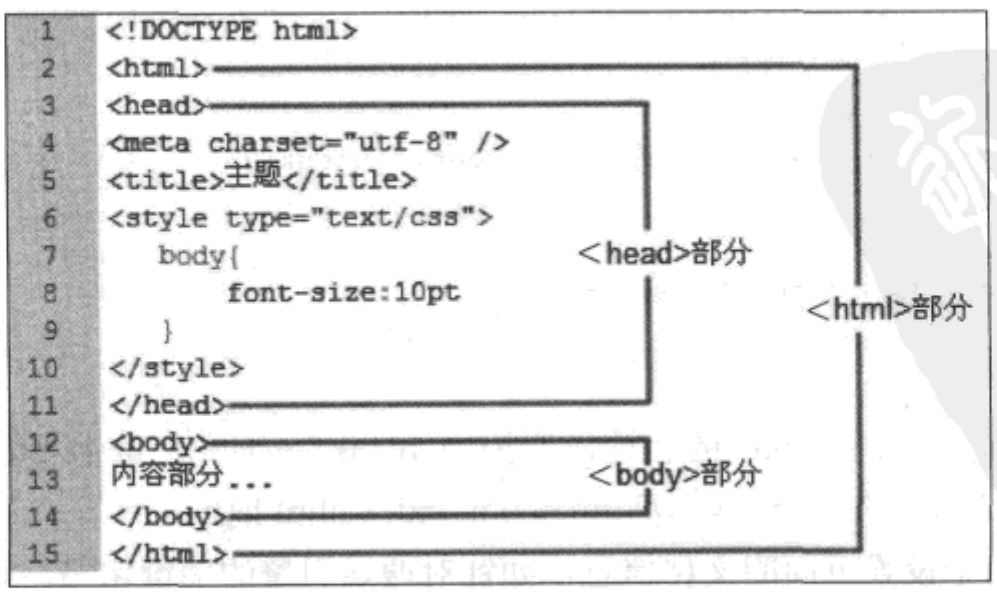


图 3-2　<html> 元素所包含的结构

说明 在 HTML 5 中，如果要设置 <html> 元素的 xmlns 属性，该属性值必须是 http://www.w3.org/1999/xhtml，因为只有该属性值符合 W3C 规范命名空间，才可以用于整个页面文档的元素。

3.2 文档元素

一个 HTML 文档通常包含两个部分：一个为头部分，由 <head> 元素包含；另一个为主体部分，由 <body> 元素包含。

由于在说明文档的相关信息时经常用到 <head>、<title>、<base>、<link>、<meta>、<style> 等元素，因此，这些元素也称为**文档元素**。

<head> 元素是所有头部元素的载体，其中的元素可以包含 JavaScript 脚本或文件、CSS 样式或文件，以及其他说明文档的信息。<head> 元素包含的标签及描述如表 3-2 所示。

表 3-2 < head > 元素包含的标签及描述

标 签	描 述
base	定义页面中所有链接的默认地址或目标
link	导入页面中的样式文件
meta	定义页面中的相关信息
script	定义客户端脚本代码或文件
style	定义 HTML 文档中的样式信息
title	设置文档的标题内容

<base> 元素可以设置页面中 URL 为空时的值。该元素有两个属性：一个是 href，表示当页面的 URL 为空时的链接地址；另一个是 target，表示打开页面链接的方式，常用方式有“_blank”、“_self”、“_parent”等，其中“_blank”表示在新窗口中打开，“_self”表示在当前窗口中打开，“_parent”表示在父窗口中打开。下面用一个例子说明该属性的用法。

```
...省略部分代码
<head>
  <base href="http://www.html5.com/" target="_blank"/>
</head>
<body>
  <a href="index.html">testPage</a>
</body>
...省略部分代码
```

如上述代码所示，由于 target 的属性设置为“_blank”，因此，单击“testPage”字符链接时，将以新窗口的方式打开 http://www.html5.com/ index.html 地址。

<meta> 元素用于设置页面的文档信息，如针对搜索引擎的关键字等。在 HTML 5 中，该元素不再支持 scheme 属性；同时，该元素新增了一个 charset 属性，在该属性中简化了很多冗

余的字符，如“http-equiv="Content-Type" content="text/html;”，只需要加入“utf-8”即可，使用页面字符集的定义更加方便。下面看一个使用 <meta> 元素的例子。

```
...省略部分代码
<head>
  <meta name="keywords" content="HTML5, UI, Web 前端开发 "
              charset="utf-8"/>
</head>
...省略部分代码
```

上述代码通过 <meta> 元素定义了针对搜索引擎的关键字，值为“HTML5, UI, Web 前端开发”，便于搜索引擎对该页面的检索；同时，使用“charset”属性指定页面的字符集。

下面通过实例 3-2 介绍 <head> 元素包含其他文档信息元素的方法。

实例 3-2　元素 <head> 的使用

1. 功能描述

在新建的页面 <head> 元素中，加入该元素所包含的各类标签，并定义超级链接的样式。当单击“请点击我”标签时，将展示相应样式效果并进入 <base> 元素设置的默认地址。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-2.html，加入代码如代码清单 3-2 所示。

代码清单 3-2　文档元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>文档元素的使用</title>
<base href="http://www.html5.com/" target="_blank"/>
<meta name="keywords" content="HTML5, CSS, JavaScript"/>
<meta name="description" content="用于检测页面的文档元素"/>
<style type="text/css">
  a{
     padding:8px;
     font-size:13px;
     text-decoration:none;
  }
  a:hover{
     border:solid 1px #ccc;
    background-color:#eee;
  }
</style>
</head>
<body>
<a href="index.html">请点击我</a>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-3 所示。

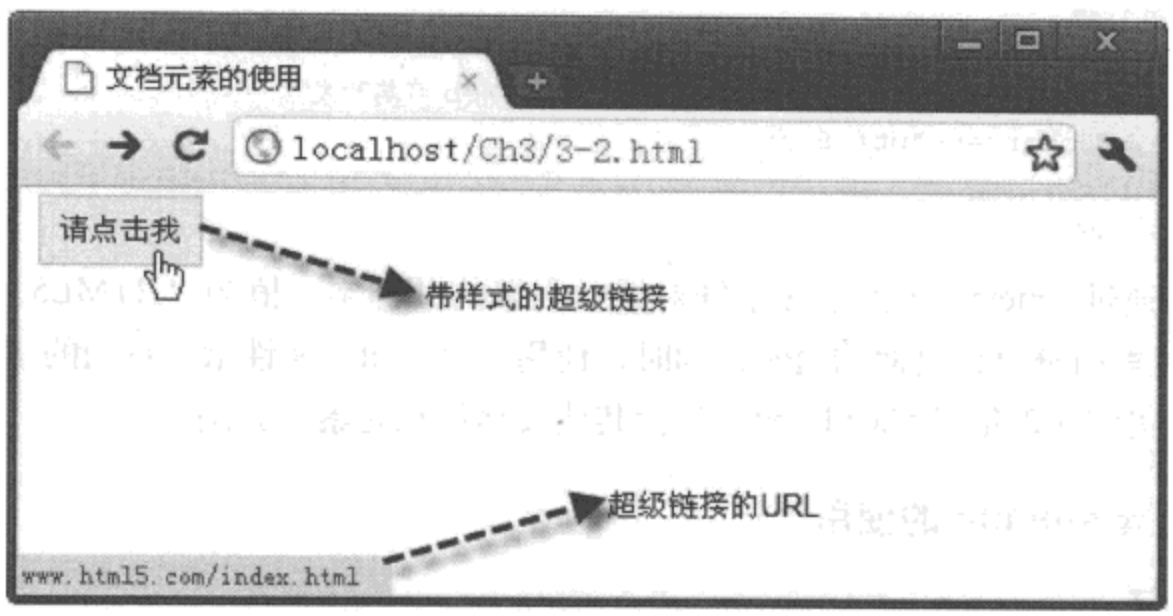


图 3-3　使用文档元素的页面效果

4. 源码分析

在 <head> 元素所包含的文档元素中，许多元素增加了用于 HTML 5 中执行的新属性，同时，原有属性中，有些属性在 HTML 5 中不再被支持。例如，<link> 元素在 HTML 5 中不再支持的属性有 charset、rev、target，其他属性同样可以在 HTML 5 中执行。

在 <base> 元素中设置的“href”属性值对整体页面元素生效，将鼠标指针移至超级链接时可以看到，默认网站地址就是“href”属性的值，效果如图 3-3 所示，实现源码如加粗部分所示。

说明　在一个页面文档中，<base> 与 <title> 元素只能使用一次，并且必须包含在 <head> 元素中，而其他元素可以重复使用多次。<base> 元素应排在其他元素之前，以便于其他元素能调用 <base> 元素的属性。

3.3　脚本

为了增加页面的互动性，需要对文档编写客户端脚本，最常用的语言是 JavaScript。通过编写客户端的脚本语言，可以实现对页面文档进行验证表单、变更内容等操作。

在页面文档中，用于标志脚本的标签有以下两个：

- <script> 元素，它既可以包含脚本语言，也可以通过 src 属性导入一个脚本文件；同时，选择元素的必选属性 type 与 MIME 类型。
- <noscript> 元素，它是一个检测工具，用于 <script> 中的脚本内容未被执行时显示的内

容，即浏览器如果支持 <script> 中的脚本，则不会显示 <noscript> 中的内容。<script> 元素的常用属性及描述如表 3-3 所示。

表 3-3　<script> 元素的属性及描述

属　性	值	描　述
async	true，false	定义是否异步执行脚本，此属性为 HTML 5 新增
charset	charset	设置脚本中使用的字符编码，此属性 HTML 5 不再支持
language	JavaScript 等	定义脚本的语言类型，此属性 HTML 5 不再支持
xml:space	preserve	此属性 HTML 5 不再支持

<script> 元素中的 async 属性为 HTML 5 新增的内容。该属性有两个取值：true 和 false。当取值为 true 时，脚本在页面中执行的方式是异步的，即在页面解析的过程中执行；当取值为 false 时，脚本将立即执行，页面也会等脚本执行完成后继承解析。

下面通过实例 3-3 介绍 <script> 与 <noscript> 元素在页面中的使用方法。

实例 3-3　元素 <script> 与 <noscript> 的使用

1. 功能描述

在新建的页面中增加一个文本框“txtContent”和一个按钮“请点击我”；当单击按钮时，通过页面中加入的 JavaScript 脚本代码获取文本框中的内容，并且显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-3.html，加入代码如代码清单 3-3 所示。

代码清单 3-3　脚本元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>脚本元素的使用</title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
<script type="text/javascript" async="true">
  function Btn_Click(){
   var strTxt=document.getElementById("txtContent").value;
   var strDiv=document.getElementById("divShow");
   strDiv.style.display="block";
   strDiv.innerHTML=" 您输入的字符是："+strTxt;
  }
</script>
<noscript>您的浏览器不支持 JavaScript!</noscript>
</head>
<body>
<input type="text" id="txtContent"
```

```
        class="inputtxt" />
<input type="button" value="请点击我"
        class="inputbtn" onClick="Btn_Click();">
<div id="divShow" class="divShow"></div>
</body>
</html>
```

为了增加页面的浏览效果，在实例 3-3 中通过 <link> 元素导入了一个 CSS 样式文件 css3.css，文件地址是 /Ch3/Css/css3.css，其代码如下：

```
@charset "utf-8";
/* CSS Document */
body{
  font-size:12px
}
.inputbtn {
   border:solid 1px #ccc;
   background-color:#eee;
   line-height:18px;
   font-size:12px
}
.inputtxt {
   border:solid 1px #ccc;
   line-height:18px;
   font-size:12px
}
.divShow{
   border:solid 1px #666;
   background-color:#eee;
   margin-top:5px;
   padding:5px;
   width:196px;
   display:none
}
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-4 所示。

4. 源码分析

在本实例的 <script> 元素中，设置 async 属性的值为 true，即允许脚本在页面解析时异步执行。HTML 5 中新增的这个属性，可以在很大程度上缓解页面解析的压力，加速页面加载的速度；同时，又不会阻碍 <script> 元素中脚本的执行。如果是执行大量的 JavaScript 代码，其效果将更加明显。在 <script> 元素中，定义了一个函数 Btn_Click()，用于单击页面按钮时显示文本框中输入的内容。如果浏览器支持 <script> 元素，单击按钮时，该函数将被执行，否则将显示 <noscript> 元素中的内容，实现过程如代码中加粗部分所示。

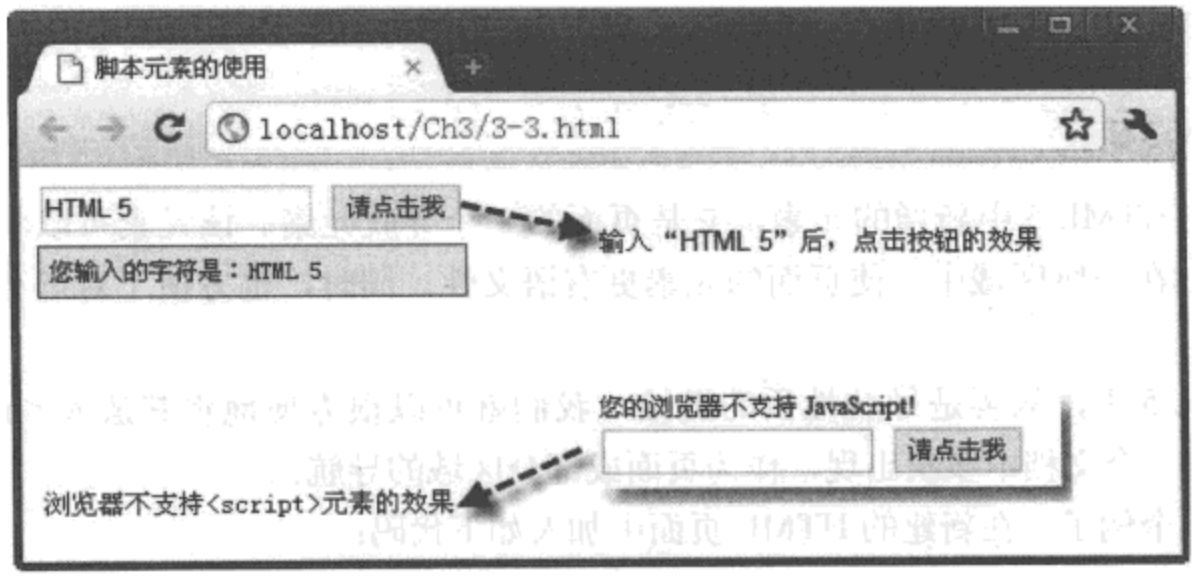


图 3-4　使用脚本元素的页面效果

3.4　节点

在 HTML 5 中，新增了许多用于标志节点的元素，如 <section>、<nav> 等，这些元素可以用于页面内容的分段或分区，接下来详细介绍这些新增的节点元素的使用方法。

3.4.1　section 元素

<section> 元素是 HTML 5 中新增的元素。该元素用于标记文档中的区段或段落，例如文章中的章节、页眉、页脚的设置。

<section> 元素的常用属性及描述如表 3-4 所示。

表 3-4　< section> 元素的属性及描述

属　性	值	描　述
cite	URL	<section> 元素的内容来源于 Web 的地址
hidden	true，false	显示或隐藏 <section> 元素，默认值为 false
draggable	true，false	是否可以拖动 <section> 元素，默认值为 false

下面以“draggable”属性为例，说明该元素的使用方法。在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<section draggable="true">
  <h1>陶国荣</h1>、
  <p>性别：男，邮箱地址：tao_guo_rong@163.com...</p>
</section>
...省略部分代码
```

上述代码创建了一个可以拖动的区段，其中包含两个元素：一个元素是 <h1>，用于显示标题，即“陶国荣”；另一个元素是 <p>，用于对标题元素进行说明，此处对应的内容是“性

别：男，邮箱地址：tao_guo_rong@163.com...”。

3.4.2 nav 元素

<nav> 是 HTML 5 中新增的元素，它是页面的一个导航元素。该元素可以将具有导航性质的链接归纳在一块区域中，使页面的元素更有语义性，同时，也方便了对屏幕阅读器设备的支持。

在 HTML 5 中，只要是导航性质的链接，我们就可以很方便地将其放入 <nav> 元素中。该元素可以在一个文档中多次出现，作为页面或部分区域的导航。

看下面这个例子。在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<nav draggable="true">
  <a href="index.html">首页</a>
  <a href="book.html">图书</a>
  <a href="bbs.html">论坛</a>
</nav>
...省略部分代码
```

上述代码创建了一个可以拖动的导航区域，<nav> 元素中包含了三个用于导航的超级链接，即“首页”、“图书”和“论坛”。该导航可用于全局导航，也可放在某个段落，作为区域导航。

3.4.3 hgroup 元素

<hgroup> 元素是 HTML 5 中新增的元素。该元素用于对页面的标题进行分组，从而形成一个组群。为了更好地说明各组群的功能，该元素常常与元素 <figcaption> 结合使用，通过 <figcaption> 元素说明各组群的功能。

例如，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<hgroup draggable="true">
  <figcaption>标题组一</figcaption>
  <h1>标题 h1</h1>
  <h2>标题 h2</h2>
</hgroup>
...省略部分代码
```

上述代码通过元素 <hgroup> 创建了一个标题组，命名为“标题组一”。该标题组中包含两个标题，分别为“标题 h1”与“标题 h2”。

3.4.4 address 元素

<address> 是 HTML 5 中新增的元素。该元素用于定义文档作者或所有者的相关联系方

式，常用于 <article> 元素外部；如果用于 <article> 元素内部，则表示文章作者的联系方式。该元素在通常情况下显示为斜体，绝大部分的浏览器在解析该元素时，都会在该元素的前后增加换行标记。

例如，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<address title="作者联系方式">
  Written by taoguorong<br />
  <a href="mailto:tao_guo_rong@163.com">Email me</a>
</address>
...省略部分代码
```

上述代码通过 <address> 元素，创建了一个用于显示作者或所有者联系信息的标签“作者联系方式”，该标签中的内容“mailto:tao_guo_rong@163.com”将以斜体的形式显示在页面中。

说明　<article>、<aside>、<header>、<footer> 也属于节点元素，这些元素的使用方法已在前面的章节中有所介绍，在此不再赘述。

3.5　分组内容

HTML 5 对原有的分组内容元素 、<ol>、<dl> 进行了改良，有的元素增加了许多新的属性，有的元素废除了一些不合理的原有特征。下面分别进行详细介绍。

3.5.1　ul 元素

在 HTML 5 中， 元素用于定义页面中的无序列表，其用法与 HTML 4 相类似；区别在于，HTML 5 不再支持“type”与“compact”这两个属性。由于 <ul> 元素通常与 <li> 元素组合使用，因此，HTML 5 也不支持 <li> 元素的“type”属性，而是改用 CSS 样式来定义列表的类型。

例如，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<ul>
  <li>数码</li>
  <li>图书
    <ul>
      <li>程序设计</li>
      <li>网络与数据通信</li>
    </ul>
  </li>
  <li>百货</li>
```

```
</ul>
...省略部分代码
```

上述代码通过 <ul> 元素创建了一个带嵌套的列表“数码”，其中又分为“图书”和“百货”两个列表项。在“图书”列表项中，又通过 <ul> 元素新增加了一个子列表，用于展示上级“图书”列表项的子项信息，这个例子中的子项信息包括“程序设计”和“网络与数据通信”。

3.5.2 ol 元素

 元素用于页面中有序列表的创建。与 HTML 4 相比，在 HTML 5 中新增加了两个属性：一个为“start”，另一个为“reversed”，前者用于自定义列表项开始的编号，后者用于设置列表是否进行反向排序。

下面通过一个简单的实例 3-4 介绍 <ol> 元素新增属性的详细使用过程。

实例 3-4 元素 <ol> 的使用

1. 功能描述

在页面中，通过 <ol> 元素创建一个“各类图书销量排名”列表，并添加三个选项（计算机、社科、文艺）作为列表的内容。另外，增加一个文本框“设置开始值”与一个“确定”按钮；在文本框中输入一个值，单击“确定”按钮后，将以文本框中的值为列表项开始的编号显示各类图书销量排名。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-4.html，加入代码如代码清单 3-4 所示。

代码清单 3-4 <ol> 列表元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>ol 列表的使用 </title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
<script type="text/javascript" async="true">
 function Btn_Click(){
   var strNum=document.getElementById("txtOrderNum").value;
   var strDiv=document.getElementById("olList");
   strDiv.setAttribute("start",strNum);
  }
</script>
</head>
<body>
```

```
<h5> 各类图书销量排名 </h5>
<ol id="olList">
  <li> 计算机 </li>
  <li> 社科 </li>
  <li> 文艺 </li>
</ol>
<h5> 设置开始值 </h5>
<input type="text" id="txtOrderNum"
       class="inputtxt" style="width:60px" />
<input type="button" value=" 确定 "
       class="inputbtn" onClick="Btn_Click();">
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-5 所示。

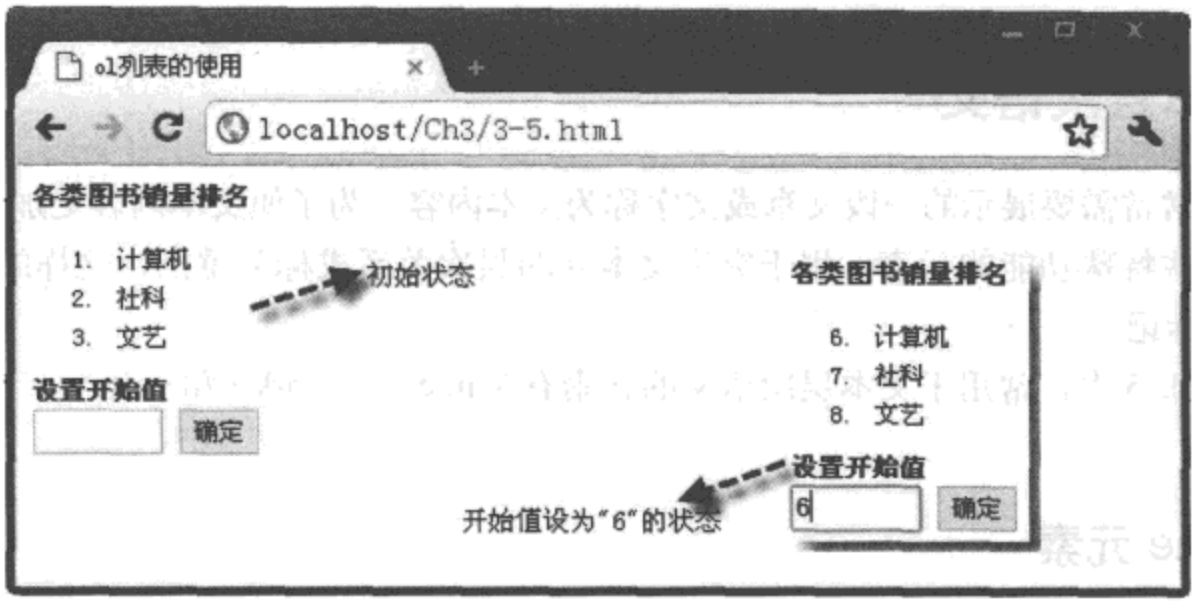


图 3-5　使用 <ol> 列表元素的页面效果

4. 源码分析

页面文件 3-4.html 中的 JavaScript 代码先定义一个函数 Btn_Click()，在单击“确定”按钮时调用。在该函数中，先获取输入文本的值与 <ol> 列表元素，并分别保存至变量“strNum”与“strDiv”中；然后，通过 setAttribute 方法，将列表元素的“start”属性设置为变量“strNum”的值，从而改变了列表项元素编号的开始值。例如，本实例在文本框中输入数字“6”，那么，列表项元素的编号起始值将从 6 开始，实现过程如代码中加粗部分所示。

3.5.3　dl 元素

<dl> 元素用于创建一个术语列表。HTML 5 对该元素进行了改良，允许在一个 <dl> 元素

表中包含多个带名字的术语 <dt> 元素，每个术语元素后面可以跟一个或多个定义 <dd> 元素，并且术语元素 <dt> 与定义元素 <dd> 都不允许重复出现。

例如，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<dl>
  <dt>HTML 5</dt>
  <dd>在当今时代，无论是前端设计还是 Web 开发都应掌握它 </dd>
  <dt>jQuery</dt>
  <dd>一种非常优秀的 Web 开发前端框架 </dd>
</dl>
...省略部分代码
```

上述代码表示在列表 <dl> 元素中创建了两个术语 <dt> 元素，即“HTML 5”和“Query”；在每个术语元素的后面又分别创建了一个定义 <dd> 元素，用于对术语元素进行内容上的详细说明。注意，列表 <dl> 元素中允许在一个术语 <dt> 元素的后面创建多个定义 <dd> 元素，或在一个列表 <dl> 元素中定义多个术语 <dt> 元素。

3.6 文本层次语义

页面中常常需要展示的一段文章或文字称为文本内容。为了使文本内容更加形象、生动，需要增加一些特殊功能的元素，用于突出文本间的层次关系或标为重点，这样的元素称为**文本层次语义标记**。

在 HTML 5 中，常用于文本层次语义的元素有 <time>、<mark> 和 <cite>。下面分别进行详细说明。

3.6.1 time 元素

<time> 是 HTML 5 新增加的一个标记，用于定义时间或日期。该元素可以代表 24 小时中的某一时刻，在表示时刻时，允许有时间差。在设置时间或日期时，只需将该元素的属性“datetime”设为相应的时间或日期即可。

例如，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<p id="p1">
  <time datetime="2011-4-9">
    今天是 2011 年 4 月 9 日
  </time>
<p>
<p id="p2">
  <time datetime="2011-4-9T21:00">
    现在时间是 2011 年 4 月 9 日晚上 9 点
  </time>
<p>
```

```
<p id="p3">
        <time datetime="2011-12-31">
          新款 PSP2 掌上游戏机将于今年年底上市
        </time>
</p>
 <p id="p4">
        <time datetime="2011-5-1" pubdate="true">
          本消息发布于 2011 年 5 月 1 日
        </time>
</p>
...省略部分代码
```

<p> 元素 ID 号为“p1”中的 <time> 元素表示的是日期。页面在解析时，获取的是属性“datetime”中的值，而标记之间的内容只是用于显示在页面中。

<p> 元素 ID 号为“p2”中的 <time> 元素表示的是日期和时间，它们之间使用字母“T”进行分隔。如果在整个日期与时间的后面加上一个字母 Z，则表示获取的是 UTC（世界统一时间）格式。

<p> 元素 ID 号为“p3”中的 <time> 元素表示的是将来时间。

<p> 元素 ID 号为“p4”中的 <time> 元素表示的是发布日期。

为了在文档中将这两个日期进行区分，在最后一个 <time> 元素中增加了“pubdate”属性，表示此日期为发布日期。

说明　<time> 元素中的可选属性“pubdate”表示时间是否为发布日期，它是一个布尔值，该属性不仅可以用于 <time> 元素，还可用于 <article> 元素。

3.6.2　mark 元素

<mark> 元素是 HTML 5 中新增的元素，主要功能是在文本中高亮显示某个或某几个字符，旨在引起用户的特别注意。其使用方法与 和 有相似之处，但相比而言，HTML 5 中新增的 <mark> 元素在突出显示时，更加随意与灵活。

下面通过一个完整的实例 3-5 来介绍该元素的使用方法。

实例 3-5　元素 <mark> 的使用

1. 功能描述

在页面中，首先使用 <h5> 元素创建一个标题“优秀开发人员的素质”，然后通过 <p> 元素对标题进行阐述。在阐述的文字中，为了引起用户的注意，使用 <mark> 元素高亮处理字符“素质”、“过硬”和“务实”。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-5.html，加入代码如代码清单 3-5 所示。

代码清单 3-5　mark 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>mark 元素的使用 </title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
</head>
<body>
 <h5> 优秀开发人员的 <mark> 素质 </mark></h5>
 <p class="p3_5">
    一个优秀的 Web 页面开发人员，必须具有
    <mark> 过硬 </mark> 的技术与
    <mark> 务实 </mark> 的专业精神
 </p>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-6 所示。

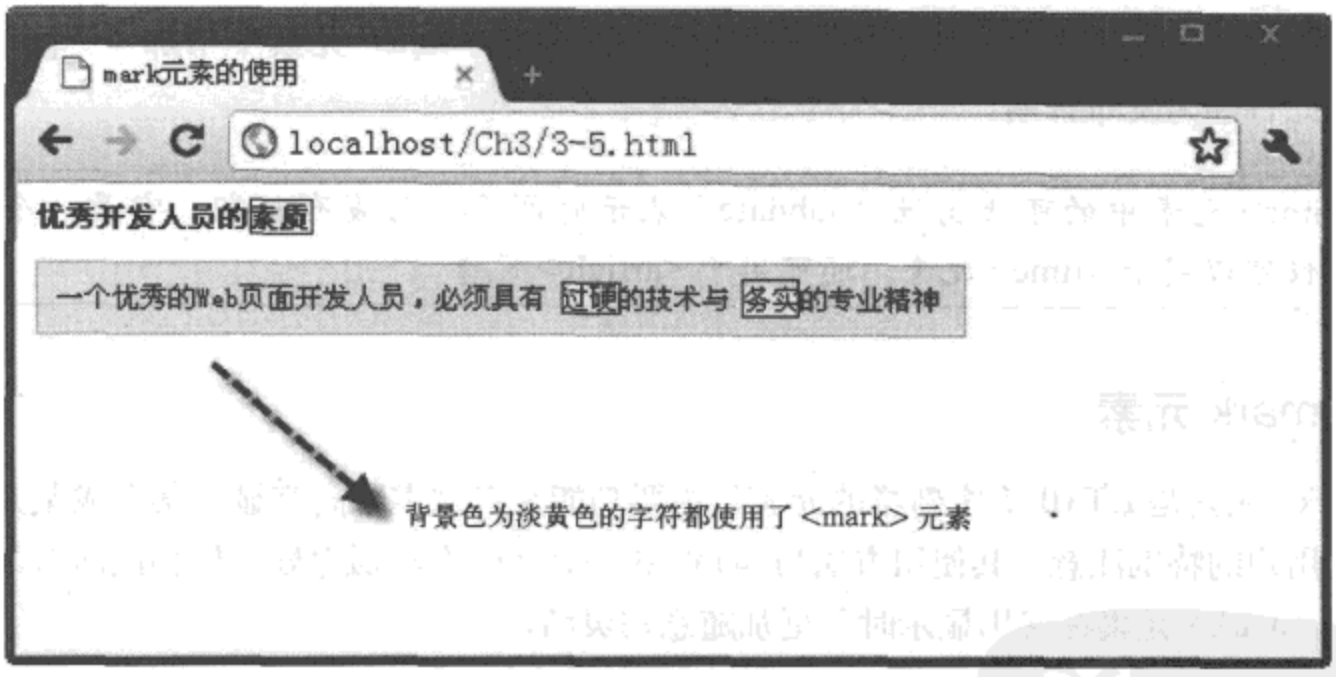


图 3-6　使用 <mark> 元素的页面效果

4. 源码分析

在本实例中，通过使用 <mark> 元素，将文字中的“素质”、“过硬”和“务实”三个字符进行了高亮显示的处理，实现方法如源码中加粗部分所示。

<mark> 元素的这种高亮显示的特征，除用于文档中突出显示外，还常用于查看搜索结果页面中关键字的高亮显示，其目的主要是引起用户的注意。

虽然 <mark> 元素在使用效果上与 或 元素有相似之处，但三者的出发点是不一样的。 元素是作者对文档中某段文字的重要性进行的强调； 元素是作者为

了突出文章的重点而进行的设置；<mark> 元素是数据展示时，以高亮的形式显示某些字符，与原作者本意无关。

3.6.3　cite 元素

<cite> 元素可以创建一个引用标记，用于文档中参考文献的引用说明，如书名或文章名称。一旦在文档中使用了该标记，被标记的文档内容将以斜体的样式展示在页面中，以区别于段落中的其他字符。

下面通过一个完整的实例 3-6 来介绍该元素的使用方法。

实例 3-6　元素 <cite> 的使用

1. 功能描述

在页面中，首先通过 <p> 元素显示一段文档；然后，在文档的下面使用 <cite> 元素标识这段文档所引用的书名。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-6.html，加入代码如代码清单 3-6 所示。

代码清单 3-6　cite 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>cite 元素的使用 </title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
</head>
<body>
 <h5>jQuery </h5>
 <p>
    jQuery 是继 Prototype 之后的一个优秀的 JavaScript 框架，
    深受全球开发者的欢迎 ...</p>
 <p>
    --- 引自 << <cite>jQuery 权威指南 </cite> >> ---
 </p>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-7 所示。

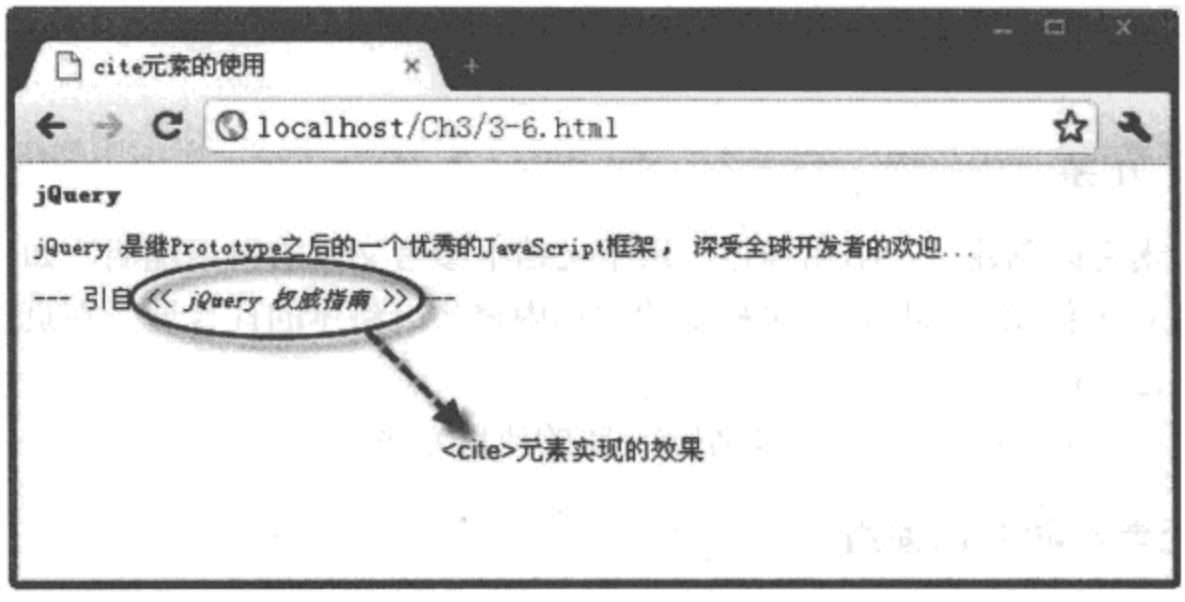


图 3-7　使用 <cite> 元素的页面效果

4. 源码分析

在上述代码中，使用 <cite> 元素标识这段文档所引用的书名“jQuery 权威指南”，因此，该书名在页面中显示为斜体。

注意　HTML 5 中的 <cite> 元素基本兼容了 HTML 4 中的全部功能，但定义时更加严格。在使用该元素时，只允许包含标题或书名，不允许包含更多的其他引用信息，如作者姓名、出版日期等。

3.7　嵌入内容

在页面中，除了显示文档或字符外，还需要放入一些其他元素，如图片 <img>、页面 <iframe>、多媒体 <object> 等，这些元素对于整个 DOM 文档来说，属于嵌入内容。在 HTML 5 中，这些嵌入内容的有些属性不再被支持，下面我们通过实例逐个进行介绍。

3.7.1　img 元素

<img> 元素的功能是在页面中导入一幅图像，它是页面开发中使用较为频繁的一个元素。在 HTML 5 中，该元素的“border”、“align”、“hspace”、“vspace”属性不再被支持，这些功能需要通过 CSS 样式来实现。

例如，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<style type="text/css">
  img{
     padding:5px;
```

```
        border:solid 1px #666;
        text-align:center
    }
</style>
<a href="javascript:void(0)">
    <img src="Images/img_01.jpg" alt="HTML 5 下的图片 " />
</a>
...省略部分代码
```

上述代码中，元素 <img> 创建的图片可以被单击，通过 CSS 样式代码控制页面中该元素的效果。因为在页面的 <style> 元素中，定义了 <img> 元素的展示样式，包括图片元素 4 个方向的边距为“5px”，添加一个带颜色的边框，以及整个图片元素居中，实现方法见代码中加粗部分所示，这种做法比较符合 HTML 5 对元素 <img> 的支持。

3.7.2　iframe 元素

<iframe> 元素用于在页面中创建包含另一文档的框架。出于对页面安全性的考虑，HTML 5 不再支持 <frame> 框架元素，包括 <frameset> 框架集元素，但仍然支持 <iframe> 元素，只是该元素的一些原有属性不再被支持，而仅仅支持“src”属性。

例如，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<iframe src="3-1.html" sandbox="allow-forms"></iframe>
...省略部分代码
```

在上述代码中，用到了“src”属性，HTML 5 还设置了另外一个属性“sandbox”，该属性是 <iframe> 元素在 HTML 5 中新增加的属性。

众所周知，使用 <iframe> 元素时包含另一个页面，这一操作的安全性一直让开发者担忧。为了规避这一不足，HTML 5 新增加了一个元素的属性“sandbox”，通过该属性的设置，可以避免私自访问父页面、执行异样脚本、通过脚本嵌入表单或控制表单。该属性有 4 个属性值可供选择，如表 3-5 所示。

表 3-5　“sandbox”属性及描述

属　性　值	描　　述
allow-forms	允许脚本嵌入自己的表单或操纵表单
allow-same-origin	允许将嵌入内容视为同一个数据源
allow-scripts	允许执行脚本
allow-top-navigation	允许最外层浏览器的上下文导航功能

在设置时，根据实际需求选择允许的操作，从而有效避免 <iframe> 元素嵌入的文档有安全性问题。

3.7.3 object 元素

<object> 元素用于在页面中嵌入一个对象，这个对象包括视频、音频、Flash 等多媒体文件。该元素在 HTML 5 中仍然被保留，只是有些原有的属性不再被支持。

在 HTML 5 中，元素 <object> 被支持的原有属性如表 3-6 所示。

表 3-6 < object > 元素在 HTML 5 中被支持的属性与描述

属 性	值	描 述
data	URL	设置多媒体对象的数据源
type	TypeValue	选择数据源中的多媒体在页面中展示的类型
usemap	URL	设置多媒体对象在客户端映射的地址
height	HeigtValue	设置对象的高度（单位：像素）
width	WidthValue	设置对象的宽度（单位：像素）

举个例子，在新建的 HTML 页面中加入如下代码：

```
...省略部分代码
<object data="Video/swf_01.swf" type="all"
        width="200px" height="200px">
</object>
...省略部分代码
```

上述代码按照 HTML 5 的支持特征，设置了 <object> 元素的关键属性“data”值。如果按照设置的多媒体路径找到了该对象，在支持 HTML 5 属性的浏览器中可以实现播放功能。

在 HTML 5 中，新增了专门用于播放多媒体文件的标签——<video> 元素与 <audio> 元素。前者用于播放视频或影视，后者用于播放音频文件，这两个元素的使用方法将在第 6 章中详细介绍。<video> 元素与 <audio> 元素将逐步取代 <object> 元素，从而真正展示 HTML 5 在处理视频或音频方面的强大优势。

3.8 公共属性

在 HTML 5 取代 HTML 4 的过程中，无论是新增还是改良的元素，都有一些共同的属性，我们称为**公共属性**，如 draggable、hidden、spellcheck、contenteditable 属性。接下来逐一介绍这些常用的公共属性。

3.8.1 draggable 属性

“draggable”属性用于设置是否允许用户拖动元素，该属性有两个值，分别为：“true”表示鼠标选中元素后，可以进行拖动的操作；“false ”表示鼠标选中元素后，不能进行拖动的操作，该属性值为默认值。

下面通过一个完整的实例 3-7 来介绍该属性在元素中的使用方法。

实例 3-7　公共属性 draggable 的使用

1. 功能描述

在页面中，使用 <article> 元素显示一段文字，并设置 <article> 元素的属性“draggable”值为“true”，当用户选中这段文字移动鼠标时，可以实现拖动的效果。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-7.html，加入代码如代码清单 3-7 所示。

代码清单 3-7　draggable 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>draggable 属性的使用 </title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
</head>
<body>
 <h5> 元素的拖动属性 </h5>
  <article draggable="true" class="p3_7">
     这是一段可以拖动的文字，选中后进行拖动。
  </article>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-8 所示。

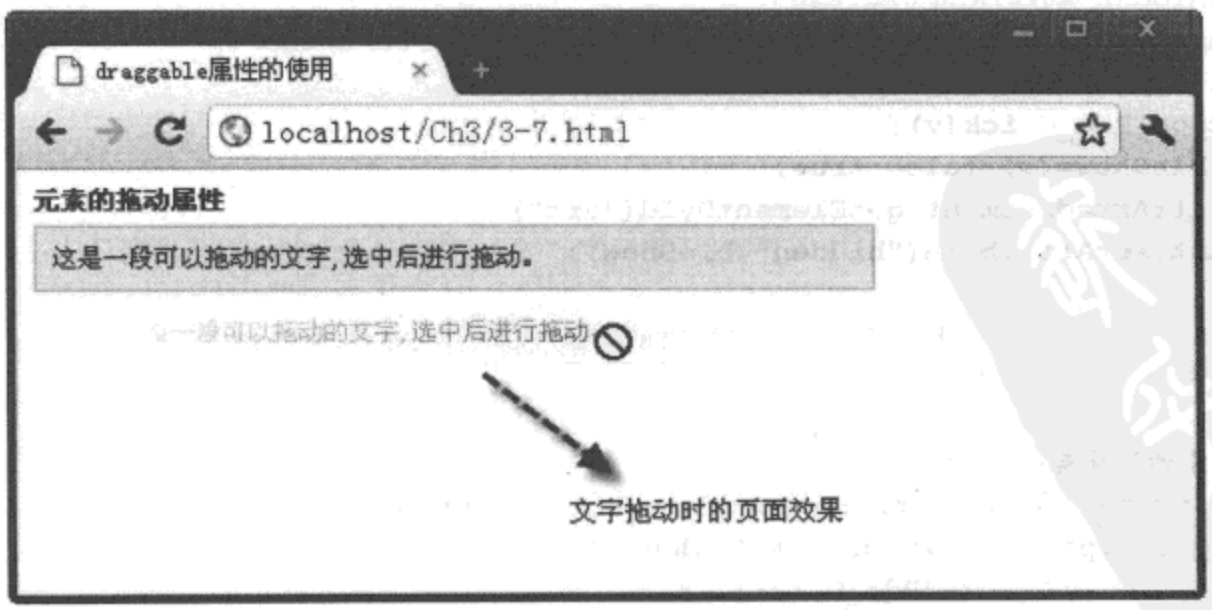


图 3-8　使用 draggable 属性的页面效果

4. 源码分析

在代码中，为了实现 <article> 元素的可拖动效果，必须在元素中添加一个“draggable”属性，并将该属性的值设置为“true”，表示该元素可以进行拖动，实现过程如代码中加粗部分所示。设置完成后，在浏览器中就可以选中该元素，进行拖动的操作。

3.8.2 hidden 属性

在 HTML 5 中，绝大部分的元素都支持“hidden”属性，该属性只有两个取值：“true”和“false”。当“hidden”的取值为“true”时，元素不在页面中显示，但还存在于页面中；反之，则显示于页面中。该属性的默认值为“false”，即元素创建时便显示出来。

下面通过一个完整的实例 3-8 来介绍该属性在元素中的使用方法。

实例 3-8 公共属性 hidden 的使用

1. 功能描述

在页面的 <nav> 元素中设置两个相互排斥的单选按钮，一个用于显示 <article> 元素，另一个用于隐藏 <article> 元素。通过编写相应的 JavaScript 代码实现上述功能。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-8.html，加入代码如代码清单 3-8 所示。

代码清单 3-8 hidden 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>hidden 属性的使用 </title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
<script type="text/javascript" async="true">
  function Rdo_Click(v){
   var blnShow=(v)?false:true;
   var strArt=document.getElementById("art");
   strArt.setAttribute("hidden",blnShow);
  }
</script>
</head>
<body>
 <h5> 元素的隐藏属性 </h5>
  <nav style="padding-top:5px;padding-bottom:5px">
    <input type="radio" id="rdoHidden_1"
             onClick="Rdo_Click(1)"
           name="rdoHidden" value="1" checked="true"/> 显示
    <input type="radio" id="rdoHidden_2"
             onClick="Rdo_Click(0)"
           name="rdoHidden" value="0"/> 隐藏
```

```
    </nav>
    <article id="art" class="p3_8">
        今天是一个好天气啊，蓝蓝的天空，飘着朵朵白云。
    </article>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-9 所示。

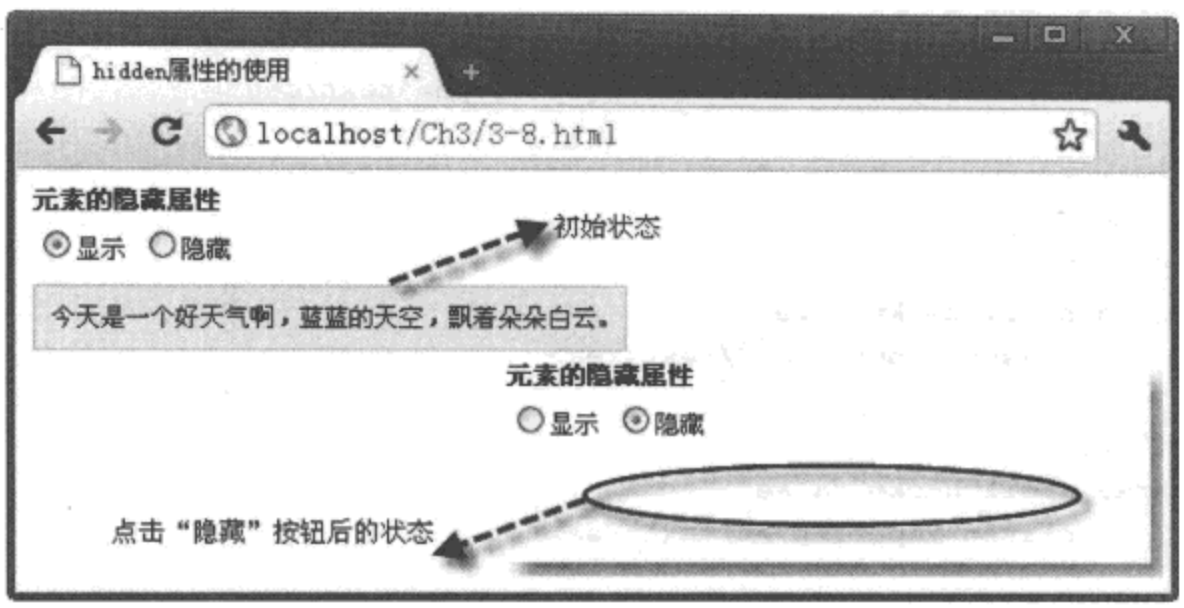


图 3-9　使用 hidden 属性的页面效果

4. 源码分析

上述的 JavaScript 代码中，自定义了一个函数 Rdo_Click()，用于单击单选按钮时调用。在该函数中，先获取单击单选按钮时传回的变量“v”值，然后将“v”值转成“hidden”属性对应的布尔值“true”或“false”；最后，通过 setAttribute() 方法，将该值设置到 <article> 元素的“hidden”属性中，从而实现隐藏的效果，实现过程如代码中加粗部分所示。

3.8.3　spellcheck 属性

“spellcheck”属性用于检测文本框或输入框中的拼音或语法是否正确，该属性的值为布尔值，即“true”或“false”。如果为“true”，则检测对应输入框中的语法；反之，则不检测。

下面通过一个完整的实例 3-9 来介绍该属性在元素中的使用方法。

实例 3-9　公共属性 spellcheck 的使用

1. 功能描述

在页面中，分别创建两个 <textarea> 输入框元素。第一个元素将“spellcheck”属性设置为“true”，即需要语法检测；另外一个元素的“spellcheck”属性设置为“false”，即不需要语

法检测。分别在两个输入框中录入文字，可以显示不同的检测效果。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-9.html，加入代码如代码清单 3-9 所示。

代码清单 3-9　spellcheck 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>spellcheck 属性的使用 </title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
</head>
<body>
 <h5> 输入框中语法检测属性 </h5>
 <p> 需要检测 <br/>
   <textarea spellcheck="true"
             class="inputtxt"></textarea>
 </p>
 <p> 不需要检测 <br/>
   <textarea spellcheck="false"
             class="inputtxt"></textarea>
 </p>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-10 所示。

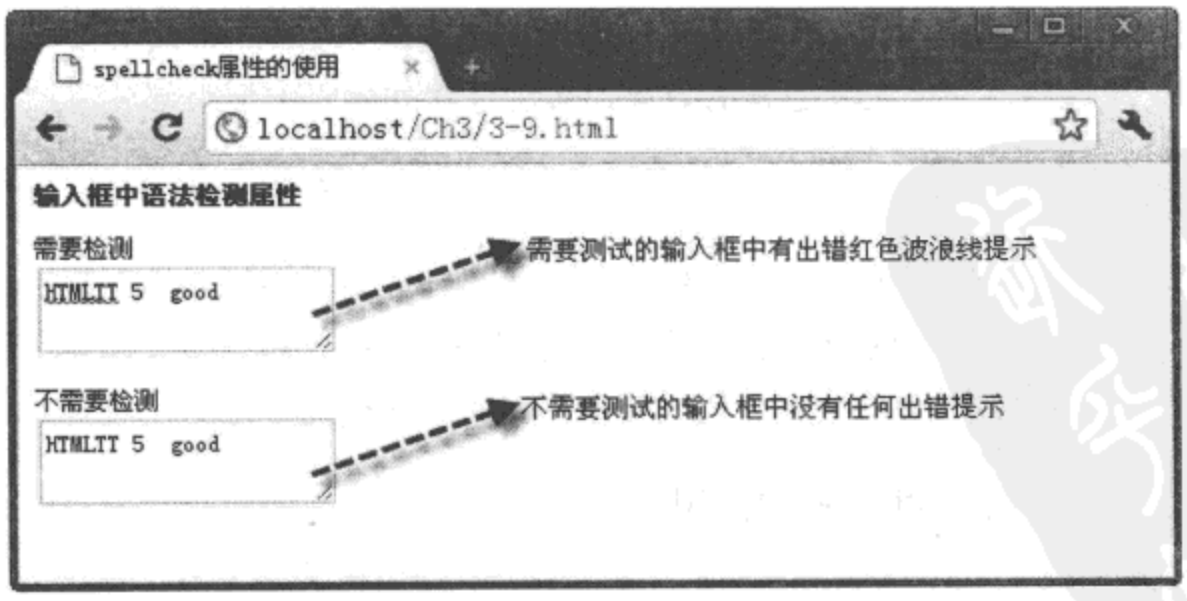


图 3-10　使用 spellcheck 属性的页面效果

4. 源码分析

在代码清单中，为了形成对比效果，将第一个 <textarea> 输入框元素中的“spellcheck”属性设置为“true”值，将第二个 <textarea> 输入框元素中的“spellcheck”属性设置为“false”值。从图 3-10 可以看出，在两个输入框中输入同样的内容，第一个输入框中的内容显示出错的红色波浪线，而第二个输入框没有任何提示，表明第一个 <textarea> 输入框中设置的“spellcheck”属性值已生效。

在 HTML 5 中，虽然各浏览器对“spellcheck”属性进行了很好的支持，但各浏览器支持的元素是有差异的。在 Chrome 浏览器中，支持 <textarea> 输入框元素，而不支持 <input> 元素中的文本框；Firefox、Opera 浏览器需要在“选项”菜单中手动进行设置，才能显示效果。

3.8.4　contenteditable 属性

HTML 5 中有一个非常便捷的属性“contenteditable”，利用它可以直接对显示在页面中的文字进行编辑。该属性的取值为布尔型，即“true”或“false”。如果在元素中将该属性的值设置为“true”，那么就可以对该元素显示的文本内容直接进行编辑了。

下面通过一个完整的实例 3-10 来介绍该属性在元素中的使用方法。

实例 3-10　公共属性 contenteditable 的使用

1. 功能描述

在页面中，分别创建两个 <article> 内容元素。第一个元素将“contenteditable”属性设置为“true”，用于直接内容的编辑；第二个 <article> 元素保存编辑后的内容。当用户编辑完成后，单击“保存”按钮，则将编辑后的内容显示在第二个 <article> 元素中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 3-10.html，加入代码如代码清单 3-10 所示。

代码清单 3-10　contenteditable 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>contenteditable 属性的使用 </title>
<link href="Css/css3.css" rel="stylesheet" type="text/css">
<script type="text/javascript" async="true">
  function Btn_Click(){
   var strArt=document.getElementById("art_0").innerHTML;
   var objArt=document.getElementById("art_1");
   objArt.innerHTML='<b> 编辑后：</b>'+strArt;
  }
```

```
</script>
</head>
<body>
 <h5>元素的内容编辑属性</h5>
  <article contenteditable="true" class="p3_10" id="art_0">
     一段可编辑的文字
  </article>
  <article class="p3_10" id="art_1">
  </article>
  <input type="button" value="保存"
        class="inputbtn" onClick="Btn_Click();">
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器下执行的页面效果如图 3-11 所示。

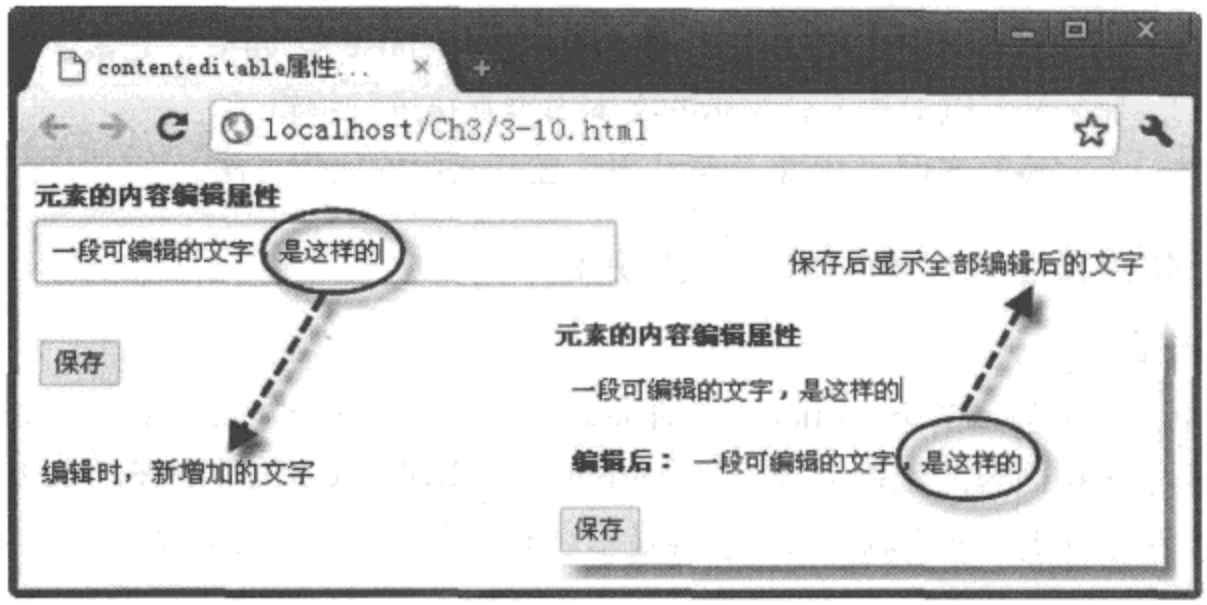


图 3-11　使用 contenteditable 属性的页面效果

4. 源码分析

在 JavaScript 代码中，自定义了一个函数 Btn_Click()，在单击“保存”按钮时调用。在该函数中，为了获取编辑后的内容，先将修改完成的内容保存至变量“strArt”中，然后，通过元素 ID 号获取用于显示结果的 <article> 元素，并将该元素显示的内容设置为变量“strArt”的值。实现过程如代码中加粗部分所示。

在 HTML 5 中，大部分显示文本内容的元素都支持“contenteditable”属性，这给页面中用户的交互体验带来了极大的方便。

注意　目前，暂无相关的 API 对编辑后的内容进行直接保存；如果需要保存，只能借助于 Ajax 或 jQuery 中的异步操作来更新对应的后台数据。

3.9　本章小结

本章先从最基本的根元素 <html> 讲起，分门别类地对 HTML 5 中各重要元素采用理论辅助实例的方式进行介绍，最后，还介绍了 HTML 5 中元素的公共属性。

通过对这些 HTML 5 中新增或改良元素的学习，可以进一步加深对各元素的理解，为第 4 章 HTML 5 中表单元素的 学习打下扎实的实践基础。

第4章 HTML 5中的表单

本章内容

- input 元素的新增类型
- input 元素新增的公用属性
- 新增表单元素
- 表单新增的验证方法和属性
- 本章小结

无论是实现功能还是展示页面中的元素，表单在 HTML 中都有不可替代的作用。HTML 4 中，表单包含的元素非常有限，并且属性与方法都不多，尤其是表单交互过程中数据的验证，需要编写大量的 JavaScript 代码；因此，可能面临被客户端禁用的危险，还可能导致最后实现不了验证数据的功能。

上述的种种不足都在 HTML 5 中被克服了。首先，HTML 5 中的表单在兼容原有元素的基础上，又新增了许多新的元素类型，如 email、url、range 等；其次，还新增加许多新的方法与属性，如 autofocus、placeholder 等。另外，HTML 5 表单元素中数据的有效性验证不再依赖于编写脚本语言的方式，完全可以通过 HTML 5 自身内建验证机制实现；这一功能为开发人员带来极大的方便，在很大程度上提高了开发的效率。

接下来，详细介绍 HTML 5 表单中各种新元素、新方法的使用。

4.1 input 元素的新增类型

在 HTML 5 中，<input> 元素在原有类型的基础上添加了许多新的类型成员，如表 4-1 所示。

表 4-1 HTML 5 中 <input> 元素新增的类型

类型名称	HTML 代码	功能描述
邮件输入框	<input type="email">	输入 E-mail 地址的文本框
Web 地址输入框	<input type="url">	输入 URL 地址的文本框
数字输入框	<input type="number">	输入数字的文本框，可以设置输入值的范围
数字滑动条	<input type="range">	通过拖动滑动条改变一定范围内的数字
日期选择器	<input type="date">	选择日期（包括年、月、日、星期）的文本框
搜索输入框	<input type="search">	输入搜索关键字操作的文本框

说明 上表所列的类型，目前只有 Opera 浏览器可以很完美地进行支持，其他浏览器只能显示常规的文本框。

接下来，详细介绍 <input> 元素每个新增加类型的使用方法与展示效果。

4.1.1 email 邮件类型

如果将 <input> 元素中的“type”类型设置为“email”，将在页面中创建了一专门用于输入邮件地址的文本输入框。该文本框与其他文本框在页面显示时没有区别，专门用于接收 E-mail 地址信息。因此，当表单在提交时，将会自动检测文本框中的内容是否符合 E-mail 邮件地址格式；如果不符，则提示相应错误信息。

此外，表单提交前，并不检测“email”类型文本框的内容是否为空；而是在不为空的情况下，检测其内容是否符合标准的 E-mail 格式。如果该元素的“multiple”属性设置为

“true”，则允许用户输入一串用逗号分隔的 E-mail 地址。

下面通过简单的实例 4-1 介绍“email”类型的 <input> 元素的使用过程。

实例 4-1　email 类型的 <input> 元素的使用

1. 功能描述

在新创建的页面表单中加入一个“email”类型的 <input> 元素，用于输入邮件地址；另外，新建一个表单提交按钮，当单击“提交”按钮时，将自动检测“email”类型的文本框中输入的字符是否符合邮件格式，如果不符，则显示错误提示信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-1.html，加入代码如代码清单 4-1 所示。

代码清单 4-1　email 类型的 <input> 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>email 类型的 input 元素 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend> 请输入邮件地址： </legend>
   <input name="txtEmail" type="email"
          class="inputtxt" multiple="true">
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 提交 ">
 </fieldset>
</form>
</body>
</html>
```

在代码清单 4-1 中，通过调用一个样式文件 css4.css 实现页面样式的控制，其实现的代码如下所示：

```
@charset "utf-8";
/* CSS Document */
body {
   font-size:12px
}
.inputbtn {
   border:solid 1px #ccc;
   background-color:#eee;
```

```
    line-height:18px;
    font-size:12px
}
.inputtxt {
    border:solid 1px #ccc;
    line-height:18px;
    font-size:12px;
    width:180px
}
fieldset{
    padding:10px;
    width:260px
}
```

3. 页面效果

该页面在不同浏览器中执行的页面效果如图 4-1 所示。

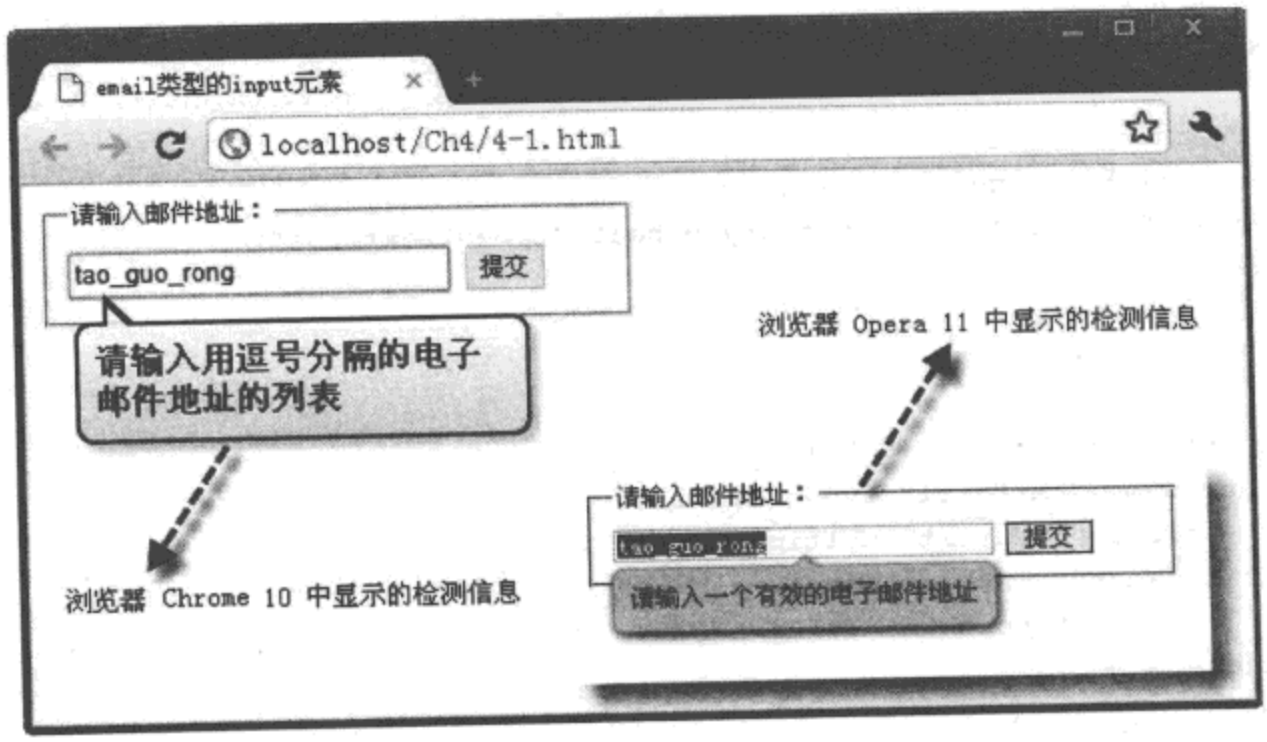


图 4-1　email 类型的 <input> 元素在不同浏览器中的页面效果

4. 源码分析

如代码清单 4-1 加黑部分所示，由于将 <input> 元素“multiple”的属性值设置为“true”，因此，在 Chrome 10 浏览器中，单击“提交”按钮时，显示的提示信息为“请输入用逗号分隔的电子邮件地址的列表”；而如果不设置该属性值时，则提示信息为“请输入电子邮件地址”。

4.1.2　url 地址类型

<input> 元素中的“url”类型是一种新增类型，该类型表示 <input> 元素是一个专门用于输入 Web 站点地址的输入框。Web 地址的格式与普通文本有些区别，如文本中有反斜杠“/”和点“.”。为了确保“url”类型的输入框能够正确提交符合格式的内容，表单在提交数据前，

会对其内容格式的有效性进行自动验证，如果不符合对应的格式，则会出现相应的错误提示信息；另外，与“email”类型一样，有效性检测并不判断输入框的内容是否为空，而是针对非空的内容进行格式检测。

下面通过简单的实例 4-2 介绍“url ”类型的 <input> 元素的使用过程。

实例 4-2　url 类型的 <input> 元素的使用

1. 功能描述

在新建的页面表单中，创建一个“url”类型的 <input> 元素，同时，新建一个表单“提交”按钮。当单击“提交”按钮时，将自动检测输入框中的元素是否符合 Web 地址格式，如果不符，则显示错误提示信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-2.html，加入代码如代码清单 4-2 所示。

代码清单 4-2　url 类型的 <input> 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>url 类型的 input 元素 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend> 请输入网址：</legend>
   <input name="txtUrl" type="url"
          class="inputtxt" />
   <input name="frmSubmit" type="submit"
           class="inputbtn" value=" 提交 " />
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在不同浏览器中执行的页面效果如图 4-2 所示。

4. 源码分析

在本实例中，由于将文本框的“type”属性设置为“url”值，浏览器将自动检验文本框中的内容是否符合“url”的格式，如果不符，则弹出提示信息，如图 4-2 左部分所示。

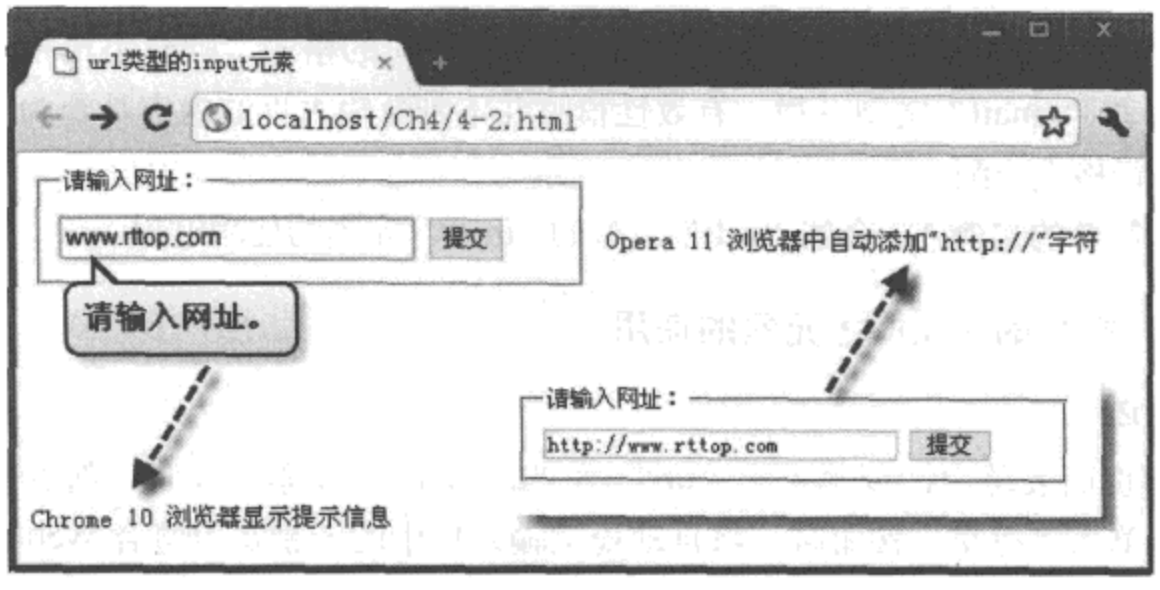


图 4-2　url 类型的 <input> 元素在不同浏览器中的页面效果

需要说明的是，目前对 <input> 元素新增类型提供支持的只有 Chrome 10 与 Opera 11 浏览器，其他浏览器暂时还不支持，而两个浏览器对“url”类型的 <input> 元素在页面展示时，效果并不一样。Chrome 10 浏览器必须输入完整的 URL 地址路径（包括“http://”），不介意前面有空格；而 Opera 11 浏览器不必输入完整的 URL 地址路径，提交时自动会在前面添加，但介意开始处有空格，如果文本输入框中的开始处有空格，将提示格式出错信息。

4.1.3　number 数字类型

在 HTML 4 以前的版本中，如果想要在表单中输入一个指定范围的整数，需要在表单提交前，进行复杂的代码数据检测，确定输入框中是否是一个符合要求的整数；而在 HTML 5 中，只要创建一个“number”类型的 <input> 元素，便可以实现以上操作。该类型的元素在 HTML 5 中还将显示一个微调控件，如果指定了最大与最小范围值，就可以点击微调控件的上限与下限按钮，以指定的步长（step）增加或减少输入框中的值，极大方便了用户的操作。

在“number”类型的输入框中，用户不可以输入其他非数字型的字符，并且当输入的数字大于设定的最大值或小于设置的最小值时，都将出现数字输入出错的提示信息。同样，该类型不进行输入内容是否为空值的自动检测。

下面通过简单的实例 4-3 介绍“number”类型的 <input> 元素的使用过程。

实例 4-3　number 类型的 <input> 元素的使用

1. 功能描述

在新建的表单中，创建三个“number”类型的 <input> 元素，分别用于输入日期中“年”、“月”、“日”的数字；同时，新建一个表单的“提交”按钮。单击该按钮时，将检测这三个输入框中的数字是否属于各自设置的整数范围，如果不符合，将显示错误提示信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-3.html，加入的代码如代码清单 4-3 所示。

代码清单 4-3　number 类型的 <input> 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>number 类型的 input 元素 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend> 请出生日期: </legend>
   <input name="txtYear" type="number"
          class="inputtxt" min="1960" max="1990"
          step="1" value="1990" /> 年
   <input name="txtMonth" type="number"
          class="inputtxt" min="1" max="12"
          step="1" value="4"/> 月
   <input name="txtDay" type="number"
          class="inputtxt" min="1" max="31"
          step="1" value="23"/> 日
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 提交 " />
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在不同浏览器中执行的页面效果如图 4-3 所示。

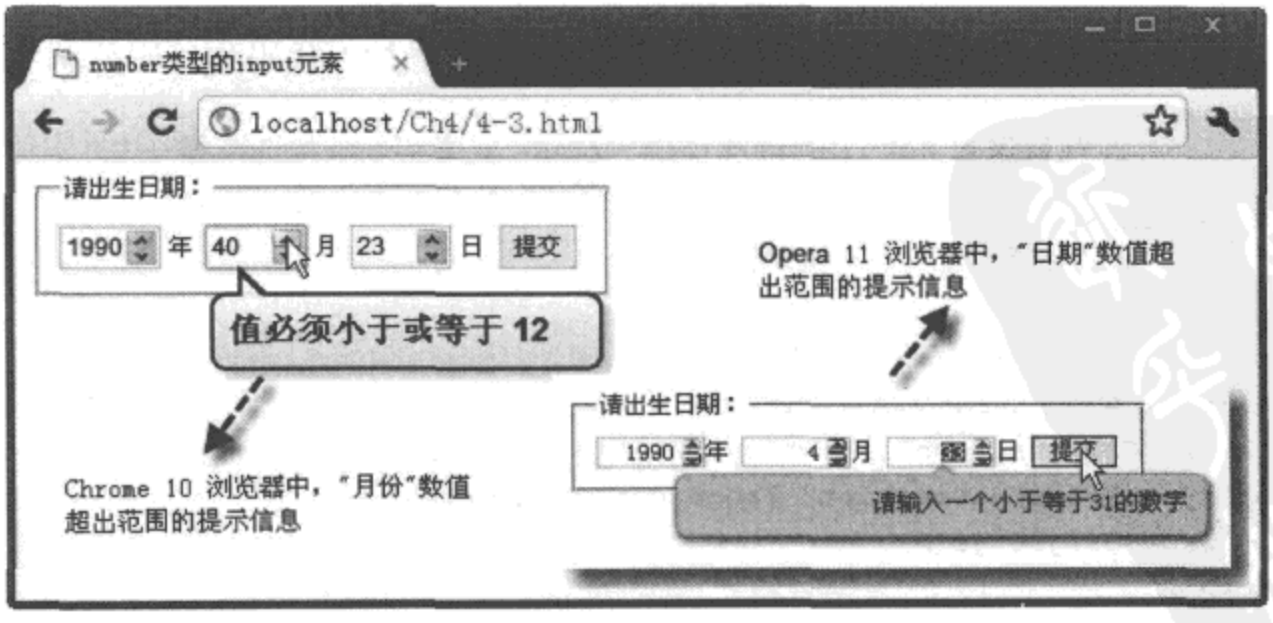


图 4-3　number 类型的 <input> 元素在不同浏览器中的页面效果

4. 源码分析

如代码清单 4-3 中的加粗代码所示，定义了三个“number”类型的 <input> 元素输入框，分别设置了“min”、“max”、“value”、“step”属性值。其中“step”属性值表示步长值，默认值为 1，即当用户点击微调控件时，向上增加或向下减少的值。所有这些属性值都是可选项，如果不需要指定数字上限则可以省略“max”属性。

在使用 Opera 11 浏览器展示时，数字输入框的微调控件中，如果数字不能向上调，那么向上箭头按钮变灰，表示不可用，向下箭步则变为可用；反之，向下箭头变灰色，表示不可用，向上箭头则表示为可用。

4.1.4 range 数字滑动条

如果要在 HTML 5 中输入整数，除了使用 4.1.3 节中提到的“number”类型的 <input> 元素外，还可以使用“range”类型。这两种数字类型的 <input> 元素基本属性都一样，唯一不同之处在于页面中的展示形式。“number”类型在页面中以输入框加微调控件显示，而“range”类型则以滑动条的形式展示数字，通过拖动滑块实现数字的改变。

下面通过简单的实例 4-4 介绍“range”类型的 <input> 元素的使用过程。

实例 4-4 range 类型的 <input> 元素实现颜色选择器

1. 功能描述

在新建的页面表单中，创建三个“range”类型的 <input> 元素，分别用于设置颜色中的“红色”（r）、“绿色”（g）、“蓝色”（b）；另外，新建一个 <p> 元素，用于展示滑动条改变时的颜色区。当用户任意拖动某个绑定颜色的滑动条时，对应的颜色区背景色都会随之发生变化，同时，颜色区下面显示对应的色彩值（rgb）。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-4.html，加入代码如代码清单 4-4 所示。

代码清单 4-4 range 类型的 <input> 元素实现颜色选择器

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>range 类型的 input 元素 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="javascript"
        src="Js/js4.js">
</script>
</head>
<body>
```

```
<form id="frmTmp">
 <fieldset>
   <legend>选择颜色值：</legend>
   <span id="spnColor">
   <input id="txtR" type="range" value="0"
          min="0" max="255" onChange="setSpnColor()" >
   <input id="txtG" type="range" value="0"
          min="0" max="255" onChange="setSpnColor()">
   <input id="txtB" type="range" value="0"
          min="0" max="255" onChange="setSpnColor()">
   </span>
   <span id="spnPrev"></span>
   <P id="pColor">rgb(0,0,0)</P>
 </fieldset>
</form>
</body>
</html>
```

在代码清单 4-4 中，通过调用一个 JavaScript 文件 js4.js，实现单击某个颜色滑动条动态改变预览颜色区背景色的功能，其实现的代码如下所示：

```
// JavaScript Document
function $$(id){
   return document.getElementById(id);
}
// 定义变量
var intR,intG,intB,strColor;
// 根据获取变化的值，设置预览方块的背景色函数
function setSpnColor(){
   intR=$$("txtR").value;
   intG=$$("txtG").value;
   intB=$$("txtB").value;
   strColor="rgb("+intR+","+intG+","+intB+")";
   $$("pColor").innerHTML=strColor;
   $$("spnPrev").style.backgroundColor=strColor;
}
// 初始化预览方块的背景色
setSpnColor();
```

3. 页面效果

该页面在不同浏览器中执行的页面效果如图 4-4 所示。

4. 源码分析

回顾代码清单 4-4 加粗代码，其中分别使用“range”类型定义了三个 <input> 元素，这些元素都以滑动条的形式展示在页面中。拖动滑块时，将触发 JavaScript 的一个自定义函数

setSpnColor()，该函数可以根据获取滑动条的值动态改变颜色块的背景色，实现过程见文件 js4.js 中代码加粗部分所示。

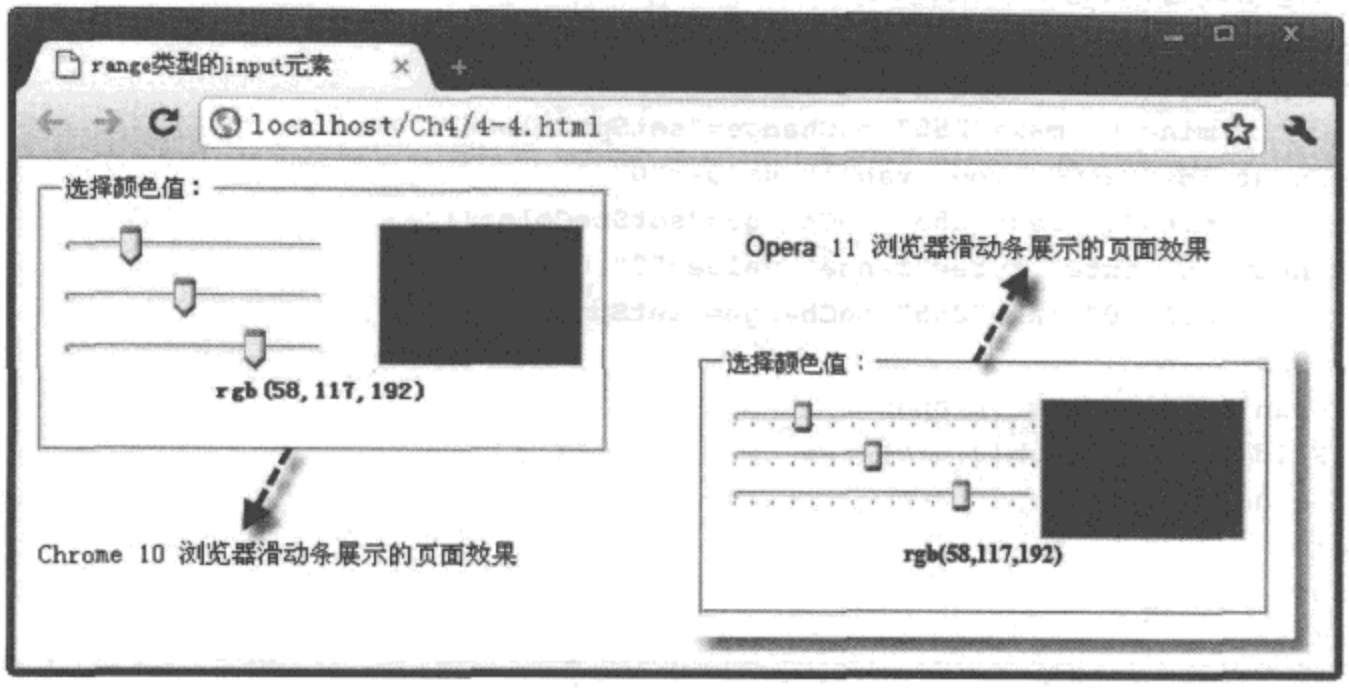


图 4-4　range 类型的 <input> 元素在不同浏览器中的页面效果

虽然 Chrome 10 与 Opera 11 浏览器都支持 <input> 元素的“range”类型，但在页面中展示的效果是不一样的。在 Opera 11 浏览器中，“range”类型的 <input> 元素带刻度，支持按左右方向键增加或减小滑动条的值，而 Chrome 10 浏览器则不支持。

4.1.5　date 日期类型

在 HTML 4 之前的版本中，没有专门用于显示日期的文本输入框，如果要实现这样的输入框，需要编写大量的 JavaScript 代码或导入相应的插件，实现过程较为复杂。

在 HTML 5 中，只需要将 <input> 元素的类型设置为“date”，便可创建一个日期型的文本输入框。当用户单击该文本框时，会弹出一个日历选择器，选择日期后关闭该日历选择器，会将所选择的日期显示在文本框中。

下面通过简单的实例 4-5 介绍“date”类型的 <input> 元素的使用过程。

实例 4-5　分类展示不同形式的选择日期

1. 功能描述

在新建的页面表单中，分三组创建六个不同展示形式的日期类型输入框。第一组为“日期”与“时间”类型，展示类型为“date”与“time”值的日期输入框；第二组为“月份”与“星期”类型，展示类型为“month”与“week”值的日期输入框；第三组为“日期时间”类型，分别展示类型为“datetime”与“datetime-local”值的日期输入框。所有这些日期类型的输入框在表单提交时，都将对输入的日期或时间进行有效性检测，如果不符将弹出错误提示信息。

2. 实现代码

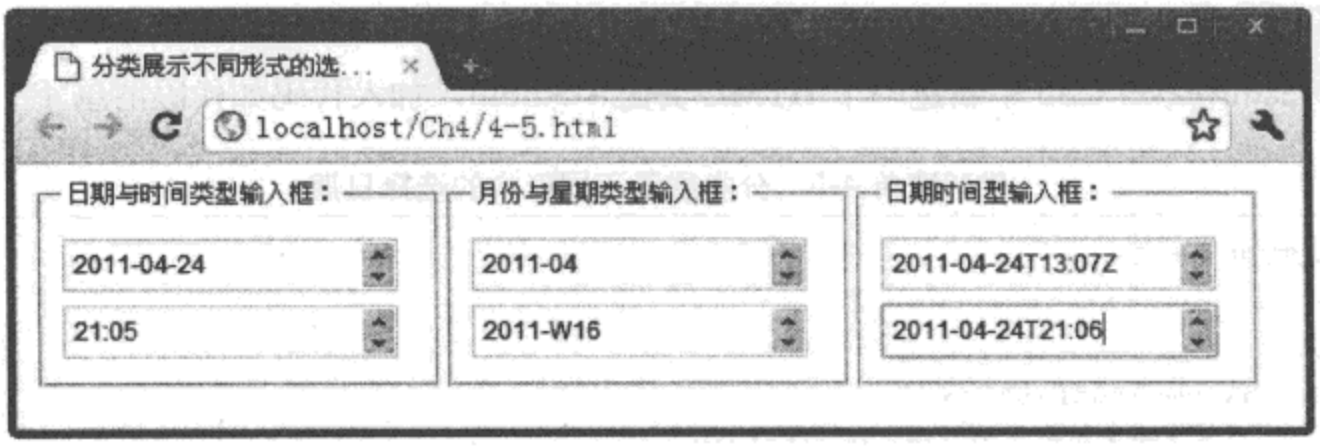
在 Dreamweaver CS5 中新建一个 HTML 页面 4-5.html，加入代码如代码清单 4-5 所示。

代码清单 4-5　分类展示不同形式的选择日期

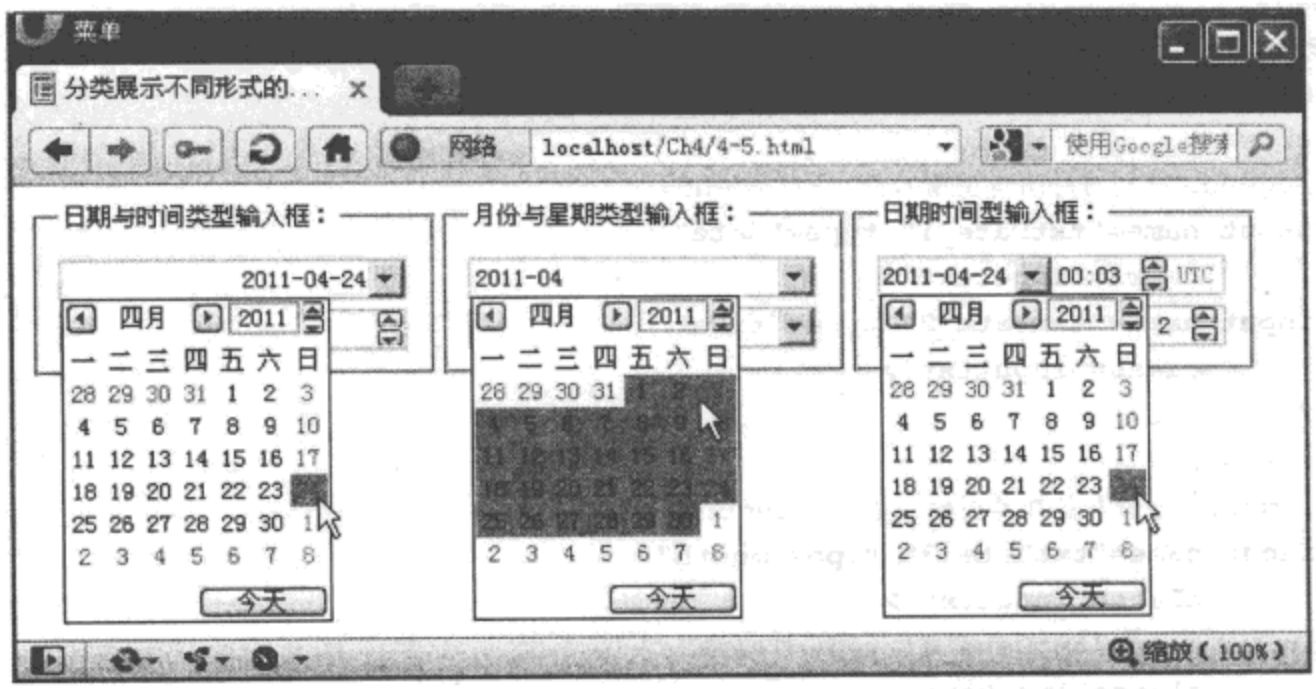
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>分类展示不同形式的选择日期</title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>日期与时间类型输入框：</legend>
   <input name="txtDate_1" type="date"
          class="inputtxt">
   <input name="txtDate_2" type="time"
          class="inputtxt">
   </fieldset>
   <fieldset>
   <legend>月份与星期类型输入框：</legend>
   <input name="txtDate_3" type="month"
          class="inputtxt">
   <input name="txtDate_4" type="week"
          class="inputtxt">
   </fieldset>
   <fieldset>
   <legend>日期时间型输入框：</legend>
   <input name="txtDate_5" type="datetime"
          class="inputtxt">
   <input name="txtDate_6" type="datetime-local"
          class="inputtxt">
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在不同浏览器中执行的页面效果如图 4-5 所示。



a) Chrome 10 浏览器



b) Opera 11 浏览器

图 4-5　不同类型的日期输入框在不同浏览器中的页面效果

如图 4-5a 所示，日期类型的输入框在 Chrome 10 浏览器中不能弹出日期选择器，只能点击向上或向下的箭头按钮，从而改变相应的输入值。如图 4-5b 所示，在 Opera 11 浏览器中，可以弹出日期选择器，更加方便用户选择日期或时间。

4. 源码分析

如代码清单 4-5 加粗源码所示，“datetime”类型是专门用于 UTC 日期与时间的输入文本框，而“datetime-local”类型则是用于本地日期与时间的输入文本框，默认值为本地的日期与时间。另外，在“week”类型的输入文本框中，如果值为“2011-W17”表示在 2011 年的第 17 个星期，以此类推，该星期的值表示当年的第几个星期。

注意　在 Opera 11 浏览器中，如果可以弹出日期选择器，则该输入框为只读型，即只允许通过弹出的日期选择器进行日期或时间的选择，不允许手动输入数字。

4.1.6　search 搜索类型

“search”类型的 <input> 元素专门用于关键字的查询，该类型的输入框与“text”类型的输入框在功能上没有太大的区别，都是用于接收用户输入的查询关键字，但在页面展示时，却有细微的区别。在 Chrome 10 与 Safari 5 浏览器中，当开始在输入框中填写内容时，输入框的右侧将会出现一个“×”按钮，单击该按钮，将清空在输入框中的内容，使用十分方便。

下面通过简单的实例 4-6 介绍 search 类型的 <input> 元素的使用过程。

实例 4-6　search 类型的 <input> 元素的使用

1. 功能描述

在新建的表单中，增加一个“search”类型的 <input> 元素，用于查询关键字的输入；同时，增加一个表单“提交”按钮，当单击按钮时，显示输入的关键字内容。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-6.html，加入代码如代码清单 4-6 所示。

代码清单 4-6　search 类型的 <input> 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>search 类型的 input 元素 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="javascript"
        src="Js/js6.js">
</script>
</head>
<body>
<form id="frmTmp" onSubmit="return ShowKeyWord();">
 <fieldset>
   <legend> 请输入搜索关键字：</legend>
   <input id="txtKeyWord" type="search"
          class="inputtxt">
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 提交 ">
 </fieldset>
 <p id="pTip"></p>
</form>
</body>
</html>
```

在代码清单 4-6 中，通过调用一个 JavaScript 文件 js6.js，实现单击表单“提交”按钮时，显示输入的查询关键字功能，其实现的代码如下所示：

```
// JavaScript Document
function $$(id){
  return document.getElementById(id);
}
// 将获取的内容显示在页面中
function ShowKeyWord(){
  var strTmp="<b>您输入的查询关键字是：</b>";
  strTmp=strTmp+$$('txtKeyWord').value;
  $$('pTip').innerHTML=strTmp;
  return false;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 4-6 所示。

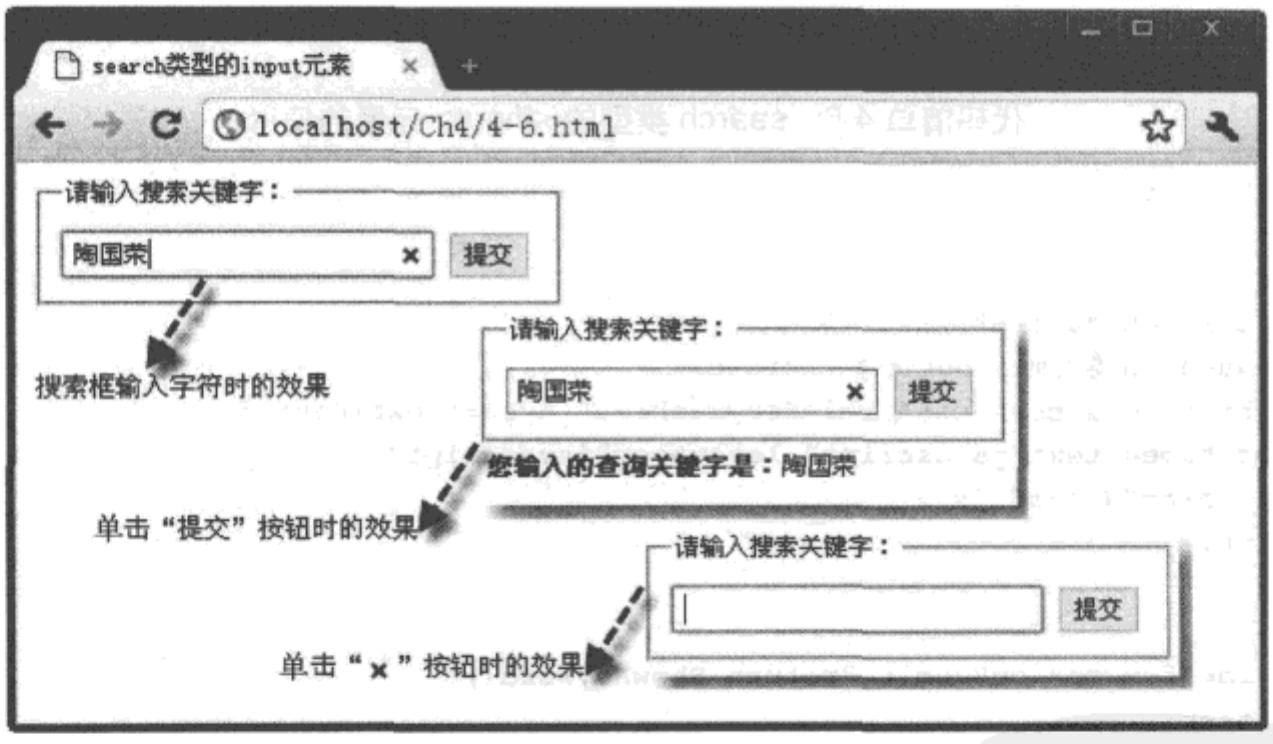


图 4-6　search 类型的输入框在 Chrome 10 浏览器中的页面效果

4. 源码分析

当表单提交时，为了获取“search”类型的 <input> 输入框的值，在表单的“onSubmit”事件中，调用了一个 JavaScript 自定义函数 ShowKeyWord()，如代码清单 4-6 加粗代码所示。

在自定义函数 ShowKeyWord() 中，先获取查询输入框的值，然后将该值设置为展示元素 <p> 的内容，并通过“return false”方法终止表单提交的过程，显示该函数执行结果，实现代码如文件 js6.js 中加粗部分所示。

由此可见，“search”类型的 <input> 输入框与一般的“text”文本输入框在使用时没有太

大的区别，只是在页面展示时，输入框的右侧增加了一个“×”按钮，用于快速清空输入框中的内容。

4.2　input 元素新增的公用属性

4.1 节介绍了新增的 <input> 元素，其中介绍了许多新增的元素属性，如“range”类型中的“min”、“max”、“step”属性等。除此之外，<input> 元素在 HTML 5 中还添加一些公用的属性，如表 4-2 所示。

表 4-2　<input> 元素在 HTML 5 中新增的公用属性

属性名称	HTML 代码	功能说明
autofocus	<input autofocus= "true">	页面加载成功后，设置对应的元素自动获取焦点
pattern	<input pattern =" 正则表达式 ">	使用正则表达式验证 <input> 元素的内容
placeholder	<input placeholder = " 默认内容 ">	设置文本输入框中的默认内容
required	<input required = "true">	是否检测文本输入框中的内容为空
novalidate	<input novalidate = "true">	是否验证文本输入框中的内容

接下来通过实例详细介绍每个新增加的属性在 <input> 元素中的使用方法。

4.2.1　autofocus 属性

表单中的所有 <input> 元素都具有“autofocus”属性，该属性是一个布尔值，主要功能是页面加载完成后，光标是否自动锁定 <input> 元素，即是否使元素自动获取焦点。在 <input> 元素中，如果将该属性的值设置为“true”或直接输入“autofocus”属性名称，那么对应的元素将自动获取焦点。

下面通过简单的实例 4-7 介绍 <input> 元素中“autofocus”属性的使用过程。

实例 4-7　<input> 元素中 autofocus 属性的使用

1. 功能描述

在新建的表单中，创建两个文本输入框，一个用于输入“姓名”，另一个用于输入“密码”。输入“姓名”的文本框设置“autofocus”属性，当页面加载完成或单击表单“提交”按钮后，拥有“autofocus”属性的“姓名”输入文本框会自动获取焦点。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-7.html，加入代码如代码清单 4-7 所示。

代码清单 4-7　<input> 元素中 autofocus 属性的使用

```
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="utf-8" />
<title>input 元素中 autofocus 属性的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>autofocus 属性：</legend>
   <p> 姓名：<input type="text" name="txtName"
             class="inputtxt" autofocus="true"></p>
   <p> 密码：<input type="password" name="txtPws"
             class="inputtxt"></p>
   <p class="p_center">
      <input name="frmSubmit" type="submit"
              class="inputbtn" value=" 提交 " />
      <input name="frmReset" type="reset"
              class="inputbtn" value=" 取消 " />
   </p>
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 4-7 所示。

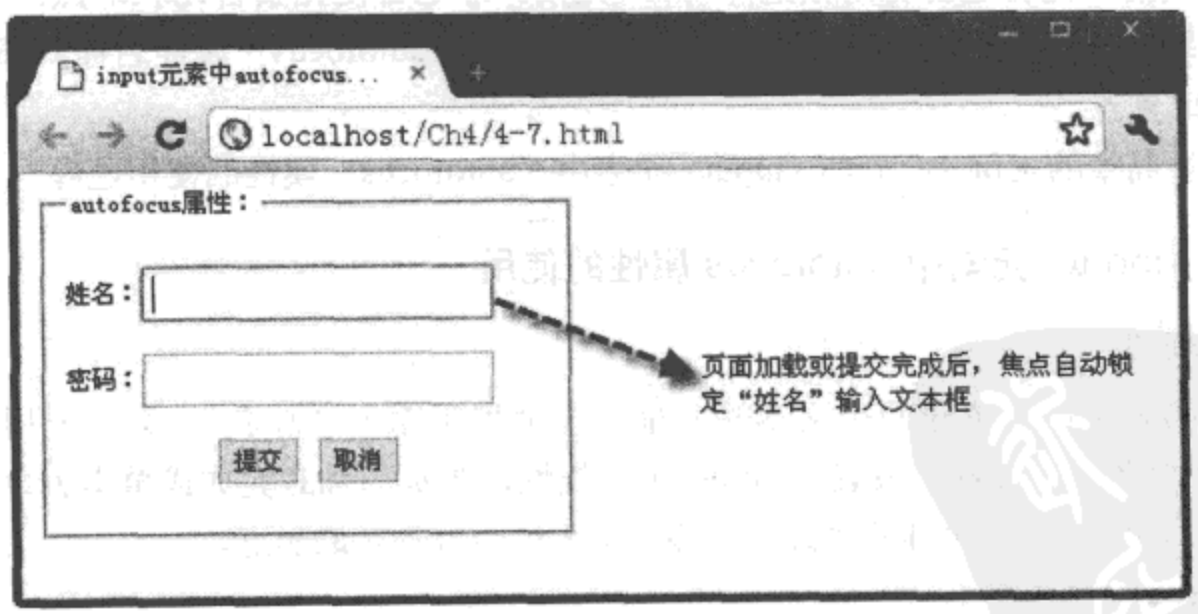


图 4-7 <input> 元素的 autofocus 属性在 Chrome 10 浏览器中使用时的页面效果

4. 源码分析

在 HTML 4 中，如果要使某个元素自动获取焦点，需要编写 JavaScript 代码实现。虽然这一功能的实现方便了用户的操作，但也带来了不少的弊端，例如，需要通过按空格键滚动页

面时，而焦点还在表单的输入文本框中，因此，输入的空格只能显示在文本框中，并不能实现页面的滚动。

在 HTML 5 中，由于实现这一功能的不再是 JavaScript 代码，而是元素的属性，因此，所有页面实现该功能的方法都是一致的，避免了由于代码实现的不同而效果不一样的情况。

注意　在一个页面表单中，建议只对一个输入框设置“autofocus”属性，例如在资料录入页面中，只对第一个文本输入框设置“autofocus”属性。

4.2.2　pattern 属性

在 <input> 元素中，“pattern”是元素的验证属性。使用该属性中的正则表达式，可以验证文本输入框中的内容。

在前面的章节中，“email”、“url”等类型的 <input> 元素都内置了正则表达式，当创建这些元素时，用来与内容进行匹配，进行有效性验证。因此，换一个角度说，这些元素都使用了“pattern”属性，只是内置的而已。

内置验证的元素毕竟较少，并且如果要进行组合式的验证，就需要使用“pattern”属性。该属性支持各种类型的组合正则表达式，用来验证对应的文本输入框中的内容。

下面通过简单的实例 4-8 介绍 <input> 元素中“pattern”属性的使用过程。

实例 4-8　<input> 元素中 pattern 属性的使用

1. 功能描述

在表单中，创建一个“text”类型的 <input> 元素，用于输入“用户名”，并设置元素的“pattern”属性，其值为一个正则表达式，用来验证“用户名”是否符合“以字母开头，包含字符或数字和下划线，长度在 6 ～ 8 之间”规则。单击表单“提交”按钮时，输入框中的内容与表达式进行匹配，如果不符，则提示错误信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-8.html，加入代码如代码清单 4-8 所示。

代码清单 4-8　<input> 元素中 pattern 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>input 元素中 pattern 属性的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
```

```
<form id="frmTmp">
 <fieldset>
   <legend>pattern 属性：</legend> 输入用户名：
   <input name="txtAge" type="text"
          class="inputtxt" pattern="^[a-zA-Z]\w{5,7}$" />
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 提交 " />
   <p class="p_color">
      以字母开头，包含字符或数字和下划线，长度在 6~8 之间
   </p>
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 4-8 所示。

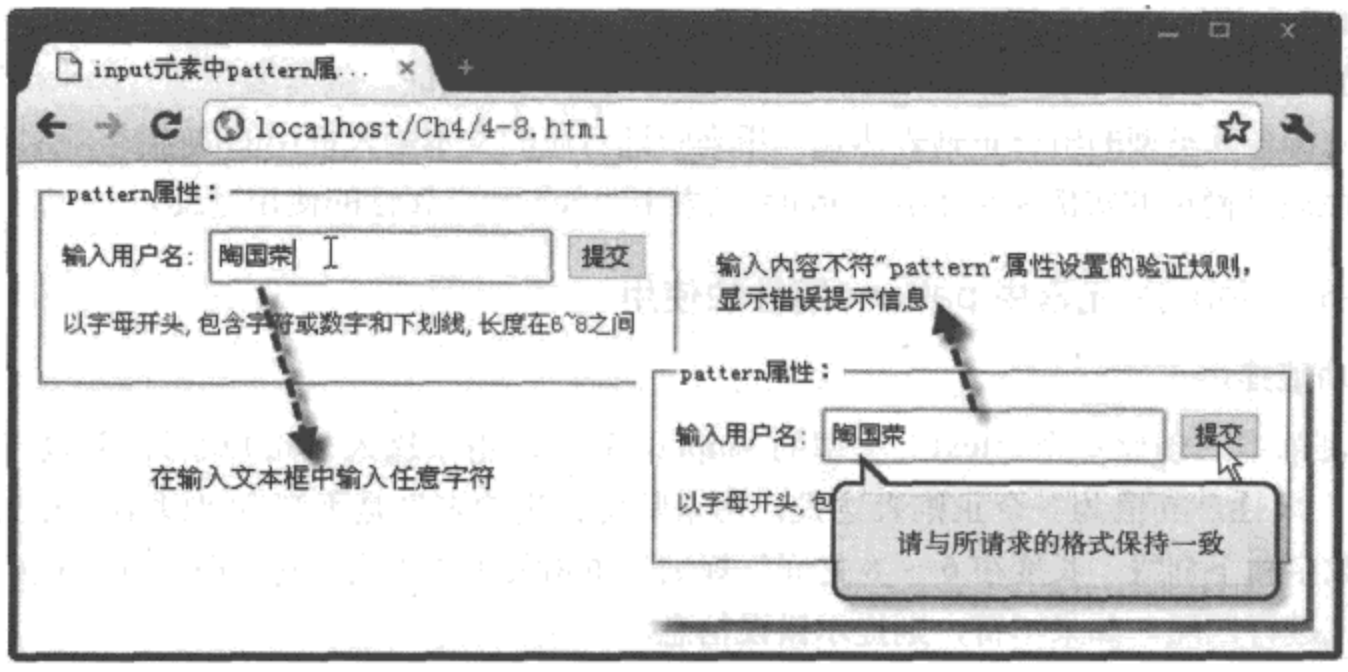


图 4-8 <input> 元素的 pattern 属性在 Chrome 10 浏览器中使用时的页面效果

4. 源码分析

在 <input> 元素中，所有的输入框类型都支持“pattern”属性，使用时，只要在输入框中添加一个“pattern”属性（如代码中加粗部分所示），就可以通过属性中各种组合类型的正则表达式验证输入框中的内容。

与新增加的各类型 <input> 元素一样，目前只有 Chrome 10 与 Opera 11 浏览器支持该属性；对于不支持该属性的浏览器，只能通过编写 JavaScript 代码实现输入内容的验证。

4.2.3　placeholder 属性

<input> 元素中的“placeholder”属性称为“占位”属性，其属性值称为“占位文本”。顾名思义，“占位文本”就是显示在输入框中的提示信息，当输入框获取焦点时，该提示信息自动消失；而当输入框丢失焦点时，提示信息又重新显示。

在 HTML 4 中，要实现这样的效果，需要编写不少的 JavaScript 代码，而在 HTML 5 中，只要设置元素的“placeholder”属性就可以实现了。

下面通过简单的实例 4-9 介绍 <input> 元素中“placeholder”属性的使用过程。

实例 4-9　<input> 元素中 placeholder 属性的使用

1. 功能描述

在新建的表单中，创建一个类型为“email”的 <input> 元素，设置该元素的“placeholder”属性值为“请正确输入您的邮件地址”。当页面初次加载时，该元素的占位文本显示在输入框中，单击输入框时，占位文本自动消失。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-9.html，加入代码如代码清单 4-9 所示。

代码清单 4-9　<input> 元素中 placeholder 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>input 元素中 placeholder 属性的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>placeholder 属性：</legend> 输入邮箱 :
   <input name="txtEamil" type="Email"
          class="inputtxt"
          placeholder=" 请正确输入您的邮件地址 " />
   <input name="frmSubmit" type="submit"
         class="inputbtn" value=" 提交 " />
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 4-9 所示。

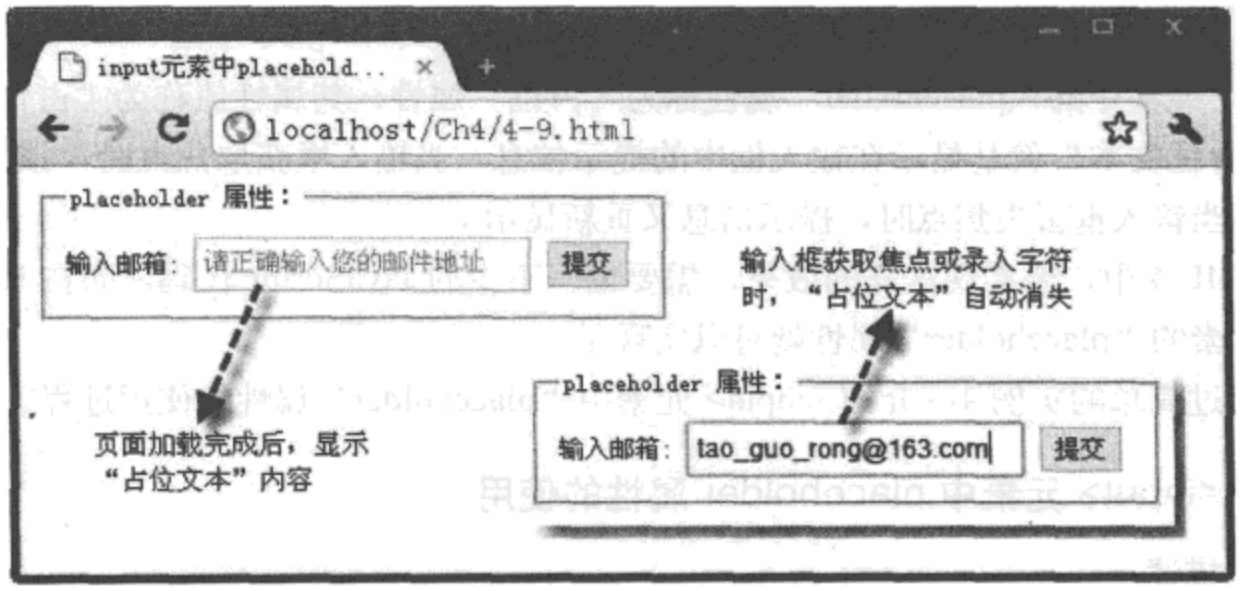


图 4-9 <input> 元素的 placeholder 属性在 Chrome 10 浏览器中使用时的页面效果

4. 源码分析

如果需要在表单中设置输入框元素的默认提示信息（占位文本），只要添加该元素“placeholder”属性，并设置属性的内容就可以，详细见代码中加粗部分所示。当输入框中的内容没有消失，而单击“提交”按钮时，也不会将占位文本作为输入框中的内容提交给服务器。

虽然利用输入框中的“placeholder”属性可以很方便地实现动态显示提示信息的功能，但如果内容过长，还是建议使用元素的“title”属性来显示。

注意 文本输入框中的“placeholder”属性值只支持纯文本，目前还不支持 HTML 语法，也不能修改输入框中占位文本的样式。

4.2.4 required 属性

在页面的表单中，输入文本框的“required”属性，用于检测输入内容是否为空。“email”或“url”类型的 <input> 元素在提交表单时，都要进行内容验证。这种验证仅针对输入框中的内容是否符合各自所属的类型，对输入框中的文本内容是否为空，并不验证，即只验证非空内容。

以前，为了验证表单中输入文本框的内容是否为空，通常的做法是编写 JavaScript 代码，计算输入文本框中的字符长度，如果等于 0 则返回一个“false”值，这种方法较为复杂。在 HTML 5 中，只要在验证元素中添加一个“required”属性，就可以对其内容是否为空自动进行验证；如果为空，在表单提交数据时，会显示错误提示信息。

下面通过简单的实例 4-10 介绍 <input> 元素中“required”属性详细的使用过程。

实例 4-10　<input> 元素中 required 属性的使用

1. 功能描述

在表单中，创建一个用于输入“姓名”的“text”类型 <input> 元素，并在该元素中添加一个“required”属性，将属性值设置为“true”。当用户单击表单“提交”按钮时，将自动验证输入文本框中内容是否为空；如果为空，会显示错误信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-10.html，加入代码如代码清单 4-10 所示。

代码清单 4-10　<input> 元素中 required 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>input 元素中 required 属性的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>required 属性：</legend> 输入姓名 :
   <input name="txtUserName" type="text"
          class="inputtxt" required="true" />
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 提交 " />
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在不同浏览器中执行的页面效果如图 4-10 所示。

4. 源码分析

在页面的表单中，如果需要验证某个输入框的内容（必须不为空值），只要添加一个“required”属性，并将该属性的值设置为“true”或只是增加属性名称“required”，实现方法如代码中加粗部分所示。设置完成后，在表单提交时，将自动检测该输入框中的内容是否为空。

目前，只有 Chrome 10 与 Opera 11 浏览器支持输入框中的“required”属性。在 HTML 5 中，<input> 元素的绝大部分类型（如 text、password、search、url、email、date pickers、number、checkbox、radio）都支持“required”属性。

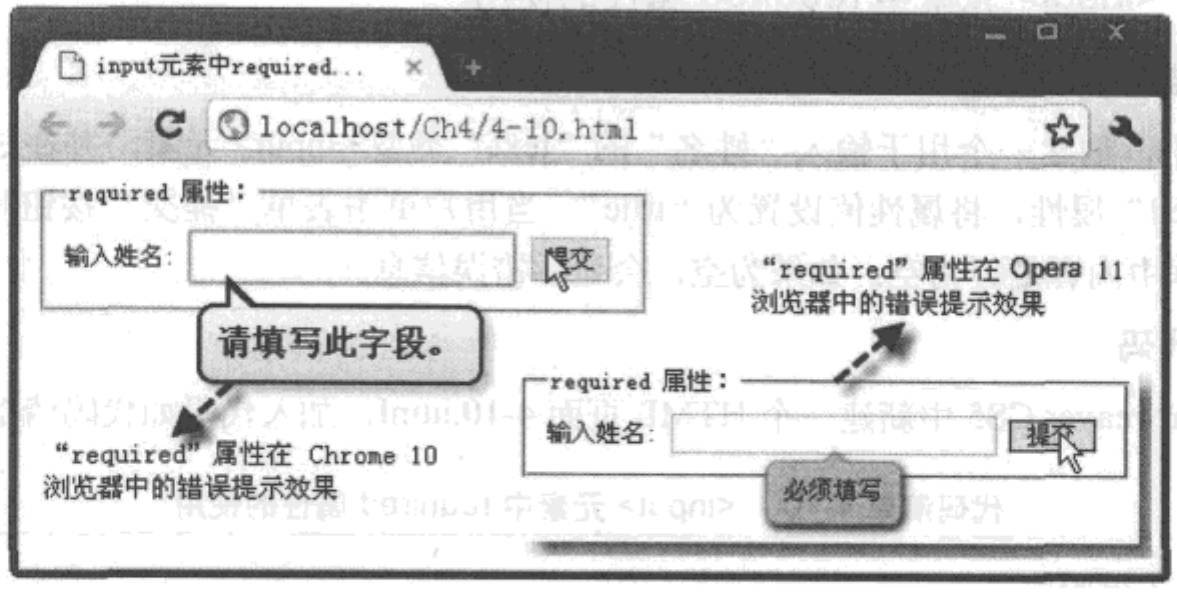


图 4-10 <input> 元素的 required 属性在不同浏览器中使用时的页面效果

4.3 新增表单元素

在 HTML 5 的表单中，除新增 <input> 元素的类型外，还添加许多新的表单元素，如 <datalist>、<keygen>、<output> 等。这些元素的加入，极大丰富了表单数据的操作，优化了用户的体验，下面分别介绍这些新增元素的使用方法与技巧。

4.3.1 datalist 元素

<datalist> 是 HTML 5 中新增加的元素，该元素的功能是辅助表单中文本框的数据输入。<datalist> 元素本身是隐藏的，与表单文本框的“list”属性绑定，即将“list”属性值设置为 <datalist> 元素的“ID”号。绑定成功后，用户单击文本框准备输入内容时，<datalist> 元素以列表的形式显示在文本框的底部，提示输入字符的内容。当用户选中列表中的某个选项后，<datalist> 元素将自动隐藏，同时，文本框中显示所选择的内容。

<datalist> 元素中的列表内容可以动态进行修改，支持与表单中的各类型的输入框（如“email”、“url”、“text”等）进行绑定。

下面通过简单的实例 4-11 介绍 <datalist> 元素的使用过程。

实例 4-11 <datalist> 元素的使用

1. 功能描述

在页面的表单中，新增一个 ID 号为“lstWork”的 <datalist> 元素；另外，创建一个文本输入框，并将文本框的“list”属性设置为“lstWork”，即将文本框与 <datalist> 元素进行绑定。当单击输入框时，将显示 <datalist> 元素中的列表项。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-11.html，加入代码如代码清单 4-11 所示。

代码清单 4-11　<datalist> 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>datalist 元素的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<form id="frmTmp">
 <fieldset>
   <legend> 请输入职业： </legend>
   <input type="text" id="txtWork"
          list="lstWork" class="inputtxt" />
     <datalist id="lstWork">
       <option value=" 程序开发员 "></option>
       <option value=" 系统架构师 "></option>
       <option value=" 数据维护员 "></option>
     </datalist>
 </fieldset>
</form>
<body>
</body>
</html>
```

3. 页面效果

目前，只有在 Opera 11 浏览器中，才支持表单中的文本输入框通过“list”属性绑定 <datalist> 元素。在 Opera 11 浏览器中执行的页面效果如图 4-11 所示。

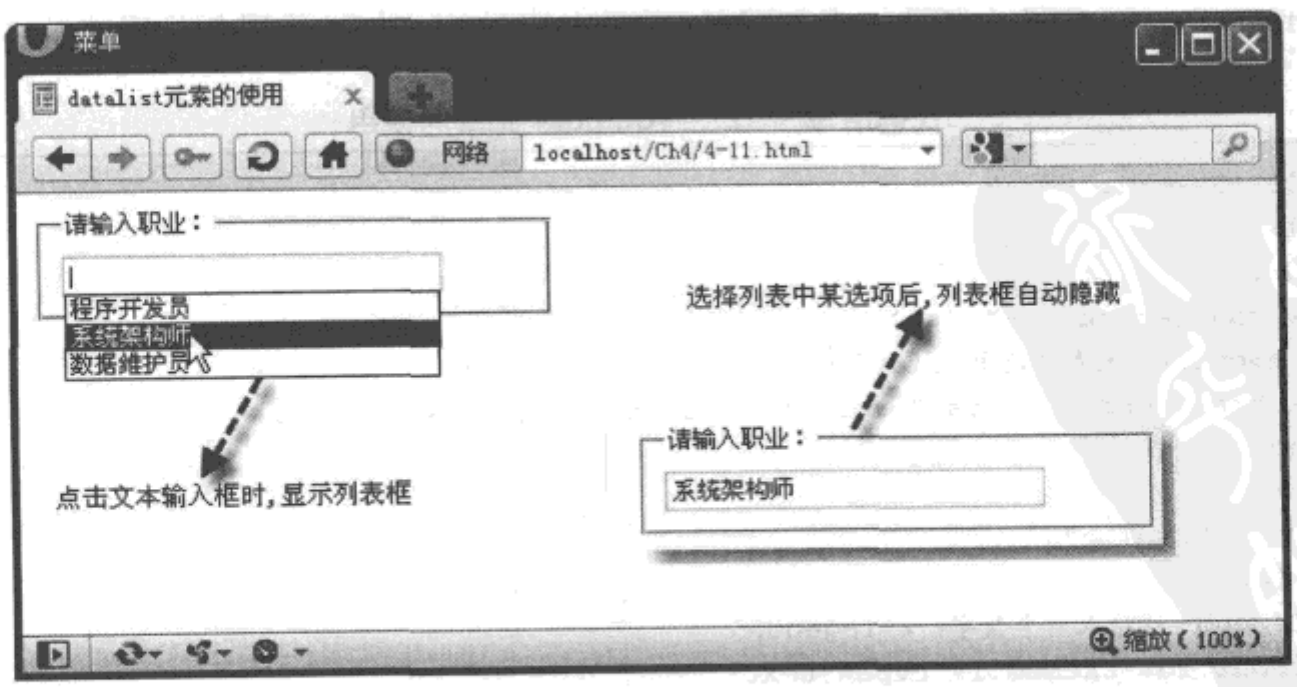


图 4-11　使用 <datalist> 元素绑定输入框在 Opera 11 浏览器中的页面效果

4. 源码分析

如果需要将 <datalist> 元素与文本输入框相绑定，只要将输入框的“list”属性设置为 <datalist> 元素的 ID 号，便完成了两者的绑定，具体实现过程如代码中加粗部分所示。

从实例 4-11 中不难看出，虽然 <datalist> 与 <input> 元素的关系十分密切，但二者还是不同实体的两个元素，无法融合成一个独立的新元素。这也是出于对浏览器兼容性的考虑，因为，如果合成为一个元素，那么，不兼容 <datalist> 元素的浏览器也无法使用与其绑定的文本输入框，这约束了 <input> 元素中文本框的使用范围。

4.3.2 output 元素

<output> 元素是 HTML 5 中新增的表单元素，该元素必须从属于某个表单，或通过属性指定某个表单。该元素的功能是在页面中显示各种不同类型表单元素的内容，如输入框的值、JavaScript 代码执行后的结果值等。为了获取这些值，需要设置 <output> 元素的“onFormInput”事件。在表单输入框中录入内容时，触发该事件，从而十分方便地实时侦察到表单中各元素的输入内容。

下面通过一个简单的实例 4-12 介绍 <output> 元素的使用过程。

实例 4-12 <output> 元素的使用

1. 功能描述

在新建的表单中，创建两个输入文本框，用于输入两个数字；另外，新建一个 <output> 元素，用于显示两个输入文本框中数字相乘后的结果。当改变两个输入框中任意一个数值时，<output> 元素显示的计算结果也将自动进行变化。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-12.html，加入代码如代码清单 4-12 所示。

代码清单 4-12 <output> 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>output 元素的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend> 请输入两个数字：</legend>
   <input id="txtNum_1" type="text"
          class="inputtxt" /> X
```

```
    <input id="txtNum_2" type="text"
           class="inputtxt" /> =
    <output onFormInput=
            "value=txtNum_1.value*txtNum_2.value">
    </output>
  </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 4-12 所示。

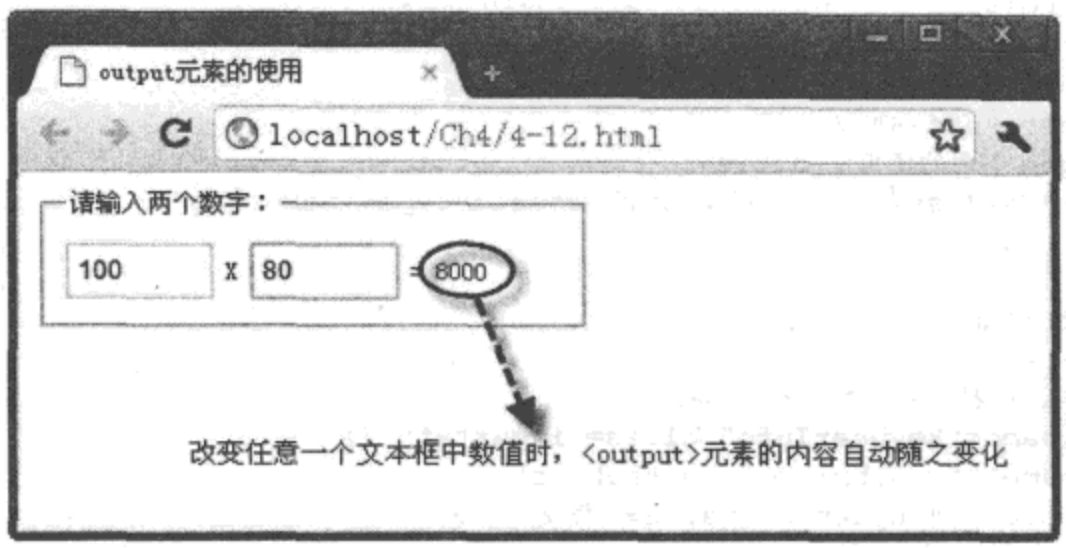


图 4-12　<output> 元素在 Chrome 10 浏览器中使用时的页面效果

4. 源码分析

由于将 <output> 元素的内容通过“onFormInput”事件绑定了两个输入文本框，因此，当输入框中的值发生变化时，<output> 元素的内容根据绑定的规则迅速响应，从而实现了一种联动的效果，详细规则如代码中加粗部分所示。

在代码清单 4-12 中，<output> 元素的“value”值为“txtNum_1.value*txtNum_2.value”，表示将显示的内容绑定为两个输入文本框值的相乘，类似于 this.innerHTML 的表示方法；也可以通过编写 JavaScript 自定义函数，与“onFormInput”事件绑定来实现。

4.3.3　keygen 元素

<keygen> 元素是 HTML 5 中新增加的元素，用于生成页面的密钥。一般情况下，如果在表单中创建该元素，在表单提交时，该元素将生成一对密钥：一个保存在客户端，称为**私密钥**（Private Key）；另一个发送至服务器，由服务器进行保存，称为**公密钥**（Public Key），公密钥可以用于客户端证书的验证。

下面通过简单的实例 4-13 介绍 <keygen> 元素的使用过程。

实例 4-13 <keygen> 元素的使用

1. 功能描述

在页面的表单中，新建一个“name”值为“keyUserInfo”的 <keygen> 元素，该元素将在页面中创建一个选择密钥位数的下拉列表框。当选择列表框中某选项值，并单击表单的“提交”按钮时，将根据所选密钥的位数生成对应密钥，提交给服务器。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-13.html，加入代码如代码清单 4-13 所示。

代码清单 4-13 <keygen> 元素的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>keygen 元素的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend> 请选择密钥位数：</legend>
   <keygen name="keyUserInfo" class="inputtxt" />
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 提交 " />
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 4-13 所示。

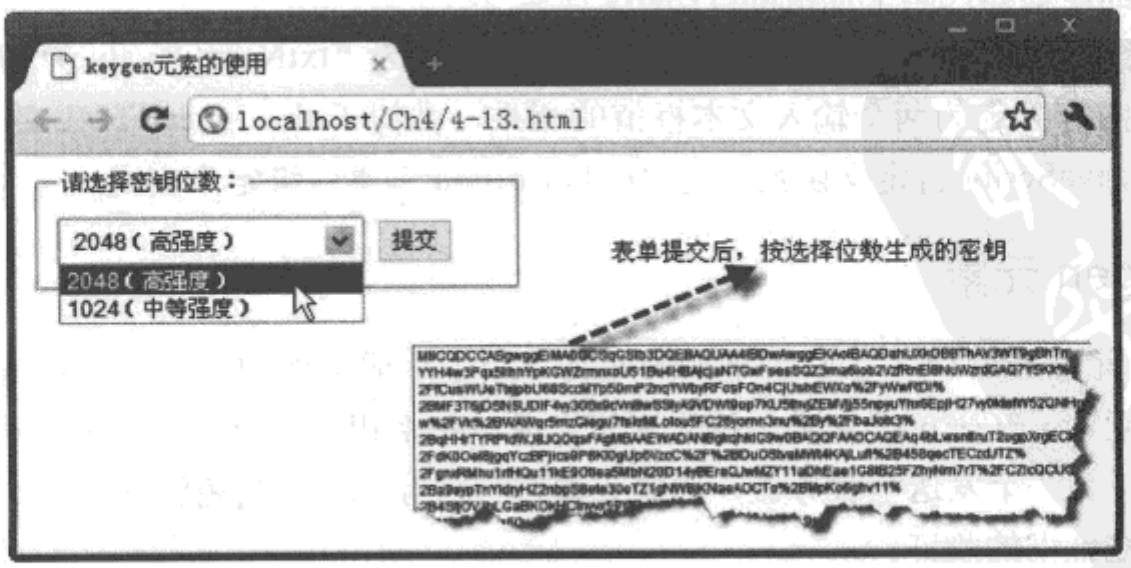


图 4-13 <keygen> 元素在 Chrome 10 浏览器中使用时的页面效果

4. 源码分析

<keygen> 元素在表单中以列表的形式展示，提供密钥位数的选择；当表单提交时，可以通过 <keygen> 元素在表单中的“name”值，获取该元素生成的对应位数密钥，代码如加粗部分所示。

另外，<keygen> 元素中“keyType”属性表明生成密钥的类型，如设置为“rsa”，则以“rsa”加密类型生成相应位数的密钥。

目前，只有 Chrome 10 与 Opera 11 浏览器支持该元素，因此，如果把 <keygen> 元素作为客户端安全保护的一种有效措施，还需要时间。

4.4　表单新增的验证方法和属性

在 HTML 5 中，表单是一个十分重要的容器型元素，其中各元素的验证结果与信息反馈，直接影响到数据是否能够顺利提交。为了加强元素属性的验证效果，在 <form> 表单元素中，新增了许多与表单验证相关的方法与属性，如方法 checkValidity() 和 setCustomValidity()、属性“novalidate”等。接下来，详细介绍这些方法与属性在表单中的使用过程。

4.4.1　checkValidity 显式验证法

表单中的各元素可以利用“pattern”与“required”属性，验证元素内容的有效性，此外，每个元素都可以通过调用一个公用方法 checkValidity()，核实本身是否与验证条件匹配。如果一致，返回 true，否则返回 false。开发人员可以编写 JavaScript 代码调用元素的该方法，将验证结果展示出来。因此，这种方法又称为“显式验证法”。

下面通过简单的实例 4-14 介绍调用表单中 checkValidity() 方法的过程。

实例 4-14　调用表单的 checkValidity 方法

1. 功能描述

在页面的表单中，创建一个用于输入“密码”的文本框，并使用“pattern”属性自定义相应的“密码”验证规则。另外，使用 JavaScript 代码编写一个表单提交时触发的函数 chkPassWord()，该函数将显式地检测“密码”输入文本框的内容是否与自定义的验证规则匹配。如果不符合，在文本输入框的右边显示一个“×”，否则，显示一个“√”。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-14.html，加入代码如代码清单 4-14 所示。

代码清单 4-14　调用表单的 checkValidity 方法

```
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="utf-8" />
<title>form 中 checkValidity 方法的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="javascript"
        src="Js/js14.js">
</script>
</head>
<body>
<form id="frmTmp" onSubmit="return chkPassWord()">
 <fieldset>
   <legend>checkValidity() 方法 </legend>
   <p> 输入密码 :
   <input name="txtPassWord" id="txtPassWord"
          type="password" class="inputtxt"
          pattern="^[a-zA-Z]\w{5,7}$" required /> *
   <span id="spnPassWord"></span></p>
   <p class="p_center">
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 确定 " />
   </p>
 </fieldset>
</form>
</body>
</html>
```

代码清单 4-14 中引用一个 JavaScript 文件 js14.js，在该文件中，调用 checkValidity() 方法，显式地展示元素本身与验证规则是否匹配的结果，其实现的代码如下所示：

```
// JavaScript Document
function $$(id){
   return document.getElementById(id);
}
// 检测密码是否验证成功
function chkPassWord(){
  var $$Pass=$$("txtPassWord");
  var $$spnP=$$("spnPassWord");
  var strP;
  if($$Pass.checkValidity()){
    strP=" √ ";
   }
  else{
    strP="×";
   }
   $$spnP.innerHTML=strP;
   return false;
}
```

3. 页面效果

Opera 11 浏览器中执行该页面的效果如图 4-14 所示。

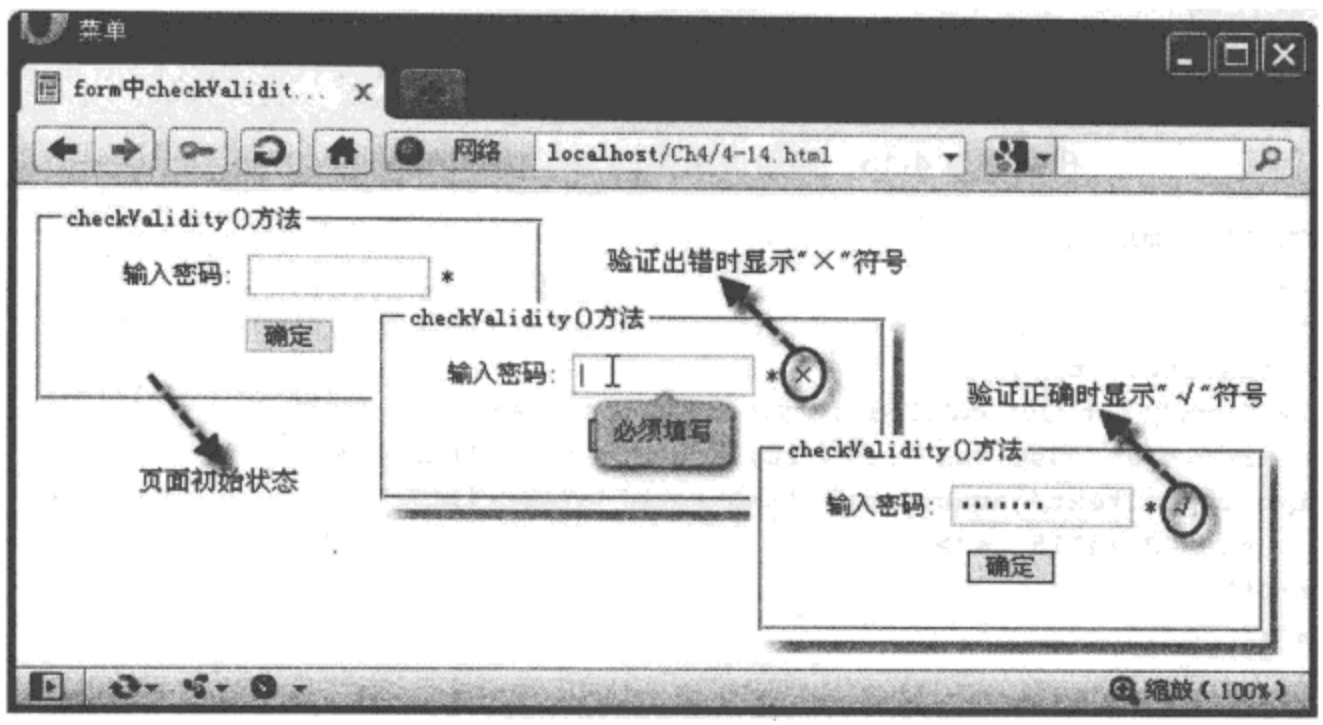


图 4-14　调用 checkValidity 方法展示验证结果

4. 源码分析

表单或元素调用 checkValidity() 方法后，将返回一个布尔值，即“true”或“false”。如果为“true”，说明表单全部元素或指定元素的内容符合验证规则；否则，说明与验证规则不相符。

注意　目前为止，只有 Opera 11 才支持该方法，其他浏览器暂时还不支持该方法。

4.4.2　使用 setCustomValidity 方法修改提示信息

在表单（或表单元素内容）与相应规则验证时，由于使用的是系统内置的验证方法，因此，出错提示信息也是由系统自带的，一般情况下不修改。但有时，为了与系统出错信息的格式一致，需要修改验证出错信息内容，实现的方法是调用元素或表单的 setCustomValidity() 方法，该方法括号中的内容就是修改后的错误提示信息。

下面通过简单的实例 4-15 介绍调用表单中 setCustomValidity() 方法的过程。

实例 4-15　调用表单的 setCustomValidity 方法

1. 功能描述

在表单中，创建两个“text”类型的 <input> 元素，用于输入两次“密码”值。在提交表单时，调用一个 JavaScript 代码编写的自定义函数 setErrorInfo()；该函数获取两次输入的

“密码”值后，检测两次输入是否一致，并调用元素的 setCustomValidity() 方法修改系统验证的错误信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-15.html，加入代码如代码清单 4-15 所示。

代码清单 4-15　调用表单中的 setCustomValidity 方法

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>form 中 setCustomValidity 方法的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="javascript"
        src="Js/js15.js">
</script>
</head>
<body>
<form id="frmTmp" onSubmit="return setErrorInfo()">
 <fieldset>
   <legend>setCustomValidity() 方法 </legend>
   <p> 输入密码：
   <input name="txtPassWord_1" id="txtPassWord_1"
          type="password" class="inputtxt"
          pattern="^[a-zA-Z]\w{5,7}$" required /> *
   </p>
   <p> 再次输入：
   <input name="txtPassWord_2" id="txtPassWord_2"
          type="password" class="inputtxt"
          pattern="^[a-zA-Z]\w{5,7}$" required /> *
   </p>
   <p class="p_center">
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 确定 " />
   </p>
 </fieldset>
</form>
</body>
</html>
```

代码清单 4-15 中引用一个 JavaScript 文件 js15.js，在该文件中，调用 setCustomValidity () 方法修改验证后的系统提示信息，其实现的代码如下所示：

```
// JavaScript Document
function $$(id){
   return document.getElementById(id);
```

```
}
// 检测密码是否验证成功
function setErrorInfo(){
  var $$Pass_1=$$("txtPassWord_1");
  var $$Pass_2=$$("txtPassWord_2");
  if($$Pass_1.value==$$Pass_2.value){
    $$Pass_2.setCustomValidity(" √，两次密码相同！");
   }
  else{
    $$Pass_2.setCustomValidity("×，两次密码不一样！");
   }
}
```

3. 页面效果

Opera 11 浏览器中执行该页面的效果如图 4-15 所示。

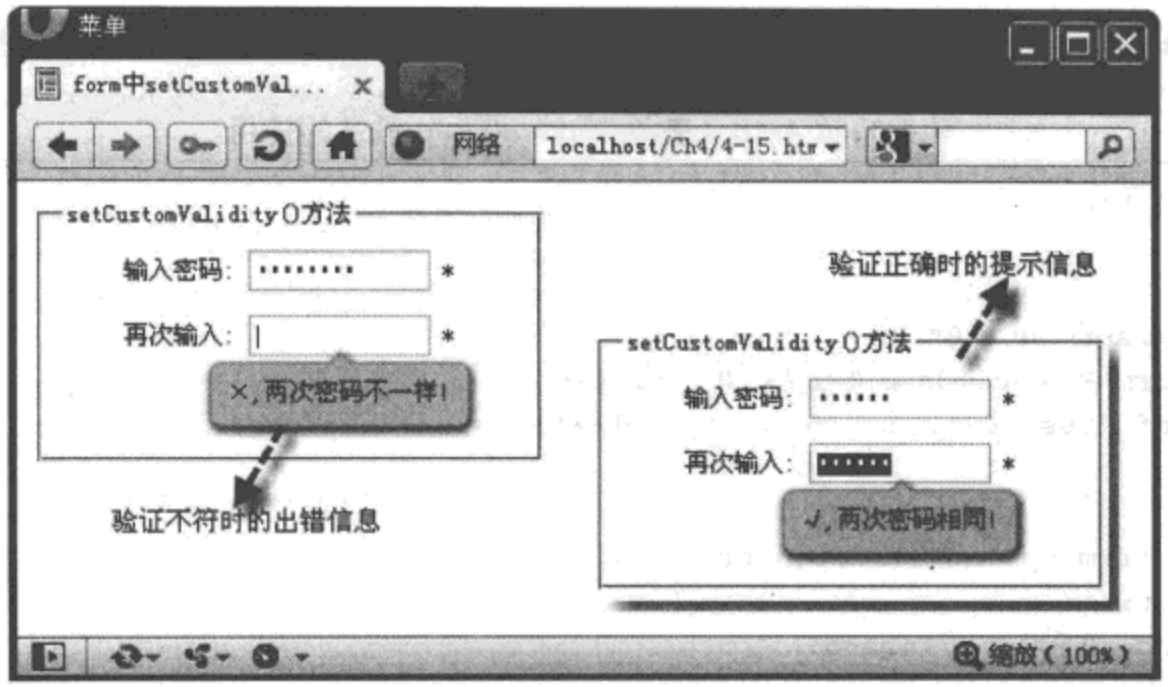


图 4-15　调用 setCustomValidity 方法修改验证结果

4. 源码分析

使用 JavaScript 代码调用元素的 setCustomValidity() 方法，可以很方便地修改元素在验证时展示的系统提示信息，其调用格式如下：

```
页面元素 .setCustomValidity(" 系统提示信息 ");
```

注意　目前为止，只有 Opera 11 才支持该方法，其他浏览器暂时还不支持该方法。

4.4.3　表单的 novalidate 属性

表单中新增加的“novalidate”属性，用于取消表单全部元素的验证。在表单提交时，通常需要逐个验证元素的内容是否与既定的规则一致，如果所有元素都相符，才能进行表单数

据的提交，否则，提示出错信息。但有时在表单提交时，并不希望元素进行验证，这时，如果是单个元素，可以动态设置其验证方法，但如果是表单，则可以通过设置该属性来实现。

下面通过简单的实例 4-16 介绍设置表单中“novalidate”属性的过程。

实例 4-16　表单中 novalidate 属性的使用

1. 功能描述

在表单中创建一个用户登录界面，其中包括两个“text”类型的输入文本框，一个用于输入“用户名”，另一个用于输入“密码”，并都通过“pattern”属性设置相应的输入框验证规则。另外，将表单的“novalidate”属性设置为“true”，单击表单“提交”按钮后，表单中的元素将不会进行内置的验证，而直接进行数据的提交。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 4-16.html，加入代码如代码清单 4-16 所示。

代码清单 4-16　表单中 novalidate 属性的使用

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>form 中 novalidate 属性的使用 </title>
<link href="Css/css4.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp" novalidate="true">
 <fieldset>
   <legend> 用户登录 </legend>
   <p> 姓名 :
   <input name="txtUserName" id="txtUserName"
          type="text" class="inputtxt"
          pattern="^[a-zA-Z]\w{5,7}$" required /> *
   </p>
   <p> 密码 :
   <input name="txtPassWord" id="txtPassWord"
          type="password" class="inputtxt"
          pattern="^[a-zA-Z]\w{5,7}$" required /> *
   </p>
   <p class="p_center">
   <input name="frmSubmit" type="submit"
          class="inputbtn" value=" 登录 " />
   <input name="frmReset" type="reset"
          class="inputbtn" value=" 取消 " />
   </p>
 </fieldset>
```

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

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 4-16 所示。

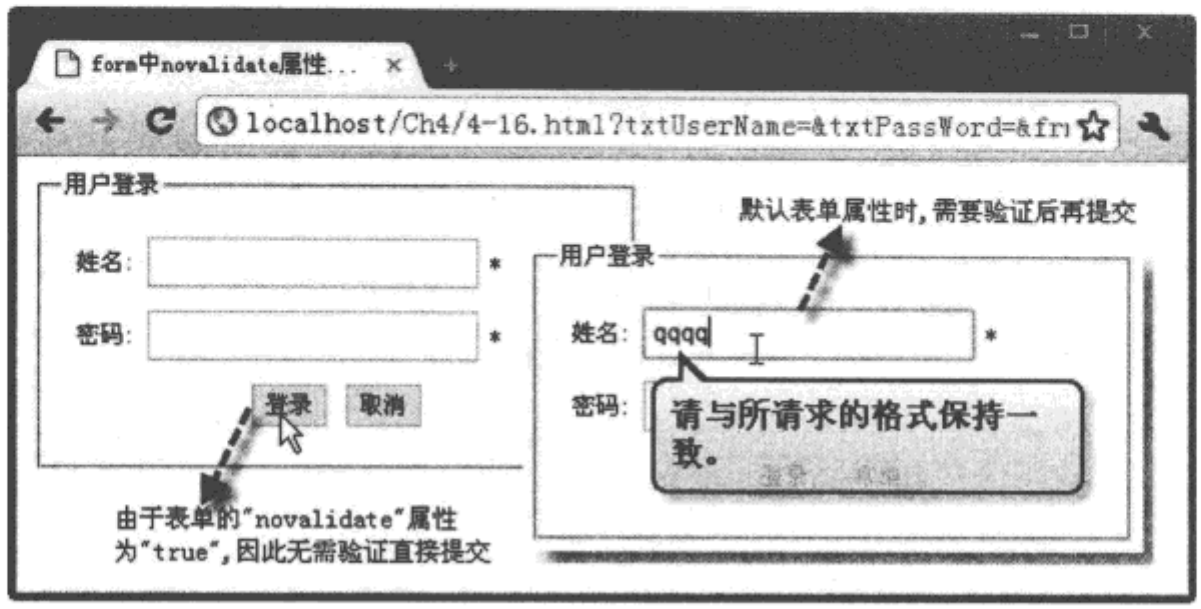


图 4-16　表单中 novalidate 属性设置不同值时的页面效果

4. 源码分析

不仅表单具有“novalidate”属性，大部的文本输入框元素如 tex、search、url、email、password、date pickers、range 等元素都支持该属性。由于表单中的元素不经过验证而直接提交给服务器，可能会带来安全隐患，因此，考虑数据的安全性，建议谨慎使用此属性。

4.5　本章小结

无论是在 HTML 4 还是 HTML 5 中，表单都是一个十分重要的页面元素。本章先从 <input> 的新增类型开始，由浅入深地介绍了表单中各种新增加元素与属性的使用方法；最后，详细介绍了表单在提交数据时两个重要的验证方法，进一步巩固前面章节中所学的知识。

PDG

第5章 HTML 5中的文件

本章内容

- 选择文件
- 使用 FileReader 接口读取文件
- 使用 DataTransfer 对象拖放上传图片文件
- 文件读取时的错误与异常
- 本章小结

在 HTML 5 中，专门提供了一个页面层调用的 API 文件，通过调用这个 API 文件中的对象、方法和接口，可以很方便地访问文件的属性或读取文件内容。本章节将详细介绍在 HTML 页面中，使用“file”类型的 <input> 元素，访问或操纵本地文件的方法。

5.1 选择文件

与 HTML 4 之前版本相同，在 HTML 5 中，可以创建一个“file”类型的 <input> 元素实现文件的上传功能；只是在 HTML 5 中，该类型的 <input> 元素新添加了一个“multiple”属性，如果将属性的值设置为“true”，那么，则可以在一个元素中实现多个文件的上传。另外，通过访问 Blob 对象，可以获取上传文件的类型、大小属性，下面分别进行详细的介绍。

5.1.1 选择单个文件

在 HTML 5 中，创建一个“file”类型的 <input> 元素上传文件时，该元素在页面中的展示方式发生了变化，不再有文本框，而是一个“选择文件”的按钮，按钮的右侧显示选择上传文件的名称。初始化页面时，没有上传文件，因此，显示“未选择文件”字样。

下面通过实例 5-1 介绍选择单个文件上传的过程。

实例 5-1 选择单个文件上传

1. 功能描述

在页面中，创建一个“file”类型的 <input> 元素，单击该元素的“选择文件”按钮，选择一个图片文件。单击“打开”按钮或双击该文件后，在“选择文件”按钮的右侧，如果显示该图片文件的名称，表明已将该文件选中，正在等待上传。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-1.html，加入代码如代码清单 5-1 所示。

代码清单 5-1 选择单个文件上传

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>选择单个文件上传</title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>上传单个文件：</legend>
   <input type="file" name="fleUpload" id="fleUpload"/>
```

```
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-1 所示。

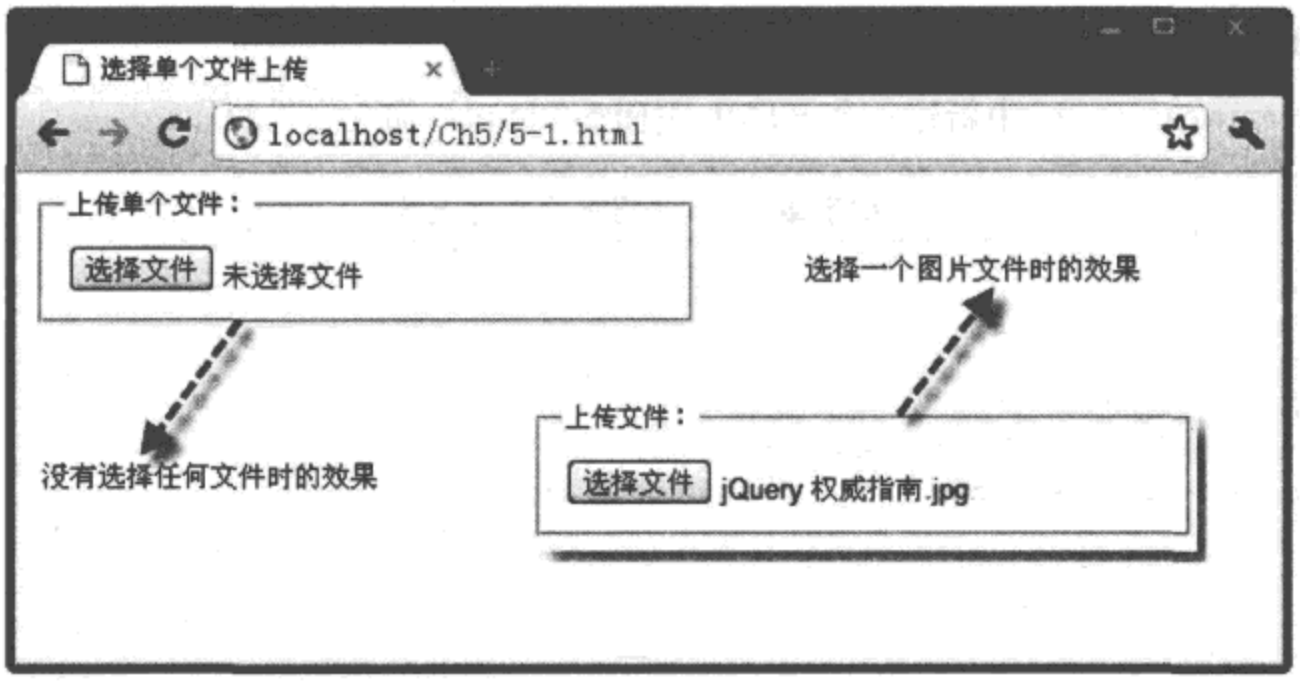


图 5-1　使用 file 类型的 <input> 元素实现选择单个文件

4. 源码分析

在本示例中，单击“选择文件”按钮选中上传文件后，没有编写任何 JavaScript 代码，就可以在页面中显示所选择的文件名称。如果名称太长，在页面显示时，将采取两端显示、中间省略的形式展示。例如，图片名称为“jQuery 权威指南 _2010 年作品 .jpg”，页面将显示为“jQuery 权威 ...10 年作品 .jpg”。

5.1.2　选择多个文件

在 HTML 5 中，除了可以选择单个文件外，还可以通过添加元素的“multiple”属性，将该属性值设为“true”，实现选择多个文件的功能。

一个文件对应一个“file”对象，该对象中有两个重要的属性：一个是“name”，表示不包含路径的文件名称；另外一个属性是“lastModifiedDate”，表示文件最后修改的时间。当使用“file”类型的 <input> 元素选择多个文件时，该元素中就含有多个“file”对象，从而形成了“FileList”对象，即“file”对象的列表。

下面通过实例 5-2 介绍选择多个文件上传的过程。

实例 5-2 选择多个文件上传

1. 功能描述

在页面中，创建一个“file”类型的<input>元素，添加“multiple”属性，并将该属性的值设置为“true”。单击“选择文件”时，同时选择 3 个文件；单击“打开”按钮后，在“选择文件”按钮的右侧将显示“3 个文件”的字样。移动鼠标至文字上时，显示这 3 个文件的详细名称与类型。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-2.html，加入代码如代码清单 5-2 所示。

代码清单 5-2 选择多个文件上传

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>选择多个文件上传</title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>上传多个文件：</legend>
   <input type="file" name="fleUpload" id="fleUpload" multiple="true"/>
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-2 所示。

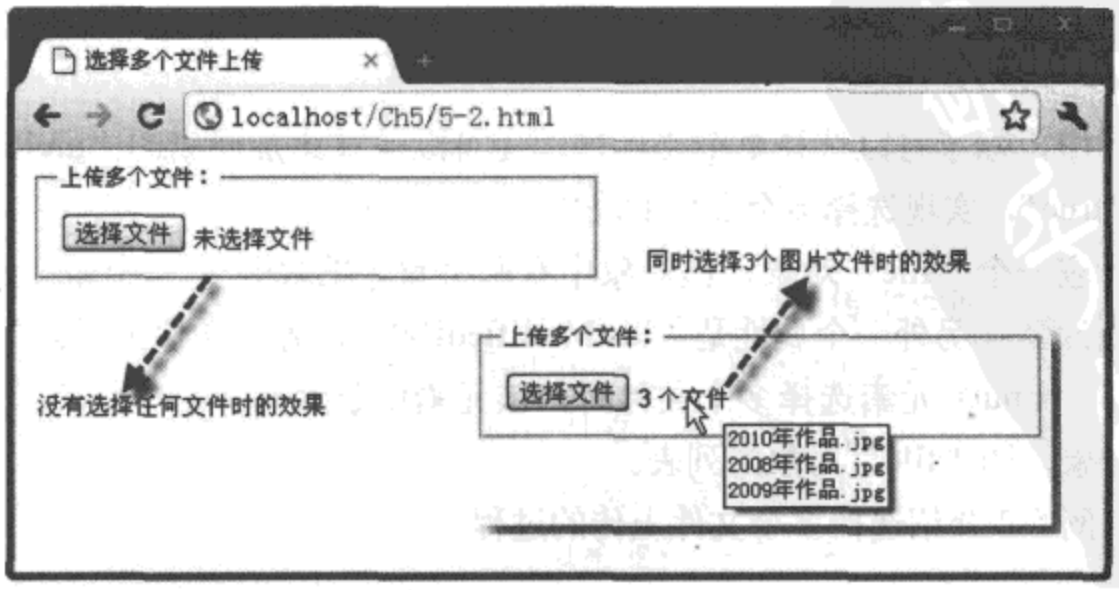


图 5-2 使用 multiple 属性实现选择多个文件

4. 源码分析

在本示例中，由于“file”类型的 <input> 元素中添加了“multiple”属性，因此，可以通过该元素选择多个文件。选择成功后，“选择文件”按钮右侧不再显示文件的名称，而是显示成功选择文件的总量；当将鼠标移至总量时，显示全部上传文件的详细列表。

当多个文件被选中时，在上传文件元素中，将会产生一个“FileList”对象，用来装载各文件的基本信息，如文件名称、类型、大小等，在上传文件总量的文字上移动鼠标时，将调用该对象的列表信息，展示在页面中。

5.1.3　使用 Blob 接口获取文件的类型与大小

Blob 表示二进制数据块，Blob 接口中提供了一个 slice 方法，通过该方法可以访问指定长度与类型的字节内部数据块。该接口提供了两个属性：一个为“size”，表示返回数据块的大小；另外一个为“type”，表示返回数据块的 MIME 类型，如果不能确定数据块的类型，则返回一个空字符串。实例 5-1 与实例 5-2 中的“file”对象，实质上是 Blob 接口的一个实体，完全继承了该接口中的方法与属性。

下面通过实例 5-3 介绍“file”对象继承 Blob 接口获取文件类型与大小的过程。

实例 5-3　获取上传文件的类型与大小

1. 功能描述

在页面的表单中，创建一个“file”类型的 <input> 元素，并将该元素的“multiple”属性设置为“true”，表示允许选择多个文件。单击“选择文件”按钮，选取多个需要上传的文件后，在页面中将以列表的方式展示所选文件的名称、类型、大小信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-3.html，加入代码如代码清单 5-3 所示。

代码清单 5-3　获取上传文件的类型与大小

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>获取上传文件的类型与大小</title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js3.js"/>
</script>
</head>
<body>
<form id="frmTmp">
 <fieldset>
```

```
    <legend>上传多个文件：</legend>
    <input type="file" name="fleUpload" id="fleUpload"
          onChange="fileUpload_GetFileList(this.files);"
          multiple="true"/>
    <ul id="ulUpload"></ul>
  </fieldset>
</form>
</body>
</html>
```

在代码清单 5-3 中，页面导入一个 JavaScript 文件 js3.js，其中调用 fileUpload_GetFileList() 方法，以列表的形式展示上传文件的数据信息，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_GetFileList(f) {
    var strLi = "<li class='li'>";
    strLi = strLi + "<span>文件名称</span>";
    strLi = strLi + "<span>文件类型</span>";
    strLi = strLi + "<span>文件大小</span>";
    strLi = strLi + "</li>";
    for (var intI = 0; intI < f.length; intI++) {
        var tmpFile = f[intI];
        strLi = strLi + "<li>";
        strLi = strLi + "<span>" + tmpFile.name + "</span>";
        strLi = strLi + "<span>" + tmpFile.type + "</span>";
        strLi = strLi + "<span>" + tmpFile.size + " KB</span>";
        strLi = strLi + "</li>";
    }
    $$("ulUpload").innerHTML = strLi;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-3 所示。

4. 源码分析

如页面加粗代码所示，“file”类型的 <input> 元素选择上传文件时，将触发“onChange”事件。在该事件中，调用自定义的函数 fileUpload_GetFileList(this.files)，其中，实参“this.files”表示所选择的上传文件集合，即 FileList 对象。在函数 fileUpload_GetFileList() 中，遍历传回的 FileList 文件集合，获取单个的“file”对象。该对象通过继承 Blob 接口的属性，返回文件的名称、类型、大小信息，并将这些信息以叠加的方式保存在变量“strLi”中；最后，将变量的内容赋值给 ID 号为“ulUpload”的列表元素，通过该元素，将上传文件的信息展示在

页面中，详细实现过程如 JavaScript 代码中加粗部分所示。

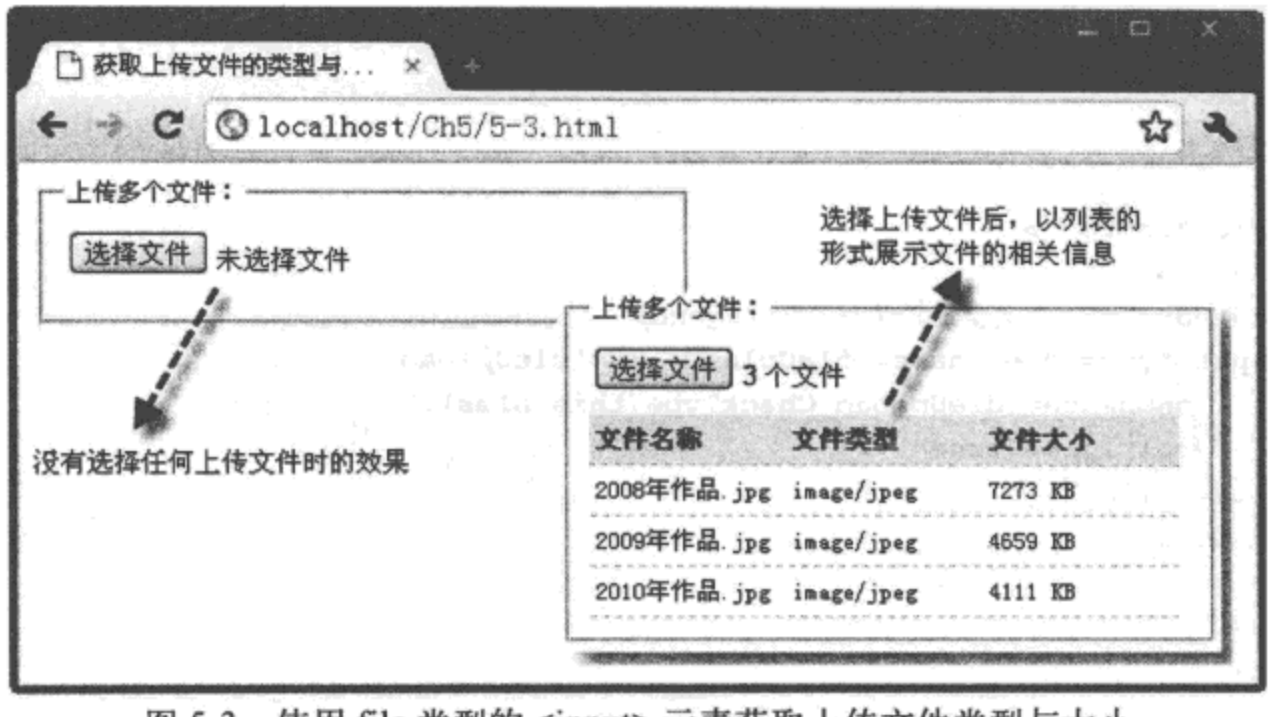


图 5-3　使用 file 类型的 <input> 元素获取上传文件类型与大小

5.1.4　通过类型过滤选择的文件

在实例 5-3 中，通过“file”对象可以获取每个上传文件的名称、类型、大小，根据这个特征，可以过滤上传文件的类型。具体流程是：当选择上传文件后，遍历每一个“file”对象，获取该对象的类型，并将该类型与设置的过滤类型进行匹配；如果不符合，则提示上传文件类型出错或拒绝上传，从而实现在选择文件时过滤掉不需要上传的文件。

下面通过实例 5-4 介绍“file”对象通过类型过滤选择文件的过程。

实例 5-4　通过类型过滤上传文件

1. 功能描述

在页面表单中，创建一个“file”类型的 <input> 元素，并设置“multiple”属性为“true”，用于选择多个文件。当单击“选择文件”按钮，并选取了需要上传的文件后，如果选取的文件中存在不符合“图片”类型的文件，将在页面中显示总数量与文件名称。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-4.html，加入代码如代码清单 5-4 所示。

代码清单 5-4　通过类型过滤上传文件

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>通过类型过滤上传文件</title>
```

```
<link href="Css/css5.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
          src="Js/js4.js"/>
</script>
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>上传过滤类型后的文件：</legend>
   <input type="file" name="fileUpload" id="fileUpload"
         onChange="fileUpload_CheckType(this.files);"
         multiple="true" />
   <p id="pTip"/>
 </fieldset>
</form>
</body>
</html>
```

在代码清单5-4中，页面导入一个JavaScript文件js4.js，其中调用fileUpload_CheckType()方法，按设置的类型格式过滤需要上传的文件，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 选择上传文件时调用的函数
function fileUpload_CheckType(f) {
    var strP = "",
    strN = "",
    intJ = 0;
    var strFileType = /image.*/;
    for (var intI = 0; intI < f.length; intI++) {
        var tmpFile = f[intI];
        if (!tmpFile.type.match(strFileType)) {
            intJ = intJ + 1;
            strN = strN + tmpFile.name + "<br>";
        }
    }
    strP = "检测到（" + intJ + "）个非图片格式文件。";
    if (intJ > 0) {
        strP = strP + "文件名如下：<p>" + strN + "</p>";
    }
    $$("pTip").innerHTML = strP;
}
```

3. 页面效果

该页面在Chrome 10浏览器中执行的页面效果如图5-4所示。

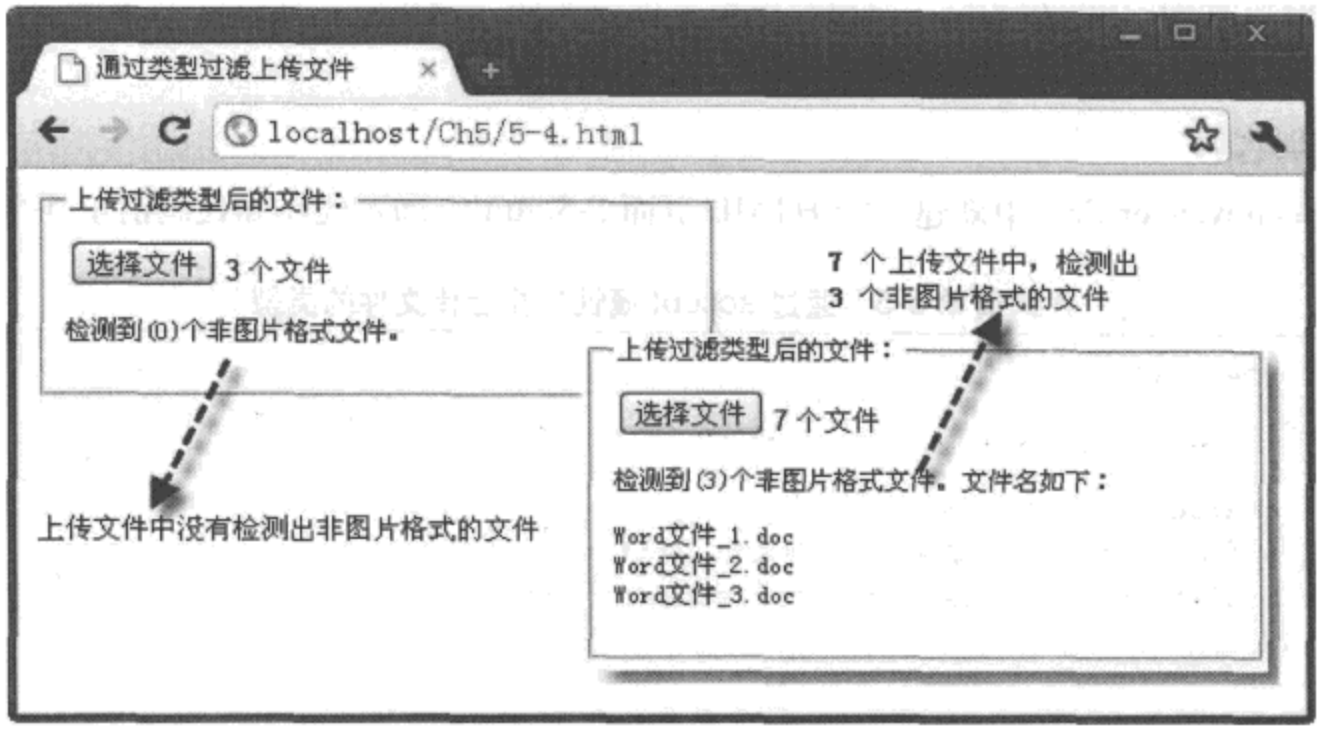


图 5-4　使用 file 类型的 <input> 元素过滤上传文件类型

4. 源码分析

在实例 5-3 中我们知道，如果上传文件是图片类型，则“file”对象返回的类型均以“image/”开头，后面添加“*”表示所有的图片类型，或添加“gif”表示某种类型图片；因此，如果是一个图片文件，该文件返回的类型必定以“image/”字样开头。

根据这一特点，在本实例中，当遍历传回的文件集合时，通过 match() 方法检测每个文件返回的类型中是否含有“image/*”字样。如果没有，说明是非图片类型文件，分别将总量与文件名称以叠加的形式保存在变量中；然后，将变量的内容赋值给 ID 号为“pTip”的元素；最后，通过该元素显示全部过滤文件的总量与名称表，详细实现过程如 JavaScript 代码中加粗部分所示。

5.1.5　通过 accept 属性过滤选择文件的类型

在选择上传文件后，如果能根据文件返回的类型过滤所选择的文件，将是一种不错的方法，但需要编写不少的代码。在 HTML 5 中，可以设置“file”类型 <input> 元素的“accept”属性为文件的过滤类型；设置完“accept”属性值后，在打开窗口选择文件时，默认的文件类型就是所设置的过滤类型。

下面通过一个实例 5-5 简单介绍“file”类型 <input> 元素通过属性过滤选择文件类型的过程。

实例 5-5　通过 accept 属性过滤上传文件的类型

1. 功能描述

在页面表单中，创建一个“file”类型的 <input> 元素，并在元素中添加一个“accept”属

性，属性值设置为“image/gif”。当用户单击“选择文件”按钮时，在打开的文件选择窗口中，文件类型为“accept”属性所设置的值。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-5.html，加入代码如代码清单 5-5 所示。

代码清单 5-5　通过 accept 属性过滤上传文件的类型

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>通过 accept 属性过滤某类型上传文件 </title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>选择某类型上传文件：</legend>
   <input type="file" name="fleUpload"
          id="fleUpload" accept="image/gif" />
 </fieldset>
</form>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-5 所示。

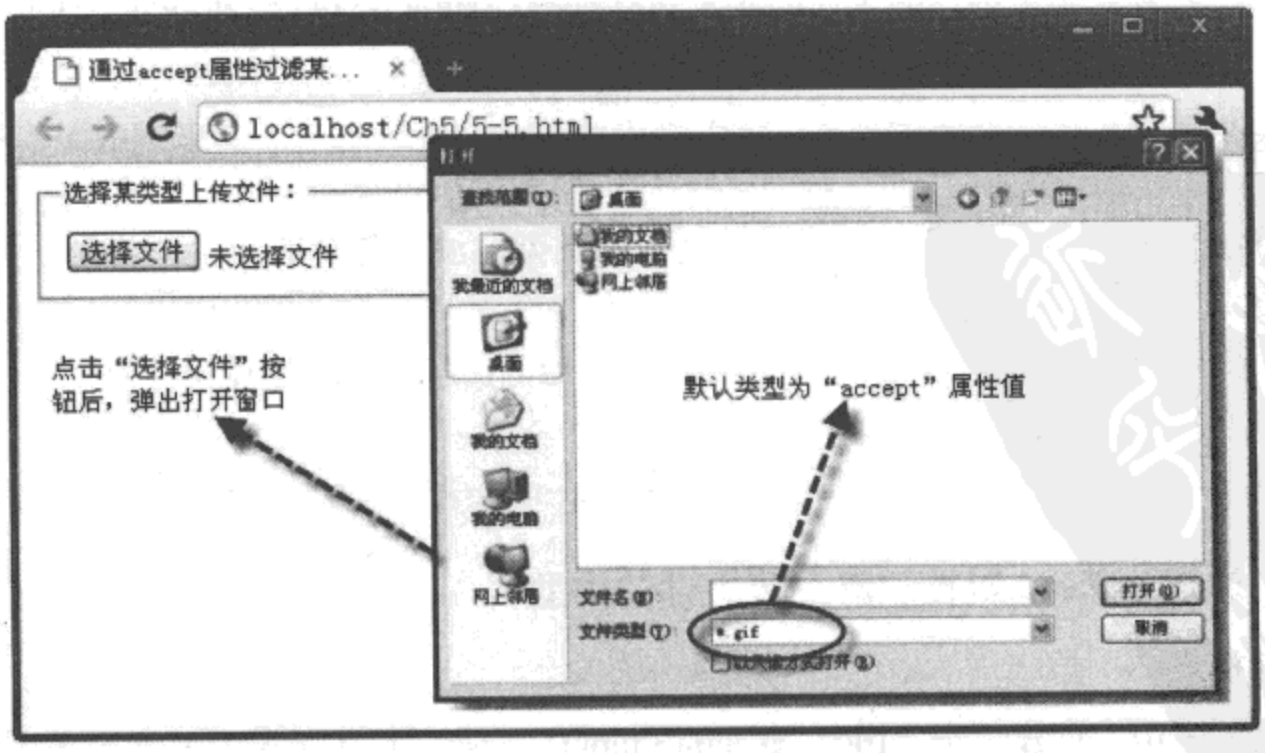


图 5-5　通过设置属性过滤上传文件类型

4. 源码分析

在本实例中，由于对文件元素添加了“accept”属性，并设置“image/gif”类型作为该属性的值，因此，当单击“选择文件”按钮打开窗口时，其默认的选择文件类型就是所设置“accept”属性值，实现过程如代码中加粗部分所示。

通过简单设置元素的一个属性，就可以在文件选择前过滤所选文件的类型，这种方法比代码简单，操作方便，但是在目前的浏览器中，该方法不是很有效。原因在于，即便通过属性设置了文件选择的类型，但不是该类型的文件同样也可以被选中，也能被文件元素所接受。因此，使用这种过滤上传文件类型方法时，还需要谨慎。

5.2　使用 FileReader 接口读取文件

使用 Blob 接口可以获取文件的相关信息，如文件名称、大小、类型，但如果想要读取或浏览文件，则需要通过 FileReader 接口。该接口不仅可以读取图片文件，还可以读取文本或二进制文件；同时，根据该接口提供的事件与方法，可以动态侦察文件读取时的详细状态。本节详细介绍 FileReader 接口的使用方法。

5.2.1　FileReader 接口的方法

FileReader 接口提供了一个异步的 API，通过这个 API 可以从浏览器主线程中异步访问文件系统中的数据。因此，FileReader 接口可以读取文件中的数据，并将读取的数据放入内存中。

当访问不同文件时，必须重新调用 FileReader 接口的构造函数。因为每调用一次，FileReader 接口都将返回一个新的 FileReader 对象，只有这样，才能实现访问不同文件的数据。

在 FileReader 接口中拥有许多常用的方法，用于读取文件或响应事件，如 onabort 事件触发时，需要调用 abort() 方法。FileReader 接口的常用方法如表 5-1 所示。

5.2.2　使用 readAsDataURL 方法预览图片

通过 fileReader 接口中的 readAsDataURL() 方法，可以获取 API 异步读取的文件数据，另存为数据 URL；将该 URL 绑定 <img> 元素的“src”属性值，就可以实现图片文件预览的效果。

下面通过实例 5-6 介绍使用 readAsDataURL() 方法预览图片的过程。

表 5-1 FileReader 接口的常用方法

方法名称	参数	功能描述	使用说明
readAsBinaryString()	file	以二进制的方式读取文件内容	调用该方法时，将 file 对象返回的数据块，作为一个二进制字符串的形式，分块读入内存中
readAsArrayBuffer()	file	以数组缓冲的方式读取文件内容	调用该方法时，将 file 对象返回的数据字节数，以数组缓冲的方式读入内存中
readAsDataURL()	file	以数据 URL 的方式读取文件内容	调用该方法时，将 file 对象返回的数据块，以一串数据 URL 字符的形式展示在页面中，这种方法一般读取数据块较小的文件
readAsText()	file, encoding	以文本编码的方式读取文件内容	调用该方法时，其中 encoding 参数表示文本文件编码的方式，默认值为 utf-8, 即以 utf-8 编码格式将获取的数据块按文本方式读入内存中
abort()	无	读取数据中止时，将自动触发该方法	如果在读取文件数据过程中出现异常或错误时，触发该方法，返回错误代码信息

实例 5-6 使用 readAsDataURL 方法预览图片

1. 功能描述

在页面表单中，添加一个“file”类型的 <input> 元素，用于选择上传文件，设置属性“multiple”的值为“true”，表示允许选择多个文件。单击“选择文件”按钮后，如果选择的是图片文件，将在页面中显示。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-6.html，加入代码如代码清单 5-6 所示。

代码清单 5-6 使用 readAsDataURL 方法预览图片

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 readAsDataURL() 方法预览图片 </title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
          src="Js/js6.js"/>
</script>
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>预览图片文件：</legend>
   <input type="file" name="fleUpload" id="fleUpload"
```

```
            onChange="fileUpload_PrevImageFile(this.files);"
            multiple="true"/>
    <ul id="ulUpload"></ul>
  </fieldset>
</form>
</body>
</html>
```

在代码清单 5-6 中，页面导入一个 JavaScript 文件 js6.js，其中调用 fileUpload_PrevImageFile() 方法，该方法访问 fileReader 接口，将文件以数据 URL 的方式返回页面，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_PrevImageFile(f) {
    //检测浏览器是否支持 FileReader 对象
    if (typeof FileReader == 'undefined') {
        alert("检测到您的浏览器不支持 FileReader 对象！ ");
    }
    var strHTML = "";
    for (var intI = 0; intI < f.length; intI++) {
        var tmpFile = f[intI];
        var reader = new FileReader();
        reader.readAsDataURL(tmpFile);
        reader.onload = function(e) {
            strHTML = strHTML + "<span>";
            strHTML = strHTML + "<img src='" + e.target.result
                    + "' alt=''/>";
            strHTML = strHTML + "</span>";
            $$("ulUpload").innerHTML = "<li>" + strHTML + "</li>";
        }
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-6 所示。

4. 源码分析

在本实例中，图片预览的过程实质上是图片文件被读取后展示在页面中的过程，为了实现这一过程，需要引用 FileReader 接口提供的读取文件方法 readAsDataURL()。在引用接口前，考虑到各浏览器对接口的兼容性不一样，首先利用 JavaScript 代码检测用户的浏览器是否支持 FileReader 对象，如果不支持，则提示出错信息。

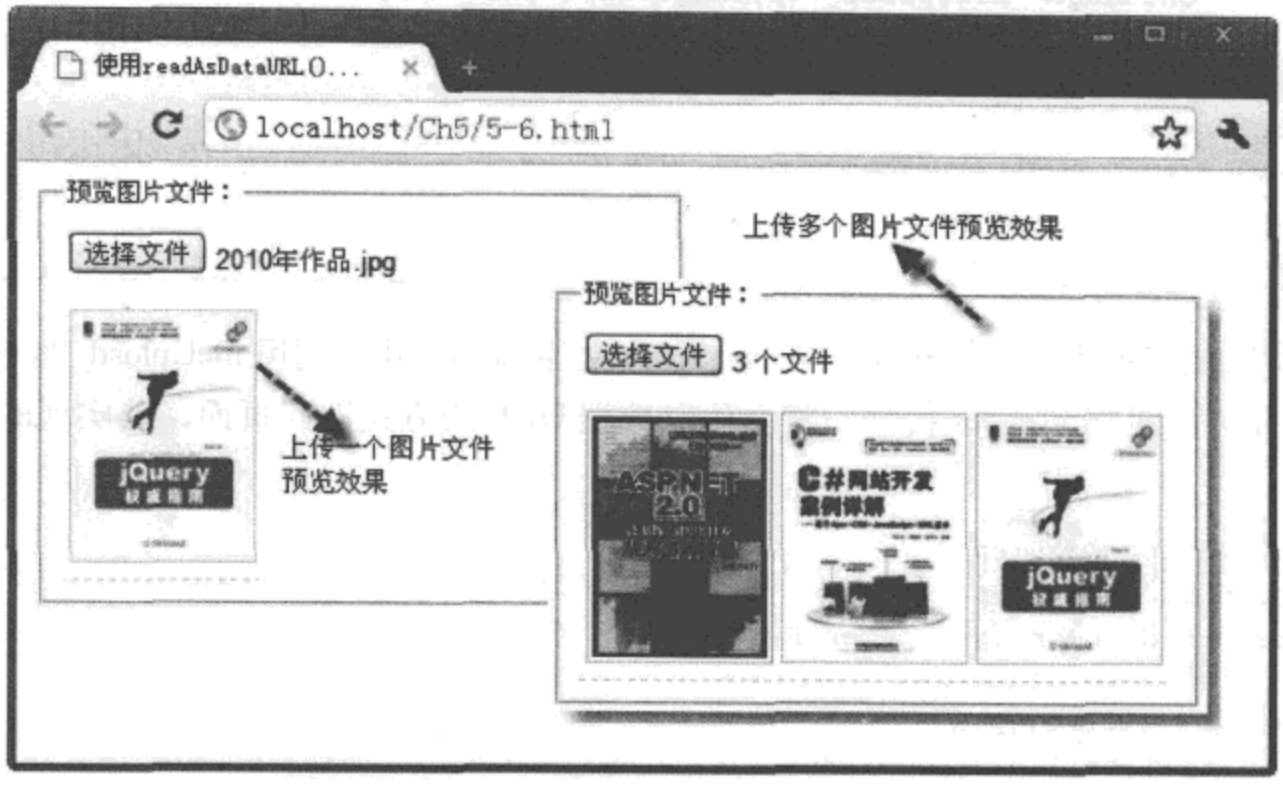


图 5-6　使用 readAsDataURL 方法预览图片

在 JavaScript 代码中，遍历传回的上传文件集合，获取每个“file”对象。由于每个文件返回的数据块都不同，因此，每次在读取文件前，必须先重构一个新的 FileReader 对象，然后，将每个文件以数据 URL 的方式读入页面中。当读取成功时，触发 onload 事件，在该事件中，通过“result”属性获取文件读入页面中的 URL 地址，并将该地址与 <img> 元素进行绑定，最后，通过列表 ID 号为“ulUpload”的列表元素展示在页面中，从而实现上传图片文件预览的效果，详细实现过程如 JavaScript 代码中加粗部分所示。

5.2.3　使用 readAsText 方法读取文本文件

使用 FileReader 接口中的 readAsText() 方法，可以将文件以文本编码的方式进行读取，即可以读取上传文本文件的内容；其实现的方法与读取图片基本相似，只是读取文件的方式不一样。

下面通过实例 5-7 介绍使用 readAsText () 方法读取文本文件内容的过程。

实例 5-7　使用 readAsText 方法读取文本文件

1. 功能描述

在页面表单中，新建一个“file”类型的 <input> 元素，用于获取上传的文本文件。当单击“选择文件”按钮，选中一个文本文件后，在页面中将显示该文本文件的内容。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-7.html，加入代码如代码清单 5-7 所示。

代码清单 5-7　使用 readAsText 方法读取文本文件内容

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 fileReader 方法读取文本文件 </title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
          src="Js/js7.js"/>
</script>
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend> 读取文本文件：</legend>
   <input type="file" name="fleUpload" id="fleUpload"
         onChange="fileUpload_ReadTxtFile(this.files);"/>
   <article id="artShow"></article>
 </fieldset>
</form>
</body>
</html>
```

在代码清单 5-7 中，页面导入一个 JavaScript 文件 js7.js，其中调用 fileUpload_ReadTxtFile() 方法将文件以文本编码方式读取并返回页面，其实现的代码如下所示：

```
 // JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 选择上传文件时调用的函数
function fileUpload_ReadTxtFile(f) {
   // 检测浏览器是否支持 FileReader 对象
    if (typeof FileReader == 'undefined') {
        alert(" 检测到您的浏览器不支持 FileReader 对象！ ");
    }
    var tmpFile = f[0];
    var reader = new FileReader();
    reader.readAsText(tmpFile);
    reader.onload = function(e) {
        $$("artShow").innerHTML = "<pre>" +
        e.target.result + "</pre>";
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-7 所示。

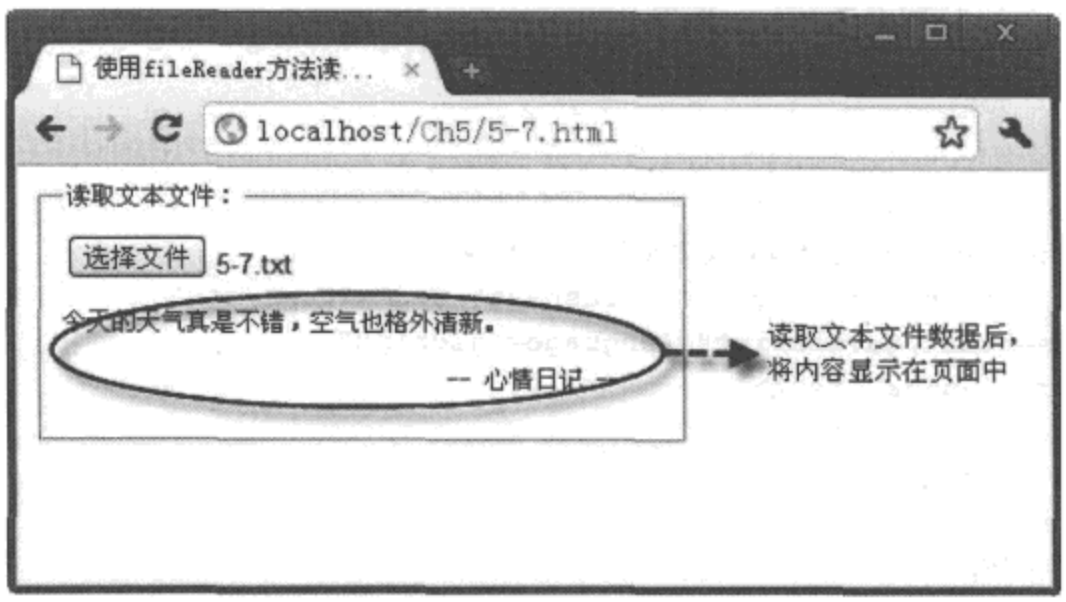


图 5-7　使用 readAsText 方法读取文本文件时的效果

4. 源码分析

在本实例中，由于“file”类型的 <input> 文件上传元素没有添加“multiple”属性，因此，单击“选择文件”按钮后，将返回单个“file”文件。

与实例 5-6 一样，在 JavaScript 代码中，首先检测浏览器是否支持 FileReader 对象。如果支持，则重构一个新的 FileReader 对象，调用该对象的 readAsText() 方法，将文件以文本编码的方式读入页面中；然后，通过“result”属性获取读入的内容，并将该内容赋值给 ID 号为“artShow”的元素；最后，通过该元素将文本文件的内容显示在页面。

5.2.4　侦听 FileReader 接口中的事件

在 FileReader 接口中，提供了很多常用的事件，以及一套完整的事件处理机制。通过这些事件的触发，可以清晰地侦听 FileReader 对象读取文件的详细过程，以便更加精确地定位每次读取文件时的事件先后顺序，为编写事件代码提供有力的支持。FileReader 接口的常用事件如表 5-2 所示。

表 5-2　FileReader 接口的常用事件

事件名称	描　述
onloadstart	当读取数据开始时，触发该事件
onprogress	当正在读取数据时，触发该事件
onabort	当读取数据中止时，触发该事件
onerror	当读取数据失败时，触发该事件
onload	当读取数据成功时，触发该事件
onloadend	当请求操作成功时，无论操作是否成功，都将触发该事件

经过反复测试证明，一个文件通过 FileReader 接口中的方法正常读取时，触发事件的先后顺序如图 5-8 所示。

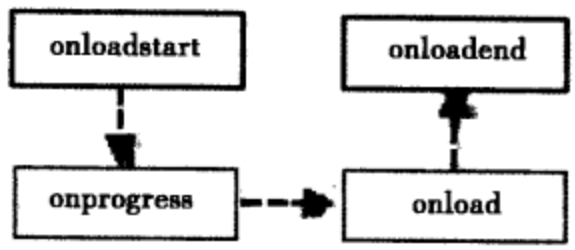


图 5-8　调用 FileReader 接口中的方法正常读取文件时事件的先后顺序

针对图 5-8 的说明如下：

- 大部分的文件读取过程都集中在 onprogress 事件中，该事件耗时最长。
- 如果文件在读取过程中出现异常或中止，那么 onprogress 事件将结束，直接触发 onerror 或 onabort 事件，而不会触发 onload 事件。
- onload 事件是文件读取成功时触发，而 onloadend 虽然也是文件操作成功时触发，但该事件不论文件读取是否成功，都将触发。因此，想要正确获取文件数据，必须在 onload 事件中编写代码。

下面通过实例 5-8 介绍文件读取时触发事件的先后顺序。

实例 5-8　展示文件读取时触发事件的先后顺序

1. 功能描述

在页面的表单中，添加一个“file”类型的 <input> 元素，当用户单击“选择文件”按钮，并通过打开的窗口选取一个文件后，页面中将展示读取文件过程中所触发事件的内容。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-8.html，加入代码如代码清单 5-8 所示。

代码清单 5-8　展示文件读取时触发事件的先后顺序

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>展示文件读取时触发事件的先后顺序</title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
          src="Js/js8.js"/>
</script>
</head>
<body>
<form id="frmTmp">
 <filedset>
```

```
    <legend>文件读取事件的顺序：</legend>
    <input type="file" name="fleUpload" id="fleUpload"
           onChange="fileUpload_ShowEvent(this.files);"/>
            <p id="pStatus"></p>
 </fieldset>
</form>
</body>
</html>
```

在代码清单 5-8 中，页面导入一个 JavaScript 文件 js8.js，其中调用 fileUpload_ShowEvent() 方法列出文件在正常读取过程中的全部事件，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_ShowEvent(f) {
    if (typeof FileReader == 'undefined') {
        alert("检测到您的浏览器不支持 FileReader 对象！ ");
    }
    var tmpFile = f[0];
    var reader = new FileReader();
    reader.readAsText(tmpFile);
    reader.onload = function(e) {
        $$("pStatus").style.display="block";
        $$("pStatus").innerHTML = "数据读取成功！";
    }
    reader.onloadstart = function(e) {
        $$("pStatus").style.display="block";
        $$("pStatus").innerHTML = "开始读取数据 ...";
    }
    reader.onloadend = function(e) {
        $$("pStatus").style.display="block";
        $$("pStatus").innerHTML = "文件读取成功！";
    }
    reader.onprogress = function(e) {
        $$("pStatus").style.display="block";
        $$("pStatus").innerHTML = "正在读取数据 ...";
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-9 所示。

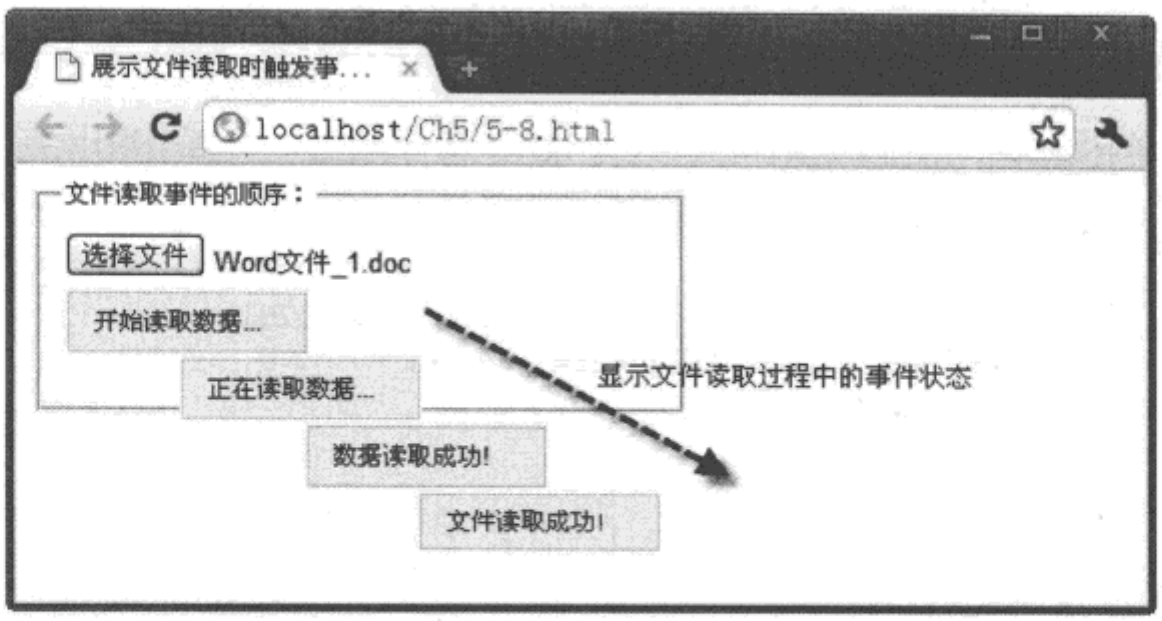


图 5-9　文件读取过程中显示各事件执行的先后顺序

4. 源码分析

在本实例的页面中，单击“选择文件”按钮后，触发一个自定义的函数 fileUpload_ShowEvent()。在该函数中，首先检测浏览器是否支持 FileReader 对象，如果不支持，则弹出错误提示信息；然后，重新构造一个新的 FileReader 对象，并对传回的文件以文本编码的方式读入页面；最后，列出文件在正常读取过程中将触发的 4 个事件。在每个事件中，首先显示 ID 号为“pStatus”元素，然后将事件的状态内容设置为该元素的文本内容；当 FileReader 对象执行 readAsText() 方法读取文件时，各个不同事件将按执行顺序被触发，设置的状态内容以动态的方式显示在 ID 号为“pStatus”的页面元素中。

5.3　使用 DataTransfer 对象拖放上传图片文件

在 HTML 5 中，借助于 DataTransfer 对象中提供的方法，可以实现浏览器与其他应用程序之间文件的拖动。虽然 HTML 4 之前的版本也支持拖放数据的操作，但该操作仅局限在整个浏览器中，不支持浏览器之外的数据。

下面通过实例 5-9 介绍使用 DataTransfer 对象拖放上传图片文件的过程。

实例 5-9　使用 DataTransfer 对象拖放上传图片文件

1. 功能描述

在页面的表单中，创建一个 <ul> 元素，用于接收并预览拖入的图片文件。当用户从电脑的文件夹中选择图片文件后，以拖动的方式将文件放入该元素内，并以预览的方式显示。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 5-9.html，加入代码如代码清单 5-9 所示。

代码清单 5-9　使用 DataTransfer 对象拖放上传图片文件

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>拖放选择上传文件</title>
<link href="Css/css5.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js9.js"/>
</script>
</head>
<body>
<form id="frmTmp">
 <fieldset>
   <legend>拖动选择文件：</legend>
   <ul id="ulUpload" ondrop="dropFile (event)"
       ondragenter="return false"
       ondragover="return false">
   </ul>
 </fieldset>
</form>
</body>
</html>
```

在代码清单 5-9 中，页面导入一个 JavaScript 文件 js9.js，其中调用 fileUpload_MoveFile() 方法，将拖动的文件数据放入 DataTransfer 对象，然后调取。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_MoveFile(f) {
    //检测浏览器是否支持 FileReader 对象
    if (typeof FileReader == 'undefined') {
        alert("检测到您的浏览器不支持 FileReader 对象！ ");
    }
    for (var intI = 0; intI < f.length; intI++) {
        var tmpFile = f[intI];
        var reader = new FileReader();
        reader.readAsDataURL(tmpFile);
        reader.onload = (function(f1) {
            return function(e) {
                var eleSpan = document.createElement('span');
                eleSpan.innerHTML = ['<img src="',
                      e.target.result,
                      '" title="',
```

```
                        f1.name, '"/>'].join('');
                    $$('ulUpload').insertBefore(eleSpan, null);
                }
            })(tmpFile);
        }
}
function dropFile(e) {
    // 调用预览上传文件的方式
     fileUpload_MoveFile(e.dataTransfer.files);
    // 停止事件的传播
     e.stopPropagation();
    // 阻止默认事件的发生
     e.preventDefault();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 5-10 所示。

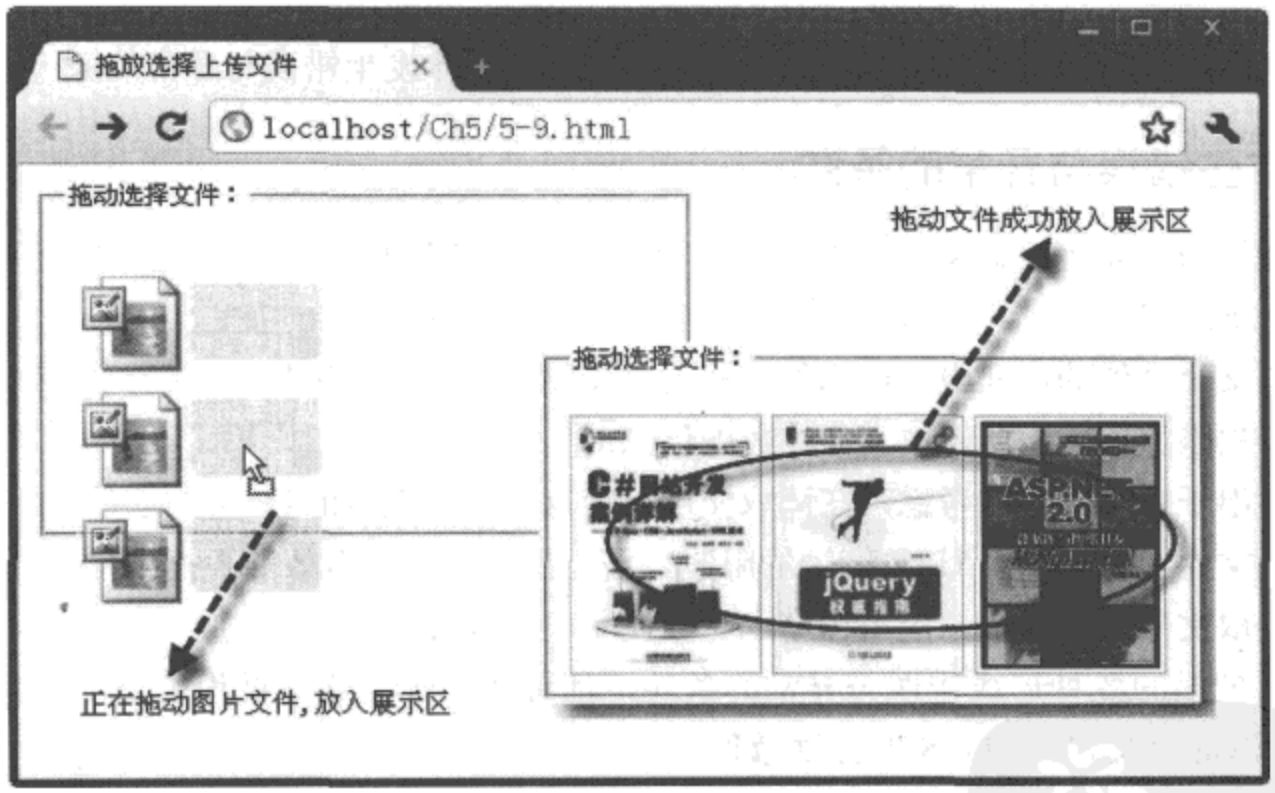


图 5-10　拖放选择上传图片文件

4. 源码分析

在本实例中，把图像文件从文件夹拖入页面目标元素的过程中，通过 DataTransfer 对象中的 setData() 方法保存数据。为了接收被保存的数据，页面中的目标元素在调用元素的拖放事件 ondrop 中，调用了一个自定义的函数 dropFile ()；同时，为了确保目标元素顺利接收拖放文件，必须将目标元素的 ondragenter 与 ondragover 事件都返回“return false”，实现方法如 HTML 源码中加粗部分所示。

在自定义的函数 dropFile () 中，先调用另一个自定义的函数 fileUpload_MoveFile()；同时，

要实现文件的拖放过程，还要在目标元素的拖放事件中，停止其他事件的传播并且关闭默认事件。其实现的过程如 JavaScript 代码中自定义的函数 dropFile () 所示。

在自定义的函数 fileUpload_MoveFile() 中，先从 DataTransfer 对象中获取被保存的文件集合，然后，遍历整个集合中的文件成员，获取每一个单独的文件，通过重构一个 FileReader 对象，调用该对象中的 readAsDataURL()，将文件以数据地址的形式读入页面中；同时，创建页面元素 ，将数据地址与 <img> 元素绑定，通过 join() 方法，写入 元素的内容中；最后，将全部获取的内容写入 ID 号为“ulUpload”的列表元素中，通过该元素展示在页面中。其实现的过程如 JavaScript 代码中自定义的函数 fileUpload_MoveFile () 所示。

有关 DataTransfer 对象更多的属性与方法，将在本书的第 11 章进行详细的介绍。

5.4 文件读取时的错误与异常

虽然使用 FileReader 接口中的方法可以快速实现各类文件的读取，但在文件读取的过程中，不可避免地会出现各种类型的错误与异常；而通过 FileError 对象可以获取这些错误与异常所产生的错误代码，根据返回的错误代码，可以分析具体发生错误与异常的原因。

5.4.1 发生错误与异常的条件

在使用 FileReader 接口中的 API 异步访问文件数据的过程中，出现下列情况时，可能会出现潜在的错误与异常。

- 访问某个文件的过程中，该文件被移动或删除及被其他应用程序并发式修改。
- 由于权限原因，无法读取文件的数据信息。
- 文件出于安全因素的考虑，在读取文件时，返回一个无效的数据信息。
- 读取文件太大，超出 URL 网址的限制，将无法返回一个有效的数据结构信息。
- 在读取文件的过程中，应用程序本身触发了中止读取文件 abort() 事件。

以上所列举的各种形成错误与异常的条件，都可能在读取文件过程中出现，从而导致无法使用 FileReader 接口中的对象与方法读取文件数据。

在使用 API 异步读取文件的过程中，一旦出现错误与异常，无法成功返回文件数据，可以使用 FileError 接口。该接口主要用于异步报告错误，当 FileReader 对象无法返回数据时，将形成一个错误属性，而该属性则是一个 FileError 对象，通过该对象列出错误与异常的错误代码信息。

下面通过一个实例说明上传图片文件时，为了预览图片效果自定义函数 fileUpload_TmpPrevImageFile()，该函数如何在读取文件过程中捕获出现的错误与异常。

在 JavaScript 代码中加入如下代码：

```
// JavaScript Document
function $$(id) {
```

```
    return document.getElementById(id);
}
//选择上传文件时调用的函数
function fileUpload_TmpPrevImageFile(f) {
    //检测浏览器是否支持 FileReader 对象
    if (typeof FileReader == 'undefined') {
        alert("检测到您的浏览器不支持 FileReader 对象! ");
    }
    var strHTML = "";
    var tmpFile = f[0];
    var reader = new FileReader();
    reader.readAsDataURL(tmpFile);
    reader.onload = function(e) {
        strHTML = strHTML + "<span>";
        strHTML = strHTML + "<img src='" +
                  e.target.result + "' alt=''/>";
        strHTML = strHTML + "</span>";
        $$("ulUpload").innerHTML = "<li>" + strHTML + "</li>";
    }
    //出现错误与异常时，返回错误代码
    reader.onerror = function(e){
        alert("FileError: " + e.target.error.code);
    }
}
```

在上述 JavaScript 代码中，文件数据读取成功后，将触发 onload 事件，以数据 URL 的方式读取到页面中，并将该 URL 与 <img> 元素绑定，进行图片文件的预览功能；一旦在读取过程中发生了错误与异常，将触发 onerror 事件。在该事件中，通过 e.target.error.code 的方式返回出现异常的错误代码说明，实现过程如代码中加粗部分所示。

5.4.2　错误代码说明

在 HTML 5 中，可以调用 FileError 接口中的对象，捕获数据文件在读取过程中出现的错误代码信息，这些错误代码信息说明如表 5-3 所示。

表 5-3　FileError 对象获取的错误代码及说明

错误常数	说　明
NOT_FOUND_ERR	文件无法找到或原文件已被修改
SECURITY_ERR	出于安全的考虑，无法获取数据文件
ABORT_ERR	触发了 abort 事件，中止文件读取的过程
NOT_READABLE_ERR	由于权限原因，不能获取数据文件
ENCODING_ERR	读取的文件太大，超出读取时地址的限制

文件在读取过程中，如果出现错误与异常，可以在 onerror 事件中，通过 e.target.error.code 方式捕获异常的错误代码。获取的错误代码就是如表 5-3 所示的错误常数，用户可以根据返回错误代码的不同，分别进行后续代码的处理与修改。

5.5 本章小结

文件在 HTML 5 中占有很重要的地位，本章首先介绍选择单个与多个文件的实现方法；然后，结合实例详细阐述选择文件后，以各类方式读取数据的过程；同时，讲解了如何在文件读取过程中，捕获出现的错误与异常的方法。学习本章之后，可以为读者完整掌握 HTML 5 中文件的使用打下扎实的理论与实践的基础。

第6章

HTML 5中的视频和音频

本章内容

- 多媒体元素基本属性
- 多媒体元素常用方法
- 多媒体元素重要事件
- 本章小结

在 HTML 4 或之前版本中，如果要在 Web 页面中播放视频或音频文件，常用的方法是引用 flash 插件，通过 <object>、<embed> 元素设置一些低效、冗余的属性来实现，使用起来十分不方便，插件的引用还存在页面安全的隐患。

为了解决这一问题，HTML 5 中新增了 <video> 与 <audio> 两个元素和对应的 API 文件，分别用于在页面中播放视频与音频文件。虽然这两个新增的元素播放文件不一样，但是它们有很多相似的功能，因此，本章将这两个新增的元素统称为“多媒体元素”同时介绍。

6.1 多媒体元素基本属性

在 HTML 5 中播放视频与音频文件十分简单，只需要添加 <video> 或 <audio> 元素，并简单设置元素的一些基本属性，就可以在页面中播放多媒体文件。接下来详细介绍这两个多媒体元素在页面中的使用格式与属性。

6.1.1 元素格式

<video> 是 HTML 5 中新增加的元素，用于电影文件、其他视频流的播放；<audio> 也是 HTML 5 中新增加的元素，用于音乐文件、其他音频流的播放。在两个多媒体元素的开始标记与结束标记间放置文本内容，可以在不支持该元素的浏览器中使用。即如果浏览器不支持 HTML 5 中新增的这两个多媒体元素，那么将显示开始与结果标记之间的文本内容。

下面通过实例 6-1 介绍使用多媒体元素播放文件的过程。

实例 6-1 使用多媒体元素播放文件

1. 功能描述

在页面中，创建两个多媒体元素 <video> 与 <audio>，并在元素的“src”属性中，设置各自播放的视频与音频文件，页面加载完成后自动播放这两个文件。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-1.html，加入代码如代码清单 6-1 所示。

代码清单 6-1 使用多媒体元素播放文件

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用多媒体元素播放文件</title>
</head>
<body>
<video id="vdoMain" src="Video/6-test_1.mov"
       autoplay="true">
```

```
        你的浏览器不支持视频
</video>
<audio id="adoMain" src="Audio/6-test_4.mp3"
        autoplay="true" controls="true">
        你的浏览器不支持音频
</audio>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-1 所示。

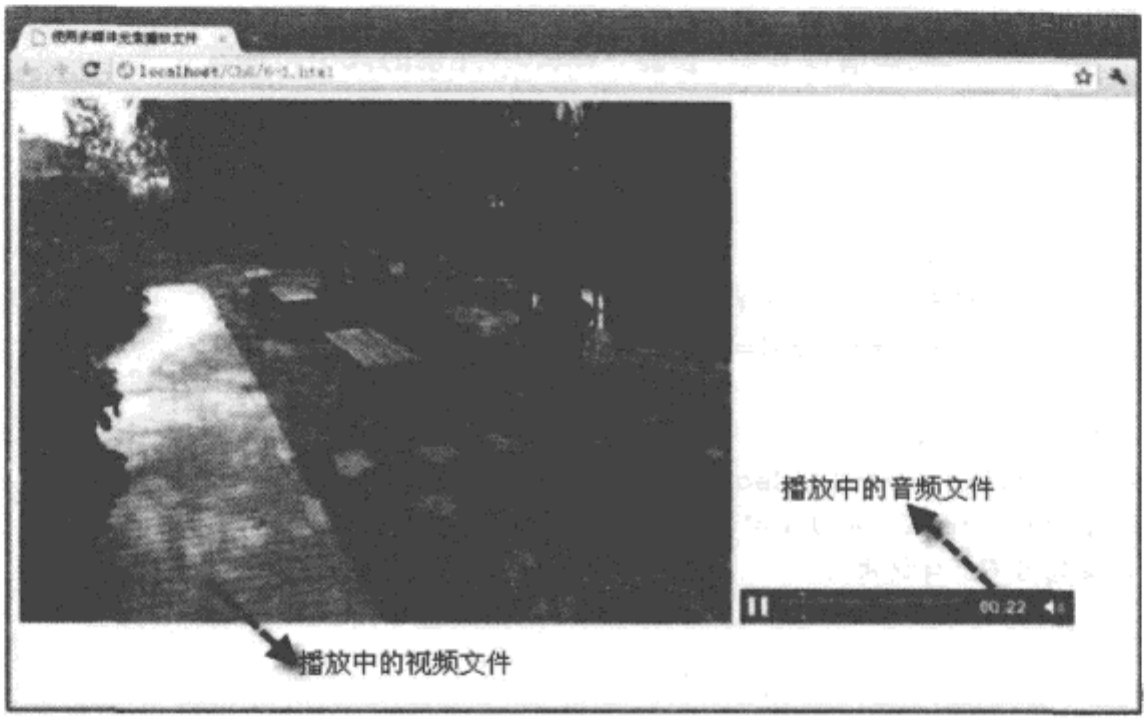


图 6-1　使用多媒体元素播放文件的效果

4. 源码分析

在本实例中，创建多媒体元素后，通过“src”属性指定播放文件的 URL。为了在页面加载完成后实现自动播放功能，需要添加一个“autoplay”属性，并将该属性的值设置为“true”；该属性的功能是告诉浏览器是否自动播放视频或音频文件，默认值为“false”，表示不自动播放。注意，在 <audio> 音频元素中，为了能正常播放音频文件，必须将“controls”属性设置为“true”，表示显示音频元素自带的控制条工具。

6.1.2　width 与 height 属性

“width”与“height”属性只适用于 <video> 元素，表示设置媒体元素的大小，单位为像素。如果不设置该属性，则使用播放源文件的大小；如果播放源文件的大小不可用，则使用“poster”属性中的文件大小。如果仅设置一个宽度值，那么，将根据播放源文件的长宽比例，自动生成一个与之对应的高度值，以等比例的方式控制视频文件的大小。

下面通过实例 6-2 介绍设置 <video> 元素的大小与样式的过程。

实例 6-2　设置 <video> 元素的大小与样式

1. 功能描述

在页面中，创建一个媒体元素 <video>，先在元素的“src”属性中设置需要播放的视频文件，然后，分别设置 <video> 元素的长度与宽度，并导入一个样式文件 css6 控制媒体元素的样式。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-2.html，加入代码如代码清单 6-2 所示。

代码清单 6-2　设置 <video> 元素的大小与样式

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>设置 <video> 元素的大小与样式 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
</head>
<body>
<video id="vdoMain" src="Video/6-test_1.mov"
       autoplay="true" width="360px" height="220px">
       你的浏览器不支持视频
</video>
</body>
</html>
```

在代码清单 6-2 中，页面导入一个样式文件 css6.css，其中设置了 <video> 元素的部分样式。其实现的代码如下所示：

```
@charset "utf-8";
/* CSS Document */
body {
   font-size:12px
}
video{
   border:solid #ccc 5px;
   padding:3px;
   background-color:#eee
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-2 所示。

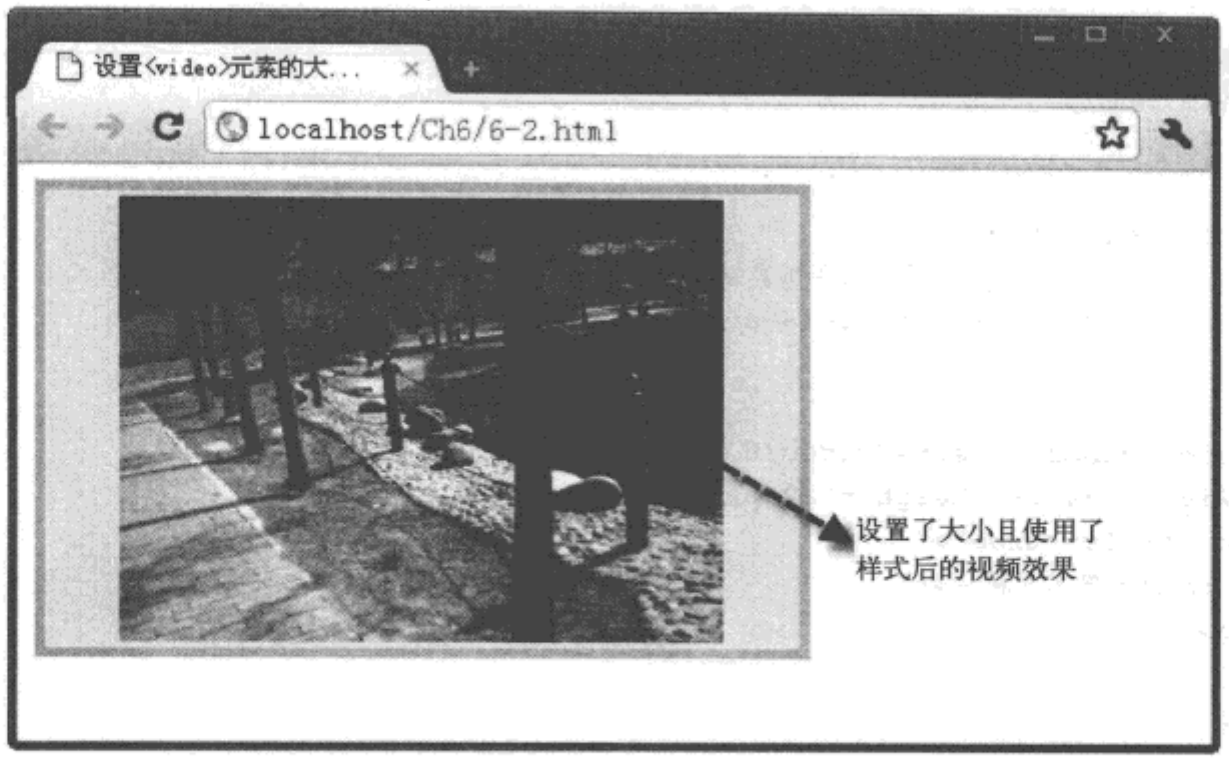


图 6-2　设置多媒体元素的大小与样式后的效果

4. 源码分析

在本实例中，被播放的媒体源文件根据 <video> 元素所设置的长宽尺寸，以等比例的方式进行缩放，并在页面中播放；同时，使用样式文件 css6.css 美化了 <video> 标记的边框与背景色，并设置了媒体文件播放时各个方位的边距长度。

6.1.3　controls 属性

在实例 6-1 中，我们为 <audio> 音频元素添加了一个“controls”属性，同样，<video> 视频元素也拥有“controls”属性，如果将该属性的值设置为“true”或“controls”，将在视频元素的底部展示一个元素自带的控制条工具。虽然由于浏览器不同，控制条工具的样式会有些变化，但所有控制条都有一个播放 / 暂停控件、一个进度条和音量开关，这些特征是不变的。

下面通过实例 6-3 介绍设置 <video> 元素控制条工具属性的过程。

实例 6-3　设置 <video> 元素的控制条工具属性

1. 功能描述

在实例 6-2 的基础上，取消“autoplay”属性，同时，为 <video> 元素新增一个“controls”属性，并将该属性的值设置为“true”或“controls”。移动鼠标至播放视频时，将在视频底部出现一个自带的控制条工具。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-3.html，加入代码如代码清单 6-3 所示。

代码清单 6-3　设置 <video> 元素的控制条工具属性

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>设置 <video> 元素的控制条工具属性 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
</head>
<body>
<video id="vdoMain" src="Video/6-test_1.mov"
       width="360px" height="220px"
       controls="true">
       你的浏览器不支持视频
</video>
</body>
</html>
```

3. 页面效果

该页面在不同浏览器中执行的页面效果如图 6-3 所示。

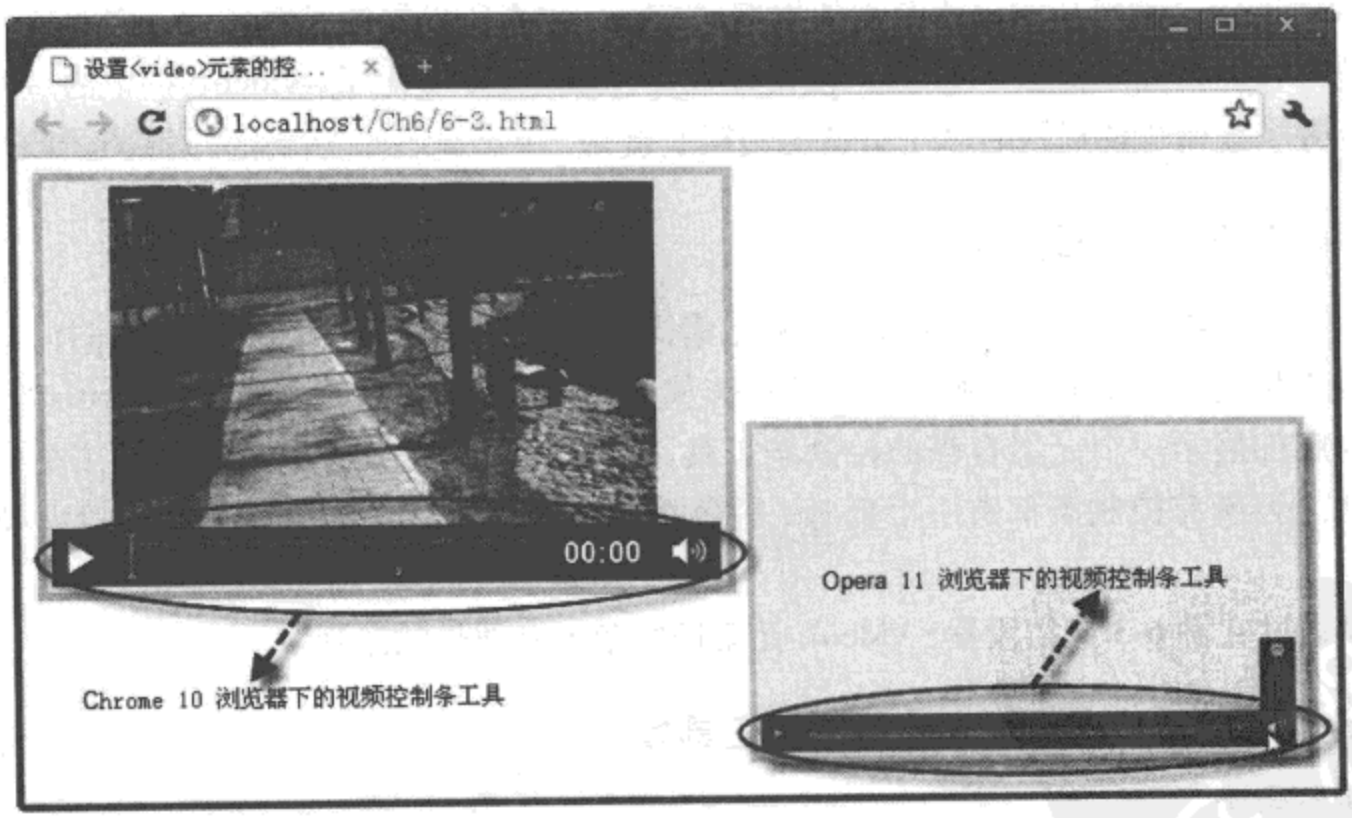


图 6-3　不同浏览器展示多媒体控制条工具的效果

4. 源码分析

在本实例中，只有当用户将鼠标悬停或 Tab 键聚焦至播放的视频上时，所设置的控制条工具才能显示出来；一旦从视频上移开，控制条工具又隐藏起来。

在 <audio> 元素中，“controls”属性十分重要，如果没有设置该属性，页面中将不会显示任何效果，但该元素却存在于 DOM 中，通过 JavaScript 代码可以访问相关的元素。

6.1.4　poster 属性

在 HTML 5 的 <video> 元素中，“poster”属性表示所选图片 URL，如果添加该属性，将在视频文件播放前显示该图片，而不是默认显示视频文件的第一帧。另外，添加该属性，还可以避免在播放的视频文件不可用时，出现一片空白区域，影响用户体验。

下面通过实例 6-4 介绍增加 <video> 元素的“poster”属性使用过程。

实例 6-4　设置 <video> 元素的 poster 属性

1. 功能描述

在实例 6-3 的基础上，为 <video> 元素新增一个“poster”属性，并选取一幅图片作为该属性的值，当播放视频文件时，在视频播放区域中，首先将显示“poster”属性指定的图片。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-4.html，加入代码如代码清单 6-4 所示。

代码清单 6-4　设置 <video> 元素的 poster 属性

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>设置 <video> 元素的 poster 属性 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
</head>
<body>
<video  id="vdoMain" src="Video/6-test_1.mov"
        width="360px" height="220px"
        controls="true" poster="Images/2010 年作品 .jpg">
        你的浏览器不支持视频
</video>
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-4 所示。

4. 源码分析

在本实例中，设置了 <video> 媒体元素的“poster”属性，该属性是视频元素 <video> 所独有的属性。利用该属性不仅可以在视频文件开始播放前设置图片，还可以通过视频元素的事件机制，指定在某事件中改变该属性的图片 URL。例如，当用户单击“暂停”或播放完成时，在相应的事件中编写 JavaScript 代码，通过 setAttribute() 方法重置“poster”属性中图片

的 URL，可以根据不同事件，动态变换图片的效果。具体实例将在 6.3.3 节中介绍。

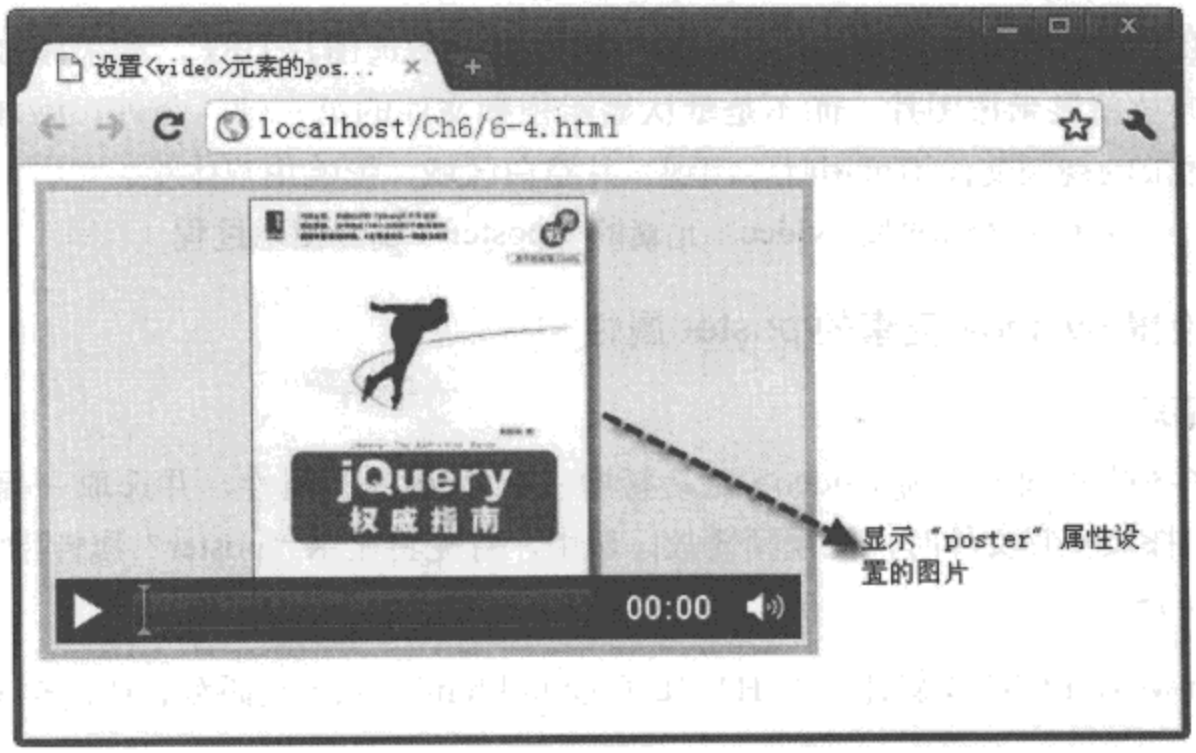


图 6-4　设置多媒体元素的 poster 属性后的效果

6.1.5　networkState 属性

多媒体元素 <video> 的“networkState”属性可以返回视频文件的网络状态。当浏览器读取视频文件时，将触发一个“progress”事件，通过该事件，可以获取视频文件在被打开过程中，各个不同阶段的网络状态值。“networkState”为只读属性，该属性对应 4 个返回值，如表 6-1 所示。

表 6-1　networkState 属性返回值

字符常量	返回值	描　述
NETWORK_EMPTY	0	数据加载初始化
NETWORK_IDLE	1	文件加载成功，等待请求播放
NETWORK_LOADING	2	文件正在加载过程中
NETWORK_NO_SOURCE	3	加载出错，一般原因是没有找到支持的编码格式

下面通过实例 6-5 介绍获取 <video> 元素“networkState”属性返回值的过程。

实例 6-5　获取 <video> 元素 networkState 属性的返回值

1. 功能描述

在页面中，添加一个多媒体元素 <video>，同时，新增一个 元素；当使用 <video> 元素加载视频文件时，在触发的“progress”事件中，通过 元素显示文件在加载过程中返回的“networkState”属性值。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-5.html，加入代码如代码清单 6-5 所示。

代码清单 6-5　获取 <video> 元素 networkState 属性的返回值

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title><video> 元素的 networkState 属性 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js5.js"/>
</script>
</head>
<body>
<div>
  <video  id="vdoMain" src="Video/6-test_1.mov"
          width="360px" height="220px"
          onProgress="Video_Progress(this)"
          controls="true" poster="Images/2010 年作品 .jpg">
          你的浏览器不支持视频
  </video>
  <span id="spnStatus"></span>
<div>
</body>
</html>
```

在代码清单 6-5 中，页面导入一个 JavaScript 文件 js5.js，其中自定义了一个函数 Video_Progress()，用于媒体元素在“progress”事件中调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function Video_Progress(e) {
    var intState = e.networkState;
    $$("spnStatus").style.display = "block";
    $$("spnStatus").innerHTML = StrByNum(intState)
    if (intState == 1) {
       $$("spnStatus").style.display = "none";
    }
}
function StrByNum(n) {
    switch (n) {
    case 0:
        return " 正在初始化 ...";
```

```
        case 1:
            return "数据加载完成！";
        case 2:
            return "正在加载中 ...";
        case 3:
            return "数据加载失败！";
        }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-5 所示。

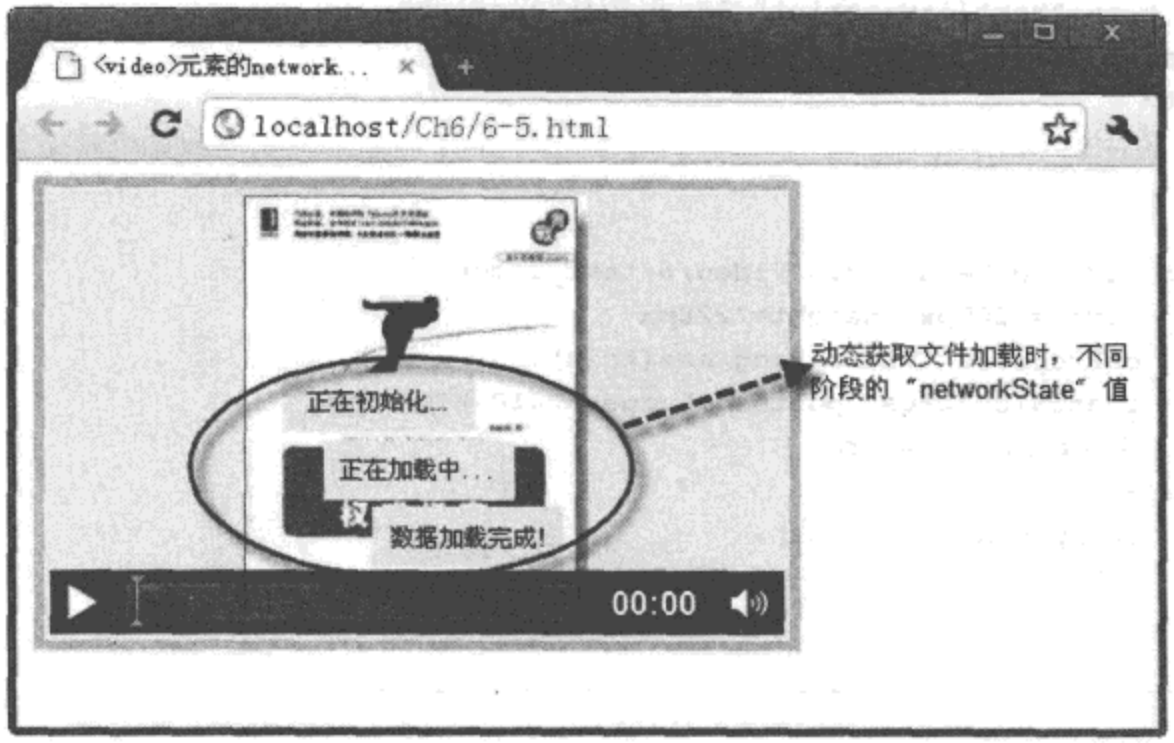


图 6-5　获取多媒体元素 networkState 属性值

4. 源码分析

在本实例中，媒体元素 <video> 在触发加载视频文件事件“progress”时，调用一个自定义的函数 Video_Progress()。在该函数中，首先，将 <video> 元素的“networkState”属性值保存至变量“intState”中，并将显示状态信息 元素的可见样式设置为“block”，表示可见；然后，调用另一个自定义的函数 StrByNum()，将保存至变量“intState”中的“networkState”属性值转成相应的文字说明信息，并赋值给显示状态信息元素 ，用于页面中的动态显示；最后，当返回的“networkState”属性值为“1”时，表示数据加载完成，再将显示状态信息 元素的可见样式设置为“none”，即隐藏该元素，实现的关键过程如 JavaScript 代码中加粗部分所示。

6.1.6　error 属性

“error”属性是一个只读属性，在使用多媒体元素加载或读取文件过程中，如果出现异常

或错误，将触发元素的“error”事件。在该事件中，可以通过元素的“error”属性返回一个 MediaError 对象，根据该对象的“code”返回当前的错误值。MediaError 对象中“code”对应 4 个返回值，如表 6-2 所示。

表 6-2　error 属性返回值

字符常量	返回值	描　述
MEDIA_ERR_ABORTED	1	媒体资源文件获取过程中出现异常而被中止
MEDIA_ERR_NETWORK	2	出现网络错误，获取媒体资源出错
MEDIA_ERR_DECODE	3	媒体资源可用，解码出错
MEDIA_ERR_SRC_NOT_SUPPORTED	4	没有找到可以播放的媒体文件格式

下面通过实例 6-6 介绍获取 <video> 元素“error”属性返回值的过程。

实例 6-6　获取 <video> 元素 error 属性的返回值

1. 功能描述

在页面中，添加一个多媒体元素 <video>，同时，新增一个 元素。当使用 <video> 元素加载一个不支持的播放格式文件时，在触发的“error”事件中，通过 元素显示加载出错后“error”属性返回的错误代码信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-6.html，加入代码如代码清单 6-6 所示。

代码清单 6-6　获取 <video> 元素 error 属性的返回值

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title><video> 元素的 error 属性 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js6.js"/>
</script>
</head>
<body>
<div>
  <video  id="vdoMain" src="Video/6-test_0.MM"
         width="360px" height="220px"
         onError="Video_Error(this)"
         controls="true" poster="Images/2010 年作品 .jpg">
         你的浏览器不支持视频
  </video>
  <span id="spnStatus"></span>
```

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

在代码清单 6-6 中，页面导入一个 JavaScript 文件 js6.js，其中自定义了一个函数 Video_Error()，用于媒体元素在“error”事件中调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function Video_Error(e) {
    var intState = e.error.code;
    $$("spnStatus").style.display = "block";
    $$("spnStatus").innerHTML = ErrorByNum(intState);
}
function ErrorByNum(n) {
    switch (n) {
    case 1:
        return "加载异常，用户请求中止！";
    case 2:
        return "加载中止，网络错误！ ";
    case 3:
        return "加载完成，解码出错 ";
    case 4:
        return "没有找到支持的播放格式！";
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-6 所示。

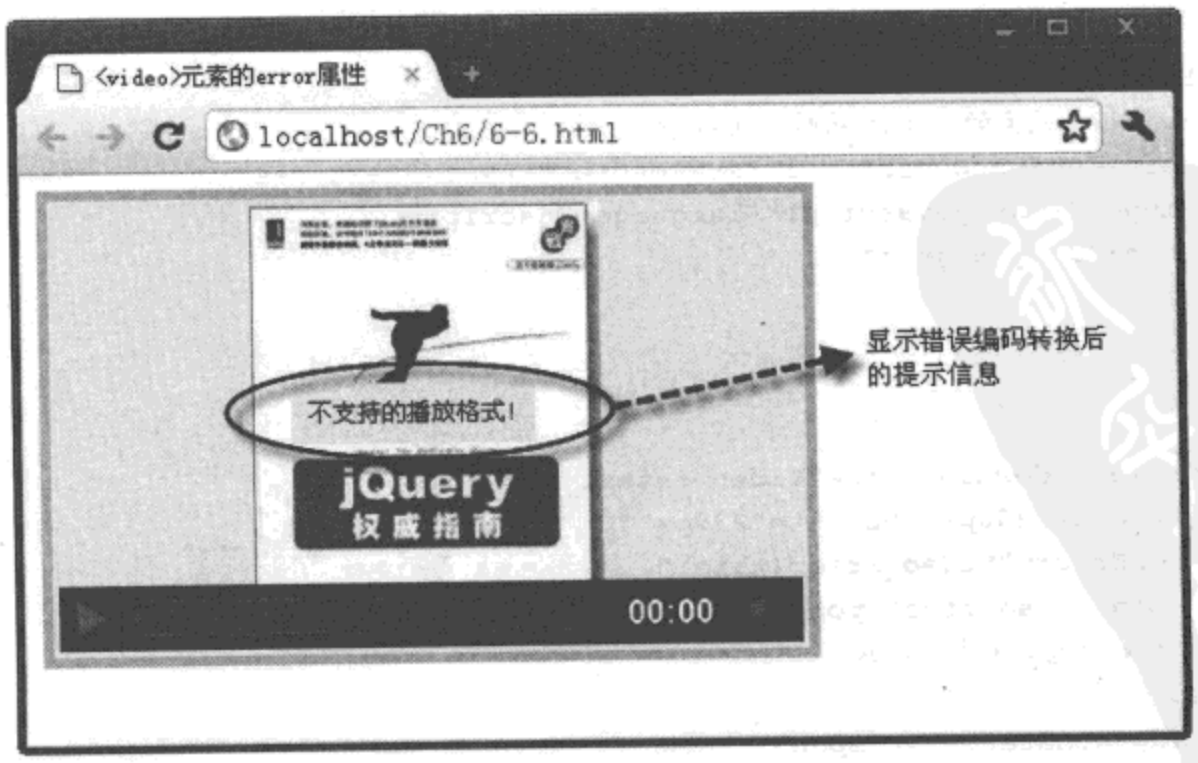


图 6-6　获取多媒体元素 error 属性值

4. 源码分析

在本实例中，由于视频元素 <video> 不支持载入文件“6-test_0.MM”的播放格式，因此，触发了“error”事件。在该事件中将调用函数 Video_Error()，在该函数中，首先，通过变量“intState”保存 MediaError 对象“code”返回的错误代码值；然后，将该值通过另一个函数 ErrorByNum() 返回对应的文字说明信息；最后，将获取的说明信息显示在页面元素 中。实现的关键过程如 JavaScript 代码中加粗部分所示。

6.1.7　其他属性

除以上介绍的多媒体元素属性外，还有许多其他属性可以在播放多媒体文件时使用，例如，返回当前播放媒体状态的属性“readyState”、播放过程中返回的当前时间“currenTime”、播放速率属性“playbackReat”、播放音量属性“volume”，下面分别进行详细介绍。

1. readyState 属性

多媒体元素可以通过“readyState”属性返回当前播放文件的各种状态，根据这些状态，可以反映媒体文件在播放过程中是否正常。该属性为只读属性，共返回 5 种可能出现的状态值，如表 6-3 所示。

表 6-3　readyState 属性返回值

字符常量	返回值	描　述
HAVE_NOTHING	0	表示当前没有可播放的媒体文件
HAVE_METADATA	1	表示正在加载中，还不具备开始播放的条件
HAVE_CURRENT_DATA	2	已加载完成部分数据，还不具备完整播放的条件
HAVE_FUTURE_DATA	3	数据加载全部完成，可以进行正常播放
HAVE_ENOUGH_DATA	4	数据加载全部完成，并且能以一种较流畅的速率进行播放

可以通过下列代码获取页面中多媒体元素的“readyState”属性值。

页面代码为：

```
<video id="vdoMain" src="Video/6-test_1.mov"
       onLoadStart="load_start(this);" >
       你的浏览器不支持视频与音频
</video>
```

JavaScript 代码为：

```
function load_start(e) {
   alert(e.readyState);
}
```

由于在开始加载媒体文件“LoadStart”事件中，还尚未形成加载文件的过程，因此，该事件将返回一个“0”状态值。

2. currentTime、startTime、duration 属性

“currentTime”属性可以返回媒体文件当前播放时间，也可以修改该时间属性。如果修改

成功，当前播放位置就指向所修改时间，该属性为可读写属性。

“startTime”属性可以返回多媒体元素开始播放的时间。默认情况下，该时间值为 0。该属性为只读属性，不可进行修改。

“duration”属性可以返回多媒体元素总体播放时间。在加载媒体文件过程中，该值将不断发生变化，如果加载完成，将返回播放整个文件所需的总时间。

上述多媒体元素的三个时间属性值单位均为秒，在使用过程中，通常需要转成分钟，以便于在页面中进行展示，实现代码如下所示。

页面代码为：

```
<video id="vdoMain" src="Video/6-test_1.mov"
       onTimeUpdate="progress(this)" >
       你的浏览器不支持视频与音频
</video>
```

JavaScript 代码为：

```
function progress(e){
   var strCurrTime=RuleTime(Math.floor(e.currentTime/60),2)
                   +":"+
                    RuleTime(Math.floor(e.currentTime%60),2);
   alert(strCurrTime);
}
// 转换时间显示格式
function RuleTime(num, n) {
    var len = num.toString().length;
    while(len < n) {
        num = "0" + num;
        len++;
    }
    return num;
}
```

在上述 JavaScript 代码中，先将获取的当前时间 e.currentTime 值除以 60 秒，获取分钟值，并以两位数的格式显示；然后，通过与当前时间 e.currentTime 求余数据的方式，获取剩余的秒数，并转换成两位数字的显示格式，链接在分钟的后面。

3. palyed、paused、ended 属性

通过多媒体元素中的“palyed”属性可以获取媒体文件已播放完成的时间段。“palyed”属性返回一个 TimeRanges 对象，通过该对象，可以获取已播放文件的开始时间与结束时间。

“paused”属性可以返回当前播放的文件是否处于暂停状态。该返回值是一个布尔值，如果为“true”，表示当前的播放文件处于暂停状态，否则为等待播放或正在播放状态。

“ended”属性可以返回当前播放的文件是否已结束。该返回值是一个布尔值，如果为“true”，表示当前播放的文件已结束，否则为还没有结束文件的播放。

以上三个属性均为只读属性，不可以进行修改，只能在各事件中获取，如以下代码所示。

页面代码为：

```
<video id="vdoMain" src="Video/6-test_1.mov"
       onPlaying="palying(this)">
       你的浏览器不支持视频与音频
</video>
```

JavaScript 代码为：

```
function palying(e){
   alert(e.ended);
}
```

4. preload、autoplay、loop 属性

“preload”属性表示在页面打开时，多媒体元素是否需要将所指定的媒体文件进行预加载，该属性有 3 个可选择值，如表 6-4 所示。

表 6-4　preload 属性值及描述

属 性 值	描　　述
none	表示不进行先期播放文件的加载
metadata	表示只加载播放文件基本信息（如总字节数、持续时间、第一帧信息等）
auto	表示需要将播放文件进行预加载，默认值为“auto”

“autoplay”属性可以使多媒体元素加载完成播放的文件后，实现自动播放的功能。打开页面后，无需单击“播放”按钮，自动开始播放。添加时，可以单独加入该属性名称“autoplay”或者将该属性的值设置为“true”均可。

“loop”属性表示媒体的文件在播放结束后，是否还要进行循环播放，一旦添加该属性名称或将该属性的值设置为“true”时，播放过程将不断进行重复，直到手动结束为止。

下面代码表示在页面中，添加一个可以自动预加载播放文件、自动播放、循环播放的视频元素，实现播放指定的媒体文件功能。

页面代码为：

```
<video id="vdoMain" src="Video/6-test_1.mov"
       preload="auto" autoplay="true" loop="true">
       你的浏览器不支持视频与音频
</video>
```

5. defaultPlaybackRate、playbackRate 属性

“defaultPlaybackRate”属性可以返回页面媒体元素默认的文件播放速度频率，即默认播放速率。一般情况下，该属性值为 1，也可以修改该属性值，从而改变其默认的播放速率值。

“playbackRate”属性返回当前正在播放的媒体文件的速度频率，即当前播放速率。也可以修改该属性值，如果修改成功，将可以实现“快进”或“慢进”的播放效果，默认值为 1。

需要说明的是，目前绝大多数的浏览器并不支持修改“defaultPlaybackRate”与“playbackRate”属性的功能，因此，目前只能获取该属性值。如下列代码，可获取“defaultPlaybackRate”属性的值。

页面代码为：

```
<video id="vdoMain" src="Video/6-test_1.mov"
       onPlaying="palying(this)">
       你的浏览器不支持视频与音频
</video>
```

JavaScript 代码为：

```
function palying(e){
   alert(" 当前播放速率为：  "+e.playbackRate);
}
```

6. volume、muted 属性

“volume”属性表示媒体元素播放时的音量。该属性的取值范围为（0 ～ 1），0 为最低音量，1 为最高音量，也可在取值范围内修改该属性值，从而调整当前播放媒体的音量大小。

“muted”属性是一个布尔值，表示是否设置为静音。如果为“true”，表示是静音，否则不是静音，默认值为“false”。

可以通过单击按钮调节媒体播放时的音量大小，单击复选框设置媒体播放时是否开启静音效果，代码如下所示。

页面代码为：

```
<video id="vdoMain" src="Video/6-test_1.mov">
       你的浏览器不支持视频与音频
</video>
<input name="btnVolume" id="btnVolume" type="button"
       value=" 调节音量 "  onClick="btnVolume_click();">
<input name="chkMuted" id="chkMuted" type="checkbox"
       onChange="chkMuted_change(this);"> 开启静音
```

JavaScript 代码为：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function btnVolume_click(){
   $$("vdoMain").volume=0;
}
function chkMuted_change(e){
   $$("vdoMain").muted=(e.checked)?true:false;
}
```

6.2　多媒体元素常用方法

多媒体元素可以通过添加“controls”属性显示控制条工具栏，单击工具栏中的按钮控制媒体文件的播放过程。除此之外，也可以自定义播放按钮，调用多媒体元素在播放文件时的方法，实现控制文件播放过程的功能。

6.2.1　媒体播放时的方法

当一个媒体文件需要通过多媒体元素播放时，可以调用元素 load 方法进行文件的加载。load 方法用于加载媒体文件，且自动将元素的“playbackRate”属性值设置为“defaultPlaybackRate”属性对应的值；同时，将元素的“error”属性设置为“null”值。

如果播放加载完成的媒体文件，还需要调用多媒体元素的“play”方法，同时将元素的“paused”属性设置为“false”值。

如果暂停正在播放中的媒体文件，还需要调用元素的“pause”方法，即将元素的“paused”属性设置为“true”值。

下面通过实例 6-7 介绍自定义 <video> 元素控制条工具栏的过程。

实例 6-7　自定义 <video> 元素控制条工具栏

1. 功能描述

在页面中，添加一个多媒体元素 <video>，用于播放指定的媒体文件，且不设置“controls”属性；同时，在多媒体元素的底部创建两个 元素，前者用于“加载”媒体文件，后者用于“播放”加载完成的媒体文件，或者“暂停”正在播放中的媒体文件。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-7.html，加入代码如代码清单 6-7 所示。

代码清单 6-7　自定义 <video> 元素控制条工具栏

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title> 自定义 <video> 元素控制条工具栏 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js7.js"/>
</script>
</head>
<body>
<div>
  <video id="vdoMain" src="Video/6-test_1.mov"
```

```
            width="360px" height="220px"
            poster="Images/2010 年作品 .jpg">
            你的浏览器不支持视频
        </video>
        <p id="pTool">
            <span onClick="v_load();">加载 </span>
            <span id="spnPlay" onClick="v_play(this);">播放 </span>
        </p>
    <div>
    </body>
    </html>
```

在代码清单 6-7 中，页面导入一个 JavaScript 文件 js7.js，其中自定义了两个函数 v_load() 与 v_play()，分别在单击两个 元素时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function v_load() {
    $$("spnPlay").innerHTML = "播放";
    $$("vdoMain").load();
}
function v_play(e) {
    if (e.innerHTML == "播放") {
        $$("vdoMain").play();
        e.innerHTML = "暂停";
    } else {
        $$("vdoMain").pause();
        e.innerHTML = "播放";
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-7 所示。

4. 源码分析

在本实例中，单击“加载”标记时，将触发自定义的函数 v_load()。在该函数中，先改变“播放”标记中显示的文字信息，因为一旦重新加载，需要再次单击“播放”标记；然后，调用多媒体元素的 load 方法，从而实现重新加载一次媒体文件的功能。

当媒体文件加载完成，单击“播放”标记时，将触发自定义的函数 v_play()。在该函数中，根据传回单击标记中的内容触发对应方法；如果传回标记内容是“播放”，那么执行元素的“play”方法，并将标记内容修改为“暂停”；当单击“暂停”标记时，由于传回标记内容是“播放”，因此，执行元素的“pause”方法，并将标记内容修改为“播放”，从而实现同一

个标记根据不同显示内容，完成“播放”或“暂停”媒体文件的功能，实现过程如 JavaScript 代码中加粗部分所示。

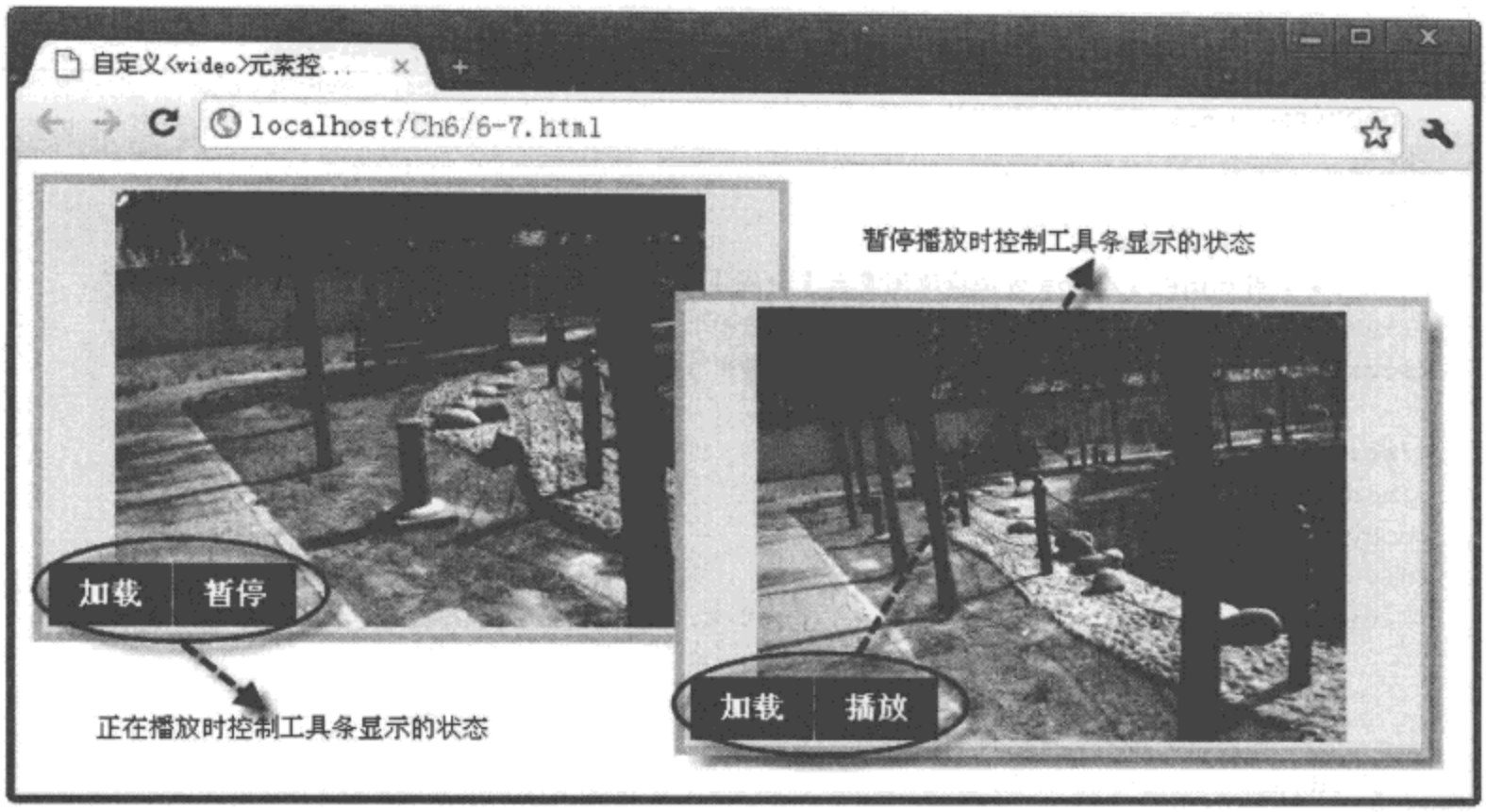


图 6-7　自定义 <video> 元素控制条工具栏

6.2.2　canPlayType 方法

由于浏览器对多媒体元素加载媒体文件的类型支持不同，因此，在使用多媒体元素加载文件前，需要检测当前浏览器是否支持媒体文件类型。检测的方法是通过调用多媒体元素的 canPlayType(type) 方法，其中“type”参数表示需要浏览器检测的类型，该类型与媒体文件的 MIME 类型一致；通过多媒体元素的 canPlayType(type) 方法，可以返回如下三个值：

- 空字符：表示浏览器不支持该类型的媒体文件。
- maybe：表示浏览器可能支持该类型的媒体文件。
- Probably：表示浏览器支持该类型的媒体文件。

下面通过实例 6-8 介绍使用 canPlayType 方法检测浏览器支持媒体类型的过程。

实例 6-8　使用 canPlayType 方法检测浏览器支持媒体类型

1. 功能描述

在页面中，添加一个多媒体元素 <video>，另外，在多媒体元素的底部创建一个 元素，用于检测浏览器是否支持各种媒体类型。单击 元素后，将检测后的结果显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-8.html，加入的代码如代码清单 6-8 所示。

代码清单 6-8　使用 canPlayType 方法检测浏览器支持媒体类型

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 canPlayType 方法检测浏览器支持媒体类型 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js8.js"/>
</script>
</head>
<body>
<div>
  <video id="vdoMain" src="Video/6-test_1.mov"
         width="360px" height="220px"
         poster="Images/2010 年作品 .jpg">
         你的浏览器不支持视频
  </video>
  <p id="pTool">
     <span onClick="v_chkType();">检测 </span>
  </p>
  <span id="spnResult"></span>
<div>
</body>
</html>
```

在代码清单 6-8 中，页面导入一个 JavaScript 文件 js8.js，其中自定义了一个函数 v_chkType()，在单击内容为“检测”的 元素时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var i = 0,j = 0,k = 0;
function v_chkType() {
   var strHTML="";
   var arrType = new Array('audio/mpeg;', 'audio/mov;',
   'audio/mp4;codecs="mp4a.40.2"', 'audio/ogg;codecs="vorbis"',
   'video/webm;codecs="vp8,vorbis"', 'audio/wav;codecs="1"');
   for (intI = 0; intI < arrType.length; intI++) {
        switch ($$("vdoMain").canPlayType(arrType[intI])) {
        case "":
            i = i + 1;
```

```
            break;
        case "maybe":
            j = j + 1;
            break;
        case "probably":
            k = k + 1;
            break;
        }
    }
    strHTML+=" 空字符: "+i+"<br>";
    strHTML+="maybe: "+j+"<br>";
    strHTML+="probably: "+k;
    $$("spnResult").style.display="block";
    $$("spnResult").innerHTML=strHTML;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-8 所示。

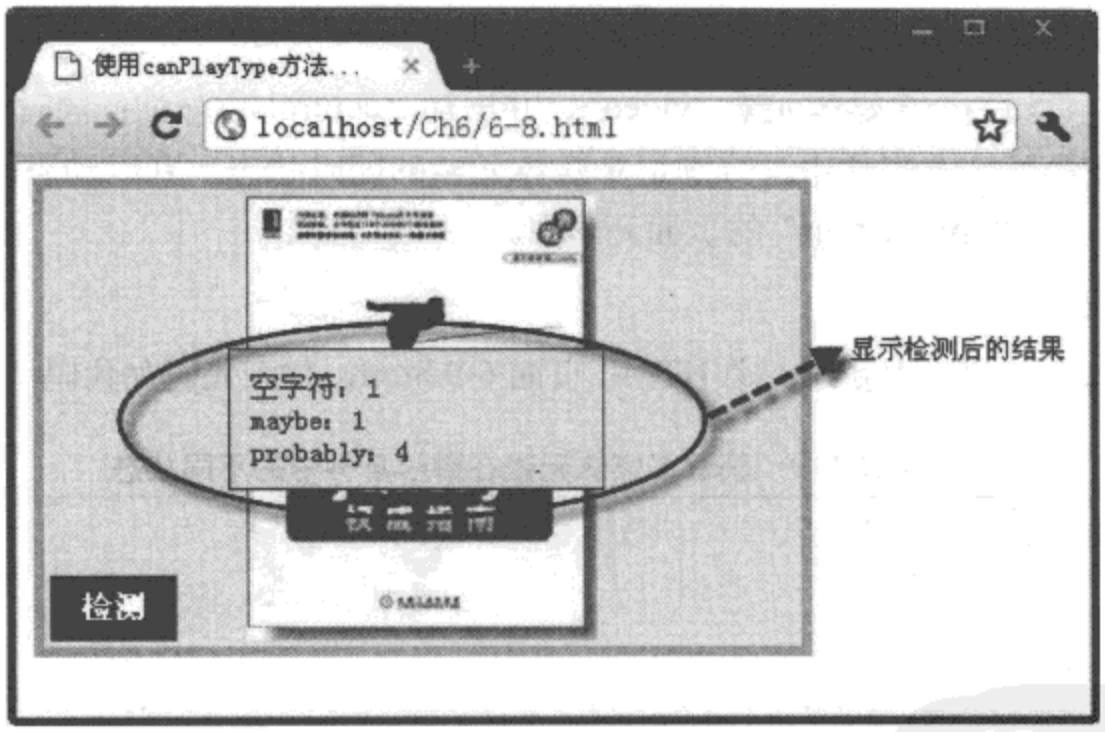


图 6-8　使用 canPlayType 方法检测浏览器支持媒体类型

4. 源码分析

在本实例中，当用户在页面中单击内容为“检测”的 元素时，将调用一个自定义函数 v_chkType()。在该函数中，首先，定义一个数组“arrType”，用于保存各种媒体类型及编码格式；然后，遍历该数组中的元素。在遍历过程中，调用多媒体元素的 canPlayType() 方法，对每种类型及编码格式进行检测，并将返回检测结果值的累加总量保存至各自变量中；最后，将这些变量值数据通过 ID 号为“spnResult”的元素显示在页面中，实现过程如 JavaScript 代码中加粗部分所示。

6.3 多媒体元素重要事件

多媒体元素不仅有相关的属性、方法，而且该元素还有一系列完备的事件机制。在6.1节介绍多媒体元素的“networkState”与“error”属性时，分别触发了“progress”与“error”事件。除此之外，还有许多记录媒体文件播放过程的事件，如“playing”等，下面分别进行介绍。

6.3.1 媒体播放事件

在媒体文件被浏览请求加载、开始加载、开始播放、暂停播放、播放结束这一系列的流程中所触发的事件，称为“媒体播放事件”，也是多媒体元素的核心事件。通过对这些事件的跟踪，可以很方便地获取媒体文件在各个阶段的播放状态。

下面通过实例6-9介绍获取多媒体元素在播放事件中不同状态的过程。

实例6-9 获取多媒体元素在播放事件中的不同状态

1. 功能描述

在页面中，添加一个多媒体元素<video>，并增加“controls”属性；同时，通过自定义函数绑定多个播放事件。在事件中，分别记录媒体元素的即时状态，并以动态的方式，将状态内容显示在ID号为“spnPlayTip”的页面元素中。

2. 实现代码

在Dreamweaver CS5中新建一个HTML页面6-9.html，加入代码如代码清单6-9所示。

代码清单6-9 获取多媒体元素在播放事件中的不同状态

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>获取多媒体元素在播放事件中的不同状态</title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js9.js"/>
</script>
</head>
<body>
<div>
  <video id="vdoMain" src="Video/6-test_1.mov"
         width="360px" height="220px" controls="true"
         onMouseOut="v_move(0)" onMouseOver="v_move(1)"
         onPlaying="v_palying()" onPause="v_pause()"
         onLoadStart="v_loadstart();"
```

```
            onEnded="v_ended();"
            poster="Images/2010 年作品 .jpg">
            你的浏览器不支持视频
    </video>
    <p id="pTip">
        <span id="spnPlayTip" class="spnL"></span>
    </p>
</div>
</body>
</html>
```

在代码清单 6-9 中，页面导入一个 JavaScript 文件 js9.js，其中自定义了多个函数，分别响应各个播放事件被触发时的调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function v_move(v){
    $$("pTip").style.display=(v)?"block":"none";
}
function v_loadstart() {
    $$("spnPlayTip").innerHTML=" 开始加载 ";
}
function v_palying(){
    $$("spnPlayTip").innerHTML=" 正在播放 ";
}
function v_pause(){
    $$("spnPlayTip").innerHTML=" 已经暂停 ";
}
function v_ended(){
    $$("spnPlayTip").innerHTML=" 播放完成 ";
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-9 所示。

4. 源码分析

在本实例中，实现了鼠标移至多媒体元素时，显示媒体播放状态，移出元素时，隐藏播放状态的效果。实现方法是在多媒体元素的 onMouseOut 与 onMouseOver 事件中，传递不同的参数值，调用同一个自定义的函数 v_move()。在该函数中，将根据传回的参数值，显示或隐藏 ID 号为“pTip”的页面元素，从而实现鼠标移至或移出多媒体元素的效果。

为了在多媒体元素触发播放事件的过程中，动态显示媒体文件的播放状态，需要在绑定的事件中，修改 ID 号为“spnPlayTip”的元素内容，实现过程如 JavaScript 代码中加粗部分所示。

图 6-9　获取多媒体元素在播放事件中的不同状态

6.3.2　timeupdate 事件

在多媒体元素的众多事件中，timeupdate 事件是一个十分重要的事件。在媒体文件播放过程中，如果播放位置发生变化，就会触发该事件。利用该事件，结合多媒体元素的“currentTime”与“duration”属性，可以动态显示媒体文件播放的当前时间与总量时间。

下面通过实例 6-10 介绍通过 timeupdate 事件动态显示媒体文件播放时间的过程。

实例 6-10　通过 timeupdate 事件动态显示媒体文件播放时间

1. 功能描述

在实例 6-9 的基础上，为多媒体元素 <video> 添加一个“onTimeUpdate”事件，用于改变播放文件位置时调用。另外，新增加一个 ID 号为“spnTimeTip”的 元素，用于动态显示媒体文件播放的当前时间与总量时间。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 6-10.html，加入代码如代码清单 6-10 所示。

代码清单 6-10　通过 timeupdate 事件动态显示媒体文件播放时间

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="utf-8" />
<title>通过 timeupdate 事件动态显示媒体文件播放时间 </title>
<link href="Css/css6.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js10.js"/>
</script>
</head>
<body>
<div>
  <video id="vdoMain" src="Video/6-test_1.mov"
         width="360px" height="220px" controls="true"
         onMouseOut="v_move(0)" onMouseOver="v_move(1)"
         onPlaying="v_palying()" onPause="v_pause()"
         onLoadStart="v_loadstart();"
         onEnded="v_ended();"
         onTimeUpdate="v_timeupdate(this)"
         poster="Images/2010 年作品 .jpg">
         你的浏览器不支持视频
  </video>
  <p id="pTip">
     <span id="spnPlayTip" class="spnL"></span>
     <span id="spnTimeTip" class="spnR">00:00 / 00:00</span>
  </p>
<div>
</body>
</html>
```

在代码清单 6-10 中，页面导入一个 JavaScript 文件 js10.js，其中，除其他自定义函数外，新增了一个函数 v_timeupdate()，在改变播放位置时调用。其实现的代码如下所示：

```
... 与实例 6-9 的代码相同部分已省略
function v_timeupdate(e){
   var strCurTime=RuleTime(Math.floor(e.currentTime/60),2)+":"+
                  RuleTime(Math.floor(e.currentTime%60),2);
    var strEndTime=RuleTime(Math.floor(e.duration/60),2)+":"+
                  RuleTime(Math.floor(e.duration%60),2);
   $$("spnTimeTip").innerHTML=strCurTime+" / "+strEndTime;
}
//转换时间显示格式
function RuleTime(num, n) {
    var len = num.toString().length;
    while(len < n) {
        num = "0" + num;
        len++;
    }
```

```
    return num;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 6-10 所示。

图 6-10　通过 timeupdate 事件动态显示媒体文件播放时间

4. 源码分析

在本实例中，当多媒体元素触发 timeupdate 事件时，调用一个自定义函数 v_timeupdate()。在该函数中，分别使用整除与求余数的方法，分割多媒体元素当前时间 (currentTime) 属性与时间总量 (duration) 属性返回的秒值，组成分与秒的格式。在组成过程中，又调用了另外一个自定义函数 RuleTime()，该函数可以将长度不足 2 位的数字，在前面加“0”进行补充，实现过程如 JavaScript 代码中加粗部分所示。

6.3.3　其他事件

除了在 6.3.1 节与 6.3.2 节中介绍的事件外，多媒体元素还有很多实用的事件。例如，播放总长度改变时，将触发 durationchange 事件；音量大小改变或启动静音时，将触发 volumechange 事件等。多媒体元素的相关事件如表 6-5 所示。

在 HTML 5 中，多媒体元素 <video> 与 <audio> 可以利用这些自带的事件，方便快捷地实现在原来 HTML 4 以前版本中较为复杂的功能。

表 6-5　多媒体元素的相关事件

事件名称	描　述
abort	浏览器下载媒体文件过程中，非错误引起中止下载时触发
canplay	媒体文件播放时，因速率原因需要缓存，引起不能正常播放时触发
canplaythrough	媒体文件以当前的速率，无需缓存可以正常播放时触发
durationchange	当媒体文件的播放时间总量长度发生变化时触发
emptied	当多媒体元素播放一个未知或异常的媒体文件时触发
ended	当媒体文件播放结束时触发
error	当媒体文件在加载过程中出现错误时触发
loadeddata	当多媒体元素加载完成当前指定位置的媒体文件时触发
loadedmetadata	当多媒体元素加载完成时间总长与总字节数时触发
loadstart	当浏览器开始加载媒体文件时触发
pause	当多媒体元素中止播放媒体文件时触发
play	当多媒体元素开始播放媒体文件时触发
playing	当多媒体元素正在播放媒体文件时触发
progress	当多媒体元素正在读取媒体文件时触发
ratechange	当多媒体元素的默认速率或当前速率改变时触发
readystatechange	当多媒体元素的“readystate”属性值改变时触发
seeked	当多媒体元素的“seeking”属性为 false，浏览器中止数据请求时触发
seeking	当多媒体元素的“seeking”属性为 true，浏览器正在请求数据时触发
stalled	当多媒体元素因延时原因，无法正常获取数据时触发
suspend	浏览器暂停数据的下载，但下载过程并未结束时触发
timeupdate	当多媒体元素播放媒体文件的位置发生变化时触发
volumechange	当多媒体元素音量大小改变或启动静音时触发
waiting	当多媒体元素正在等待加载下一帧播放数据时触发

6.4　本章小结

本章先从最基础的概念入手，介绍多媒体元素 <video> 与 <audio> 在页面中的使用方法；然后，通过理论与实例结合的方式，分别介绍了多媒体元素涉及的属性、方法及重要的事件，为读者熟练掌握视频与音频元素在 HTML 5 中的应用打下扎实的基础。

第7章 HTML 5绘图基础

本章内容

- 画布的基础知识
- 在画布中使用路径
- 对画布中图形的操作
- 处理画布中的图像
- 画布的其他应用
- 本章小结

在 HTML 5 中，新增一个非常重要的元素——<canvas>。之所以说该元素非常重要，是因为它是页面中的一块画布，可以在该画布上绘制任意图形（包括导入图片）；另外，借助该元素自带的 API，通过编写 JavaScript 代码，可以在 <canvas> 元素中控制各种图形、制作动画效果，这对 Web 页面来说，无疑具有划时代的意义。

7.1 画布的基础知识

想在页面中利用新增加的画布元素 <canvas> 绘画图形，需要经过以下三个步骤：

步骤 1 使用 <canvas> 元素创建一个画布区域，并获取该元素。

步骤 2 通过获取的 <canvas> 元素，取得该图形元素的上下文环境对象。

步骤 3 根据取得的上下文环境对象，在页面中绘制图形或动画。

本节结合几个简单的实例，详细介绍画布元素 <canvas> 的基本用法。

7.1.1 canvas 元素的基本用法

与创建页面中的其他元素相同，<canvas> 元素的创建也十分简单，只需要加一个标记 ID 号，设置元素的长和宽即可，代码如下所示：

```
<canvas id="cnvMain" width="280px" height="190px"></canvas>
```

画布创建完成后，就可以利用画布的上下文环境对象绘制图形了。

下面通过实例 7-1 介绍使用 <canvas> 元素绘制一个正方形的过程。

实例 7-1 使用 <canvas> 元素绘制正方形

1. 功能描述

在页面中，新建一个 <canvas> 元素，并在该元素中绘制一个指定长度的正方形。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-1.html，加入代码如代码清单 7-1 所示。

代码清单 7-1 使用 <canvas> 元素绘制正方形

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>canvas 简单示例 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js1.js"/>
</script>
</head>
```

```
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-1 中，页面导入一个 JavaScript 文件 js1.js，其中编写了一个自定义函数 pageload()，在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function pageload(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   cxt.fillStyle="#ccc";
   cxt.fillRect(30,30,80,80);
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-1 所示。

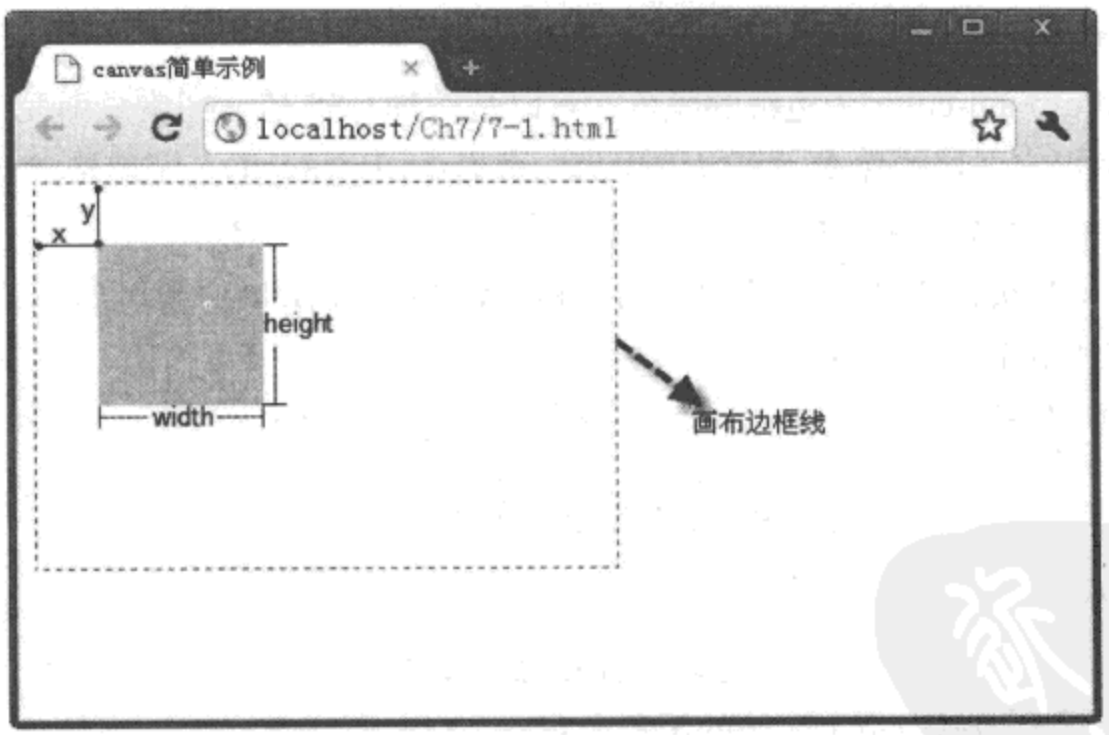


图 7-1　使用 <canvas> 元素绘制正方形

4. 源码分析

在本实例的 JavaScript 代码中，首先获取 <canvas> 元素，然后取得绘图元素的上下文环境对象“cxt”。在获取过程中，需要调用画布的 getContext() 方法，并向该方法传递一个字符串为“2d”的参数；一旦取得画布的上下文环境对象后，就可以通过该对象来使用绘图的方法与属性。例如，绘制一个矩形的方法如下：

```
cxt.fillRect(x,y,width,height);
```

其中，参数 x 表示矩形起点 x 轴与左上角（0，0）间的距离，参数 y 表示矩形起点 y 轴与左上角（0，0）的距离，参数 width 表示矩形的宽度，参数 height 表示矩形的高度，其所在位置如图 7-1 所示。在绘制矩形之前，需要设置图形的背景色，方法如下：

```
cxt.fillStyle="background-color";
```

其中，参数 background-color 可以是一种 CSS 颜色、图案、渐变色，默认值为黑色。本实例为“#ccc”，是一种 CSS 颜色。注意，设置绘制图形背景色的操作必须先于图形绘制，否则，所设置的背景色将不起作用，完整实现过程如 JavaScript 代码中加粗部分所示。

7.1.2 绘制带边框矩形

利用画布除了可以绘制有背景色的图形外，还可以绘制有边框的图形。实现过程是：在获取绘图上下文环境对象 cxt 后，调用一个 strokeRect() 方法。该方法用来绘制一个矩形，但并不填充矩形区域，而是绘制矩形的边框，其调用格式如下：

```
cxt.strokeRect(x,y,width,height);
```

其中参数 x，y 为矩形起点坐标，width 与 height 分别为矩形宽度与高度。当然，在绘制边框前，可以调用 strokeStyle 属性设置边框的颜色，格式如下：

```
cxt.strokeStyle="background-color";
```

其中，参数 background-color 为边框的颜色，可以是一种 CSS 值、图案或渐变色；另外，如果想要清空图形中指定区域的像素，可以调用另一个方法 clearRect()，调用格式如下：

```
cxt.clearRect(x,y,width,height);
```

其中，参数 x，y 为被清空色彩区域起点的坐标，width 与 height 分别为被清空像素区域的宽度与高度，清空后的区域为透明色。

下面通过实例 7-2 介绍使用 <canvas> 元素绘制带边框矩形的过程。

实例 7-2 使用 <canvas> 元素绘制带边框的矩形

1. 功能描述

在页面中，新建一个 <canvas> 元素，并在该元素中绘制一个有背景色和边框的矩形。单击该矩形时，将清空矩形中指定区域的图形色彩。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-2.html，加入代码如代码清单 7-2 所示。

代码清单 7-2 使用 <canvas> 元素绘制带边框的矩形

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="utf-8" />
<title>canvas 元素绘制带边框矩形 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js2.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"
            onClick="cnvClick();">
    </canvas>
</body>
</html>
```

在代码清单 7-2 中，页面导入一个 JavaScript 文件 js2.js，其中，除函数 pageload() 外，还新增另一个自定义函数 cnvClick()，在单击图形时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function pageload(){
    var cnv=$$("cnvMain");
    var cxt=cnv.getContext("2d");
   // 设置边框
    cxt.strokeStyle="#666";
    cxt.strokeRect(30,30,150,80);
    // 设置背景
    cxt.fillStyle="#eee";
    cxt.fillRect(30,30,150,80);
}
function cnvClick(){
     var cnv=$$("cnvMain");
     var cxt=cnv.getContext("2d");
    // 清空图形
    cxt.clearRect(36,36,138,68);
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-2 所示。

4. 源码分析

在本实例的 JavaScript 代码中，当开始加载页面时，调用一个自定义的函数 pageload()。在该函数中，使用 fillRect() 方法绘制带背景色的图形，此外，还调用了 strokeRect() 方法绘制带边框的图形。在调用 strokeRect() 方法前，先通过“strokeStyle”属性设置所绘制边框的颜色为“#666”。由于 fillRect() 方法与 strokeRect() 方法中所使用的参数值相同，因此，将绘制

一个背景色和边框重叠的矩形，实现效果如图 7-2 所示。

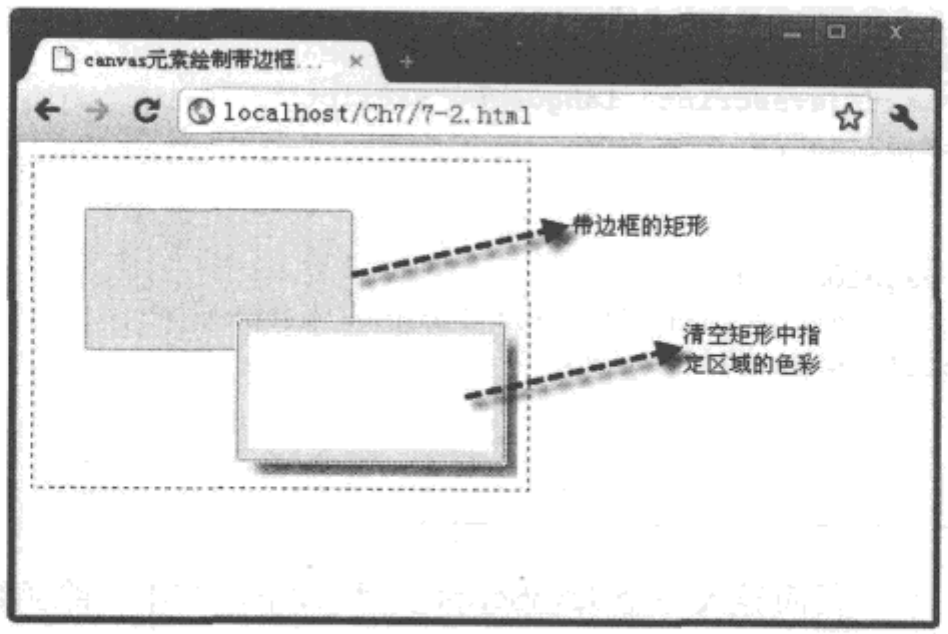


图 7-2　使用 <canvas> 元素绘制带边框矩形的效果

当用户单击绘制好的矩形时，将触发一个 onClick 事件，该事件调用自定义函数 cnvClick()。在该函数中，使用 clearRect() 方法清空指定区域的色彩，完整的实现过程如代码中加粗部分所示。

7.1.3　绘制渐变图形

在 HTML 5 中，利用 <canvas> 元素可以绘制出有渐变色的图形。渐变方式分为两种，一种是线性渐变，另一种是径向渐变。本小节介绍使用线性渐变的方式绘制图形，操作步骤如下：

步骤 1　在获取上下文环境对象 cxt 后，调用该对象的 createLinearGradient() 方法创建一个 LinearGradient 对象，调用格式如下：

```
cxt.createLinearGradient(xStart,yStart,xEnd,yEnd)
```

其中，参数 xStart，yStart 表示渐变色开始时的坐标；xEnd，yEnd 为渐变色结束时的坐标。如果 yStart 与 yEnd 相同，表示渐变色沿水平方向从左向右渐变；如果 xStart 与 xEnd 相同，表示渐变色沿纵坐标方向上下渐变；如果 xStart 与 xEnd 不相同，并且 yStart 与 yEnd 也不相同，则表示渐变色沿矩形对角线方向渐变，如图 7-3 所示。

步骤 2　创建 LinearGradient 对象并将其取名为 gnt 后，调用该对象的 addColorStop() 方法，进行渐变颜色与偏移量的设置，调用格式如下：

```
gnt.addColorStop(value,color);
```

其中，参数 value 表示渐变位置偏移量，它可以在 0 与 1 之间取任意值；参数 color 表示渐变开始与结束时的颜色，分别对应偏移量 0 与 1。为了实现颜色的渐变功能，必须调用两次该方法，第一次表示开始渐变时的颜色，第二次表示结束渐变时的颜色，如图 7-4 所示。

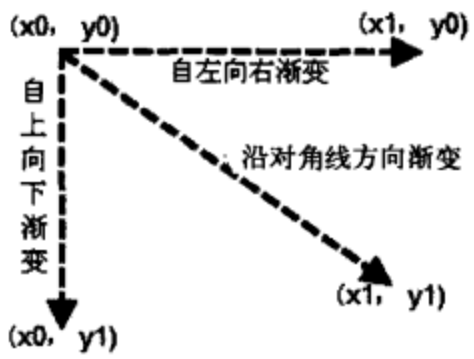


图 7-3　三种常见图形颜色渐变的方向

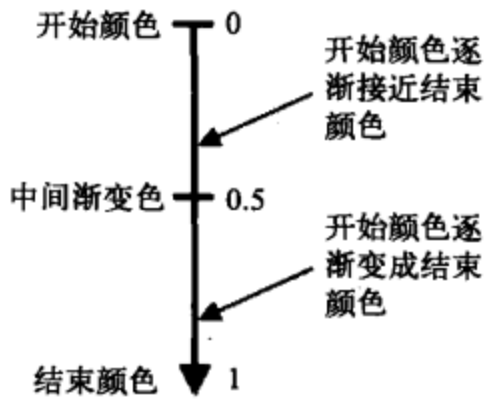


图 7-4　偏移量在颜色渐变时的变化过程

步骤 3　通过 gnt 对象将偏移量与渐变色的值设置完成后，再将 gnt 对象赋值给“fillStyle”属性，表明此次图形的样式是一个渐变对象，最后，使用 fillRect() 方法绘制出一个有渐变色的图形。

下面通过实例 7-3 介绍使用 <canvas> 元素绘制有渐变色图形的过程。

实例 7-3　使用 <canvas> 元素绘制有渐变色的图形

1. 功能描述

在页面中，新建一个 <canvas> 元素，利用该元素，以三种不同颜色渐变方向绘制图形，分别为自左向右、从上而下、沿图形对角线方向渐变。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-3.html，加入代码如代码清单 7-3 所示。

代码清单 7-3　使用 <canvas> 元素绘制有渐变色的图形

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 <canvas> 元素绘制有渐变色的图形 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js3.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-3 中，页面导入一个 JavaScript 文件 js3.js，其中自定义一个函数 pageload()，

用于页面加载时，绘制不同颜色渐变方向的图形。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function pageload(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   //绘制由左至右的颜色渐变图形
   var gnt1=cxt.createLinearGradient(20,20,150,20);
   gnt1.addColorStop(0,"#000");
   gnt1.addColorStop(1,"#fff");
   cxt.fillStyle=gnt1;
   cxt.fillRect(20,20,150,20);
   //绘制由上至下的颜色渐变图形
   var gnt2=cxt.createLinearGradient(20,20,20,150);
   gnt2.addColorStop(0,"#000");
   gnt2.addColorStop(1,"#fff");
   cxt.fillStyle=gnt2;
   cxt.fillRect(20,20,20,150);
   //绘制沿对角线的颜色渐变图形
   var gnt3=cxt.createLinearGradient(50,50,100,100);
   gnt3.addColorStop(0,"#000");
   gnt3.addColorStop(1,"#fff");
   cxt.fillStyle=gnt3;
   cxt.fillRect(50,50,100,100);
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-5 所示。

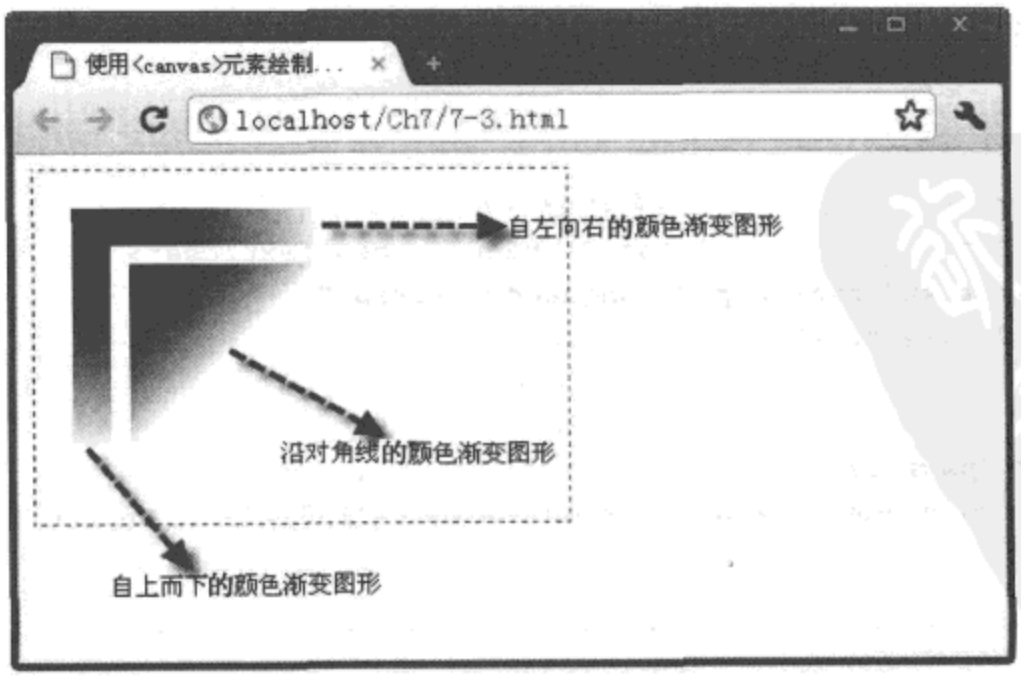


图 7-5　使用 <canvas> 元素绘制有渐变色图形

4. 源码分析

在本实例的 JavaScript 代码中，自定义了一个函数 pageload()，当页面触发 onLoad 事件时调用该函数。在该函数中，调用 createLinearGradient() 方法，传递不同的渐变色起点与终点坐标参数值，创建三个不同的 LinearGradient 对象，分别对应自左向右、自上而下、沿对角线渐变三种渐变方式，并分别将 LinearGradient 对象设置为对应 fileStyle 的属性值，从而实现绘制沿不同方向渐变色的图形，完整实现过程如代码中加粗部分所示。

7.2　在画布中使用路径

7.1 节通过使用 <canvas> 元素，绘制了不同形状与背景色的各种图形。除此之外，还可以通过该元素绘制直线与圆，这需要借助画布中的路径功能。

在页面的画布元素中，调用绘画路径的两个方法 moveTo() 与 lineTo() 可以绘制直线，调用 arc() 方法可以绘制指定位置与大小的圆形。下面详细介绍这些方法在画布中的使用。

7.2.1　moveTo 与 lineTo 的用法

在画布元素中，如果要绘制直线，通常使用 moveTo() 与 lineTo() 两个方法。moveTo() 方法用于将画笔移至指定点并以该点为直线的开始点，调用格式如下：

```
cxt.moveTo(x,y)
```

其中，cxt 为上下文环境对象名称，x 为移至起点的横坐标，y 为移至起点的纵坐标。调用该方法后，画布中即设置了一个绘制直线的开始点。如果是绘制直线还需要调用 lineTo() 方法，该方法将用画笔从指定的起点坐标与传递的终点坐标参数之间绘制一条直线，调用格式如下：

```
cxt.lineTo(x,y)
```

其中，x 为移至的终点横坐标，y 为移至终点的纵坐标。该方法可以反复调用，第一次调用后，画笔自动移至终点坐标位置；第二次调用时，又以该终点坐标位置作为第二次调用时的起点位置，开始绘制直线。当直线路径绘制完成后，再调用 stroke() 方法在画布中描边直线路径，最终在画布中展示直线效果，该方法的调用格式如下：

```
cxt.stroke()
```

stroke() 方法无参数，用于绘制完路径后对路径进行描边处理。

下面通过实例 7-4 介绍使用 <canvas> 元素绘制多条直线的过程。

实例 7-4　使用 moveTo 与 lineTo 方法绘制多条直线

1. 功能描述

在页面中，新建一个 <canvas> 元素。在该元素中，调用 moveTo() 与 moveLine() 方法绘制两条相互衔接的直线，并显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-4.html，加入代码如代码清单 7-4 所示。

代码清单 7-4　使用路径绘制多条直线

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>通过路径画直线</title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js4.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-4 中，页面导入一个 JavaScript 文件 js4.js，其中自定义一个函数 pageload()，用于页面加载时绘制两条不同位置相互衔接的直线。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function pageload(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   cxt.moveTo(130,30);
   cxt.lineTo(30,100);
   cxt.lineTo(130,160);
   cxt.lineWidth=3;
   cxt.stroke();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-6 所示。

4. 源码分析

在本实例的 JavaScript 代码中，先在坐标为（130,30）的位置，调用 moveTo() 方法绘制一个直线的开始点；然后，在该点与坐标为（30,100）的位置之间，绘制第一条直线路径，此时，画笔锁定在第一条直线结束点坐标（30,100）处，又以第一条直线结束点坐标为开始点。在该点与坐标为（130,160）的位置之间，绘制第二条直线路径，此时，画笔又锁定在第二条直线结束点坐标（130,160）处。

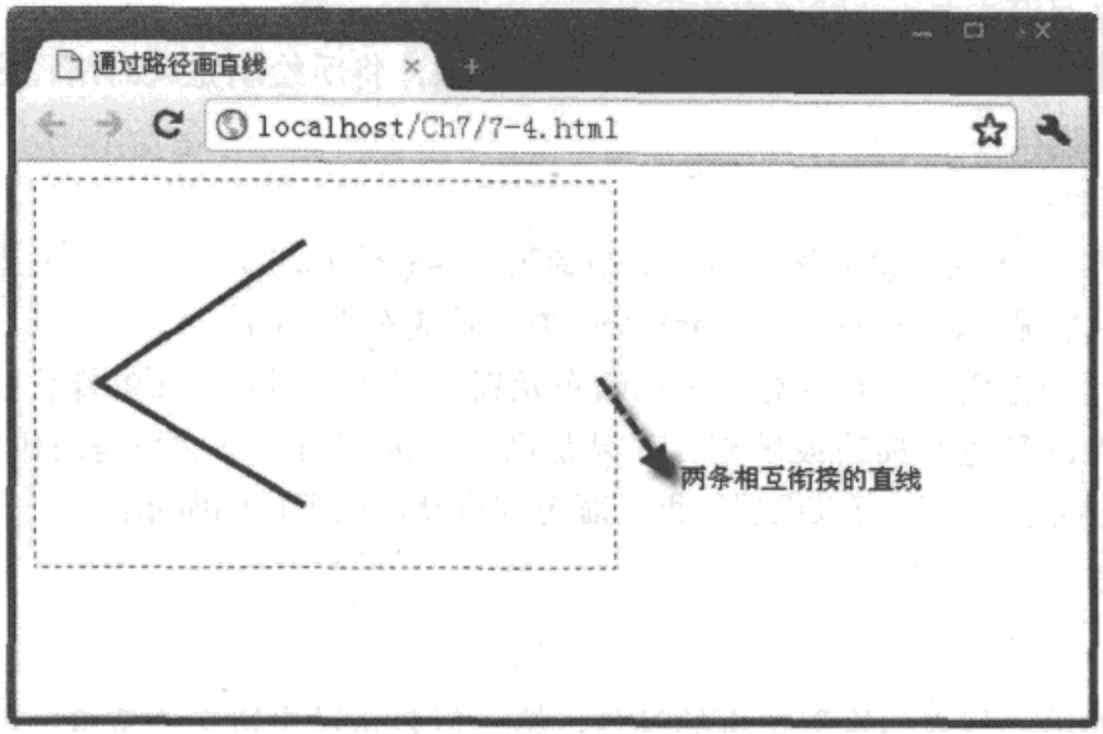


图 7-6　使用路径绘制两条直线的效果

两条直线路径绘制完成后，使用“lineWidth”属性设置直线的边框值，最后调用 stroke() 方法，对绘制完成的路径进行描边。描边时，默认颜色为黑色，也可以通过“strokeStyle”属性进行设置，描边完成后，两条绘制的直线便显示在画布中。

7.2.2　使用 arc 方法绘制圆形

在画布中，除了使用方法 moveTo() 与 lineTo() 绘制路径和描边直线外，还可以使用上下文环境对象中的 arc() 方法描绘圆形路径，以及绘制各种形状的圆形图案，该方法的调用格式如下：

```
cxt.arc(x,y,radius,startAngle,endAngle,anticlockwise)
```

其中，cxt 为上下文环境对象名称，参数 x 表示绘制圆形的横坐标，参数 y 表示绘制圆形的纵坐标；参数 radius 表示绘制圆的半径，单位为像素；参数 startAngle 表示绘制圆弧时的开始角度，参数 endAngle 表示绘制圆弧时的结束角度。

举一个例子：如果想绘制一个完整的圆形，那么，参数 startAngle 的值为 0，表示从 0 弧度开始；参数 endAngle 的值为 Math.PI*2，表示到 360 弧度时结束。如果想绘制一个半圆形，那么，参数 startAngle 的值不变，参数 endAngle 的值为 Math.PI*1，表示到 180 弧度时结束。参数 anticlockwise 是一个布尔值，表示是否按顺时针绘制，如果为“true”，表示按顺时针绘制；如果为“false”，表示按逆时针绘制。

在调用 arc() 方法绘制圆形路径之前，需要调用上下文环境对象中的 beginPath() 方法，声明开始绘制路径，其调用格式如下：

```
cxt.beginPath()
```

其中，cxt 为上下文环境对象名称，该方法无参数。需要注意的是，在使用遍历或循环绘制路

径时，每次都要调用该方法，即该方法仅对应单次的路径绘制。

绘制圆形路径完成后，还要调用 closePath() 方法，将所绘制完成的路径进行关闭，其调用的格式如下：

```
cxt.closePath()
```

其中，cxt 为上下文环境对象名称。该方法的参数与 beginPath() 方法一样，也是对应单次的路径绘制。在一般情况下，该方法与 beginPath() 方法是成对出现的。

圆形路径绘制完成后，并没有真正在画布元素中展示，因为上面的操作仅绘制了圆形的路径，还需要对路径进行描边或填充。如果是描边，则调用上下文环境对象中的 stroke() 方法；在调用该方法之前，还可以设置边框的颜色与宽度，如下代码所示：

```
cxt.strokeStyle="#ccc";
cxt.lineWidth=2;
cxt.stroke();
```

上述代码的第一行表示设置边框的颜色，第二行表示设置边框的宽度，第三行表示开始进行描边操作。需要注意的是，设置边框颜色与宽度的代码必须在描边操作前，否则将不起作用。

除了对已经绘制的圆形路径进行描边外，还可以调用上下文环境对象中的 fill() 方法进行填充操作。当然，在调用该方法之前，也可以设置填充的颜色，如下代码所示：

```
cxt.fillStyle="#eee";
cxt.fill();
```

上述代码的第一行表示设置填充圆形路径的颜色，第二行表示开始进行填充。与描边操作一样，设置填充圆形路径的颜色的代码必须在填充操作之前，否则也将不起作用。

当然，也可以对所绘制的圆形路径进行既填充又描边。下面通过实例 7-5 介绍使用 <canvas> 元素绘制多个不同样式圆的过程。

实例 7-5　使用 arc 方法绘制多个不同样式的圆形

1. 功能描述

在页面中，新建一个 <canvas> 元素，同时，创建三个 标记，内容分别设置为“实体圆”、“边框圆”、“衔接圆”；当单击某个 标记时，在画布元素中绘制对应图案的圆形。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-5.html，加入代码如代码清单 7-5 所示。

代码清单 7-5　使用路径绘制多个不同样式的圆形

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
```

```
<title>通过路径画圆形</title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js5.js"/>
</script>
</head>
<body>
   <div><p>
    <span onClick="spn1_click();">实体圆</span>
    <span onClick="spn2_click();">边框圆</span>
    <span onClick="spn3_click();">衔接圆</span></p>
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
   <div>
</body>
</html>
```

在代码清单 7-5 中，页面导入一个 JavaScript 文件 js5.js，其中自定义 3 个事件函数，分别在单击每个 元素时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function spn1_click(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   //清除画布原有图形
   cxt.clearRect(0,0,280,190);
   //开始画实体圆
   cxt.beginPath();
   cxt.arc(100,100,50,0,Math.PI*2,true);
   cxt.closePath();
   //设置填充背景色
   cxt.fillStyle="#eee";
   //进行填充
   cxt.fill();
}
function spn2_click(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   //清除画布原有图形
   cxt.clearRect(0,0,280,190);
   //开始画边框圆
   cxt.beginPath();
   cxt.arc(100,100,50,0,Math.PI*2,true);
   cxt.closePath();
   //设置边框色
   cxt.strokeStyle="#666";
```

```
        //设置边框宽度
        cxt.lineWidth=2;
        //进行描边
        cxt.stroke();
    }
    function spn3_click(){
        var cnv=$$("cnvMain");
        var cxt=cnv.getContext("2d");
        //清除画布原有图形
        cxt.clearRect(0,0,280,190);
        //开始画圆
        cxt.beginPath();
        cxt.arc(100,100,50,0,Math.PI*2,true);
        cxt.closePath();
        //设置填充背景色
        cxt.fillStyle="#eee";
        //进行填充
        cxt.fill();
        //设置边框色
        cxt.strokeStyle="#666";
        //设置边框宽度
        cxt.lineWidth=2
        //进行描边
        cxt.stroke();
        //开始画衔接的边框圆
        cxt.beginPath();
        cxt.arc(175,100,50,0,Math.PI*2,true);
        cxt.closePath();
        //设置边框色
        cxt.strokeStyle="#666";
        //设置边框宽度
        cxt.lineWidth=2
        //进行描边
        cxt.stroke();
    }
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-7 所示。

4. 源码分析

在本实例中，当单击“实体圆”标记时，将调用自定义函数 spn1_click()。在该函数中，首先，通过获取的上下文环境对象 cxt 来调用 clearRect() 方法，清空画布中原有的图形，防止图形在画布中的交叉展示；然后，调用 arc() 方法绘制一个圆形路径，其圆心坐标为（100，100）、半径为 50 像素、弧度为从 0 开始到 Math.PI*2 结束、按顺时针方向进行绘制；路径绘制完成后，设置填充颜色；最后，使用 fill() 方法将颜色填充至已绘制的圆路径中，从而在画

布中形成一个实体的圆形。

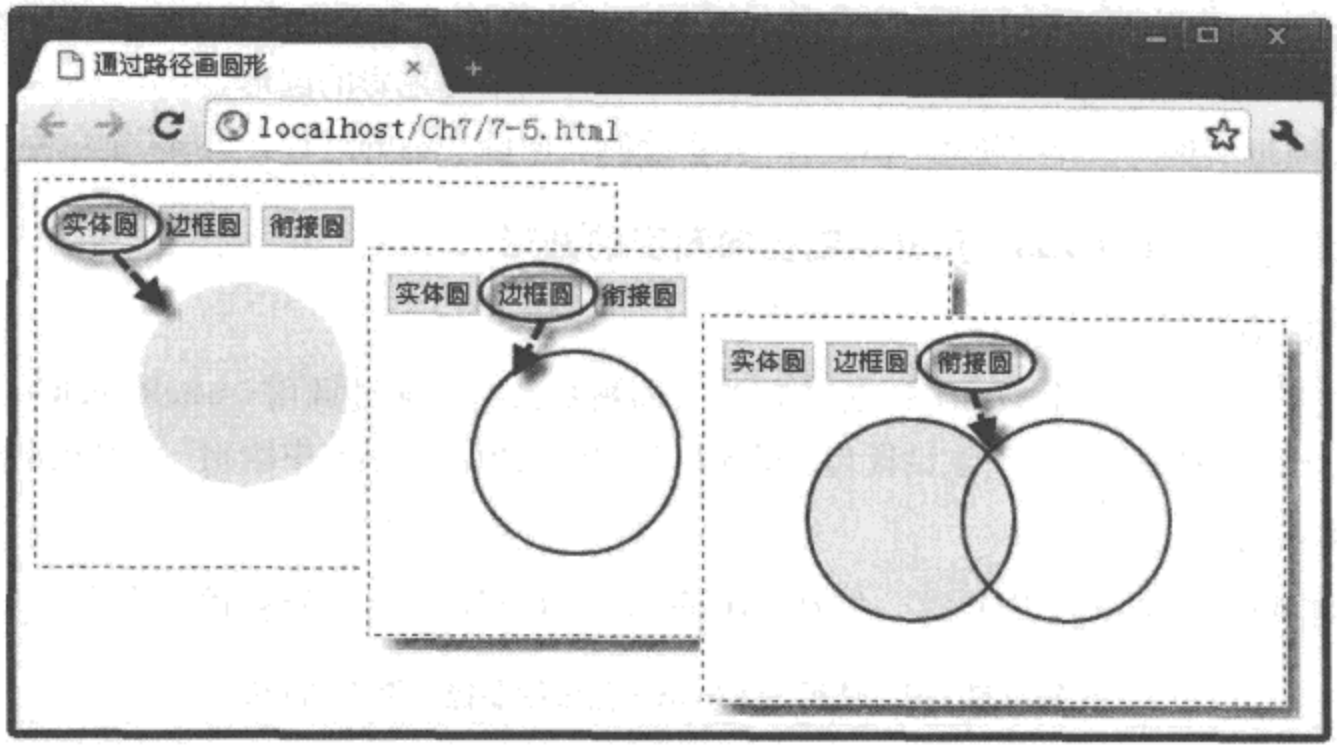


图 7-7　使用路径绘制多个不同形状的圆形

在自定义函数 spn2_click() 中，绘制圆形路径的过程与 spn1_click() 函数相同，只是在最后绘制图形时使用了 stroke() 方法对路径进行描边，而非 spn1_click() 函数中的 fill() 填充方法。在进行描边前，通过“lineWidth”与“strokeStyle”属性分别设置边框的宽度与颜色；最后，使用 stroke() 方法，按照设置的颜色与宽度对已绘制的圆路径进行描边，从而在画布中形成一个边框圆。

在自定义函数 spn3_click() 中，结合了函数 spn1_click() 与 spn2_click() 中绘制圆形的方法与过程，只是在绘制第二个圆形时，改变了圆心的横坐标距离，而其他参数值均不变化，完整的实现过程如自定义函数 spn3_click() 中加粗部分所示。

7.2.3　绘制渐变圆形

7.1.3 节中介绍了使用线形渐变的方式绘制多个不同渐变方向的矩形，同样，在画布中也可以绘制径向渐变图形。如果需要利用径向渐变的方式绘制有渐变色的圆形，只要调用上下文环境对象 cxt 中的 createRadialGradient() 方法即可，格式如下：

```
cxt.createRadialGradient(xStart,yStart,radiusStart,xEnd,yEnd,radiusEnd)
```

其中，参数 cxt 为获取的上下文对象名称，参数 xStart 为开始渐变圆心的横坐标，参数 yStart 为开始渐变圆心的纵坐标，参数 radiusStart 为开始渐变圆的半径；参数 xEnd 为结束渐变圆心的横坐标，参数 yEnd 为结束渐变圆心的纵坐标，参数 radiusEnd 为结束渐变圆的半径。

调用该方法时，从开始渐变圆心的坐标位置向结束渐变圆心的坐标位置进行颜色渐变，即两个圆之间通过各自的圆心坐标连接成一条直线，起点为开始圆心，终点为结束圆心，色彩由起点向终点进行扩散，直至终点圆外框。

当然，使用 createRadialGradient() 方法仅新建了一个径向渐变的对象，接下来需要通过 addColorStop() 方法为该对象添加偏移量与渐变色，并将该对象设置为“fillStyle”属性的值；最后，通过调用 fill() 方法，在画布中绘制出一个有径向渐变色彩的圆形。

下面通过实例 7-6 介绍使用 <canvas> 元素绘制径向渐变圆的过程。

实例 7-6　使用 <canvas> 元素绘制径向渐变的圆形

1. 功能描述

在页面中，新建一个 <canvas> 元素，当页面被加载时，通过调用 createRadialGradient() 方法创建一个渐变对象，将该对象设置为“fillStyle”属性的值，在画布中绘制一个径向渐变的圆。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-6.html，加入代码如代码清单 7-6 所示。

代码清单 7-6　使用 <canvas> 元素绘制径向渐变的圆形

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>绘制径向渐变的圆 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js6.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-6 中，页面导入一个 JavaScript 文件 js6.js，通过自定义函数 pageload() 来实现绘制径向渐变圆的功能，该函数在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function pageload(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   // 开始创建渐变对象
   var gnt=cxt.createRadialGradient(30,30,0,20,20,400);
   gnt.addColorStop(0,"#000");
   gnt.addColorStop(0.3,"#eee");
```

```
    gnt.addColorStop(1,"#fff");
    // 开始绘制实体圆路径
    cxt.beginPath();
    cxt.arc(125,95,80,0,Math.PI*2,true);
    cxt.closePath();
    // 设置填充背景色
    cxt.fillStyle=gnt;
    // 进行填充
    cxt.fill();
    // 开始绘制边框圆路径
    cxt.beginPath();
    cxt.arc(125,95,80,0,Math.PI*2,true);
    cxt.closePath();
    // 设置边框颜色
    cxt.strokeStyle="#666";
    // 设置边框宽度
    cxt.lineWidth=2;
    // 开始描边
    cxt.stroke();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-8 所示。

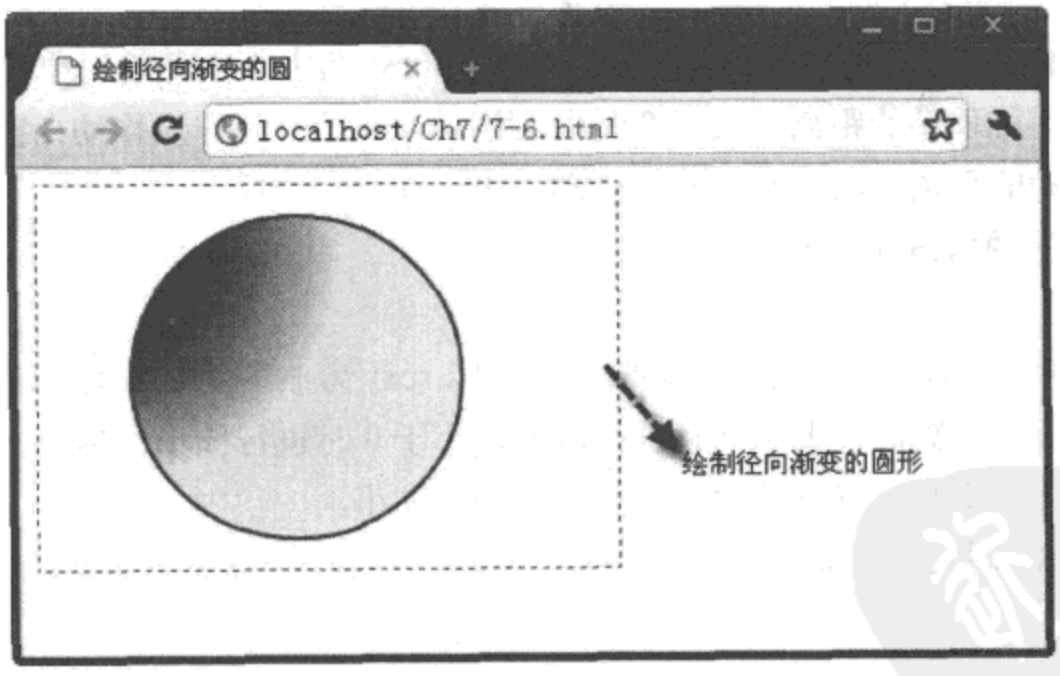


图 7-8　使用 <canvas> 元素绘制径向渐变的圆形

4. 源码分析

在本实例中，获取上下文环境对象 cxt 后，首先，调用该对象的 createRadialGradient() 方法创建一个渐变对象 gnt；然后，通过 gnt 对象的 addColorStop() 方法，为渐变对象增加三种用于渐变的偏移量与颜色值，当绘制完圆路径后，将渐变对象 gnt 赋值给“fillStyle”属性；最

后，根据“fillStyle”属性值，使用 fill() 方法在画布中绘制一个有径向渐变的圆形图案。为了增加实体圆的边框效果，以相同的参数再次调用 arc()，在实体圆的基础上，绘制一个边框圆形，详细实现过程见代码中加粗部分所示。

7.3 对画布中图形的操作

在使用画布元素 <canvas> 绘制图形时，有时需要对已绘制完成的图形进行相关的操作，如移动、缩放和旋转等，这些操作可以借助 Canvas API 中提供的相关方法来实现。通过调用 Canvas API 中提供的相关方法，可以将多块图形以不同的方式结合在一起展示，还可通过增加阴影属性值，为图形添加不同方向的阴影效果，接下来进行详细介绍。

7.3.1 变换图形原点坐标

通过前面的内容我们知道，在利用画布元素绘制图形时，先要定位图形的原点坐标，再设置其他属性才开始绘制；如果改变图形的原点坐标，图形本身也将发生变化。针对这一特点，Canvas API 提供了三种处理变换图形原点坐标的方法：translate()、scale()、rotate()，通过调用这些方法，可以实现对图形的移动、缩放、旋转。

translate() 方法的调用格式如下：

```
cxt.translate(x,y)
```

该方法的功能是实现对图形的移动操作，其中，cxt 为上下文环境对象名称，参数 x 表示将图形原点的横坐标移动的距离，大于 0 时向右移动，小于 0 时向左移动；参数 y 表示将图形的原点纵坐标移动的距离，大于 0 时向下移动，小于 0 时向上移动。

scale() 方法的调用格式如下：

```
cxt.scale(x,y)
```

该方法的功能是实现对图形的缩放操作，其中，cxt 为上下文环境对象名称，参数 x 表示向横坐标方向缩放的倍数值，大于 0 时进行放大，小于 0 时进行缩小；参数 y 表示向纵坐标方向缩放的倍数值，大于 0 时进行放大，小于 0 时进行缩小。

rotate () 方法的调用格式如下：

```
cxt.rotate(angle)
```

该方法的功能是实现对已绘制图形的旋转操作，其中，cxt 为上下文环境对象名称，参数 angle 表示图形旋转的角度，大于 0 时顺时针旋转，小于 0 时逆时针旋转。

下面通过实例 7-7 介绍使用 <canvas> 元素，实现对图形进行移动、缩放、旋转操作的过程。

实例 7-7 使用 <canvas> 元素移动、缩放、旋转图形

1. 功能描述

在页面中，新建一个 <canvas> 元素，页面被加载时，在画布中绘制一个正方形。另外，

创建三个 标记，将内容分别设置为“移动”、“缩放”、“旋转”；当单击某个 标记时，对画布中已绘制的正方形进行相应的操作。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-7.html，加入代码如代码清单 7-7 所示。

代码清单 7-7　使用 <canvas> 元素移动、缩放、旋转图形

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>移动、缩放、旋转绘制的图形 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js7.js"/>
</script>
</head>
<body onLoad="drawRect();">
    <div><p>
     <span onClick="spn1_click();">移动 </span>
     <span onClick="spn2_click();">缩放 </span>
     <span onClick="spn3_click();">旋转 </span></p>
     <canvas id="cnvMain" width="280px" height="190px"></canvas>
    <div>
</body>
</html>
```

在代码清单 7-7 中，页面导入一个 JavaScript 文件 js7.js，其中自定义了多个函数，分别在加载页面与单击各个 元素时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
//绘制一个正方形
function drawRect(){
    var cnv=$$("cnvMain");
    var cxt=cnv.getContext("2d");
    //设置边框
    cxt.strokeStyle="#666";
    cxt.lineWidth=2;
    cxt.strokeRect(105,70,60,60);
}
//上下移动已绘制的正方形
function spn1_click(){
    var cnv=$$("cnvMain");
    var cxt=cnv.getContext("2d");
```

```
    cxt.translate(-20,-20);
    drawRect();
    cxt.translate(40,40);
    drawRect();
}
// 缩放已绘制的正方形
function spn2_click(){
    var cnv=$$("cnvMain");
    var cxt=cnv.getContext("2d");
    cxt.scale(1.2,1.2);
    drawRect();
    cxt.scale(1.2,1.2);
    drawRect();
}
// 旋转已绘制的正方形
function spn3_click(){
    var cnv=$$("cnvMain");
    var cxt=cnv.getContext("2d");
    cxt.rotate(Math.PI/8);
    drawRect();
    cxt.rotate(-Math.PI/4);
    drawRect();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-9 所示。

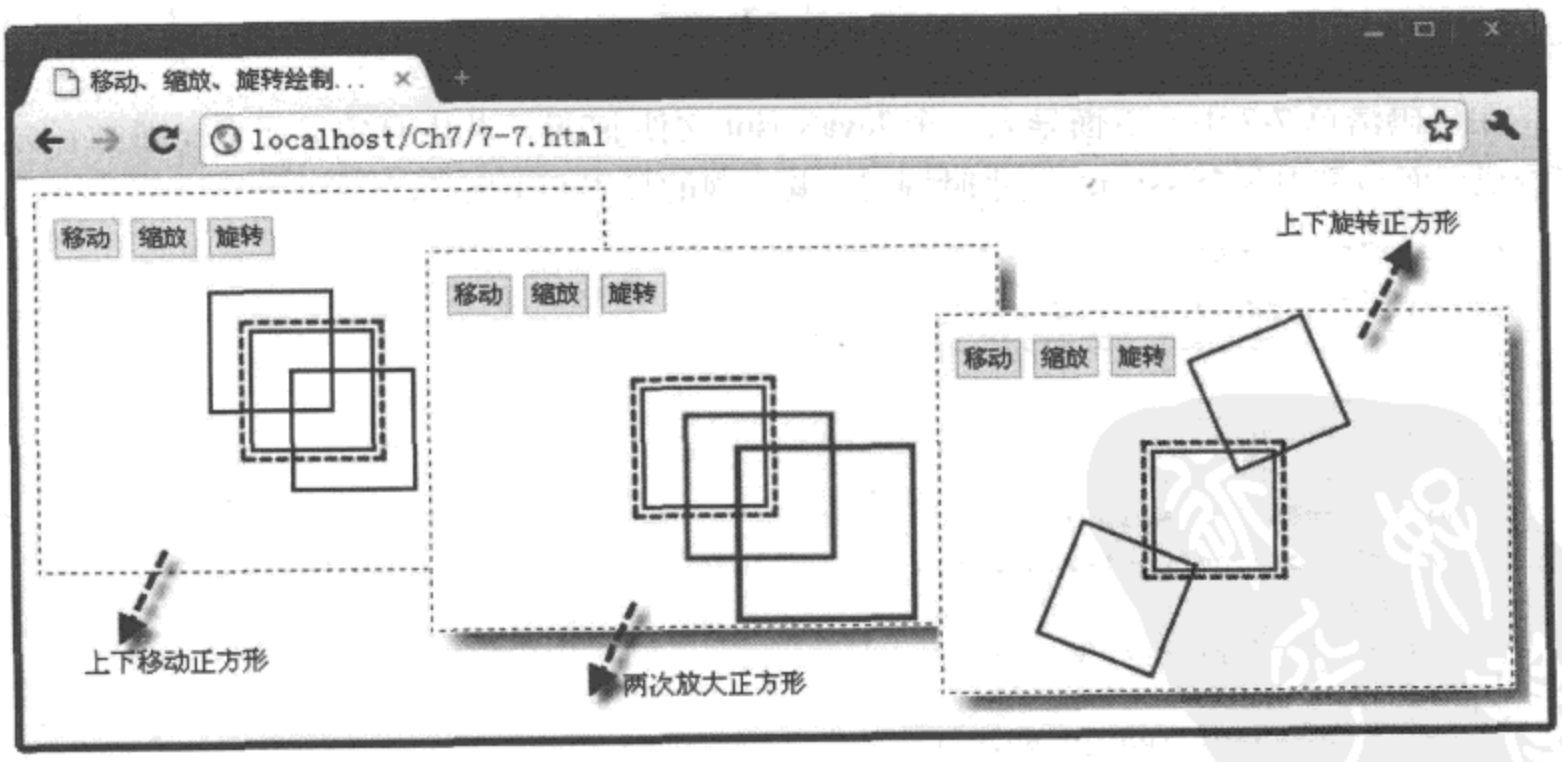


图 7-9　使用 <canvas> 元素移动、缩放、旋转图形

4. 源码分析

在本实例中，页面在触发加载事件时，调用了一个自定义函数 drawRect()。该函数的功能

是在画布中绘制一个正方形，即图 7-9 中虚线所示部分，所有对图形的移动、放大、旋转操作都是在该原型图的基础上进行的。

单击“移动”标签时，首先调用 translate() 方法，将原型图向上、向左移动 20 像素距离，并调用函数 drawRect() 对平移后的图形进行绘制；然后，再次调用 translate() 方法，将平移后的图形向下、向右移动 40 像素距离，并再次调用函数 drawRect() 对第二次平移后的图形进行绘制，实现效果如图 7-9 中的第一幅图所示。

单击“缩放”标签时，实现过程与单击“移动”标记时类似，只是调用两次 scale() 方法，分别对原型图与第一次放大图形进行放大 1.2 倍的操作，实现效果如图 7-9 中的第二幅图所示。

单击“旋转”标签时，调用了两次 rotate() 方法，第一次是对原型图按顺时针旋转 Math.PI/8 度，第二次是在已旋转图形的基础上，按逆时针再次旋转 Math.PI/4 度，实现效果如图 7-9 中的第三幅图所示。

注意　只要是针对图形的操作，无论是移动、缩放、旋转，每次操作后，都要调用函数 drawRect() 重新绘制新的图形，以确保操作后的效果显示在画布中，详细实现过程如代码中加粗部分所示。

7.3.2　组合多个图形

如果在画布中绘制多个有交叉点的图形，将根据绘制时的先后顺序显示每个图形，在交叉之处，新绘制的图形将会覆盖原图形。但有时候，这样的显示效果并不尽如人意。如果想要改变这种默认多图组合的显示形式，可以通过修改上下文环境对象的“globalCompositeOperation”属性值来实现，它有多个属性值，如表 7-1 所示。

表 7-1　globalCompositeOperation 的属性值及描述

属性值	描　述
source-over	显示图形时，新绘制的图形覆盖原先绘制的图形，这是默认值
copy	只显示新图形，其他部分作透明处理
darker	两种图形都显示，在图形重叠部分，颜色由两个图形的颜色值相减后形成
destination-atop	只显示新图形中与原图形重叠部分及新图形的其余部分，其他部分作透明处理
destination-in	只显示原图形中与新图形重叠部分，其他部分作透明处理
destination-out	只显示原图形中与新图形不重叠部分，其他部分作透明处理
destination-over	与 source-over 属性相反，原先绘制的图形覆盖新绘制的图形
lighter	两种图形都显示，在图形重叠部分，颜色由两个图形的颜色值相加后形成
source-atop	只显示原图形中与新图形重叠部分及原图形的其余部分，其他部分作透明处理
source-in	只显示新图形中与原图形重叠部分，其他部分作透明处理
source-out	只显示新图形中与原图形不重叠部分，其他部分作透明处理
xor	两种图形都绘制，并透明处理图形重叠部分

其中，source 表示新图形资源，destination 表示原图形资源。

下面通过实例 7-8 介绍使用 <canvas> 元素实现多图形组合显示的过程。

实例 7-8　使用 <canvas> 元素设置多图形组合显示的方式

1. 功能描述

在页面中，新建一个 <canvas> 元素，当页面被加载时，调用一个自定义的函数 pageload()，通过该函数创建一个正方形和圆形，将两个图形组合后的“globalComposite Operation”的属性值设为“lighter”，并将组合后的图形结果显示在画布中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-8.html，加入代码如代码清单 7-8 所示。

代码清单 7-8　使用 <canvas> 元素设置多图形组合显示的方式

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>设置多图形组合显示的方式</title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js8.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-8 中，页面导入一个 JavaScript 文件 js8.js，其中自定义了多个函数，分别在加载页面与绘制不同图形时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function pageload(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   drawRect(cxt);
   cxt.globalCompositeOperation="lighter";
   drawCirc(cxt);
}
// 绘制一个正方形
function drawRect(cxt){
```

```
    cxt.fillStyle="#666";
    cxt.fillRect(60,50,80,80);
}
// 绘制一个圆形
function drawCirc(cxt){
    cxt.beginPath()
    cxt.arc(130,120,50,0,Math.PI*2,true);
    cxt.closePath()
    cxt.fillStyle="#ccc";
    cxt.fill();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 7-10 所示。

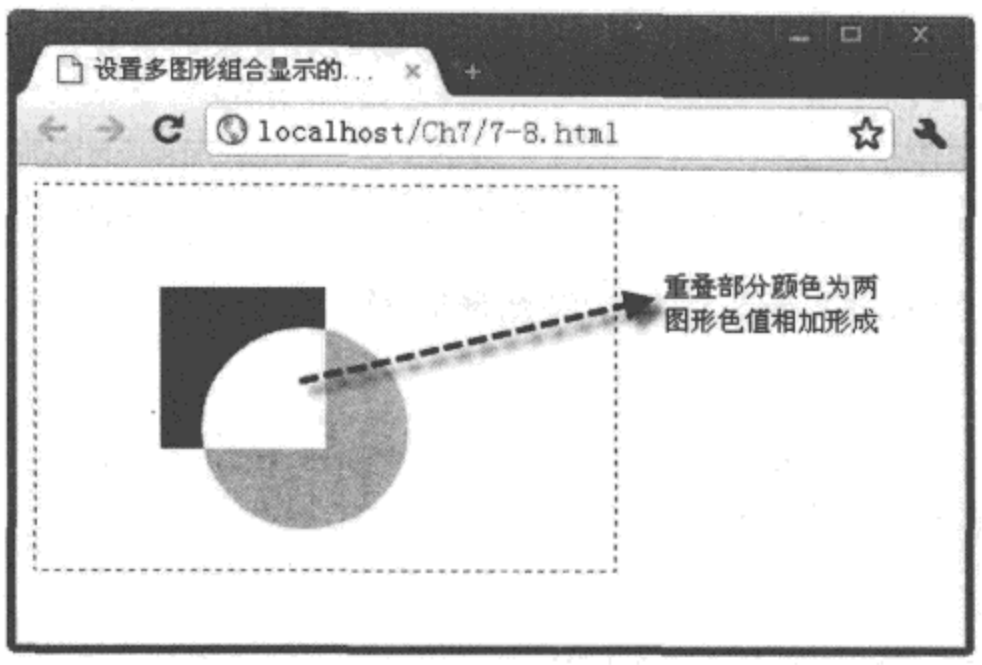


图 7-10　使用 <canvas> 元素设置多图形组合显示方式

4. 源码分析

在本实例的 JavaScript 代码中，先自定义两个函数 drawRect() 与 drawCirc()，分别用于根据传入的上下文环境参数值，绘制正方形与圆形。

当页面被加载时，将触发页面的 onLoad 事件，在该事件中调用另一个自定义函数 pageload()。在函数 pageload() 中，首先通过 ID 号获取画布元素 <canvas>，并根据画布元素取得上下文环境对象 cxt; 然后传递 cxt 对象，调用函数 drawRect() 在画布中先绘制一个正方形；接下来，设置“globalCompositeOperation”属性值为“lighter”，表明与下面图形组合时的显示方式；最后，调用 drawCirc () 函数在画布中绘制一个圆形，两个图形的重叠部分将按照设置的“globalCompositeOperation”属性值进行组合显示，详细实现过程如代码中加粗部分所示。

7.3.3 添加图形阴影

在使用画布元素 <canvas> 绘制图形时，可以为图形添加背景阴影，以达到立体显示的效果。为了实现这一功能，需要对上下文环境对象的阴影属性进行设置，相关阴影属性如表 7-2 所示。

表 7-2 设置图形阴影的相关属性及描述

属性名称	描　述
shadowOffsetX	阴影与图形的水平距离，大于 0 时向右偏移，小于 0 时向左偏移，默认值为 0
shadowOffsetY	阴影与图形的垂直距离，大于 0 时向下偏移，小于 0 时向上偏移，默认值为 0
shadowColor	阴影的颜色值，例如设置为“blue”、“#eee”、“rgb（0,0,0）”
shadowBlur	阴影的模糊值，该值越大，模糊度越强；反之，模糊度越弱，默认值为 1

说明 shadowBlur 模糊属性为可选择项，该项不影响阴影效果的实现，如果需要设置，取值通常在 1 ～ 10 之间比较适宜。

下面通过实例 7-9 介绍使用 <canvas> 元素为绘制图形添加阴影效果的过程。

实例 7-9 使用 <canvas> 元素添加绘制图形阴影

1. 功能描述

在页面中，新建一个 <canvas> 元素，绘制两个相同形状和大小的长方形，分别为两个图形添加不同方向的阴影效果，最后将两个图形同时显示在画布元素中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-9.html，加入代码如代码清单 7-9 所示。

代码清单 7-9 使用 <canvas> 元素添加绘制图形阴影

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>添加绘制图形阴影的效果</title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js9.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-9 中，页面导入一个 JavaScript 文件 js9.js，其中自定义一个实现添加绘制图形不同角度阴影效果的函数，该函数在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
function pageload(){
   var cnv=$$("cnvMain");
   var cxt=cnv.getContext("2d");
   //设置左上方向的阴影
   cxt.shadowOffsetX=-6;
   cxt.shadowOffsetY=-6;
   cxt.shadowColor="#ccc";
   cxt.shadowBlur=1;
   cxt.fillStyle="#eee";
   cxt.fillRect(20,28,100,130);
   //设置右下方向的阴影
   cxt.shadowOffsetX=6;
   cxt.shadowOffsetY=6;
   cxt.shadowColor="#ccc";
   cxt.shadowBlur=10;
   cxt.fillStyle="#eee";
   cxt.fillRect(150,28,100,130);
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-11 所示。

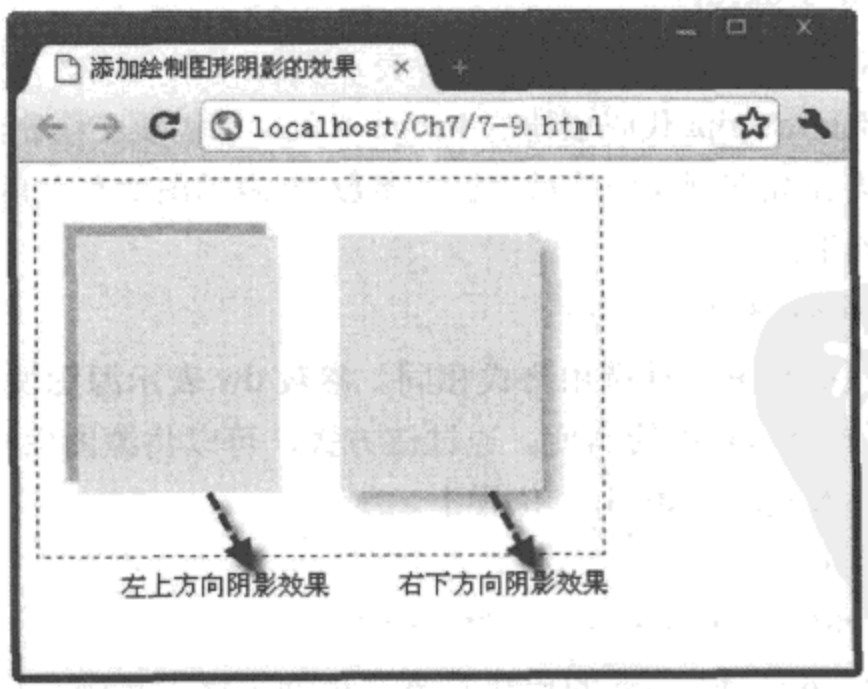


图 7-11　使用 <canvas> 元素为图形添加不同方向的阴影

4. 源码分析

在本实例的 JavaScript 代码中，首先，获取上下文环境对象 cxt 后，开始设置绘制图

形的阴影属性。为了设置第一个长方形的左上方向阴影效果，需要将“shadowOffsetX”与“shadowOffsetY”属性的值设置为负数；同时，设置阴影颜色“shadowColor”的值，并将阴影模糊度“shadowBlur”的值设置为 1，表示不进行模糊处理；然后，设置绘制图形的填充颜色，并使用 fillRect() 方法绘制一个有阴影效果的长方形，如图 7-11 左边图形所示。

为了与画布中左边图形产生对比效果，用同样的方式，在右边绘制另外一个相同的长方形，只是阴影的方向发生了变化。将“shadowOffsetX”与“shadowOffsetY”属性的值设置为正数，并将阴影模糊度“shadowBlur”的值设置为 10，表示需要轻度模糊，如图 7-11 右边图形所示。通过两个图形的对比，使用阴影参数的效果更加清晰。

7.4 处理画布中的图像

在画布元素 <canvas> 中，不仅可以绘制各种形状的图形，还可以通过 Canvas API 中的方法将磁盘或网络中的图像导入画布中，并且对导入的图片进行平铺、切割、像素处理等多项操作，接下来逐一进行详细的介绍。

7.4.1 绘制图像

在画布元素 <canvas> 中绘制图像需要调用上下文环境对象中的 drawImage() 方法，通过该方法可以将页面中存在的 <img> 元素，或者通过 JavaScript 代码创建的 Image 对象绘制在画布中。该方法有三种调用的格式，如下所示：

第一种调用格式如下：

```
cxt.drawImage(image,dx,dy)
```

其中，cxt 为上下文环境对象名称，参数 image 表示页面中的图像。无论该图像是页面中的 <img> 元素，还是编写 JavaScript 代码创建的 Image 对象，都需要在绘制图像前，完全加载该图像。参数 dx 表示图像左上角在画布中的横坐标，参数 dy 表示图像左上角在画布中的纵坐标。

第二种调用格式如下：

```
cxt.drawImage(image,dx,dy,dw,dh)
```

其中，前三个参数的用法与第一种调用格式相同。参数 dw 表示源图像缩放至画布中的宽度，参数 dh 表示源图像缩放至画布中的高度。通过该方式，可以将源图像按指定缩放的大小绘制在画布元素 <canvas> 的坐标为（dx,dy）位置上。

第三种调用格式如下：

```
cxt.drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh)
```

其中，参数 image，dx，dy，dw，dh 的用法与第二种调用格式相同。参数 sx，sy，sw，sh 表示源图像需要裁剪的范围，参数 sx 表示源图像被绘制部分的横坐标，参数 sy 表示源图像被绘制部分的纵坐标，参数 sw 表示源图像被绘制部分的宽度，参数 sh 表示源图像被绘制部分的

高度。通过该方式，可以将源图像指定的范围以映射的方式，绘制到画布中的指定区域，实现的流程如图 7-12 所示。

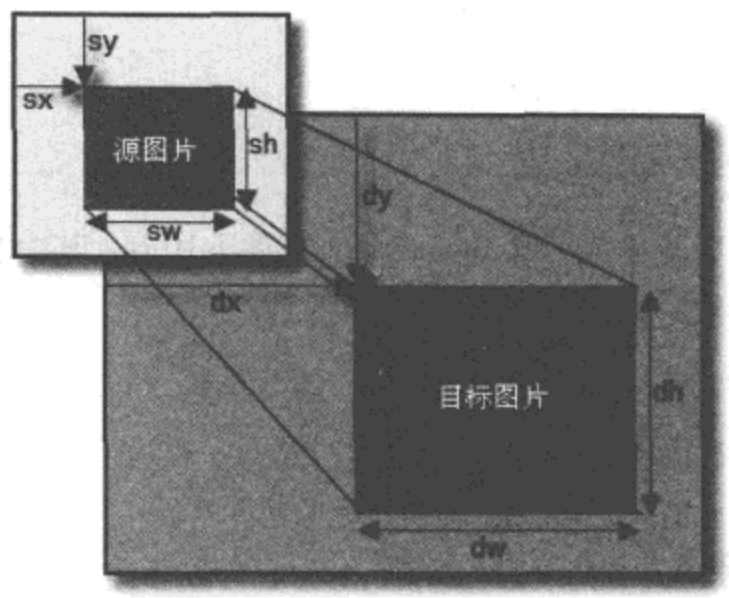


图 7-12　使用 drawImage 方法映射图像的流程

使用映射方法将源图像绘制到画布元素 <canvas> 中，可以在画布中将裁剪的源图像中的图形进行缩放，从而实现局部放大的效果。

下面通过实例 7-10 介绍在 <canvas> 元素中使用 drawImage() 方法绘制图像的过程。

实例 7-10　使用 drawImage 方法在画布中绘制图像

1. 功能描述

在页面中，新建一个 <canvas> 元素。第一次单击画布时，使用第一种调用格式在画布中绘制一个指定位置的图像；第二次单击画布时，使用第二种调用格式在画布中绘制一个指定位置与大小的图像；第三次单击画布时，使用第三种调用格式以映射的方式将源图像中的部分图像以放大方式绘制在画布中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-10.html，加入代码如代码清单 7-10 所示。

代码清单 7-10　使用 drawImage 方法在画布中绘制图像

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 drawImage 方法在画布中绘制图像 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js10.js"/>
</script>
```

```
</head>
<body>
    <canvas id="cnvMain" width="280px" height="190px"
            onClick="cnvclick(this);">
    </canvas>
</body>
</html>
```

在代码清单 7-10 中，页面导入一个 JavaScript 文件 js10.js，其中自定义一个函数 cnvclick()，该函数在单击画布时被调用。其实现的代码如下所示：

```
// JavaScript Document
// 定义保存单击次数的全局变量
var intNum = 0;
// 自定义画布单击函数
function cnvclick(cnv) {
    intNum += 1;
    intNum = (intNum == 4) ? 1 : intNum;
    var cxt = cnv.getContext("2d");
    cxt.clearRect(0, 0, cnv.width, cnv.height);
    var objImg = new Image();
    objImg.src = "Images/2010 年作品 .jpg";
    objImg.onload = function() {
        switch (intNum) {
        case 1:
            cxt.drawImage(objImg, 10, 10);
            break;
        case 2:
            cxt.drawImage(objImg, 10, 10, 94, 123);
            break;
        case 3:
            cxt.drawImage(objImg, 10, 10, 94, 123);
            cxt.drawImage(objImg, 45, 50, 100, 150, 110, 10, 160, 180);
            break;
        }
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-13 所示。

4. 源码分析

在本实例的 JavaScript 代码中，为了保存每个画布元素被单击的次数，首先定义了一个全局变量 intNum，并赋初始值为 0。当画布被单击时，触发 onClick 事件，在该事件中，调用自定义函数 cnvclick()，该函数先累加变量 intNum 的值；如果单击次数变量 intNum 大于 4，则修改为 1，实现单击画布时功能的反复操作。

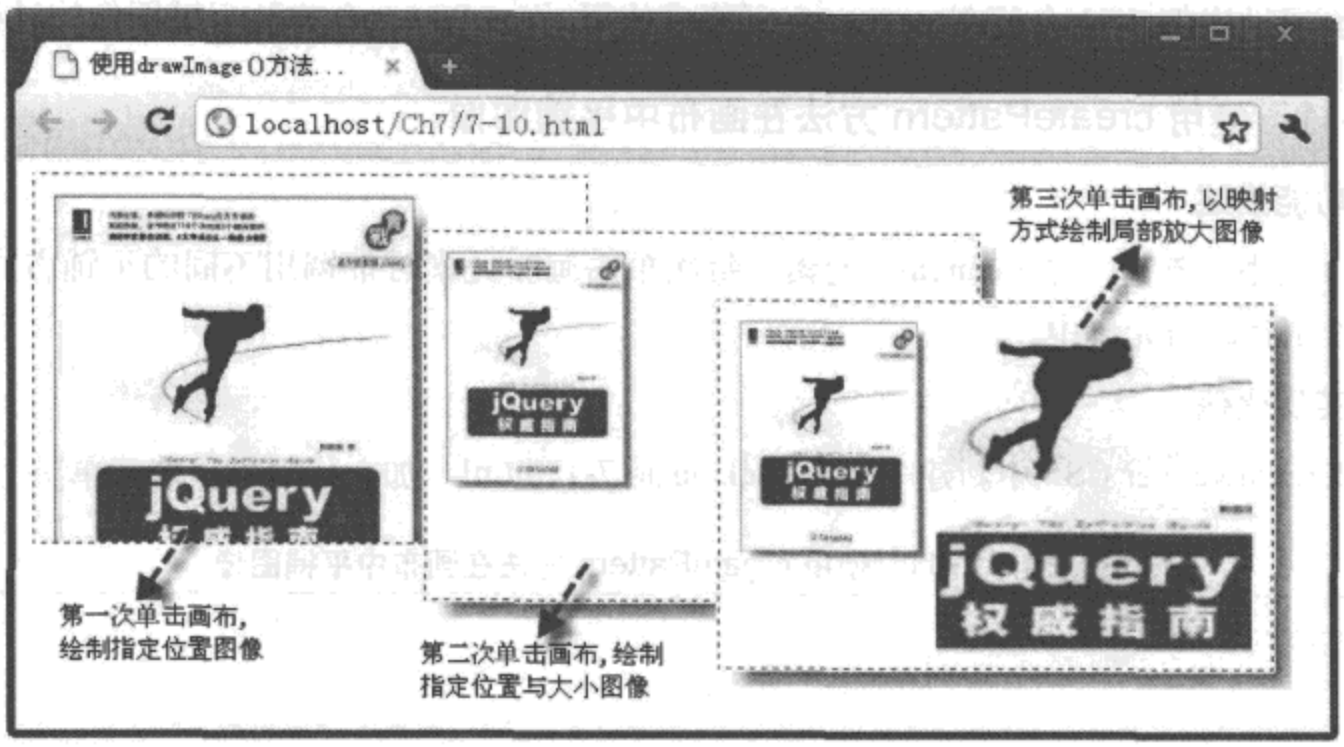


图 7-13　使用 drawImage 方法在画布中绘制图像

使用 clearRect() 方法清空每次在画布中的图形，再定义一个 Image 对象，并通过 src() 方法设置一个加载图片的路径。

为了使加载的图片可以在画布中顺利绘制，在画布加载事件 onload 中，根据累加变量 intNum 的值，调用 drawImage() 方法的不同使用格式，分别实现在画布中指定位置与大小、映射放大方式绘制图片的功能，详细实现过程如代码中加粗部分所示。

7.4.2　平铺图像

在画布中，除对绘制的图像进行缩放绘制外，还可以通过调用上下文环境对象中的 createPattern() 方法关联图像元素。选择平铺方式，创建一个平铺的对象，并将该平铺对象赋值给“fillStyle”属性。通过调用 fillRect() 方法，将该平铺对象绘制在画布中，从而实现平铺图像的效果。createPattern() 方法的调用格式如下：

```
cxt.createPattern(image,type)
```

其中，cxt 为上下文环境对象名称，参数 image 表示被平铺的图像，参数 type 表示图像平铺的方式，该参数有 4 种取值，如表 7-3 所示。

表 7-3　createPattern 方法中参数 type 的取值与描述

名　称	描　述
no-repeat	不平铺绘制的图像
repeat-x	按水平方向横向平铺所绘制的图像
repeat-y	按垂直方向纵向平铺所绘制的图像
repeat	全方位平铺所绘制的图像

下面通过实例 7-11 介绍在 <canvas> 元素中使用 createPattern() 方法平铺图像的过程。

实例 7-11 使用 createPattern 方法在画布中平铺图像

1. 功能描述

在页面中，新建一个 <canvas> 元素，每次单击画布元素时都调用不同的平铺方式，将图像显示绘制在画布元素中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-11.html，加入代码如代码清单 7-11 所示。

代码清单 7-11 使用 createPattern 方法在画布中平铺图像

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 createPattern 方法在画布中平铺图像 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js11.js"/>
</script>
</head>
<body>
    <canvas id="cnvMain" width="280px" height="190px"
            onClick="cnvclick(this);">
    </canvas>
</body>
</html>
```

在代码清单 7-11 中，页面导入一个 JavaScript 文件 js11.js，其中自定义一个函数 cnvclick()，该函数在单击画布时被调用。其实现的代码如下所示：

```
// JavaScript Document
// 定义保存单击次数的全局变量
var intNum = 0;
// 自定义单击画布函数
function cnvclick(cnv) {
    intNum += 1;
    intNum = (intNum == 5) ? 1 : intNum;
    var strPrnType = "";
    switch (intNum) {
    case 1:
        strPrnType = "no-repeat";
        break;
    case 2:
        strPrnType = "repeat-x";
```

```
            break;
        case 3:
            strPrnType = "repeat-y";
            break;
        case 4:
            strPrnType = "repeat";
            break;
        }
        var cxt = cnv.getContext("2d");
        cxt.clearRect(0, 0, cnv.width, cnv.height);
        var objImg = new Image();
        objImg.src = "Images/2010年作品_s.jpg";
        var prn = cxt.createPattern(objImg, strPrnType);
        objImg.onload = function() {
            cxt.fillStyle = prn;
            cxt.fillRect(0, 0, cnv.width, cnv.height);
        }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-14 所示。

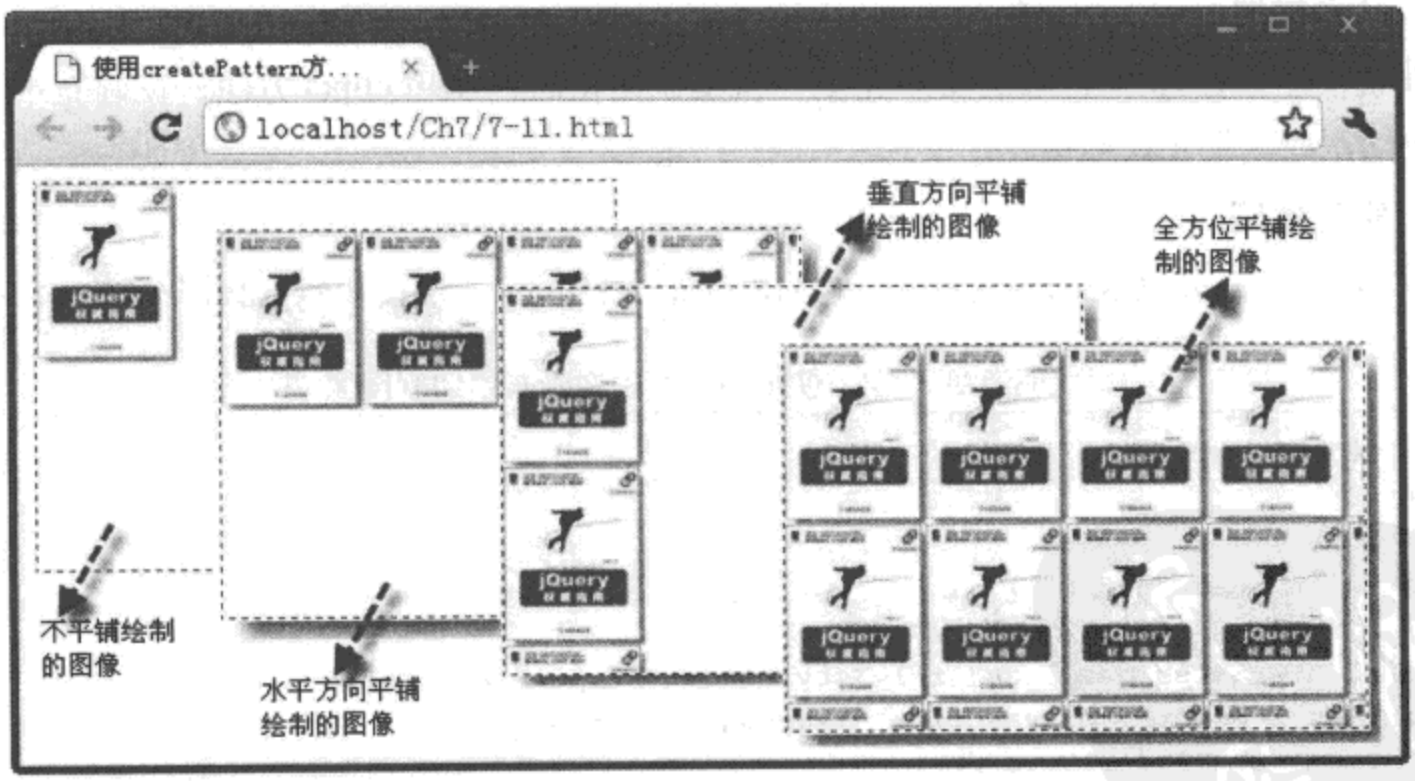


图 7-14　使用 createPattern 方法在画布中平铺图像

4. 源码分析

在本实例的 JavaScript 代码中，首先，根据单击画布的累加总量 intNum 的值，获取图像在画布中的平铺方式，并保存至变量 strPrnType 中。

使用 clearRect() 方法清空每次在画布中绘制的图形，并定义一个 Image 对象，设置该对

象加载图像的路径；再根据该图像与平铺方式变量 strPrnType 的值，新建一个平铺对象 prn，用于图像加载时使用。

在加载图像的 onload 事件中，将 prn 平铺对象赋值给“fillStyle”属性，通过调用 fillRect() 方法，将平铺对象绘制在整个画布中，“cnv.width”与“cnv.height”值分别为画布的宽与高，详细实现过程如代码中加粗部分所示。

7.4.3 切割图像

在画布元素 <canvas> 中，不仅能以各种方式平铺绘制的图像，还可以通过调用上下文环境对象中的 clip() 方法切割画布中绘制的图像。clip() 方法的调用格式如下：

```
cxt.clip()
```

其中，cxt 为上下文环境对象名称，该方法是一个无参数方法，用于切割使用路径方式在画布中绘制的区域。因此，在使用该方法前，必须使用路径的方式在画布中绘制一个区域，然后才能通过调用 clip() 方法对该区域进行切割。

下面通过实例 7-12 介绍在 <canvas> 元素中使用 clip() 方法切割图像的过程。

实例 7-12 使用 clip 方法在画布中切割图像

1. 功能描述

在页面中，新建一个 <canvas> 元素，当绘制图像时，调用 clip() 方法，在画布中将绘制的图像切割成一个光盘形状。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-12.html，加入代码如代码清单 7-12 所示。

代码清单 7-12 使用 clip 方法在画布中切割图像

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 clip 方法在画布中切割图像 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js12.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-12 中，页面导入一个 JavaScript 文件 js12.js，其中自定义一个函数 pageload()，当加载页面时调用该函数。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
//加载自定义页面时调用的函数
function pageload() {
    var cnv = $$("cnvMain");
    var cxt = cnv.getContext("2d");
    var objImg = new Image();
    objImg.src = "Images/2010年作品_m.jpg";
    objImg.onload = function() {
        drawCirc(cxt, 60, true);
        cxt.drawImage(objImg, 70, 3);
        drawCirc(cxt, 10, false);
    }
}
//根据相关参数绘制圆
function drawCirc(cxt, intR, blnC) {
    cxt.beginPath();
    cxt.arc(140, 95, intR, 0, Math.PI * 2, true);
    cxt.closePath();
    //设置边框颜色
    cxt.strokeStyle = "#666";
    //设置边框宽度
    cxt.lineWidth = 3;
    //开始描边
    cxt.stroke();
    if (blnC) {
        //切割图形
        cxt.clip();
    } else {
        //设置填充色
        cxt.fillStyle = "#fff";
        //填充图形
        cxt.fill();
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-15 所示。

4. 源码分析

在本实例的 JavaScript 代码中，当加载页面时触发 onLoad() 事件，在该事件中，调用了一个自定义的函数 pageload()。在该函数中，首先创建一个 Image 对象，并设置该对象加载图

像的路径。在加载图像过程中，第一次调用另外一个自定义函数 drawCirc()，绘制一个圆形路径，并使用 stroke() 方法将路径绘制在画布中；同时，调用 clip() 方法将画布中的圆路径进行切割。其中，在调用函数 drawCirc() 时，参数 cxt 表示上下文环境对象名称，intR 表示圆半径，blnC 表示是否需要对绘制图形进行切割，true 表示需要，false 表示不需要。

图 7-15　使用 clip 方法在画布中切割图像的效果

然后，使用 drawImage() 方法，在画布中绘制一个左上角坐标为（70，3）的图像。由于绘制图像前，画布已按照圆路径进行了切割，因此，加载的图像也按照该切割后的圆形区域进行绘制。

最后，第二次调用自定义函数 drawCirc()，绘制一个与第一个圆路径同圆心不同半径的小圆形，并设置“fillStyle”的属性值为“#fff”，通过 fill() 方法进行填充，形成光盘中大圆中的小圆部分，详细实现过程如代码中加粗部分所示。

7.4.4　处理像素

在 HTML 5 中调用 Canvas API 中的方法，还可以处理画布中所绘制图像的像素，例如做成蒙版效果、黑白色效果等。要实现这样的页面效果，需要在加载图像时调用上下文环境对象的两个方法：一个是 getImageData() 方法，用于获取图像中的像素；另一个是 putImageData() 方法，用于将处理后的像素重新绘制画布中。两种方法调用的格式分别如下所示。

getImageData() 方法的调用格式：

```
cxt.getImageData(sx,sy,sw,sh)
```

其中，cxt 为上下文环境对象，参数 sx 为所选图像区域的横坐标，参数 sy 为所选图像区域的纵坐标，参数 sw 为所选图像区域的宽度，参数 sh 为所选图像区域的高度。获取的像素区域可以通过一个变量保存，例如变量 objImgData，该变量是一个 CanvasPixelArray 对象，其中的 data 属性是一个用于保存像素数据的数组，取值如“[r1，g1，b1，a1，r2，g2，b2，a2，…]”，因此，objImgData.data.length 为获取的像素总量，在遍历时可以使用该值。

putImageData() 方法的调用格式：

```
cxt.putImageData(image,dx,dy)
```

其中，cxt 为上下文环境对象名称，参数 image 表示重新绘制的图像，参数 dx 表示重新绘制图像在画布左上角的横坐标，参数 dy 表示重新绘制图像在画布左上角的纵坐标。

下面通过实例 7-13 介绍在 <canvas> 元素中使用 getImageData() 与 putImageData() 方法处理图像像素的过程。

实例 7-13　使用 getImageData 与 putImageData 方法处理图像像素

1. 功能描述

在页面中，新建一个 <canvas> 元素。第一次单击该元素时，在画布中绘制一幅图像，再次单击时处理该图像的像素，以反色彩的形式在画布中重新绘制图像。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-13.html，加入代码如代码清单 7-13 所示。

代码清单 7-13　使用 getImageData 与 putImageData 方法处理图像像素

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 getImageData 与 putImageData 方法处理图像像素 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js13.js"/>
</script>
</head>
<body>
    <canvas id="cnvMain" width="280px" height="190px"
            onClick="cnvclick(this);">
    </canvas>
</body>
</html>
```

在代码清单 7-13 中，页面导入一个 JavaScript 文件 js13.js，其中自定义一个函数 cnvclick()，该函数在单击画布时被调用。其实现的代码如下所示：

```
// JavaScript Document
//定义保存单击次数的全局变量
var intNum = 0;
//自定义单击画布函数
function cnvclick(cnv) {
    intNum += 1;
```

```
        intNum = (intNum == 3) ? 1 : intNum;
        var cxt = cnv.getContext("2d");
        var objImg = new Image();
        objImg.src = "Images/2010年作品_1.jpg";
        var intX=cnv.width/2-objImg.width/2;
        var intY=cnv.height/2-objImg.height/2;
        objImg.onload = function() {
            switch (intNum) {
            case 1:
                cxt.drawImage(objImg, intX,intY);
                break;
            case 2:
                dealPixel(cxt,objImg,intX,intY);
                break;
            }
        }
    }
    //根据相关参数处理绘制图形像素
    function dealPixel(cxt,objImg,intX,intY) {
        var ImgData=cxt.getImageData(intX,intY,objImg.width,objImg.height);
        for(var intI=0;intI<ImgData.data.length;intI+=4){
               ImgData.data[intI+0]=255-ImgData.data[intI+0];
               ImgData.data[intI+1]=255-ImgData.data[intI+2];
               ImgData.data[intI+2]=255-ImgData.data[intI+1]
       }
       cxt.putImageData(ImgData,intX,intY);
    }
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-16 所示。

图 7-16　使用 getImageData 与 putImageData 方法处理图像像素

4. 源码分析

在本实例的 JavaScript 代码中，当画布被单击时触发 onClick() 事件，在该事件中，调用自定义的函数 cnvclick()。在该函数中，首先累计画布被单击的次数，将结果保存至变量 intNum 中；另外，创建一个 Image 对象，并设置对象加载路径。然后，分别计算 cnv.width/2-objImg.width/2 与 cnv.height/2-objImg.height/2 的值，用于获取绘制图像居中时左上角的坐标位置，并将计算后的结束值保存至变量 intX 与 intY 中。

在图像加载过程中，第一次单击画布时，调用 drawImage() 方法，在画布中绘制一个居中的图像。第二次单击画布时，调用自定义的函数 dealPixel()，该函数先通过变量 ImgData 获取绘制图像的全部像素区域，然后，按步长为 4 的方式遍历 ImgData 对象中的每个像素值，将每个像素位设置为 255 减去当前的值，从而获取反向的色值。完成遍历后，使用 putImageData() 方法将处理后的图像像素重新绘制到画布中的指定位置，详细实现过程如代码中加粗部分所示。

7.5　画布的其他应用

在 HTML 5 中，通过画布元素 <canvas> 不仅可以绘制图形、图像，还可以调用 Canvas API 中的相关方法绘制文字；同时，还能将绘制的图形保存、还原、以 base64 位的 URL 形式在浏览器中输出，以及在画布中制作简单的动画。下面逐一进行详细介绍。

7.5.1　绘制文字

要在画布中绘制文字，可以通过调用上下文环境对象的 fillText() 与 strokeText() 方法，前者用于在画布中以填充的方式绘制文字，后者用于在画布中以描边的方式绘制文字。两者的调用格式分别如下所示。

fillText() 方法的调用格式：

```
cxt.fillText(content,dx,dy,[maxLength])
```

其中，cxt 为上下文对象名称，参数 content 为文字内容，参数 dx 表示绘制文字在画布左上角的横坐标，参数 dy 表示绘制文字在画布左上角的纵坐标，可选项参数 maxLength 表示绘制文字显示的最大长度，设置后，在该长度值范围内绘制文字。

strokeText() 方法的调用格式：

```
cxt.strokeText(content,dx,dy,[maxWidth])
```

参数说明与调用 fillText() 一样，不再赘述。

在画布中绘制文字时，除调用以上两种方法外，还要设置相关的属性，其中与绘制文字相关的属性如表 7-4 所示。

表 7-4　与绘制文字相关的属性

属　性	描　述
font	设置 CSS 样式中字体的任何值，如字体样式、名称、大小、粗细、行距等
textAlign	设置文本对齐的方式，取值为 start、end、left、right、center
textBaseline	设置文本相对于起点的位置，取值为 top、bottom、middle

下面通过实例 7-14 介绍在 <canvas> 元素中使用 fillText() 与 strokeText() 方法绘制文字的过程。

实例 7-14　使用 fillText 与 strokeText 方法绘制文字

1. 功能描述

在页面中，新建一个 <canvas> 元素。当页面被加载时，设置三种不同字体的文字，分别绘制在画布元素中的不同坐标位置上。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-14.html，加入代码如代码清单 7-14 所示。

代码清单 7-14　使用 fillText 与 strokeText 方法绘制文字

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 fillText 与 strokeText 方法绘制文字 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js14.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-14 中，页面导入一个 JavaScript 文件 js14.js，其中自定义一个函数 pageload()，该函数在加载页面时被调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义加载页面时调用的函数
function pageload() {
    var cnv = $$("cnvMain");
```

```
    var cxt = cnv.getContext("2d");
    drawText(cxt, "bold 35px impact", 90, 70, false);
    drawText(cxt, "bold 35px arial blank", 130, 110, true);
    drawText(cxt, "bold 35px comic sans ms", 170, 150, true);
}
//根据参数绘制不同类型的字体
function drawText(cxt, strFont, intX, intY, blnFill) {
    cxt.font = strFont;
    cxt.textAlign = "center";
    cxt.textBaseline = "bottom";
    if (blnFill) {
        cxt.fillStyle = "#ccc";
        cxt.fillText("HTML 5 ", intX, intY);
    } else {
        cxt.strokeStyle = "#666";
        cxt.strokeText("HTML 5 ", intX, intY);
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-17 所示。

图 7-17　使用 fillText 与 strokeText 方法绘制文字

4. 源码分析

在本实例的 JavaScript 代码中，自定义了一个用于加载页面时的函数 pageload()。在该函数中，分别三次调用了另外一个用于绘制文字的函数 drawText()。该函数中有 5 个参数，其中，参数 cxt 表示上下文环境对象名称，参数 strFont 表示设置的 font 属性值，参数 intX 表示字体在画布左上角的横坐标，参数 intY 表示字体在画布左上角的纵坐标，参数 blnFill 表示是否采用 fill() 方法绘制字符，如果为 true 表示是，否则表示使用 stroke() 方法绘制文字。

第一次调用时，使用了“Impact”字体，在画布左上角坐标（90,70）处，采用 stroke() 方法，绘制了内容为“HTML 5”的文字。

第二次调用时，使用了“Arial Blank”字体，在画布左上角坐标（130,110）处，采用 fill() 方法，绘制了内容为“HTML 5”的文字。

第三次调用与第二次调用基本相同，仅是字体名称与在画布中的坐标不同，详细实现过程如代码中加粗部分所示。

7.5.2 保存、恢复及输出图形

在 HTML 5 中，有时需要在画布中绘制多个图形，并在图形之间进行切换，这时，如果不进行保存，那么切换后，原先已绘制的图形将丢失，这是我们不希望看到的。为了解决这一问题，可以通过调用上下文环境对象中的 save() 方法，保存已绘制图形；当需要使用保存的已绘制图形时，再通过调用 restore() 方法还原保存的图形，或者通过调用 toDataURL() 方法将保存的图形输出至浏览器中。以上三种方法的调用格式分别如下所示。

save() 方法的调用格式：

```
cxt.save()
```

restore() 方法的调用格式：

```
cxt.restore()
```

toDataURL() 方法的调用格式：

```
cxt.toDataURL(ImgType)
```

在上述三种方法中，cxt 均表示上下文环境对象名称。另外，前两种为无参数方法，分别实现保存画布中的图形与还原已保存图形的功能。toDataURL() 方法用于实现将画布中的图形以 base64 位编码的方式输出至浏览器中，参数 ImgType 表示输出数据的 MIME 类型，如“image/png”等。

下面通过实例 7-15 介绍在 <canvas> 元素中保存、恢复及输出图形的过程。

实例 7-15 在画布中保存、恢复及输出图形

1. 功能描述

在页面中，新建一个 <canvas> 元素与三个 元素，三个 元素分别设置“保存”、“恢复”、“输出”的内容。

当第一次单击画布时，绘制一个指定填充色与大小的图形，同时，单击“保存”标记，保存第一次单击画布时绘制的图形。

当第二次单击画布时，再次绘制一个与第一次填充色不同的图形，同时，单击“恢复”标记，用于最后一次绘制图形时使用。

当第三次单击画布时，由于还原了第一次单击画布时图形的填充色，因此，在最后一次绘制图形时，将用第一次图形的填充色填充最后一次绘制的图形。

当单击“输出”标记时，将最后绘制在画布中的图形以 base64 位编码的方式输出至浏览器中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-15.html，加入代码如代码清单 7-15 所示。

代码清单 7-15　在画布中保存、恢复及输出图形

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>在画布中保存、恢复及输出图形 </title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js15.js"/>
</script>
</head>
<body>
    <div><p>
     <span onClick="spn1_click();"> 保存 </span>
     <span onClick="spn2_click();"> 恢复 </span>
     <span onClick="spn3_click();"> 输出 </span></p>
     <canvas id="cnvMain" width="280px" height="190px"
             onClick="cnvclick(this);">
     </canvas>
    <div>
</body>
</html>
```

在代码清单 7-15 中，页面导入一个 JavaScript 文件 js15.js，其中自定义多个函数，在单击画布与标记时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 定义保存单击次数的全局变量
var intNum = 0;
// 自定义单击画布函数
function cnvclick(cnv) {
    intNum += 1;
    intNum = (intNum == 4) ? 1 : intNum;
    var cxt = cnv.getContext("2d");
    cxt.clearRect(0, 0, cnv.width, cnv.height);
    var intW = 120;
    var intH = 120;
```

```
        var intX = cnv.width / 2 - intW / 2;
        var intY = cnv.height / 2 - intH / 2;
        switch (intNum) {
        case 1:
            cxt.fillStyle = "#eee";
            cxt.fillRect(intX, intY, intW, intH);
            break;
        case 2:
            cxt.fillStyle = "#ccc";
            cxt.fillRect(intX + 15, intY + 15, intW - 30, intH - 30);
            break;
        case 3:
            cxt.fillRect(intX + 30, intY + 30, intW - 60, intH - 60);
            break;
        }
    }
    // 自定义单击 " 保存 " 标记函数
    function spn1_click() {
        var cnv = $$("cnvMain");
        var cxt = cnv.getContext("2d");
        cxt.save();
    }
    // 自定义单击 " 恢复 " 标记函数
    function spn2_click() {
        var cnv = $$("cnvMain");
        var cxt = cnv.getContext("2d");
        cxt.restore();
    }
    // 自定义单击 " 输出 " 标记函数
    function spn3_click() {
        var cnv = $$("cnvMain");
        var cxt = cnv.getContext("2d");
        window.location = cxt.canvas.toDataURL("image/png");
    }
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-18 所示。

单击页面中的“输出”标记后，将第三次绘制的图形以 base64 编码的方式输出至浏览器中。该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-19 所示。

4. 源码分析

在本实例的 JavaScript 代码中，第一次单击画布时，在画布中绘制一个长宽为 120px、填充色为“#eee”的图形；单击“保存”标记，调用 save() 方法，保存第一次单击画布时绘制的这个图形。第二次单击画布时，在画布中绘制一个长宽为 90px、填充色为“#ccc”的图形；单击“恢复”标记，调用 restore() 方法，恢复已保存的第一次单击画布时绘制的图形。第三次

单击画布时，在画布中绘制一个长宽为 60px、填充色为“#eee”的图形，该填充色与第一次单击画布时绘制的图形填充色一致。

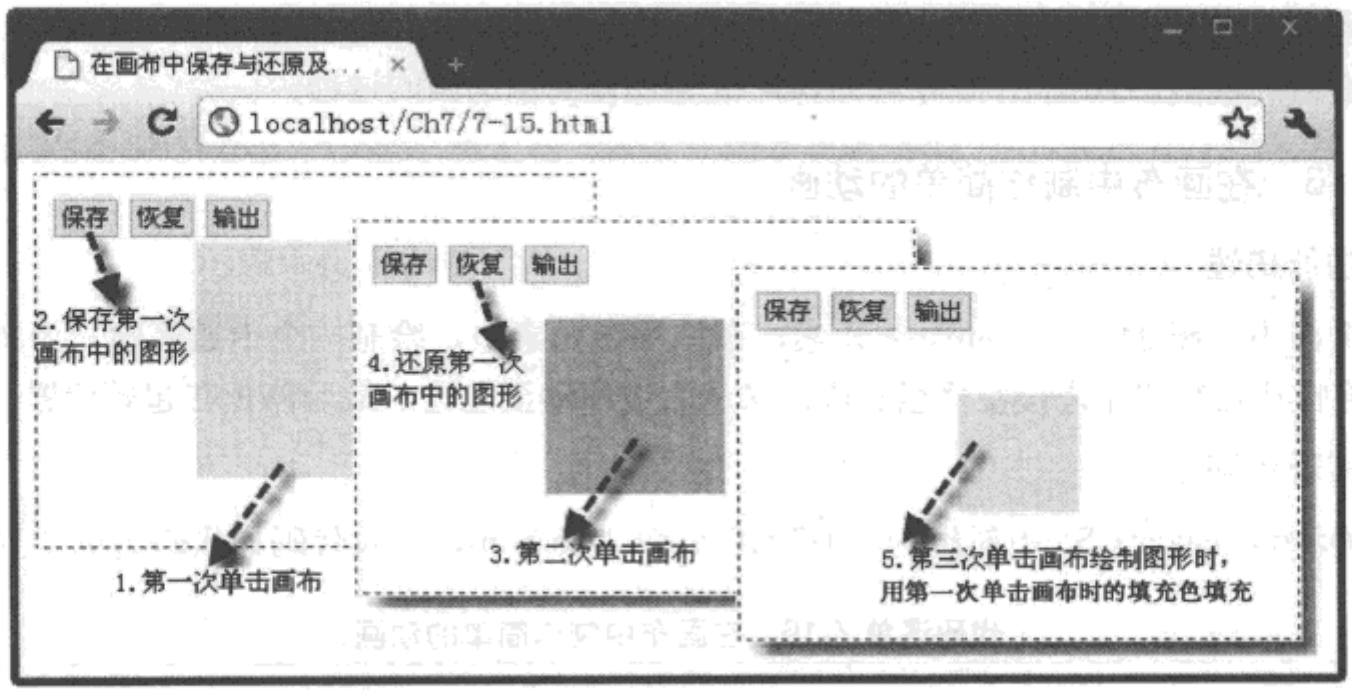


图 7-18　使用 save 与 restore 方法保存、恢复已绘制的图形

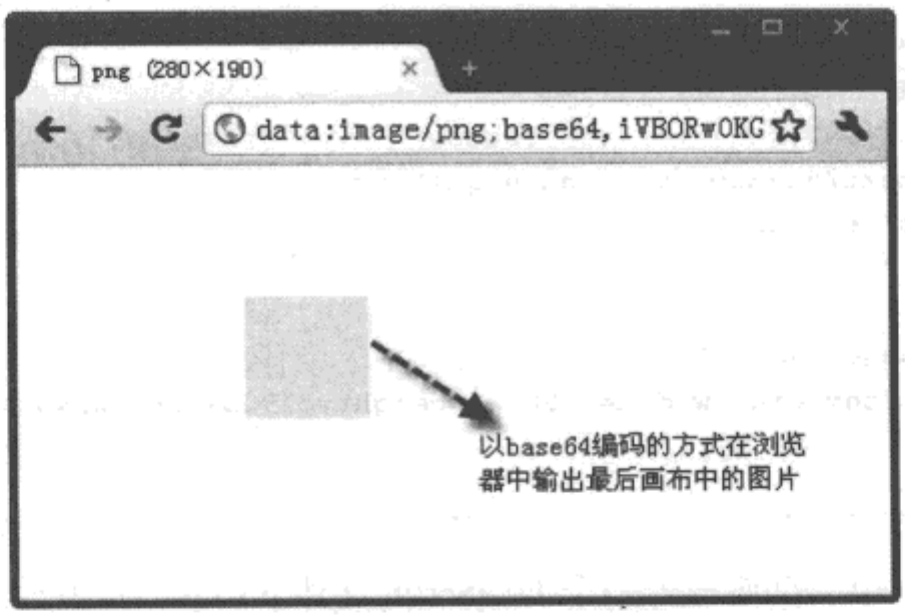


图 7-19　以 base 64 编码的方式在浏览器中输出图片

单击“输出”标记时，调用 toDataURL() 方法，将最后绘制的图形以 base64 编码的形式输出在浏览器中。可以将该编码值保存，输入浏览器中，就可以显示图形。

提示　无论是 save() 方法，还是 restore() 方法，都是对内存中栈的存取操作。执行 save() 方法时，将图形保存在内存的栈中；执行 restore() 方法时，又将保存的栈从内存中取出。

7.5.3　制作简单的动画

利用画布元素 <canvas>，除了可以绘制图形、图像、文字外，还可以制作一些简单的动

画。制作过程十分简单，主要分为两步：

步骤 1 自定义一个函数，用于图形的移动或其他动作。

步骤 2 使用 setInterval() 方法设置动画执行的间隔时间，反复执行自定义函数。

下面通过实例 7-16 介绍利用 <canvas> 元素制作简单动画的过程。

实例 7-16 在画布中制作简单的动画

1. 功能描述

在页面中，新建一个 <canvas> 元素，在该画布元素中，绘制一个卡通笑脸。当加载页面时，该笑脸从画布的左边慢慢移至右边，又从右边移动至左边，最后停止在起始位置。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 7-16.html，加入代码如代码清单 7-16 所示。

代码清单 7-16 在画布中制作简单的动画

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>在画布中制作简单的动画</title>
<link href="Css/css7.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js16.js"/>
</script>
</head>
<body onLoad="pageload();">
    <canvas id="cnvMain" width="280px" height="190px"></canvas>
</body>
</html>
```

在代码清单 7-16 中，页面导入一个 JavaScript 文件 js16.js，其中自定义多个函数，该函数在加载页面过程中被调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var intI, intJ, intX;
// 自定义加载页面函数
function pageload() {
    var cnv = $$("cnvMain");
    var cxt = cnv.getContext("2d");
    drawFace(cxt);
    intI = 1;
```

```
    intJ = 21;
    setInterval(moveFace, 100);
}
// 调用自定义函数绘制笑脸
function drawFace(cxt) {
    drawCirc(cxt, "#666", 30, 80, 30, 2, true);
    drawCirc(cxt, "#fff", 20, 70, 5, 2, true);
    drawCirc(cxt, "#fff", 40, 70, 5, 2, true);
    drawCirc(cxt, "#fff", 30, 80, 18, 1, false);
}
// 根据参数绘制各类圆形
function drawCirc(cxt, strColor, intX, intY, intR, intH, blnFill) {
    cxt.beginPath();
    cxt.arc(intX, intY, intR, 0, Math.PI * intH, blnFill);
    if (blnFill) {
        cxt.fillStyle = strColor;
        cxt.fill();
    } else {
        cxt.lineWidth = 2;
        cxt.strokeStyle = strColor;
        cxt.stroke();
    }
    cxt.closePath();
}
// 实现往返移动笑脸的功能
function moveFace() {
    var cnv = $$("cnvMain");
    var cxt = cnv.getContext("2d");
    cxt.clearRect(0, 0, 280, 190);
    if (intI < 20) {
        intI += 1;
        intX = intI;
    } else {
        if (intJ > 0) {
            intJ -= 1;
            intX = -intJ;
        }
    }
    cxt.translate(intX, 0);
    drawFace(cxt);
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的页面效果如图 7-20 所示。

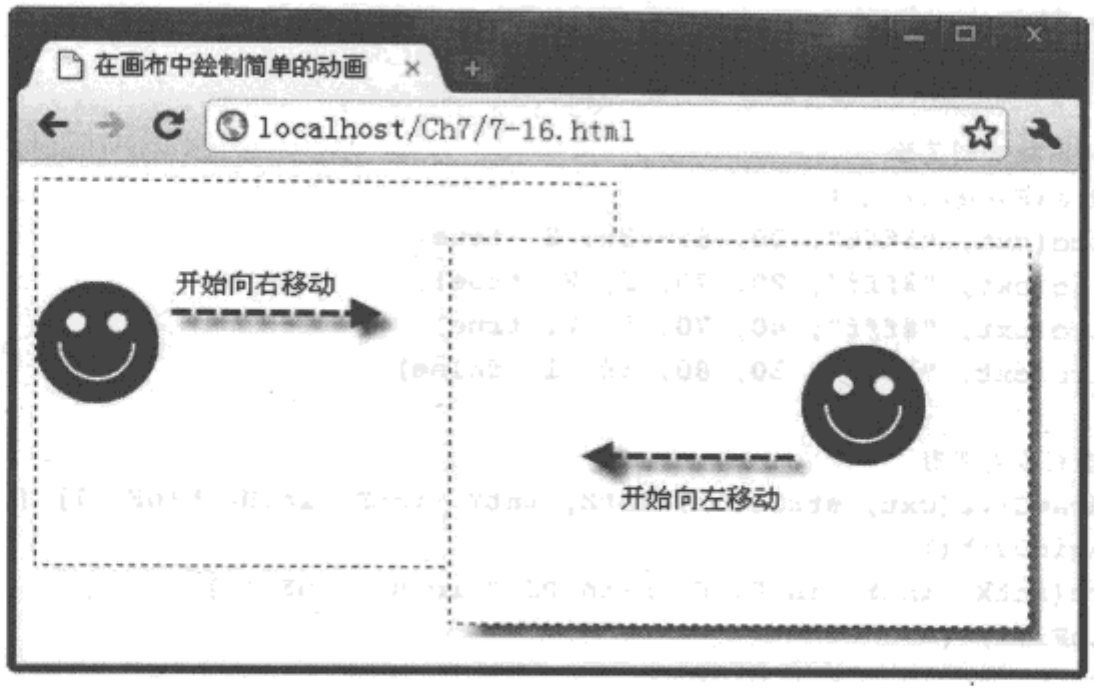


图 7-20　在画布中制作简单的动画

4. 源码分析

在本实例的 JavaScript 代码中，定义了 4 个自定义函数。其中，函数 pageload() 用于加载页面时调用；函数 drawFace() 用于根据上下文环境对象在画布中绘制卡通笑脸；函数 drawCirc() 用于根据传递的参数值，使用 fill() 与 stroke() 方法绘制指定位置、填充色、半径、弧度的圆形；函数 moveFace() 用于实现往返移动笑脸的功能。

在函数 drawFace() 中，4 次调用函数 drawCirc()，分别绘制卡通笑脸的头形、两只眼睛与嘴；在 moveFace() 函数中，先根据自增量 intI 的值，使用 translate() 方法向右移动卡通笑脸。当 intI 值大于 20 时，转为获取 intJ 值，根据自减量 intJ 的值，使用 translate() 方法向左移动卡通笑脸，直到 intJ 值小于 0，便停止移动。在自定义函数 pageload() 中，通过 setInterval() 方法，按时反复执行函数 moveFace()，最终在画布中实现简单的动画效果，详细实现过程如代码中加粗部分所示。

7.6　本章小结

本章先从最基础的画布知识讲起，介绍绘制简单的矩形与圆形的方法、路径在画布中的使用，以及通过路径的方式绘制直线与圆形的技巧；然后，介绍如何操作已绘制的各类图形；最后，通过理论与实例相结合的方式，详细介绍在画布中绘制图像、控制图像的方法，以及绘制文字、制作简单动画的操作过程。通过本章的学习，读者将全面掌握在 HTML 5 中通过画布元素 <canvas> 绘制各类图形的方法。

第 8 章

HTML 5 中的数据存储

本章内容

- Web Storage 存储简介
- localStorage 详解
- Web SQL 数据库基础
- 本章小结

随着 Web 应用的发展，如何更好地在客户端存储数据，是应用开发者非常关注的一个问题。在 HTML 4 及之前的版本中，通常使用 Cookie 存储机制将数据保存在用户的客户端。但使用 Cookie 方式存储客户端数据有一系列制约其发展的因素，例如限制保存数据空间大小、数据保密性差、代码操纵复杂等，开发者迫切需要寻找新的数据存储方式改善现状。

HTML 5 中增加了两种全新的数据存储方式：Web Storage 和 Web SQL Database。前者可用于临时或永久保存客户端的少量数据；后者是客户端本地化的一套数据库系统，通过这套数据系统，可以将大量的数据保存在客户端，而无需与服务端交互，极大地减轻了服务端的压力，加快了其他页面浏览的速度。

本章将详细介绍在 HTML 5 中使用这两种数据存储方式的方法与技巧。

8.1 Web Storage 存储简介

Web Storage 页面存储是 HTML 5 为数据存储在客户端提供的一项重要功能，由于 Web Storage API 可以区分会话数据与长期数据，因此，相应的 API 类型分为两种：

- sessionStorage（保存会话数据）
- loaclStorage（在客户端长期保存数据）

正是由于 Web Storage API 可以将客户端的数据分类型进行存储，使它在运用上更加优越于传统、单一的 Cookie 方式。下面简要介绍这两种类型数据存储的方式。

8.1.1 sessionStorage 对象

在页面进行数据存储过程中，使用 sessionStorage 对象保存的数据时间非常短暂，因为该数据实质上是被保存在 session 对象中。用户在打开浏览器时，可以查看操作过程中要求临时保存的数据；一旦关闭浏览器，所有使用 sessionStorage 对象保存的数据将全部丢失。

sessionStorage 对象保存数据的操作非常简单，只需要调用 setItem() 方法，其调用格式如下：

```
sessionStorage.setItem(key,value)
```

其中，参数 key 表示被保存内容的键名，参数 value 表示被保存内容的键值。在使用 setItem() 方法保存数据时，对应格式为（键名，键值）。一旦键名设置成功，则不允许修改，也不能重复；如果有重复的键名，那么，只能修改对应的键值，即用新增重复的键名值取代原有重复的键名值。

使用 sessionStorage 对象中的 setItem() 方法保存数据后，如果需要读取被保存的数据，应该调用 sessionStorage 对象中 getItem() 方法，其调用格式如下：

```
sessionStorage.getItem(key)
```

其中，参数 key 表示设置保存时被保存内容的键名，该方法将返回一个指定键名对应的键值，

如果不存在，则返回一个 null 值。

下面通过实例 8-1 介绍使用 sessionStorage 对象保存与读取临时数据的过程。

实例 8-1　使用 sessionStorage 对象保存与读取临时数据

1. 功能描述

在页面中，创建一个文本框与“读取”按钮。用户在文本框中输入内容时，通过 sessionStorage 对象保存文本框输入的内容，并即时显示在页面中；单击“读取”按钮时，将直接读取被保存的临时数据。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-1.html，加入代码如代码清单 8-1 所示。

代码清单 8-1　使用 sessionStorage 对象保存与读取临时数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 sessionStorage 对象保存与读取临时数据 </title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js1.js"/>
</script>
</head>
<body>
 <fieldset>
   <legend>sessionStorage 对象保存与读取临时数据 </legend>
   <input name="txtName" type="text" class="inputtxt"
          onChange="txtName_change(this);" size="30px">
   <input name="btnGetValue" type="button" class="inputbtn"
          onClick="btnGetValue_click ();" value=" 读取 ">
   <p id="pStatus"></p>
 </fieldset>
</body>
</html>
```

在代码清单 8-1 中，页面导入一个 JavaScript 文件 js1.js，其中自定义两个函数，分别在输入文本框内容与单击“读取”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 输入文本框内容时调用的函数
function txtName_change(v) {
```

```
    var strName = v.value;
    sessionStorage.setItem("strName", strName);
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = sessionStorage.getItem("strName");
}
// 单击“读取”按钮时调用的函数
function btnGetValue_click() {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = sessionStorage.getItem("strName");
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-1 所示。

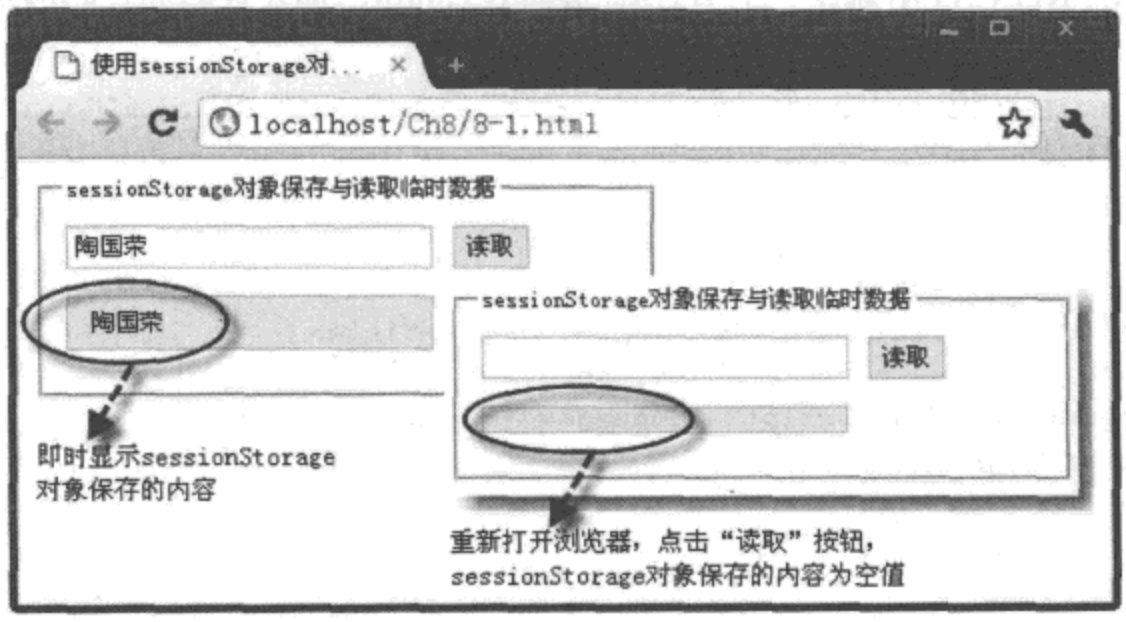


图 8-1　使用 sessionStorage 对象保存与读取临时数据

4. 源码分析

在本实例的 JavaScript 代码中，当用户在页面中的文本框中输入内容时，将触发 onChange 事件，其中调用了自定义的函数 txtName_change()。该函数首先通过变量 strName 获取传来的文本框内容，然后通过调用 sessionStorage 对象中的 setItem() 方法，将该内容值保存至 Session 对象中，键名为“strName”，对应键值为已获取内容的变量 strName。保存完成后，再通过调用 sessionStorage 对象中的 getItem() 方法，根据保存的键名，将对应的键值通过 ID 号为“pStatus”的 <p> 元素显示在页面中。

重新打开浏览器，由于原先通过 sessionStorage 对象保存的内容全部丢失，所以当用户直接单击“读取”按钮时，无法读取指定键名的键值。详细实现过程如代码中加粗部分所示。

8.1.2　localStorage 对象

使用 sessionStorage 对象只能保存用户临时的会话数据，关闭浏览器后，这些数据都将

丢失。因此，如果需要长期在客户端保存数据，不建议使用 sessionStorage 对象，应该使用 HTML 5 中新提供的 localStorage 对象。使用该对象可以将数据长期保存在客户端，直至人工清除为止。

如果使用 localStorage 对象保存数据内容，需要调用对象中的 setItem() 方法，其调用格式如下所示：

```
localStorage.setItem(key,value)
```

与 sessionStorage 对象保存数据的方法参数说明相同，localStorage 对象也是通过调用 setItem() 方法，按照（键名，键值）的方式进行设置，只是调用的对象不一样。使用 localStorage 对象保存数据后，同样可以通过调用对象中的 getItem() 方法，读取指定键名所对应的键值，其调用格式如下：

```
localStorage.getItem(key)
```

其中，参数 key 就是需要读取键值内容的键名，与 sessionStorage 对象一样，如果键名不存在，则返回一个 null 值。

localStorage 对象可以将内容长期保存在客户端，即使是重新打开浏览器也不会丢失。如果需要清除 localStorage 对象保存的内容，应该调用 localStorage 对象的另一个方法 removeItem()，其调用格式如下所示：

```
localStorage.removeItem(key)
```

其中，参数 key 表示需要删除的键名，一旦删除成功，与键名对应的相应数据将全部被删除。

下面通过实例 8-2 介绍使用 localStorage 对象保存与读取登录用户名与密码的过程。

实例 8-2　使用 localStorage 对象保存与读取登录用户名与密码

1. 功能描述

新建一个登录页面，用户在文本框中输入用户名与密码，单击“登录”按钮后，将使用 localStorage 对象保存登录时的用户名。如果选中“是否保存密码”选项，将保存登录时的密码，否则，将清空原先保存的密码。当重新在浏览器中打开该页面时，经过保存的用户名和密码数据，将分别显示在相应的文本框中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-2.html，加入代码如代码清单 8-2 所示。

代码清单 8-2　使用 localStorage 对象保存与读取登录用户名与密码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 localStorage 对象保存与读取登录用户名与密码</title>
<link href="Css/css8 css" rel="stylesheet" type="text/css">
```

```
<script type="text/javascript" language="jscript"
        src="Js/js2.js"/>
</script>
</head>
<body onLoad="pageload();">
   <form id="frmLogin" action="#">
    <fieldset>
       <legend> 登录 </legend>
       <ul>
           <li class="li_top">
               <span id="spnStatus"></span>
           </li>
           <li> 名称:
               <input id="txtName" class="inputtxt"
                      type="text">
           </li>
           <li> 密码:
               <input id="txtPass" class="inputtxt"
                      type="password">
           </li>
           <li>
               <input id="chkSave" type="checkbox">
               是否保存密码
           </li>
           <li class="li_bot">
               <input name="btnLogin" class="inputbtn" value=" 登录 "
                      type="button" onClick="btnLogin_click();">
               <input name="rstLogin" class="inputbtn"
                      type="reset" value=" 取消 ">
           </li>
       </ul>
    </fieldset>
   </form>
</body>
</html>
```

在代码清单 8-2 中，页面导入一个 JavaScript 文件 js2.js，其中自定义两个函数，分别在页面加载和单击“登录”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 页面加载时调用的函数
function pageload() {
    var strName = localStorage.getItem("keyName");
    var strPass = localStorage.getItem("keyPass");
```

```
    if (strName) {
        $$("txtName").value = strName;
    }
    if (strPass) {
        $$("txtPass").value = strPass;
    }
}
// 单击“登录”按钮后调用的函数
function btnLogin_click() {
    var strName = $$("txtName").value
    var strPass = $$("txtPass").value;
    localStorage.setItem("keyName", strName);
    if ($$("chkSave").checked) {
        localStorage.setItem("keyPass", strPass);
    } else {
        localStorage.removeItem("keyPass");
    }
    $$("spnStatus").className = "status";
    $$("spnStatus").innerHTML = " 登录成功！ ";
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-2 所示。

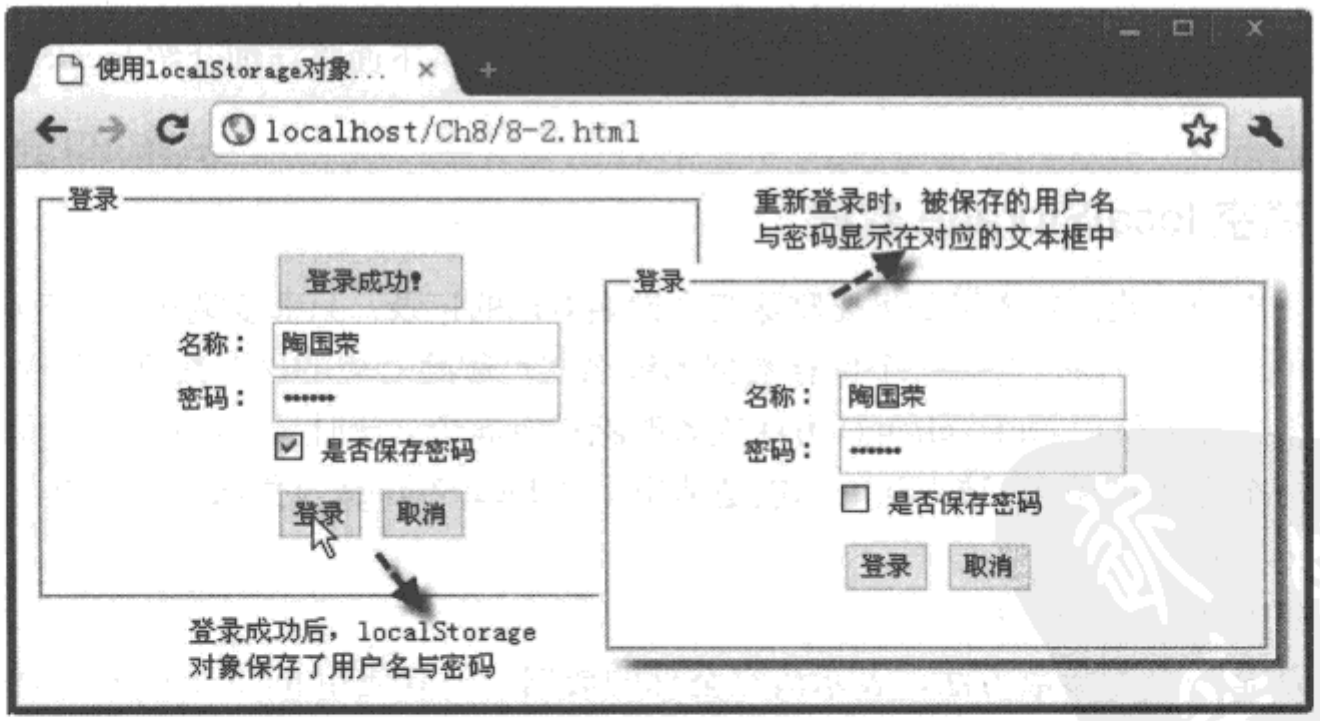


图 8-2　使用 localStorage 对象保存与读取登录用户名与密码

4. 源码分析

在本实例中，页面在加载时，将调用自定义的函数 pageload()。在该函数中，先通过 localStorage 对象中的 getItem() 方法获取指定键名的键值，并保存在变量中。如果不为空，则

将该变量值赋值于对应的文本框，用户下次登录时不用再次输入，以方便用户的操作。

用户单击“登录”按钮时，将触发 onClick 事件，在该事件中，调用另外一个自定义的函数 btnLogin_click()。在该函数中，首先分别通过两个变量保存在文本框中输出的用户名与密码，然后，调用 localStorage 对象中的 setItem() 方法，将用户名作为键名“keyName”的键值进行保存。如果选择了“是否保存密码”选项，则将密码作为键名“keyPass”的键值进行保存；否则，将调用 localStorage 对象中的 removeItem() 方法，删除键名为“keyPass”的记录。详细实现过程如代码中加粗部分所示。

注意 尽管使用 localStorage 对象可以将数据长期保存在客户端，但在跨浏览器读取数据时，被保存的数据不可共用。即每一个浏览器只能读取各自浏览器中保存的数据，不能访问其他浏览器中保存的数据。

8.2 localStorage 详解

在 HTML 5 中，通过 localStrage 对象保存的数据会长期存储在用户的客户端，直到采取手动删除的方式才能彻底清空这些保存的数据。在删除这些数据前，需要查看每个键名对应的值，即要遍历整个 localStorage 数据信息。另外，JSON 是一种以文本方式保存数据的格式，这种格式可以很便捷地与 localStorage 对象中保存的数据进行转换，从而使 localStorage 对象保存的数据结构更加合理，更方便数据在页面中进行输出。下面将详细介绍 localStorage 的相关知识。

8.2.1 清空 localStorage 数据

在上一节中，我们知道了如果要删除某个键名对应的记录，只需要调用 localStorage 对象中的 removeItem() 方法，传递一个保存数据的键名即可删除对应的保存数据。但是，有时保存数据很多，如果使用 removeItem() 方法逐条删除相对麻烦。此时，可以调用 localStorage 对象中的另一个方法 clear()，该方法的功能是清空全部 localStorage 对象保存的数据，其调用格式如下：

```
localStorage.clear();
```

该方法是一个无参数方法，表示清空全部的数据。一旦使用 localStorage 对象保存了数据，用户就可以在浏览器中打开相应的代码调试工具，查看每条数据对应的键名与键值。执行删除或清空操作后，其对应的数据也会发生变化，这些变化可以通过浏览器的代码调试工具进行侦测。

下面通过实例 8-3 介绍清空 localStorage 对象保存的全部数据的过程。

实例 8-3　清空 localStorage 对象保存的全部数据

1. 功能描述

在新建的页面中添加两个按钮，一个用于使用 localStorage 对象保存 6 条顺序记录，另一个用于清空所有 localStorage 对象保存的记录。无论是增加还是清空数据，都可以在浏览器的调试工具中查看其变化过程。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-3.html，加入代码如代码清单 8-3 所示。

代码清单 8-3　清空 localStorage 对象保存的全部数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>清空 localStorage 对象保存的全部数据 </title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js3.js"/>
</script>
</head>
<body>
    <input id="btnAdd" type="button" value=" 增加 "
           class="inputbtn" onClick="btnAdd_Click();">
    <input id="btnDel" type="button" value=" 清空 "
           class="inputbtn" onClick="btnDel_Click();">
    <p id="pStatus"></p>
</body>
</html>
```

在代码清单 8-3 中，页面导入一个 JavaScript 文件 js3.js，其中自定义两个函数，在单击“增加”和“清空”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var intNum = 0;
// 单击 " 增加 " 按钮时调用
function btnAdd_Click() {
    for (var intI = 0; intI <= 5; intI++) {
        var strKeyName = "strKeyName" + intI;
        var strKeyValue = "strKeyValue" + intI;
        localStorage.setItem(strKeyName, strKeyValue);
        intNum++;
```

```
        }
        $$("pStatus").style.display = "block";
        $$("pStatus").innerHTML = " 已成功保存 <b>" + intNum + "</b> 条数据记录！ ";
}
// 单击 " 清空 " 按钮时调用
function btnDel_Click() {
        localStorage.clear();
        $$("pStatus").style.display = "block";
        $$("pStatus").innerHTML = " 已成功清空全部数据记录！ ";
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-3 所示。

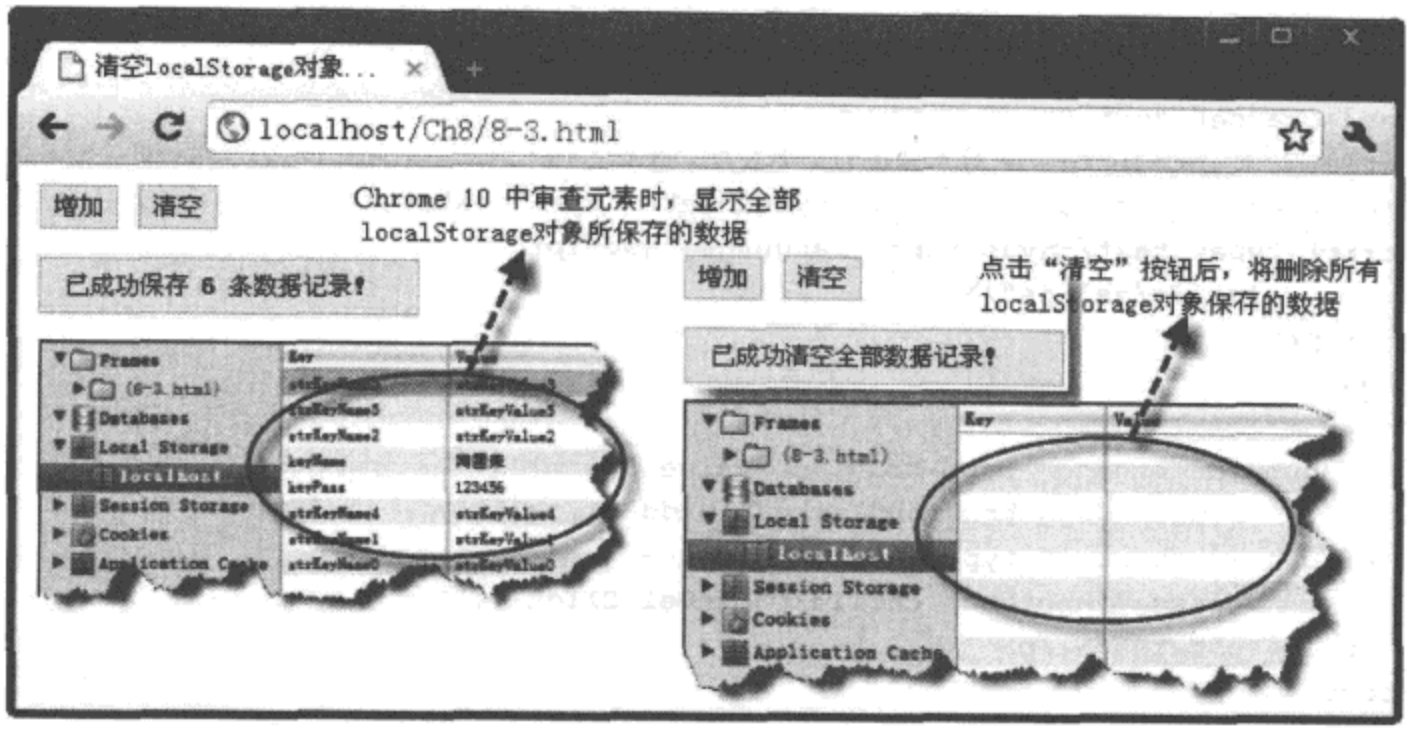


图 8-3　清空 localStorage 对象保存的全部数据

4. 源码分析

在本实例中，当用户单击“增加”按钮时，将使用循环的方式，按执行顺序保存 6 条数据记录，其键名为“strKeyName”与变量 intI 相连，即“strKeyName0”，“strKeyName1”，…，对应键值为“strKeyValue”与变量 intI 相连，即“strKeyValue 0”，“strKeyValue 1”，…。这些被 localStorage 对象保存的数据记录，可以在浏览器 Chrome 10 中，通过单击右键，选择“审查元素”选项，单击“Resources”选项卡进行查看，效果如图 8-3 左边所示。

当用户单击“清空”按钮时，调用自定义函数 btnDel_Click()。在该函数中，执行 localStrage 对象中的 clear() 方法，清空所有 localStorage 对象保存的数据，效果如图 8-3 右边所示。

需要说明的是，各浏览器查看 localStorage 对象所保存的数据方式不完全相同，Firefox 使

用 Firebug 调试工具作为存储查看器；Opera 在页面中单击右键，选择“检查元素”选项，单击“本地资源”选项卡进行查看。

8.2.2　遍历 localStorage 数据

为了查看 localStorage 对象保存的全部数据信息，通常要遍历这些数据。在遍历过程中，需要访问 localStorage 对象的另外两个属性：length 与 key。前者表示 localStorage 对象中保存数据的总量；后者表示保存数据时的键名项，该属性常与索引号（index）配合使用，表示第几条键名对应的数据记录。其中，索引号（index）以 0 值开始，如果取第 3 条键名对应的数据，index 值应该为 2。

下面通过实例 8-4 介绍遍历 localStorage 对象保存的全部数据的过程。

实例 8-4　遍历 localStorage 对象保存的全部数据

1. 功能描述

在创建的页面中，通过遍历的方式，获取 localStorage 对象保存的全部点评数据记录。在文本框中输入点评内容，单击“发表”按钮后，可以通过 localStorage 对象保存输入的数据，并实时显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-4.html，加入代码如代码清单 8-4 所示。

代码清单 8-4　遍历 localStorage 对象保存的全部数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>遍历 localStorage 数据</title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js4.js"/>
</script>
</head>
<body onLoad="getlocalData();">
   <ul id="ulMessage">
       正在读取数据 ...
   </ul>
   <p class="p4">
      <textarea id="txtContent" class="inputtxt"
                cols="37" rows="5">
      </textarea><br>
      <input id="btnAdd" type="button" value="发表"
             class="inputbtn" onClick="btnAdd_Click();">
```

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

在代码清单 8-4 中，页面导入一个 JavaScript 文件 js4.js，其中自定义多个函数，在页面加载和单击“发表”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 单击 " 发表 " 按钮时调用
function btnAdd_Click() {
    // 获取文本框中的内容
    var strContent = $$("txtContent").value;
   // 定义一个日期型对象
    var strTime = new Date();
   // 如果不为空，则保存
    if (strContent.length > 0) {
        var strKey = "cnt" + RetRndNum(4);
        var strVal = strContent + "," + strTime.toLocaleTimeString();
        localStorage.setItem(strKey, strVal);
    }
    // 重新加载
    getlocalData();
    // 清空原先内容
   $$("txtContent").value="";
}

// 获取保存数据并显示在页面中
function getlocalData() {
    // 标题部分
    var strHTML = "<li class='li_h'>";
    strHTML += "<span class='spn_a'> 编号 </span>";
    strHTML += "<span class='spn_b'> 内容 </span>";
    strHTML += "<span class='spn_c'> 时间 </span>";
    strHTML += "</li>";
    // 内容部分
    var strArr = new Array(); // 定义一数组
    for (var intI = 0; intI < localStorage.length; intI++) {
        // 获取 Key 值
        var strKey = localStorage.key(intI);
        // 过滤键名内容
        if (strKey.substring(0, 3) == "cnt") {
            var strVal = localStorage.getItem(strKey);
            strArr = strVal.split(",");
            strHTML += "<li class='li_c'>";
            strHTML += "<span class='spn_a'>" + strKey + "</span>";
```

```
            strHTML += "<span class='spn_b'>" + strArr[0] + "</span>";
            strHTML += "<span class='spn_c'>" + strArr[1] + "</span>";
            strHTML += "</li>";
        }
    }
    $$("ulMessage").innerHTML = strHTML;
}
// 生成指定长度的随机数
function RetRndNum(n) {
    var strRnd = "";
    for (var intI = 0; intI < n; intI++) {
        strRnd += Math.floor(Math.random() * 10);
    }
    return strRnd;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-4 所示。

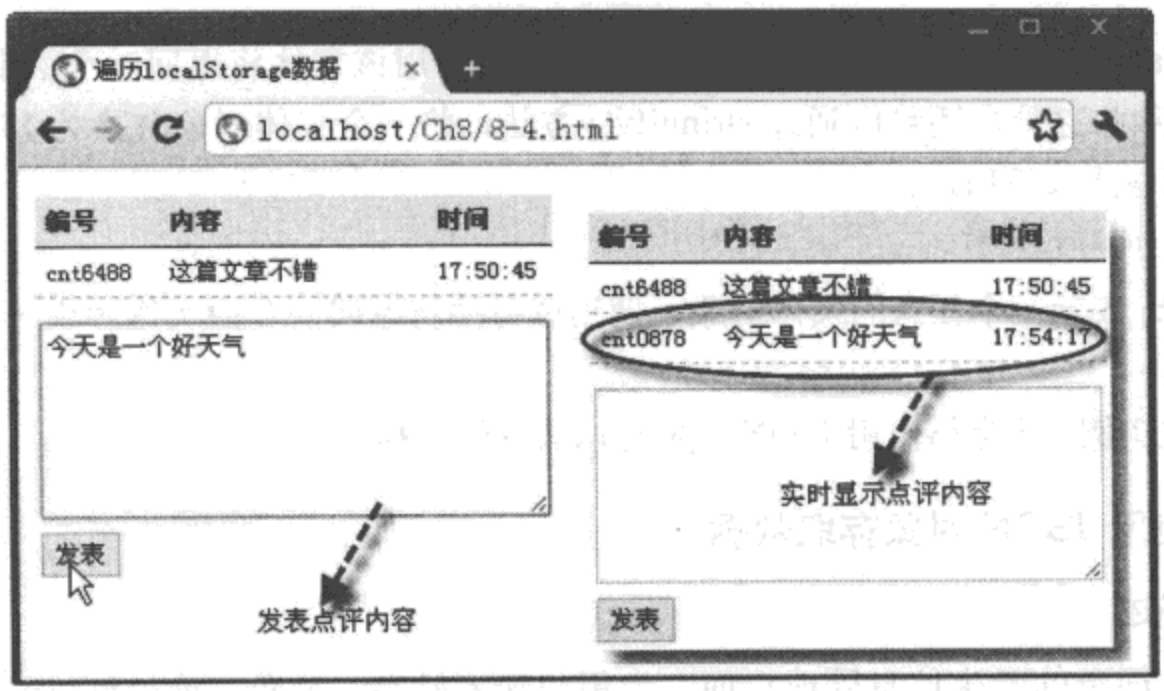


图 8-4　遍历 localStorage 对象保存的全部数据

4. 源码分析

在本实例中，当页面加载时，调用一个自定义的函数 getlocalData()。在该函数中，根据 localStorage 对象的 length 值，使用 for 语句遍历 localStorage 对象保存的全部数据。在遍历过程中，通过“strKey”变量保存每次遍历的键名。获取键名后，为了只获取 localStorage 对象中保存的点评数据，检测键名前 3 个字符是否为“cnt”。如果是，则通过 getItem() 方法获取键名对应的键值，并保存在变量“strVal”中。由于键值是由“,”组成的字符串，因此，先通过数组 strArr 保存分割后的各项数值，然后通过数组下标将各项获取的内容显示在页面中。

用户在页面中输入点评内容后单击“发表”按钮时，将调用另外一个自定义的函数

btnAdd_Click()，在该函数中，先获取点评内容并保存在变量“strContent”中。为了使保存内容的键名不重复，并且具有标记性，在生成键名时调用函数 RetRndNum()，随机生成一个4位数字，并与字符“cnt”组合成新的字符串，保存在变量“strKey”中。为了保存更多的数据信息，保存点评内容的变量“strContent”通过“,”与时间数据组合成新的字符串，保存在变量“strVal”中。最后，通过 setItem() 方法将变量“strKey”与“strVal”分别作为键名与键值保存在 localStorage 对象中。

8.2.3 使用 JSON 对象存取数据

在实例 8-4 中，虽然使用“,”逗号的方式可以存储更多的键值内容，但是处理相对复杂，拓展性差，数据的结构不合理，只能应对少量数据。为了解决这一问题，在 HTML 5 中可以通过 localStorage 数据与 JSON 对象的转换，快速实现存储更多数据的功能。

如果要将 localStorage 数据转成为 JSON 对象，需调用 JSON 对象的 parse() 方法，调用格式如下所示：

```
JSON.parse(data)
```

其中，参数 data 表示 localStorage 对象获取的数据，调用该方法将返回一个装载 data 数据的 JSON 对象。除此之外，还可以通过 stringify() 方法，将一个实体对象转换为 JSON 格式的文本数据，调用格式如下所示：

```
JSON.stringify(obj)
```

其中，参数 obj 表示一个任意的实体对象，调用该方法将返回一个由实体对象转成 JSON 格式的文本数据集。

下面通过实例 8-5 介绍使用 JSON 对象存取数据的过程。

实例 8-5 使用 JSON 对象存取数据

1. 功能描述

创建一个简单的学生信息管理页面，当用户输入姓名、分数，选择性别，单击“增加”按钮后，使用 JSON 中的 stringify() 方法，将数据保存在 localStorage 对象中；同时，调用 JSON 中的 parse() 方法实时在页面中显示新增的学生数据信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-5.html，加入代码如代码清单 8-5 所示。

代码清单 8-5 使用 JSON 对象存取数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
```

```
<title>使用 JSON 对象存取数据 </title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js5.js"/>
</script>
</head>
<body onLoad="getlocalData();">
   <ul id="ulMessage">
       正在读取数据 ...
   </ul>
   <p class="p5">
      <span class="spanl">
      学号: <input type="text" readonly="true" id="txtStuID"
                   class="inputtxt" size="10"><br>
      姓名: <input type="text" id="txtName" class="inputtxt"
                   size="15">
      </span>
      <span class="spanr">
      性别: <select id="selSex">
              <option value=" 男 "> 男 </option>
              <option value=" 女 "> 女 </option>
            </select><br>
      总分: <input type="text" id="txtScore" class="inputtxt"
                   size="8">
      </span>
      <p class="btn">
      <input id="btnAdd" type="button" value=" 增加 "
             class="inputbtn" onClick="btnAdd_Click();">
      </p>
    </p>
</body>
</html>
```

在代码清单 8-5 中，页面导入一个 JavaScript 文件 js5.js，其中自定义多个函数，在页面加载和单击“增加”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 单击 " 增加 " 按钮时调用
function btnAdd_Click() {
    var strStuID = $$("txtStuID").value;
    var strName = $$("txtName").value;
    var strSex = $$("selSex").value;
    var strScore = $$("txtScore").value;
    if (strName.length > 0 && strScore.length > 0) {
```

```
            // 定义一个实体对象，保存全部获取的值
            var SetData= new Object;
            SetData.StuID=strStuID;
            SetData.Name=strName;
            SetData.Sex=strSex;
            SetData.Score=strScore;
            var strTxtData=JSON.stringify(SetData);
            localStorage.setItem(strStuID, strTxtData);
        }
        // 重新加载
        getlocalData();
       // 清空原先内容
       $$("txtName").value="";
       $$("txtScore").value="";
}
// 获取保存数据并显示在页面中
function getlocalData() {
    // 标题部分
    var strHTML = "<li class='li_h'>";
    strHTML += "<span class='spn_a'>学号</span>";
    strHTML += "<span class='spn_b'>姓名</span>";
    strHTML += "<span class='spn_a'>性别</span>";
    strHTML += "<span class='spn_c'>总分</span>";
    strHTML += "</li>";
    // 内容部分
    for (var intI = 0; intI < localStorage.length; intI++) {
        // 获取 Key 值
        var strKey = localStorage.key(intI);
        // 过滤键名内容
        if (strKey.substring(0, 3) == "stu") {
           var GetData=JSON.parse(localStorage.getItem(strKey));
            strHTML += "<li class='li_c'>";
            strHTML += "<span class='spn_a'>" + GetData.StuID + "</span>";
            strHTML += "<span class='spn_b'>" + GetData.Name + "</span>";
            strHTML += "<span class='spn_a'>" + GetData.Sex + "</span>";
           strHTML += "<span class='spn_c'>" + GetData.Score + "</span>";
            strHTML += "</li>";
        }
    }
    $$("ulMessage").innerHTML = strHTML;
    $$("txtStuID").value="stu" + RetRndNum(4);
}
// 生成指定长度的随机数
function RetRndNum(n) {
    var strRnd = "";
    for (var intI = 0; intI < n; intI++) {
        strRnd += Math.floor(Math.random() * 10);
    }
```

```
    return strRnd;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-5 所示。

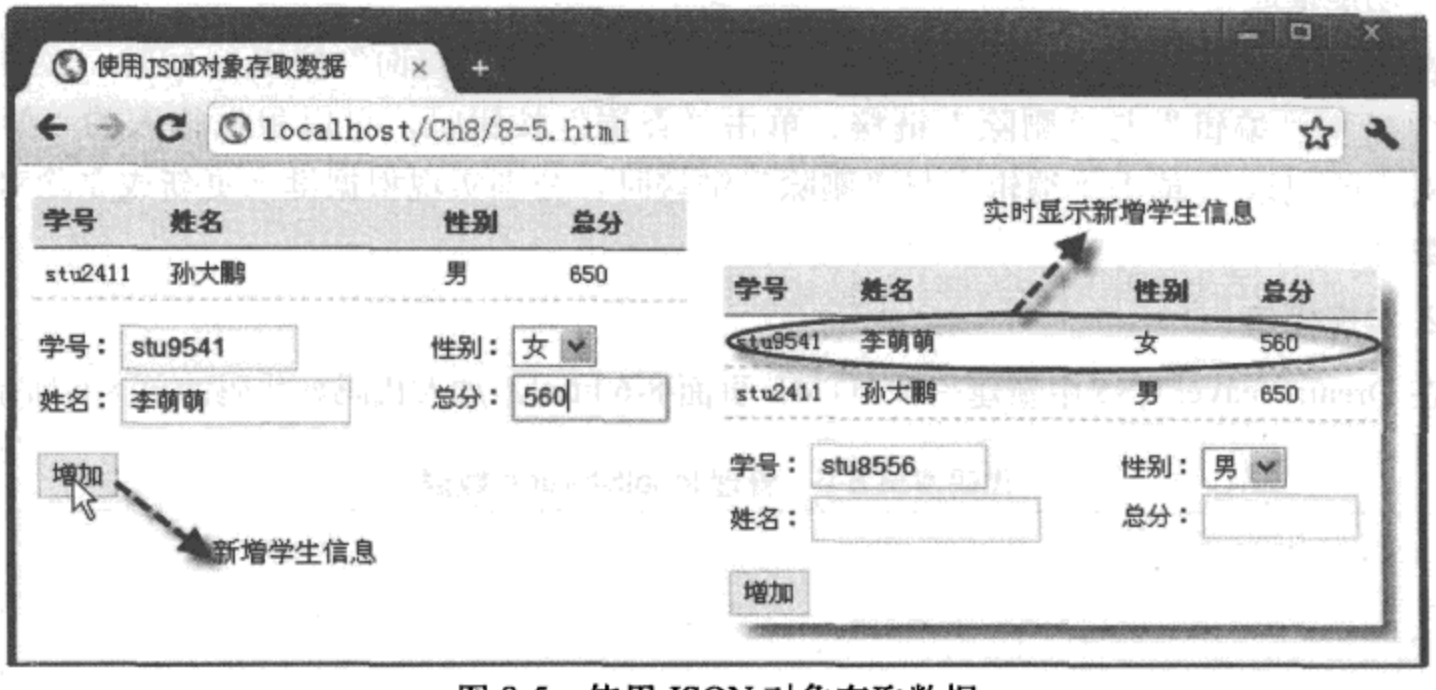


图 8-5　使用 JSON 对象存取数据

4. 源码分析

在本实例中，事件的触发与实例 8-4 基本相同，页面在加载时，都是先触发 onLoad() 事件，调用自定义的函数 getlocalData()；单击“增加”按钮时，调用自定义的函数 btnAdd_Click()。不同之处在于，调用函数 getlocalData() 遍历 localStorage 对象保存的数据时，通过 JSON 对象中的 parse() 方法，将键名对应的键值转换成一个装载全部键值数据的 JSON 对象“GetData”；调用该对象中的属性名称获取各个对应的键值数据，如“GetData.StuID”表示学生编号，显示在页面中。

在调用 btnAdd_Click() 函数时，先检测姓名与分数的内容是否为空。如果不为空，则使用“new Object”语句创建一个对象“SetData”，将输入的各种学生数据作为该对象的不同属性值进行保存；然后，通过调用 JSON 对象中的 stringify() 方法，将对象 SetData 转成 JSON 格式的文本数据保存在变量“strTxtData”中；最后，调用 setItem() 方法，将变量“strStuID”与“strTxtData”分别作为键名与键值保存在 localStorage 对象中。详细实现过程如代码中加粗部分所示。

8.2.4　管理 localStorage 数据

上一节通过引入 JSON 对象，先将保存数据转成 JSON 对象，再通过对象读取数据，极大地方便使用 localStorage 对象保存多个字段数据的操作；此外，还可以通过键名，查询、更新、

删除对应的键值记录，真正实现对 localStorage 对象保存数据的管理功能。

下面通过实例 8-6 介绍管理 localStorage 数据的过程。

实例 8-6　管理 localStorage 数据

1. 功能描述

在实例 8-5 的基础上，添加输入查询内容的文本框与“查询”按钮；同时，在列表内容项中新增“编辑”与“删除”链接。单击“查询”按钮时，可以根据输入的“学号”，返回对应的记录；单击“编辑”与“删除”链接时，分别实现根据键名更新或删除对应的键值数据。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-6.html，加入代码如代码清单 8-6 所示。

代码清单 8-6　管理 localStorage 数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>管理 localStorage 数据</title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js6.js"/>
</script>
</head>
<body onLoad="getlocalData(0);">
   <ul id="ulMessage">
       正在读取数据 ...
   </ul>
   <p class="p5">
      <span class="spanl">
      学号: <input type="text" readonly="true" id="txtStuID"
                  class="inputtxt" size="10"><br>
      姓名: <input type="text" id="txtName" class="inputtxt"
                  size="15">
      </span>
      <span class="spanr">
      性别: <select id="selSex">
              <option value="男">男</option>
              <option value="女">女</option>
            </select><br>
      总分: <input type="text" id="txtScore" class="inputtxt"
                  size="8">
      </span>
      <p class="btn">
```

```
        <input id="btnAdd" type="button" value="增加"
               class="inputbtn" onClick="btnAdd_Click();">
        </p>
      </p>
</body>
</html>
```

在代码清单 8-6 中，页面导入一个 JavaScript 文件 js6.js，其中自定义多个函数，分别在实现增加、删除、更新、查询功能时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 单击“增加”按钮时调用
function btnAdd_Click() {
    var strStuID = $$("txtStuID").value;
    var strName = $$("txtName").value;
    var strSex = $$("selSex").value;
    var strScore = $$("txtScore").value;
    if (strName.length > 0 && strScore.length > 0) {
        // 定义一个实体对象，保存全部获取的值
        var setData = new Object;
        setData.StuID = strStuID;
        setData.Name = strName;
        setData.Sex = strSex;
        setData.Score = strScore;
        var strTxtData = JSON.stringify(setData);
        localStorage.setItem(strStuID, strTxtData);
    }
    // 重新加载
    getlocalData(0);
    // 清空原先内容
    $$("txtName").value = "";
    $$("txtScore").value = "";
}
// 单击"查询"按钮时调用
function btnSearch_Click() {
    // 获取查询学号
    var strSearch = $$("txtSearch").value;
    // 根据学号键名获取数据
    getlocalData(strSearch);
}
// 获取保存数据并显示在页面中
function getlocalData(s) {
    // 标题部分
    var strHTML = "<li>";
    strHTML += "请输入学号：";
```

```
strHTML += "<input type='text' id='txtSearch'";
strHTML += "class='inputtxt' size='22'>";
strHTML += "<input id='btnSearch' type='button' value='查询' ";
strHTML += "class='inputbtn' onClick='btnSearch_Click();'>";
strHTML += "</li>";
strHTML += "<li class='li_h'>";
strHTML += "<span class='spn_a'>学号</span>";
strHTML += "<span class='spn_a'>姓名</span>";
strHTML += "<span class='spn_c'>性别</span>";
strHTML += "<span class='spn_c'>总分</span>";
strHTML += "<span class='spn_d'>操作</span>";
strHTML += "</li>";
if (s) {
    var SearchData = JSON.parse(localStorage.getItem(s));
    strHTML += "<li class='li_c'>";
    strHTML += "<span class='spn_a'>" + SearchData.StuID + "</span>";
    strHTML += "<span class='spn_a'>" + SearchData.Name + "</span>";
    strHTML += "<span class='spn_c'>" + SearchData.Sex + "</span>";
    strHTML += "<span class='spn_c'>" + SearchData.Score + "</span>";
    strHTML += "<span class='spn_d'>";
    strHTML += "<a href='#' onclick=EditData(" + s + ")>编辑</a>";
    strHTML += " | ";
    strHTML += "<a href='#' onclick=DeleteData(" + s + ")>删除</a>";
    strHTML += "</span></li>";
} else {
    for (var intI = 0; intI < localStorage.length; intI++) {
        // 获取 Key 值
        var strKey = localStorage.key(intI);
        // 过滤键名内容
        if (strKey.substring(0, 3) == "stu") {
            var GetData = JSON.parse(localStorage.getItem(strKey));
            strHTML += "<li class='li_c'>";
            strHTML += "<span class='spn_a'>" + GetData.StuID + "</span>";
            strHTML += "<span class='spn_a'>" + GetData.Name + "</span>";
            strHTML += "<span class='spn_c'>" + GetData.Sex + "</span>";
            strHTML += "<span class='spn_c'>" + GetData.Score + "</span>";
            strHTML += "<span class='spn_d'>";
            strHTML += "<a href='#' onClick=EditData('" ;
            strHTML +=  GetData.StuID ;
            strHTML +=  "')>编辑</a>";
            strHTML += " | ";
            strHTML += "<a href='#' onClick=DeleteData('" ;
            strHTML +=  GetData.StuID ;
            strHTML += "')>删除</a>";
            strHTML += "</span></li>";
        }
    }
}
```

```
    $$("ulMessage").innerHTML = strHTML;
    $$("txtStuID").value = "stu" + RetRndNum(4);
}
// 单击“编辑”链接时调用
function EditData(k) {
    // 根据键名获取对应数据
    var EditData = JSON.parse(localStorage.getItem(k));
    $$("txtStuID").value = EditData.StuID;
    $$("txtName").value = EditData.Name;
    $$("selSex").value = EditData.Sex;
    $$("txtScore").value = EditData.Score;
}
// 单击“删除”链接时调用
function DeleteData(k) {
    // 删除指定键名对应的数据
    localStorage.removeItem(k);
    // 重新加载
    getlocalData(0);
}
// 生成指定长度的随机数
function RetRndNum(n) {
    var strRnd = "";
    for (var intI = 0; intI < n; intI++) {
        strRnd += Math.floor(Math.random() * 10);
    }
    return strRnd;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-6 所示。

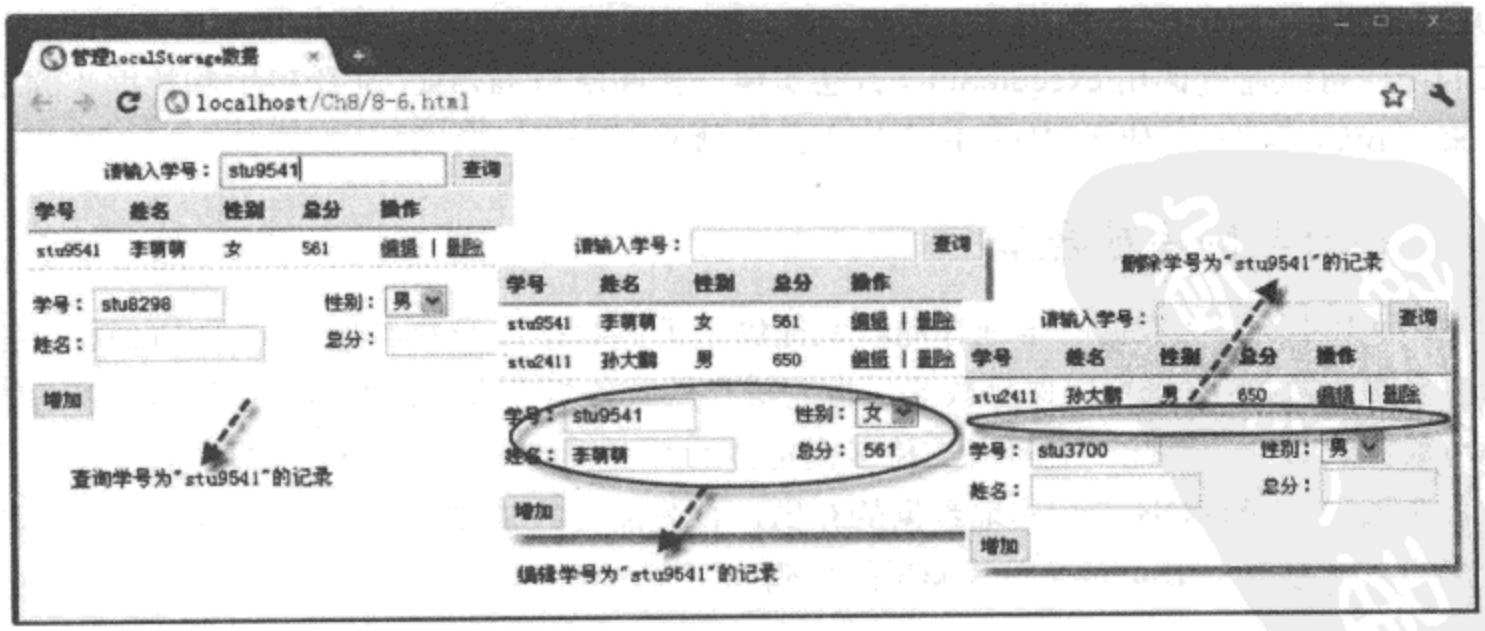


图 8-6　管理 localStorage 数据

4. 源码分析

与实例 8-5 相比，本实例增加了编辑、删除对应数据的功能。首先，为了实现根据键名查询数据的功能，对实例 8-5 中的 getlocalData() 函数据进行了改造，新添加了一个参数 s。如果这个参数有值，表示需要将该值作为键名，调用“JSON.parse(localStorage.getItem(s))”语句，获取对应的键值数据并转成 JSON 对象后，保存到对象变量 SearchData 中，通过该对象的属性显示各项保存的键值数据，并展示在页面中。

为了实现对键值数据的“编辑”功能，新增加一个自定义函数 EditData()，通过该函数中的参数 k，获取编辑时传回的键名。根据该键名，调用“JSON.parse(localStorage.getItem(k))”语句，将获取的键值数据转成 JSON 对象，并保存至对象变量 EditData 中，将对象属性的各项键值赋给页面中对应的文本框与下拉列表。当用户再次单击“增加”按钮时，将按照获取的键名更新对应的键内容，从而实现保存数据更新的功能。

为了可以根据键名删除对应键值记录，新增加另外一个自定义函数 DeleteData()，通过该函数中的参数 k，获取编辑时传回的键名。根据该键名，使用“localStorage.removeItem(k)”语句，删除 localStorage 对象中指定键名的数据；删除完成后，重新调用 getlocalData() 函数，在页面中显示删除后的数据信息。

其他功能的实现与实例 8-5 基本相同，不再赘述，详细过程如代码中加粗部分所示。

8.3 Web SQL 数据库基础

上一节详细介绍了 Web Storage 存储本地数据的方法，虽然这种方法目前可以在许多主流的浏览器、平台与设备上实现，与之相关的 API 也已经标准化，但是，Web Storage 存储空间只有 5MB，键值存储的方式带来诸多不便，未来的本地存储也不仅仅是这样一种方法。其中，最为熟知的就是 Web SQL 数据库（Web SQL DataBase，WebDB），它内置 SQLite 数据库，对数库的操作可以通过调用 executeSql() 方法实现，允许使用 JavaScript 代码控制数据库的操作。接下来，我们详细介绍使用 WebDB 实现本地存储的方法。

8.3.1 打开与创建数据库

WebDB 可以实现数据的本地存储，它提供了关系数据库的基本功能，可以存储页面中交互的、复杂的数据；它既可以保存数据，也能缓存从服务器获取的数据。WebDB 通过事务驱动，实现对数据的管理，因此，它可以支持多浏览器的并发操作，而不发生存储时的冲突。

如果要通过 WebDB 进行本地数据的存储，首先需要打开或创建一个数据库，打开或创建数据库的 API 是 openDatabase，其调用的代码如下所示：

```
openDatabase(DBName, DBVersion, DBDescribe, DBSize, Callback());
```

其中，参数 DBName 表示数据库名称，参数 DBVersion 表示版本号，参数 DBDescribe 表示

对数据库的描述，参数 DBSize 表示数据库的大小，单位为字节，如果是 2MB，必须写成 2*1024*1024，参数 Callback() 表示创建或打开数据库成功后执行的一个回调函数。

调用该方法时，如果指定的数据库名存在，则打开该数据库；否则，新创建一个指定名称的空数据库。

下面通过实例 8-7 介绍使用 openDatabase 打开与创建数据库的过程。

实例 8-7　使用 openDatabase 打开与创建数据库

1. 功能描述

在新建的页面中，增加两个按钮，一个用于打开或创建数据库，另外一个用于检测创建的数据库连接是否正常。单击这两个按钮时，将在页面中显示执行过程中的相应状态。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-7.html，加入代码如代码清单 8-7 所示。

代码清单 8-7　使用 openDatabase 打开与创建数据库

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 openDatabase 打开与创建数据库 </title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js7.js"/>
</script>
</head>
<body>
   <input id="btnCreateDb" type="button" value=" 创建数据库 "
          class="inputbtn" onClick="btnCreateDb_Click();">
   <input id="btnTestConn" type="button" value=" 测试连接 "
          class="inputbtn" onClick="btnTestConn_Click();">
   <p id="pStatus"></p>
</body>
</html>
```

在代码清单 8-7 中，页面导入一个 JavaScript 文件 js7.js，其中自定义两个函数，分别在单击“创建数据库”与“测试连接”时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var db;
// 单击“创建数据库”按钮时调用
```

```
function btnCreateDb_Click() {
    db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024,
    function() {
        $$("pStatus").style.display = "block";
        $$("pStatus").innerHTML = "数据库创建成功!";
    });
}
// 单击“测试连接”按钮时调用
function btnTestConn_Click() {
    if (db) {
        $$("pStatus").style.display = "block";
        $$("pStatus").innerHTML = "数据库连接成功!";
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-7 所示。

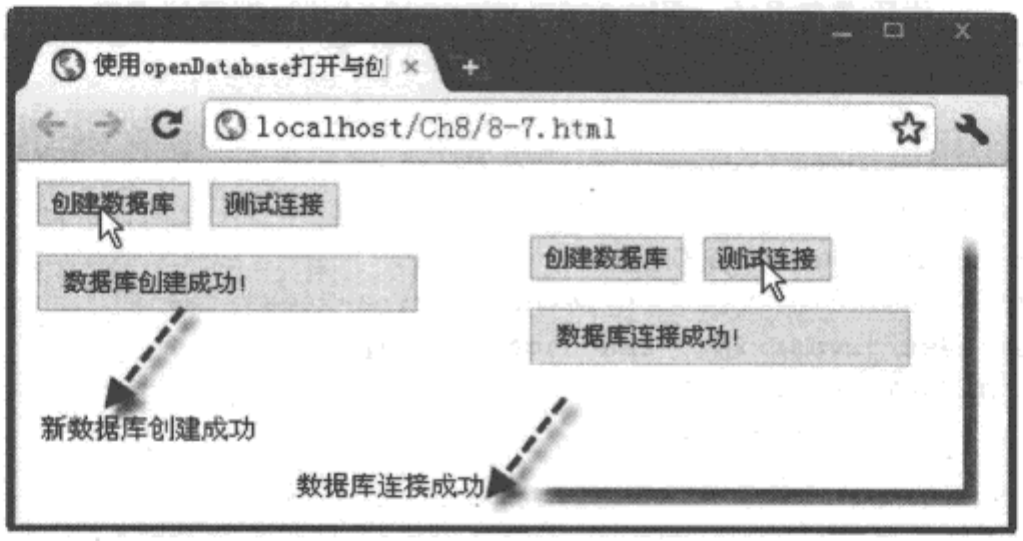


图 8-7　使用 openDatabase 打开与创建数据库

4. 源码分析

在本实例的 JavaScript 代码中，首先定义了一个全局性变量“db”，用于保存打开的数据库对象。当用户单击“创建数据库”按钮时，调用自定义的函数 btnCreateDb_Click()，在该函数中，创建或打开了一个名为“Student”，版本号为“1.0”的 2MB 的数据库对象；如果创建成功，则执行回调函数，在回调函数中显示执行成功的提示信息。

单击“测试连接”按钮时，调用另外一个自定义的函数 btnTestConn_Click()，在该函数中，直接根据全局变量“db”的状态，显示与数据库的连接是否正常的提示信息。

8.3.2　执行事务

当打开 / 创建数据库后，就可以使用数据库对象中的 transaction 方法执行事务处理。每一个事务处理请求都作为数据库的独立操作，这有效地避免在处理数据时发生冲突。其调用的

语法格式如下：

```
transaction(TransCallback, ErrorCallback, SuccessCallback);
```

其中，参数 TransCallback 表示事务回调函数，可以写入需要执行的 SQL 语句；参数 ErrorCallback 表示执行 SQL 语句出错时的回调函数，参数 SuccessCallback 表示执行 SQL 语句成功时的回调函数。

下面通过实例 8-8 介绍使用 transaction() 方法执行事务的过程。

实例 8-8　使用 transaction 方法执行事务

1. 功能描述

在新建的页面中，增加一个“执行事务”的按钮，当用户单击该按钮时，执行一条新建名为 StuInfo 表的 SQL 语句，并将执行后的结果显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-8.html，加入代码如代码清单 8-8 所示。

代码清单 8-8　使用 transaction 方法执行事务

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 transaction 方法执行事务 </title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js8.js"/>
</script>
</head>
<body>
   <input id="btnCreateTrans" type="button" value=" 执行事务 "
          class="inputbtn" onClick="btnCreateTrans_Click();">
   <p id="pStatus"></p>
</body>
</html>
```

在代码清单 8-8 中，页面导入一个 JavaScript 文件 js8.js，其中自定义一个函数，在单击“执行事务”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var db;
// 单击“执行事务”时执行
```

```
function btnCreateTrans_Click() {
    // 创建 / 打开数据库
    db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024);
    if (db) {
        var strSQL = "create table if not exists StuInfo";
        strSQL += "(StuID unique,Name text,Sex text,Score int)";
        db.transaction(function(tx) {
            tx.executeSql(strSQL)
        },
        function() {
            Status_Handle("事务执行出错!");
        },
        function() {
            Status_Handle("事务执行成功!");
        })
    }
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-8 所示。

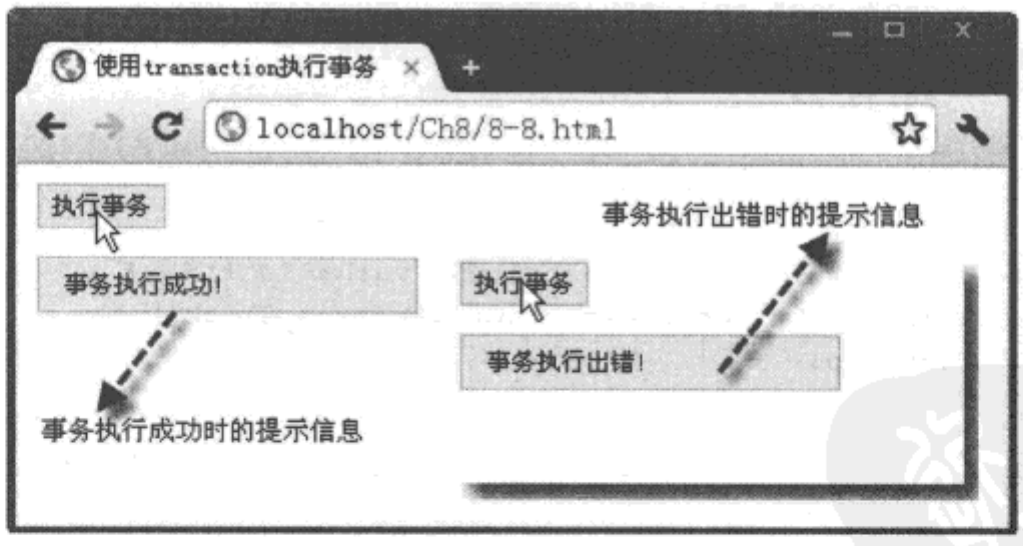


图 8-8　使用 transaction 方法执行事务

4. 源码分析

在本实例中，当用户单击“执行事务”按钮时，将调用自定义函数 btnCreateTrans_Click()。在该函数中，首先使用 openDatabase() 方法打开或创建一个名为“Student”的数据库，如果成功，即数据对象 db 不为空，则定义一个 SQL 语句，通过字符变量 strSQL 保存。该 SQL 语句的功能是：如果不存在，则新建一个名为“StuInfo”的表，该表中包含 4 个字段，

分别为“StuID”、“Name”、“Sex”、“Score”。其中，字段“StuID”为主键，不允许重复，字段“Score”为 int 类型，其他两个字段为字符型。

然后，使用 transaction() 方法执行事务，在该方法的第一个参数中获取变量 strSQL 的值，调用 executeSql 方法执行对应的 SQL 语句。最后，将事务执行过程中的结果，通过 transaction 方法中第二个与第三个回调函数显示在页面中。

注意　*在执行的 SQL 语句中，“if not exists”表示“如果该表不存在”。在新增表时，常常需要添加该语句，以避免增加重名的表。*

8.3.3　插入数据

既然可以通过事务处理的方式执行 SQL 语句创建新表，那么，如果想要给新建的表插入记录，同样也可以通过执行相应的 SQL 语句来实现。实现的关键除调用事务方法外，还要调用一个执行 SQL 语句的方法 executeSql，其调用的格式如下所示：

```
executeSql(strSQL, [Arguments], SuccessCallback, ErrorCallback);
```

其中，参数 strSQL 表示需要执行的 SQL 语句，参数 Arguments 表示语句需要的实参，参数 SuccessCallback 表示 SQL 语句执行成功时的回调函数，参数 ErrorCallback 表示 SQL 语句执行出错时的回调函数。

在使用 executeSql 方法执行 SQL 语句时，允许使用“?”作为语句中的形参，与形参相对应的实参放置在第二个参数 Arguments 中。例如，以下语句是正确的：

```
executeSql("insert into StuInfo values (?,?,?,?)",
           ["1234","张三","男","0"],,);
```

“?”形参的数量必须与对应实参完全一致，如果 SQL 语句中没有“?”形参，第二个参数 Arguments 中不允许有任何内容出错，否则，执行 SQL 语句时将会报错。

下面通过实例 8-9 介绍使用 executeSql() 方法插入记录的过程。

实例 8-9　使用 executeSql 方法插入记录

1. 功能描述

创建一个用于输入学生资料信息的页面，用户可以在页面中输入姓名、性别、总分值后，单击“提交”按钮，将提交的数据信息通过调用 executeSql() 方法插入到 StuInfo 表中，并将执行结果返回显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-9.html，加入代码如代码清单 8-9 所示。

代码清单 8-9　使用 executeSql 方法插入记录

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 executeSql 方法插入记录 </title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js9.js"/>
</script>
</head>
<body onLoad="Init_Data();">
   <p id="pStatus"></p>
   <fieldset>
      <legend>新增学生资料 </legend>
      <span class="spanl">
      学号: <input type="text" readonly="true" id="txtStuID"
                  class="inputtxt" size="10"><br>
      姓名: <input type="text" id="txtName" class="inputtxt"
                  size="15">
      </span>
      <span class="spanr">
      性别: <select id="selSex">
              <option value=" 男 "> 男 </option>
              <option value=" 女 "> 女 </option>
            </select><br>
      总分: <input type="text" id="txtScore" class="inputtxt"
                  size="8">
      </span>
      <p class="btn">
      <input id="btnAdd" type="button" value=" 提交 "
             class="inputbtn" onClick="btnAdd_Click();">
      </p>
   </fieldset>
</body>
</html>
```

在代码清单 8-9 中，页面导入一个 JavaScript 文件 js9.js，其中自定义多个函数，在单击“提交”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var db;
// 单击“提交”按钮时调用
function btnAdd_Click() {
```

```
    //创建/打开数据库
    db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024);
    if (db) {
        var strSQL = "insert into StuInfo values";
        strSQL += "(?,?,?,?)";
        db.transaction(function(tx) {
            tx.executeSql(strSQL,[
              $$("txtStuID").value,$$("txtName").value,
              $$("selSex").value,$$("txtScore").value
          ],
          function(){
             $$("txtName").value="";
             $$("txtScore").value="";
             Status_Handle("1 条记录增加成功!")
          },
          function(tx,ex){
             Status_Handle(ex.message)
          })
        })
    }
}
//自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
//生成指定长度的随机数
function RetRndNum(n) {
    var strRnd = "";
    for (var intI = 0; intI < n; intI++) {
        strRnd += Math.floor(Math.random() * 10);
    }
    return strRnd;
}
//初始化数据
function Init_Data(){
    $$("txtStuID").value=RetRndNum(6);
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-9 所示。

4. 源码分析

在本实例中，当页面加载时，先调用一个自定义的函数 Init_Data()。该函数可以随机生成一个 6 位数的字符，并将该值赋值于页面的“学号”文本框；为了使该学号不能修改，该文本框的属性设置为“只读”。

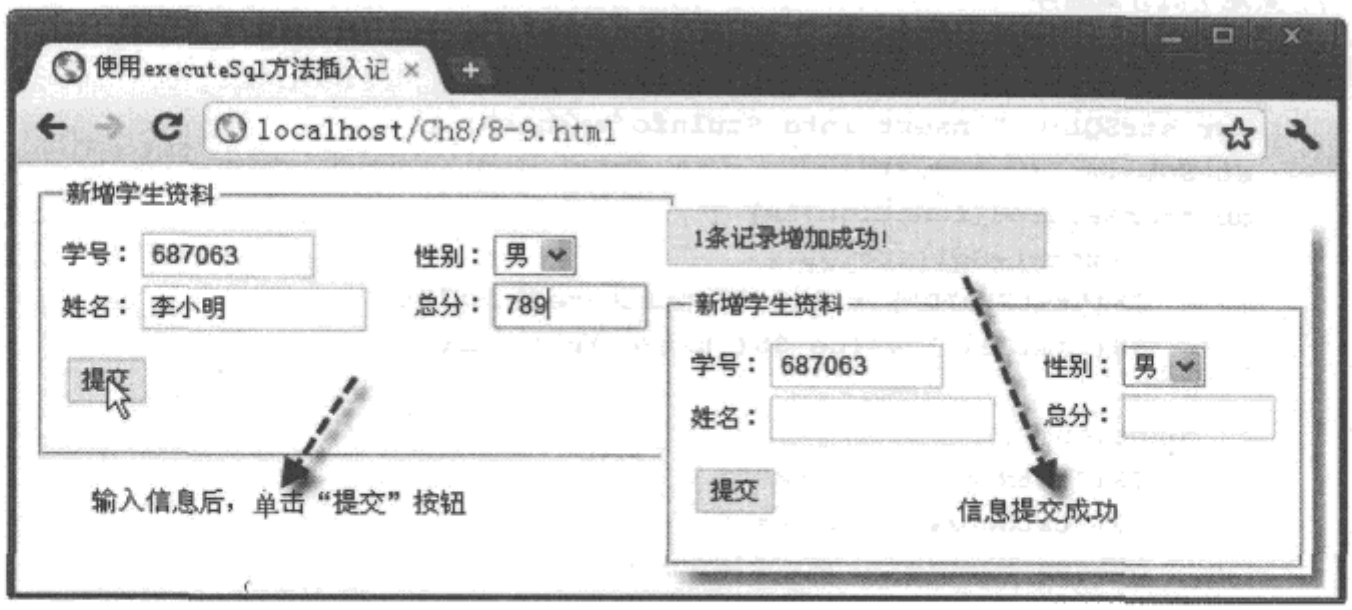


图 8-9　使用 executeSql 方法插入记录

当用户录入完学生信息的其他资料，单击“提交”按钮时，将调用自定义的函数 btnAdd_Click()。该函数先打开数据库，并确定数据库连接正常的条件下，编写向表“StuInfo”插入记录的 SQL 语句，该语句中使用了 4 个“?”形参，分别对应表中的各个字段。

在事务处理过程中，调用 executeSql() 方法执行编写好的 SQL 语句。在执行时获取页面中输入的各项信息值作为实参，传递给 SQL 语句中的形参，从而实现将页面中输入的数据插入“StuInfo”表的功能。

当 SQL 语句执行成功后，将清空页面中原有内容值，并在页面显示“1 条记录增加成功!”字样；反之，如果 SQL 语句执行出错时，将在页面显示出错时，错误对象 ex 返回的错误信息。

8.3.4　数据管理

上一节介绍了使用 executeSql() 方法执行 SQL 语句，从而实现向 WebDB 中“StuInfo”表插入数据的过程。其实，只要符合规范的 SQL 语句都可以通过 executeSql() 方法执行，例如“select”、“update”、“delete”组成的 SQL 语句，都可以带“?”形参，通过 executeSql() 方法执行。实例 8-10 将在实例 8-9 的基础上进行扩展，实现对学生数据信息的增加、查询、更新、删除操作，从而真正实现数据管理的功能。

实例 8-10　使用 executeSql 方法管理数据记录

1. 功能描述

在实例 8-9 的基础上，增加以列表形式展示学生信息的功能；同时，还要实现添加学生资料信息功能，并且能够根据学号查询学生记录。最终能够单击“编辑”链接更新记录，单击“删除”链接删除记录，实现对学生数据的全面管理。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 8-10.html，加入代码如代码清单 8-10 所示。

代码清单 8-10　使用 executeSql 方法管理数据记录

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 executeSql 方法管理数据记录 </title>
<link href="Css/css8.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js10.js"/>
</script>
</head>
<body onLoad="getWebSqlData(0);">
    <p id="pStatus"></p>
    <ul id="ulMessage">
        正在读取数据 ...
    </ul>
    <fieldset id="fstInput">
       <legend id="lgdInput"></legend>
       <span class="spanl">
       学号：<input type="text" readonly="true" id="txtStuID"
                class="inputtxt" size="10"><br>
       姓名：<input type="text" id="txtName" class="inputtxt"
                size="15">
       </span>
       <span class="spanr">
       性别：<select id="selSex">
              <option value="男">男</option>
              <option value="女">女</option>
             </select><br>
       总分：<input type="text" id="txtScore" class="inputtxt"
                size="8">
       </span>
       <p class="btn">
       <input id="btnAdd" type="button" value="提交"
              class="inputbtn" onClick="btnAdd_Click();">
       <input id="btnUpd" type="button" value="更新"
              class="inputbtn" onClick="btnUpd_Click();">
       </p>
    </fieldset>
</body>
</html>
```

在代码清单 8-10 中，页面导入一个 JavaScript 文件 js10.js，其中自定义多个功能性函数，

分别在页面加载和单击“查询”、“提交”、“更新”、“删除”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var db;
// 获取全部或查询数据
function getWebSqlData(s) {
    // 创建 / 打开数据库
    db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024);
    if (db) {
        var strSQL = "select * from StuInfo where StuID<>?";
        if (s > 0) {
            strSQL = "select * from StuInfo where StuID=?";
        }
        db.transaction(function(tx) {
            tx.executeSql(strSQL, [s],
            function(tx, rs) {
                var strHTML = "<li>";
                strHTML += "请输入学号：";
                strHTML += "<input type='text' id='txtSearch' ";
                strHTML += "class='inputtxt' size='14'>";
                strHTML += "<input id='btnSearch' type='button' value='查询' ";
strHTML += "class='inputbtn' onClick='btnSearch_Click();'>";
strHTML += "<input id='btnSearch' type='button' value='增加' ";
strHTML += "class='inputbtn' onClick='AddData();'>";
strHTML += "</li>";
strHTML += "<li class='li_h'>";
strHTML += "<span class='spn_a'>学号</span>";
strHTML += "<span class='spn_a'>姓名</span>";
strHTML += "<span class='spn_c'>性别</span>";
strHTML += "<span class='spn_c'>总分</span>";
strHTML += "<span class='spn_d'>操作</span>";
strHTML += "</li>";
// 内容部分
for (var intI = 0; intI < rs.rows.length; intI++) {
    // 定义主键
    var intId = rs.rows.item(intI).StuID;
    strHTML += "<li class='li_c'>";
    strHTML += "<span class='spn_a'>" + intId + "</span>";
    strHTML += "<span class='spn_a'>";
    strHTML += rs.rows.item(intI).Name ;
    strHTML += "</span>";
    strHTML += "<span class='spn_c'>";
    strHTML += rs.rows.item(intI).Sex;
    strHTML += "</span>";
```

```
                strHTML += "<span class='spn_c'>";
                strHTML += rs.rows.item(intI).Score ;
                strHTML += "</span>";
                strHTML += "<span class='spn_d'>";
                strHTML += "<a href='#' onClick=EditData('"
                strHTML += intId ;
                strHTML += "')>编辑</a>";
                strHTML += " | ";
                strHTML += "<a href='#' onClick=btnDel_Click('"
                strHTML += intId ;
                strHTML += "')>删除</a>";
                strHTML += "</span></li>";
            }
            $$("ulMessage").style.display = "block";
            $$("fstInput").style.display = "none";
            $$("ulMessage").innerHTML = strHTML;
        },
        function(tx, ex) {
            Status_Handle(ex.message)
        })
    })
  }
}
// 单击"提交"按钮时执行
function btnAdd_Click() {
    // 创建/打开数据库
    db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024);
    if (db) {
        var strSQL = "insert into StuInfo values";
        strSQL += "(?,?,?,?)";
        db.transaction(function(tx) {
            tx.executeSql(strSQL, [
               $$("txtStuID").value, $$("txtName").value,
               $$("selSex").value, $$("txtScore").value
           ],
            function() {
                getWebSqlData(0);
            },
            function(tx, ex) {
                Status_Handle(ex.message)
            })
        })
    }
}
// 单击“删除”链接时执行
function btnDel_Click(StuID) {
    // 创建/打开数据库
    db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024);
```

```
        if (db) {
            var strSQL = "delete from StuInfo where StuID=?";
            db.transaction(function(tx) {
                tx.executeSql(strSQL, [StuID],
                function() {
                    getWebSqlData(0);
                },
                function(tx, ex) {
                    Status_Handle(ex.message)
                })
            })
        }
    }
    // 单击"更新"链接时执行
    function btnUpd_Click() {
        // 创建/打开数据库
        db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024);
        if (db) {
            var strSQL = "update StuInfo set Name=?,Sex=?,Score=? where StuID=?";
            db.transaction(function(tx) {
                tx.executeSql(strSQL, [
                   $$("txtName").value, $$("selSex").value,
                   $$("txtScore").value, $$("txtStuID").value
                ],
                function() {
                    getWebSqlData(0);
                },
                function(tx, ex) {
                    Status_Handle(ex.message)
                })
            })
        }
    }
    // 单击"查询"按钮时执行
    function btnSearch_Click() {
        // 获取输入的学号
        var strStuID = $$("txtSearch").value;
        // 获取查询后的结果
        getWebSqlData(strStuID);
    }
    // 单击"增加"按钮时初化数据
    function AddData() {
        $$("ulMessage").style.display = "none";
        $$("fstInput").style.display = "block";
        $$("lgdInput").innerHTML = "增加学生资料";
        $$("txtStuID").value = RetRndNum(6);
        $$("txtName").value = "";
        $$("selSex").value = "";
```

```
    $$("txtScore").value = "";
    $$("btnAdd").style.display = "block";
    $$("btnUpd").style.display = "none";
}
// 单击"更新"按钮时根据 StuID 号获取对应数据
function EditData(StuID) {
    $$("ulMessage").style.display = "none";
    $$("fstInput").style.display = "block";
    $$("lgdInput").innerHTML = "修改学生资料";
    $$("btnUpd").style.display = "block";
    $$("btnAdd").style.display = "none";
    // 创建/打开数据库
    db = openDatabase('Student', '1.0', 'StuManage', 2 * 1024 * 1024);
    if (db) {
        var strSQL = "select * from StuInfo where StuID=?";
        db.transaction(function(tx) {
            tx.executeSql(strSQL, [StuID],
            function(tx, rs) {
                $$("txtStuID").value = rs.rows.item(0).StuID;
                $$("txtName").value = rs.rows.item(0).Name;
                $$("selSex").value = rs.rows.item(0).Sex;
                $$("txtScore").value = rs.rows.item(0).Score;
            },
            function(tx, ex) {
                Status_Handle(ex.message)
            })
        })
    }
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
// 生成指定长度的随机数
function RetRndNum(n) {
    var strRnd = "";
    for (var intI = 0; intI < n; intI++) {
        strRnd += Math.floor(Math.random() * 10);
    }
    return strRnd;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 8-10 所示。

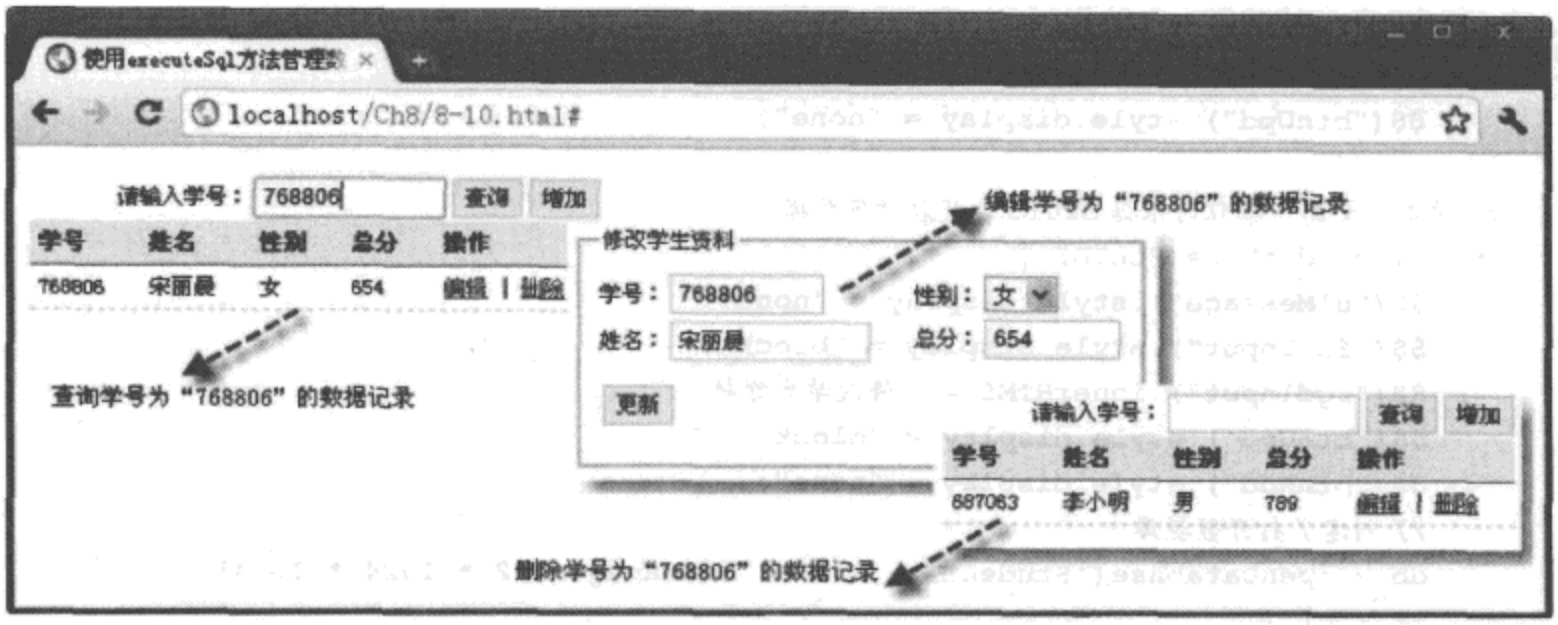


图 8-10　使用 executeSql 方法管理数据记录

4. 源码分析

在本实例中，为了可以根据学号查询数据或显示全部学生数据，当页面加载时，调用一个自定义的函数 getWebSqlData()，根据该函数形参 s 的值编写不同的查询语句。如果 s 值为 0，表示需要获取全部数据，即 SQL 语句为：

```
select * from StuInfo where StuID<>?
```

如果 s 值不为 0，表示需要根据传回的 s 值获取对应的数据，即 SQL 语句为：

```
select * from StuInfo where StuID=?
```

其中“?”形参在调用 executeSql() 方法执行 SQL 语句时，由 s 值替换。当 SQL 语句中有查询字符“select”时，如果执行成功，则可以通过访问成功回调函数中“rs”对象的“rows.length”与“rows.item”属性，遍历整个“rs”数据集对象，采用“rows.item(index). 字段名”的方式获取每一条记录中的各字段内容，并组成一个列表显示在页面中。

在展示数据的列表时，单击“编辑”链接，将调用一个自定义的函数 EditData()，该函数将根据传回的学号值编写一个 SQL 语句。执行成功后，采用“rs.rows.item(0).StuID”的方式，将获取的数据值赋给页面中对应的文本框；当单击“更新”按钮时，调用一个自定义的函数 btnUpd_Click()，在该函数中，编写一个通过学号更新记录的 SQL 语句：

```
update StuInfo set Name=?,Sex=?,Score=? where StuID=?
```

在调用 executeSql() 方法执行 SQL 语句时，将获取页面中输入的各项信息值作为实参，按顺序依次传递给 SQL 语句中的形参，实现按学号更新数据的功能。

在展示数据的列表中，单击“删除”链接时，调用一个自定义的函数 btnDel_Click ()，该函数将根据传回的学号值编写一个 SQL 语句：

```
delete from StuInfo where StuID=?
```

在调用 executeSql() 方法执行 SQL 语句时，将获取的学号值作为实参，传递给 SQL 语句

中的形参，实现按学号删除数据的功能，本实现相对复杂些。详细的实现过程如代码中加粗部分所示。

8.4　本章小结

在 HTML 5 中，本地存储是一个十分重要的内容。本章首先介绍 Web Storage 页面存储的基础知识，包括 sessionStorage 对象和 localStorage 对象存取数据的基本方法；然后，详细介绍了 localStorage 的相关知识要点，以及该对象与 JSON 数据格式的相互转换方法；最后，以理论结合实例的方式，详细介绍了 WebDB 的基础知识，包括如何编写相关 SQL 语句实现对数据的增加、查询、修改、删除功能。

PDG

第9章

HTML 5中的离线应用

本章内容

- 离线应用程序
- 本地缓存的更新及状态检测
- 检测在线状态
- 本章小结

PDG

如今，Web 应用已十分普遍，涉及日常工作中的方方面面，但这些应用有一个共同点，就是需要时刻与服务器保持交互，一旦这种交互不存在或中断，与之相关的应用也将随之停止。为了解决这样的问题，HTML 5 中新增加了一个 API，用于实现本地数据的缓存，而通过调用这个 API，也使开发离线应用成为可能。

9.1 离线应用程序

离线应用程序，简而言之就是一个URL列表，在该列表中包括HTML、CSS、JavaScript、图片及其他资源性文件的 URL 清单。当应用与服务器建立了联系时，浏览器将在本地缓存所有 URL 清单中的资源文件；当应用与服务器失去联系时，浏览器将调用缓存的文件来支撑 Web 应用。

Web 应用之所以在离线时也可以正常访问，是由于 HTML 5 的一种缓存机制 Cache Manifest，可以在线时将对应文件缓存在本地，离线时调用这些本地文件。而需要保存哪些文件，不需要保存哪些文件、在线与离线时需要调用哪些文件，这些都是由 manifest 文件来管理的。为了实现正常访问 manifest 文件，需要在服务器端进行相应的 IIS 配置；在使用 manifest 文件绑定页面后，浏览器与服务器进行数据交互。

9.1.1 manifest 文件简介

为了能在离线状态下继续访问 Web 应用，需要使用 manifest 文件将离线时需要缓存文件的 URL 写入该文件中。当浏览器与服务器建立联系后，浏览器就会根据 manifest 文件所列的缓存清单，将相应的资源文件缓存在本地。

所有创建文本文件的编辑器都可以新建 manifest 文件，只需在保存时，将扩展名设置为“.manifest”即可。例如，新建一个名为 tmp.manifest 文件，其完整的内容如下所示：

```
CACHE MANIFEST
#version 0.0.0

CACHE:
# 带相对路径的资源文件
JS/Js0.js
Css/Css0.css
Images/img0.jpg
Images/img1.png

NETWORK:
# 列出在线时需要访问的资源文件
Index.jsp
Online.do

FALLBACK:
```

```
# 以成对形式列出不可访问文件的替补资源文件
/Project/Index.jsp       /BkProject/Index.jsp
```

代码说明如下：

“CACHE:”标记表示离线时，浏览器需要缓存到本地的服务器资源文件列表。为某个页面编写 manifest 类型文件时，不需要将该页面放入列表中，因为浏览器在进行本地资源缓存时，自动将这个页面进行了缓存。

“NETWORK：”标记表示在线时，需要访问的资源文件列表。这些文件只有在浏览器与服务器之间建立联系时才能访问。如果设置为“*”，则表示除了在“CACHE:”标记中标明需要缓存的文件之外，都不进行本地缓存。

“FALLBACK：”标记表示以成对方式列出不访问文件的替补文件。前者为不可访问的文件，后者为替补文件，即当“/Project/Index.jsp”文件不可访问时，浏览器会尝试访问“/BkProject/Index.jsp”文件。

在编写代码时需要注意以下几点：

1) manifest 类型文件第一行必须为“CACHE MANIFEST”，表明这是一个通过浏览器将服务器资源进行本地缓存的格式文件。

2) 编写注释时要另起一行，并且以“#”开头。

3) manifest 类型文件内容允许重复编写分类标记，即可写多个“CACHE:”标记或另外两种标记。

4) 如果没有找到分类的标记，都视为“CACHE:”标记下的资源文件。

5) 建议通过注释的方式，标明每一个 manifest 类型文件的版本号，以便于更新文件时使用。例如，上述代码中的“#version 0.0.0”就是内定的版本号。

创建完 manifest 类型文件后，就可以通过页面中 <html> 元素的“manifest”属性，将页面与 manifest 类型文件绑定起来。这样，在浏览器中查看页面时，自动将 manifest 类型文件中所涉及的资源文件缓存在本地，其绑定代码如下所示：

```
<html manifest="tmp.manifest">
```

9.1.2　配置 IIS 服务器

创建完 manifest 文件，并将该文件与 Web 页进行了绑定，还无法实现 Web 页面的离线访问，还需要让服务器支持“.mainfest”扩展名的文件，否则，服务器无法读取 mainfest 类型的文件。下面以 Windows 版服务器为例，介绍配置 IIS 选项，使服务器支持 manifest 类型文件的方法。

步骤 1　在 IIS 中，选中“默认网站”或其他站点名称，单击右键，在弹出的“属性”窗口中选择“HTTP 头”选项卡，在该选项卡中，单击“MIME”映射区域中的“文件类型”按钮。其操作界面如图 9-1 所示。

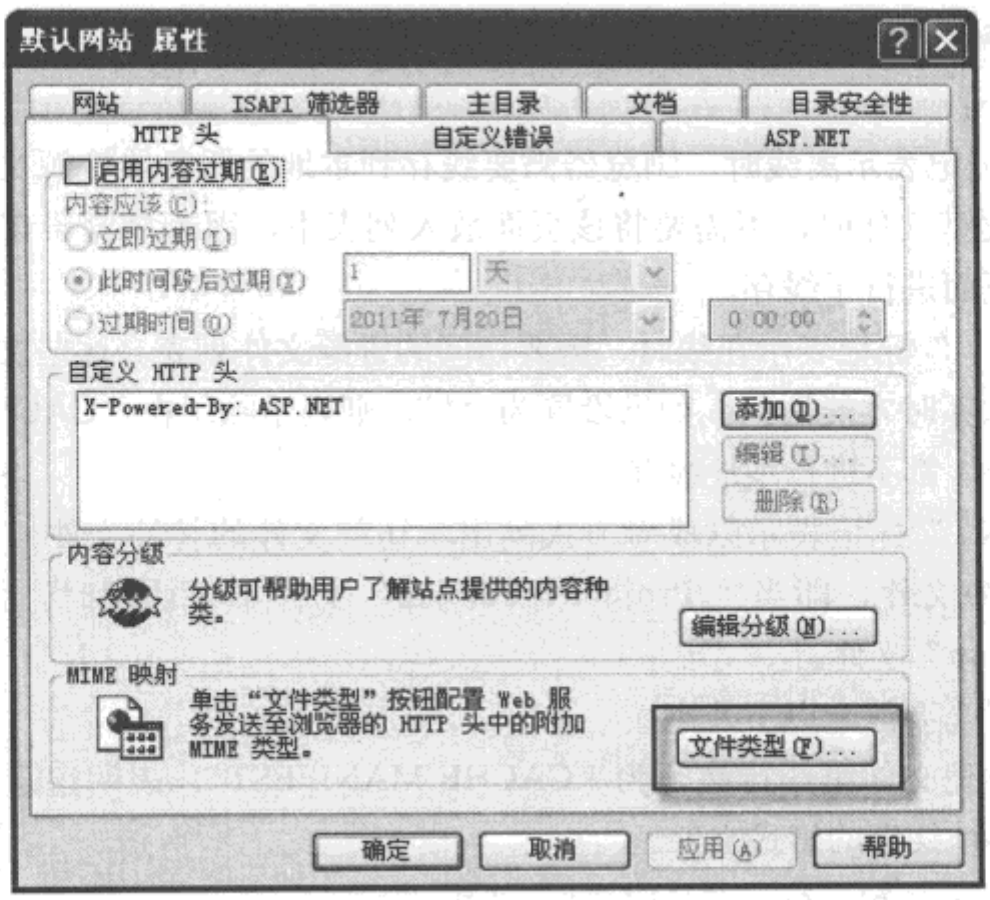


图 9-1　在 IIS 中选择“HTTP 头”选项卡的界面

步骤 2　在“HTTP 头”选项卡中单击“文件类型”按钮，在弹出的“文件类型”对话框中单击“新类型(N)...”按钮，会弹出新建文件类型的对话框；在对应的“关联扩展名”文本框中输入“.manifest”，在“内容类型”文本框中输入“text/cache-manifest”。操作界面如图 9-2 所示。

步骤 3　创建新“文件类型”时，输入“关联扩展名”与“内容类型”后，单击“确定”按钮，便完成了扩展名为“.manifest”文件类型的创建，其实现的界面如图 9-3 所示。

图 9-2　在 IIS 中新建“文件类型”的界面

图 9-3　在 IIS 中完成新建“.manifest”文件类型的界面

通过以上三个步骤，为 IIS 创建了一个扩展名为“.manifest”的文件类型，使服务器能够支持 manifest 文件，实现对应站点下 Web 页离线访问的功能。

9.1.3　离线应用的开发过程

通过前面章节的介绍，使我们明确了要开发一个离线的 Web 应用，需要具备以下几个条件：

1) 编写一个 manifest 类型文件，列出需要通过浏览器缓存至本地的资源性文件。

2) 开发一个 Web 页面，通过 <html> 元素的“manifest”属性将 manifest 文件与页面绑定。

3) 对服务器端进行配置，使其能读取 manifest 类型的文件。

当以上三个条件都具备时，就可以实现 Web 页的离线访问功能。接下来通过一个简单的实例 9-1 详细说明其实现的过程。

实例 9-1　开发一个简单的离线应用

1. 功能描述

创建一个 HTML 页面，浏览该页面时，通过编写 JavaScript 代码获取服务器时间，并按照时间的格式，动态显示在页面中；当中断与服务器的联系再次浏览该页面时，仍然可以在页面中动态地显示时间。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-1.html，加入代码如代码清单 9-1 所示。

代码清单 9-1　开发一个简单的离线应用

```
<!DOCTYPE html>
<html manifest="ce1.manifest">
<head>
<meta charset="utf-8" />
<title>开发一个简单的离线应用</title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js1.js"/>
</script>
</head>
<body>
 <fieldset>
   <legend>简单离线示例</legend>
     <output id="time">正在获取时间 ...</output>
 </fieldset>
</body>
</html>
```

在代码清单 9-1 中，页面导入一个 JavaScript 文件 js1.js，其中自定义两个函数，分别用于获取系统时间与格式化显示的时间。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 获取当前格式化后的时间并显示在页面上
function getCurTime(){
   var dt=new Date();
   var strHTML=" 当前时间是 ";
   strHTML+=RuleTime(dt.getHours(),2)+":"+
            RuleTime(dt.getMinutes(),2)+":"+
            RuleTime(dt.getSeconds(),2);
   $$("time").value=strHTML;
}
// 转换时间显示格式
function RuleTime(num, n) {
    var len = num.toString().length;
    while(len < n) {
        num = "0" + num;
        len++;
    }
    return num;
}
// 定时执行
setInterval(getCurTime,1000);
```

在实例 9-1 中，页面导入一个 CSS 文件 css9.css，该文件是一个样式文件，用于控制将获取的时间显示在页面中的样式。其实现的代码如下所示：

```
@charset "utf-8";
/* CSS Document */
body {
   font-size:12px
}
fieldset{
   padding:10px;
   width:285px;
   float:left
}
output{
   font-size:14px;
   font-family:Verdana, Geneva, sans-serif;
   padding-left:72px
}
```

在实例 9-1 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的

文件 ce1.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.0.1
CACHE:
Js/js1.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-4 所示。

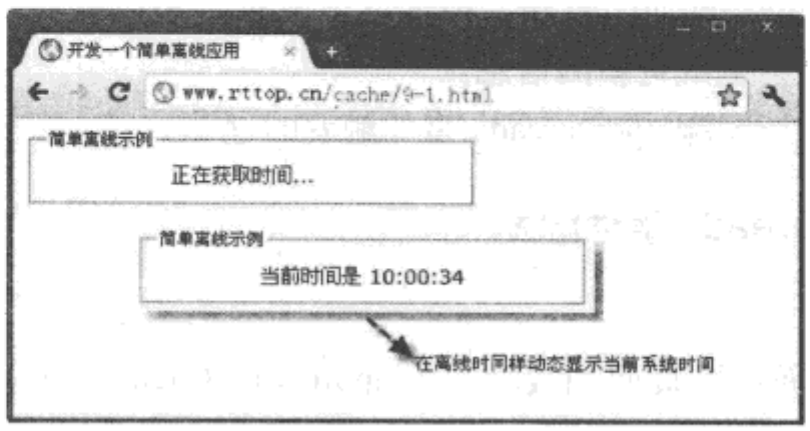


图 9-4　开发一个简单的离线应用

4. 源码分析

在本实例中，通过与页面绑定的“manifest”类型文件 ce1.manifest，缓存了 3 个资源文件，分别为 js1.js、css9.css 和 9-1.html 页面本身。因为使用了本地缓存，使浏览器与服务器之间的数据交互按照如下步骤进行：

1）浏览器：请求访问 9-1.html 页面。

2）服务器：返回 9-1.html。

3）浏览器：解析返回的 9-1.html 页面，请求服务器返回 9-1.html 页面所包含的全部资源性文件，包括 ce1.manifest 文件。

4）服务器：返回浏览器所请求的所有资源文件。

5）浏览器：解析返回的 ce1.manifest 文件，请求返回 URL 清单中的资源文件。

6）服务器：再次返回 URL 清单中的资源文件。

7）浏览器：更新本地缓存，将新获取的 URL 清单中的资源文件更新至本地缓存中。在进行更新过程中，将触发一个 onUpdateReady 事件，表示本地缓存更新完成。

8）浏览器再次查看访问 9-1.html 页面时，如果 ce1.manifest 文件没有发生变化，将直接调用本地的缓存，响应用户的请求，从而实现离线访问页面的功能。

9.2　本地缓存的更新及状态检测

applicationCache 对象表示本地缓存，在开发离线应用时，通过调用该对象的

onUpdateReady 事件，能监测到本地缓存是否完成更新。有两种手动更新本地的缓存的方法：一种是在 onUpdateReady 事件中，调用 swapCache() 方法；另外一种就是直接调用 applicationCache 对象的 update() 方法。此外，当本地缓存更新时，可以调用 applicationCache 对象的其他事件，实时监测本地缓存更新的状态。

9.2.1 updateready 事件

在实例 9-1 中，如果与页面绑定的 manifest 文件 ce1.manifest 内容发生变化，将引起本地缓存的更新，从而触发 updateready 事件。根据这一特征，可以在 updateready 事件中编写代码，实时监测本地缓存是否完成更新的信息。

下面通过实例 9-2 介绍监测 updateready 事件触发的过程。

实例 9-2 监测 updateready 事件触发

1. 功能描述

创建一个 HTML 页面，当页面加载时，为 applicationCache 对象添加一个 updateready 事件，用于监测本地缓存是否发生改变。一旦更新本地缓存，触发 updateready 事件，将在页面显示“正在触发 updateready 事件 ...”字样。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-2.html，加入代码如代码清单 9-2 所示。

代码清单 9-2 监测 updateready 事件触发

```
<!DOCTYPE html>
<html manifest="ce2.manifest">
<head>
<meta charset="utf-8" />
<title>监测 updateready 事件触发 </title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js2.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>监测 updateready 事件触发过程 </legend>
     <p id="pStatus"></p>
 </fieldset>
</body>
</html>
```

在代码清单 9-2 中，页面导入一个 JavaScript 文件 js2.js，其中自定义函数 pageload()，在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
   window.applicationCache.addEventListener("updateready",function() {
        $$("pStatus").style.display="block";
        $$("pStatus").innerHTML = " 正在触发 updateready 事件 ...";
   },true);
}
```

在代码清单 9-2 中，页面导入的 CSS 文件 css9.css，在实例 9-1 的基础上，又添加了用于控制实例 9-2 页面的部分样式代码。其实现的代码如下所示：

```
@charset "utf-8";
/* CSS Document */
... 省略实例 9-1 中已列出的代码
/* 示例 2*/
#pStatus{
   display:none;
   border:1px #ccc solid;
   width:158px;
   background-color:#eee;
   padding:6px 12px 6px 12px;
   margin-left:2px
}
```

在代码清单 9-2 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的文件 ce2.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.0.2
CACHE:
Js/js2.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-5 所示。

4. 源码分析

在本实例中，当与页面绑定的服务端 manifest 文件 ce2.manifest 内容发生改变时，才会触发本地缓存的更新。如果本地缓存更新完成，将触发设置好的 updateready 事件，显示“正在触发 updateready 事件 ...”字样。

注意　即使完成了本地缓存的更新，当前页面也不会发生任何变化，需要重新打开该页面或刷新当前页，才能执行本地缓存更新后的页面效果。

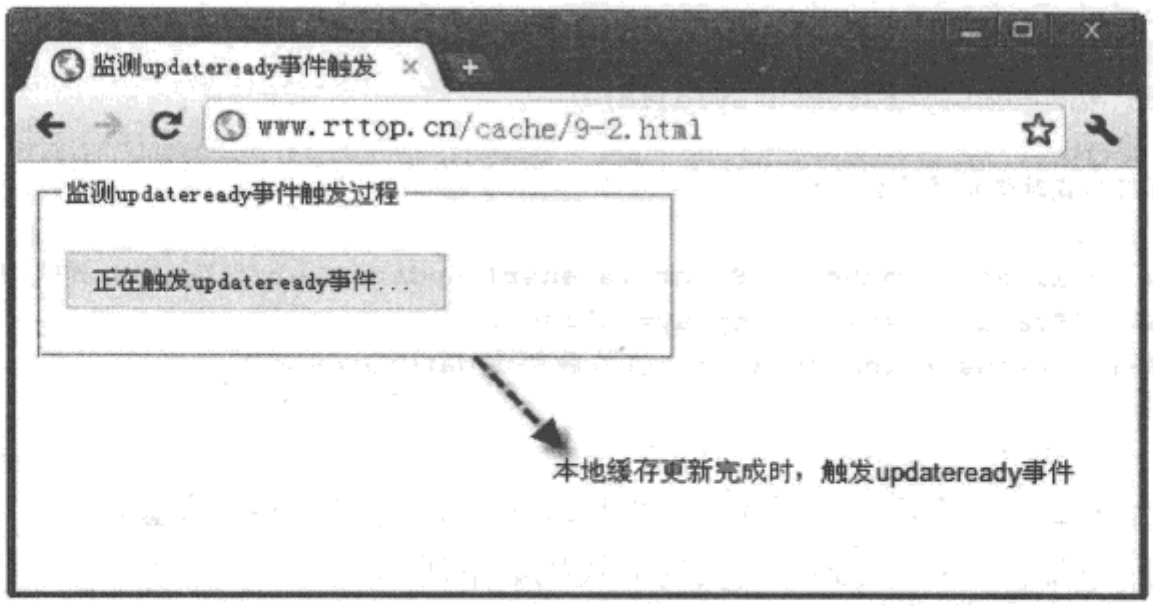


图 9-5　监测 updateready 事件触发过程

9.2.2　update 方法

上一节介绍了 updateready 事件，该事件的触发标志着本地缓存进行了更新。除此之外，还可以通过手动的方法更新本地的缓存。在 HTML 5 中，调用 applicationCache 对象中的 update 方法，可以手动更新本地缓存，其调用格式如下所示：

```
window.applicationCache.update()
```

如果有可更新的本地缓存，调用该方法将可以对本地缓存进行更新。那么如何检测是否有可更新的本地缓存，除通过 updateready 事件监测外，还可以调用 applicationCache 对象中的“status”属性。该属性有多个值，当该值为“4”时，表示有可更新的本地缓存。

下面通过实例 9-3 介绍使用 update() 方法更新本地缓存的过程。

实例 9-3　使用 update 方法更新本地缓存

1. 功能描述

创建一个 HTML 页面，当页面加载时，检测是否有可更新的本地缓存。如果存在，则显示“手动更新”按钮，单击该按钮后，将更新本地的缓存，同时，在页面中显示“手动更新完成！”字样。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-3.html，加入代码如代码清单 9-3 所示。

代码清单 9-3　使用 update 方法更新本地缓存

```
<!DOCTYPE html>
<html manifest="ce3.manifest">
<head>
```

```
<meta charset="utf-8" />
<title>使用 update() 方法更新本地缓存 </title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js3.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>检测是否有更新并手动更新缓存 </legend>
     <p id="pStatus"></p>
     <p id="pShow">
       <input id="btnUpd" value=" 手动更新 " type="button"
              class="inputbtn" onClick="btnUpd_Click()"/>
     </p>
 </fieldset>
</body>
</html>
```

在代码清单 9-3 中，页面导入一个 JavaScript 文件 js3.js，其中自定义多个函数，在页面加载与单击“手动更新”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 检测 manifest 文件是否有更新
function pageload() {
    if (window.applicationCache.status == 4) {
        Status_Handle(" 找到可更新的本地缓存 !");
        $$("pShow").style.display = "block";
    }
}
// 单击 " 手动更新 " 按钮时调用
function btnUpd_Click() {
    window.applicationCache.update();
    Status_Handle(" 手动更新完成 !");
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

在代码清单 9-3 中，页面导入的 CSS 文件 css9.css，在实例 9-2 的基础上，又添加了用于控制实例 9-3 页面的部分样式代码。其实现的代码如下所示：

```
@charset "utf-8";
/* CSS Document */
...省略实例 9-1、实例 9-2 中已列出的代码
/* 示例 3*/
.inputbtn {
   border:solid 1px #ccc;
   background-color:#eee;
   line-height:18px;
   font-size:12px
}
#pShow{
   display:none
}
```

在代码清单 9-3 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的文件 ce3.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.3.0
CACHE:
Js/js3.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-6 所示。

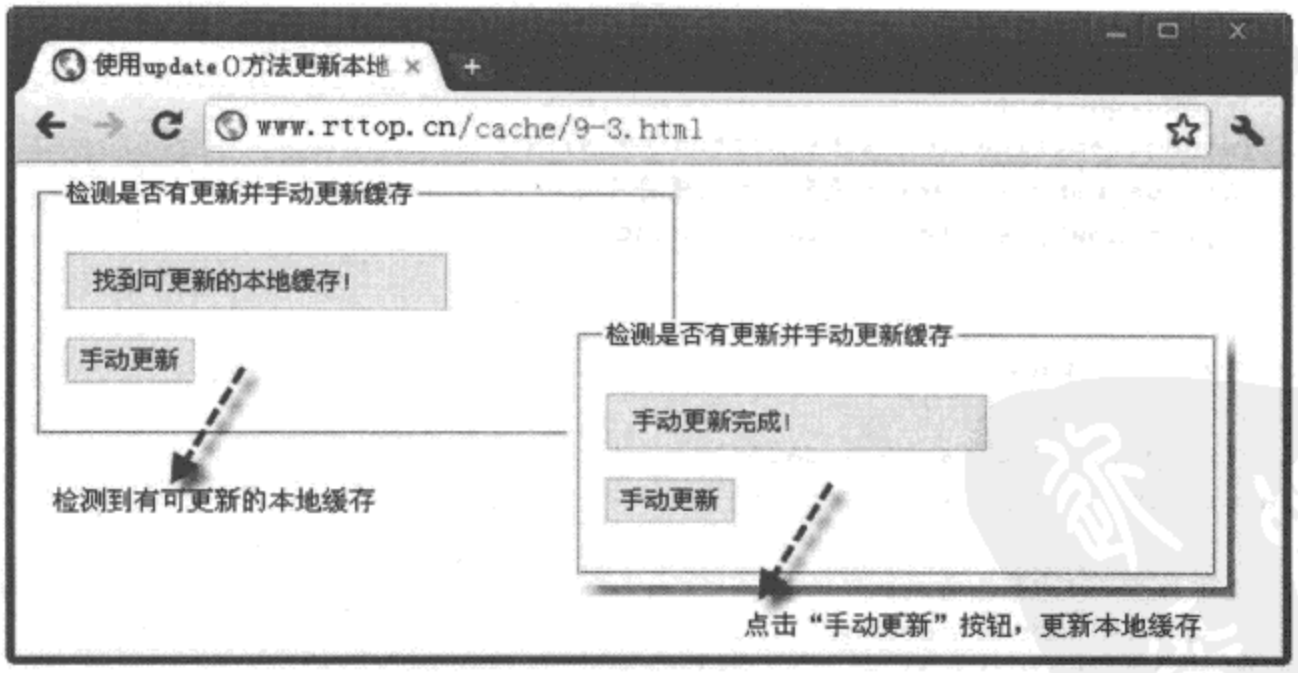


图 9-6　使用 update 方法更新本地缓存

4. 源码分析

在本实例中，当页面加载时，调用自定义的函数 pageload()。在该函数中，首先通过 applicationCache 对象中的“status”属性检测是否有可更新的本地缓存，如果存在，

即该值为“4”时，显示“手动更新”按钮。如果单击“手动更新”按钮，将触发按钮的“onClick”事件。在该事件中，调用自定义的函数 btnUpd_Click()，该函数通过使用 applicationCache 对象中的 update() 方法，更新了本地的缓存，并在页面中显示“手动更新完成！”的字样。

需要说明的是，本例中 applicationCache 对象的“status”属性值等于“4”，表示本地有可以更新的本地缓存，除此之外，该属性还有其他值如表 9-1 所示。

表 9-1　status 属性返回值及描述

返回值	描　述
0	表示空值，说明本地缓存不存在，或不可用
1	表示空闲，说明当前的本地缓存是最新的，无需更新
2	表示检测，说明正在核查 manifest 文件的状态，确定是否发生了变化
3	表示下载，说明已经确定 manifest 文件发生了变化，并且正在下载中
4	表示状态，说明本地的缓存文件已经更新，刷新页面或手动更新即可
5	表示废弃，说明当前的本地缓存已被删除或不可用

9.2.3　swapCache 方法

与上一节中的 update() 方法相同，swapCache() 方法的作用也是更新本地缓存，但与 update() 方法有两个不同之处：

1) **更新本地缓存的时间不一样**：swapCache() 方法早于 update() 方法将本地的缓存进行更新，swapCache() 方法是将本地缓存立即更新。

2) **触发事件不一样**：swapCache() 方法必须在 updateready 事件中才能调用，而 update() 方法可以随时调用。

无论是使用哪种方法，当前执行的页面都不会立即显示本地缓存更新后的页面效果，都要重新加载一次或手动刷新页面，才能发挥作用。

下面通过实例 9-4 介绍使用 swapCache() 方法更新本地缓存的过程。

实例 9-4　使用 swapCache 方法更新本地缓存

1. 功能描述

创建一个 HTML 页面，当页面加载时，检测是否有可更新的本地缓存。如果存在，则调用 swapCache() 方法立即更新本地缓存，如果更新成功，则在页面中显示“本地缓存更新完成！”字样，同时，自动刷新当前页面，展示更新后的效果。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-4.html，加入代码如代码清单 9-4 所示。

代码清单 9-4　使用 swapCache() 方法更新本地缓存

```
<!DOCTYPE html>
<html manifest="ce4.manifest">
<head>
<meta charset="utf-8" />
<title>使用 swapCache() 方法更新本地缓存 </title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js4.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>检测是否有更新并立即更新缓存 </legend>
     <p id="pStatus"></p>
 </fieldset>
</body>
</html>
```

在代码清单 9-4 中，页面导入一个 JavaScript 文件 js4.js，其中自定义函数 pageload()，在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 在添加“updateready”事件中执行 swapCache() 方法，
function pageload() {
   window.applicationCache.addEventListener("updateready",function() {
      Status_Handle("找到可更新的本地缓存！");
      window.applicationCache.swapCache();
      Status_Handle("本地缓存更新完成！");
      location.reload();
   },false);
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

在代码清单 9-4 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的文件 ce4.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.4.0
CACHE:
Js/js4.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-7 所示。

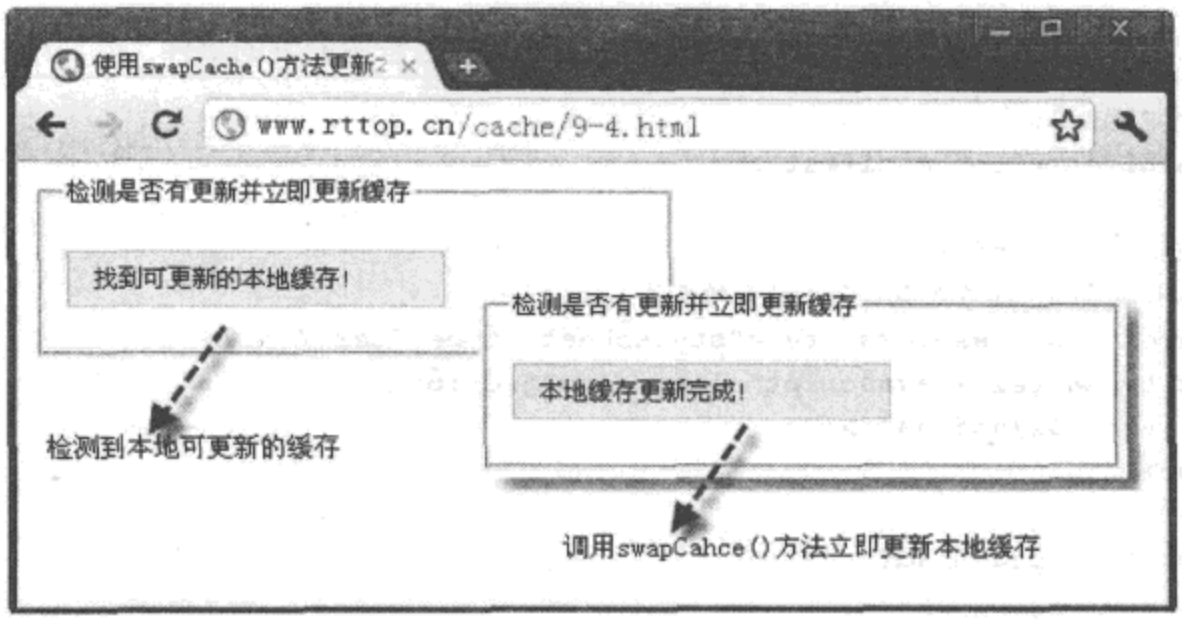


图 9-7　使用 swapCache() 方法更新本地缓存

4. 源码分析

在本实例中，如果本地有可更新的缓存，那么将触发 updateready 事件。这也是检测本地是否有可更新缓存的另外一个方法，当触发该事件时，调用 applicationCache 对象中的 swapCache() 方法，更新本地已有的缓存。

使用 swapCache() 方法更新本地缓存的好处在于：立即可以实现本地缓存的更新。但是，如果需要更新的缓存列表较多时，可能耗时很大，甚至会锁住浏览器，这时，需要在更新过程中，通过获取更新文件的进度信息，给客户进行提示，以优化用户的 UI 体验。

如何获取本地文件更新过程中的更多信息，将在接下来的章节中进行介绍。

9.2.4　更新本地缓存时触发的其他事件

上一节介绍了触发 updateready 事件时，通过 swapCache() 方法即时更新本地缓存的过程。其实，当浏览器加载一个离线应用时，还会触发许多其他的事件。例如，在更新本地缓存时，除触发 updateready 事件外，还将触发 downloading、progress、cached 等事件。

下面通过实例 9-5 介绍检测离线应用在加载过程中触发的事件。

实例 9-5　检测离线应用在加载过程中触发的事件

1. 功能描述

创建一个 HTML 页面，在页面加载时，为 applicationCache 对象添加各种可能触发的事件。当触发某一事件时，将在页面中显示该事件的名称。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-5.html，加入代码如代码清单 9-5 所示。

代码清单 9-5　检测离线应用在加载过程中触发的事件

```
<!DOCTYPE html>
<html manifest="ce5.manifest">
<head>
<meta charset="utf-8" />
<title>检测离线应用在加载过程中触发的事件</title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js5.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>检测离线应用在加载过程中触发的事件</legend>
     <p id="pStatus"></p>
 </fieldset>
</body>
</html>
```

在代码清单 9-5 中，页面导入一个 JavaScript 文件 js5.js，在该文件的自定义函数 pageload() 中添加了多个可能触发的事件，在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
   window.applicationCache.addEventListener("checking",function() {
           Status_Handle("正在检测是否有更新 ...");
   },true);
   window.applicationCache.addEventListener("downloading",function() {
           Status_Handle("正在下载可用的缓存 ...");
   },true);
   window.applicationCache.addEventListener("noupdate",function() {
           Status_Handle("没有最新的缓存更新！");
   },true);
   window.applicationCache.addEventListener("progress",function() {
           Status_Handle("本地缓存正在更新中 ...");
   },true);
   window.applicationCache.addEventListener("cached",function() {
           Status_Handle("本地缓存已更新成功！");
   },true);
```

```
    window.applicationCache.addEventListener("error",function() {
            Status_Handle(" 本地缓存更新时出错 !");
    },true);
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

在代码清单 9-5 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的文件 ce5.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.5.0
CACHE:
Js/js5.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-8 所示。

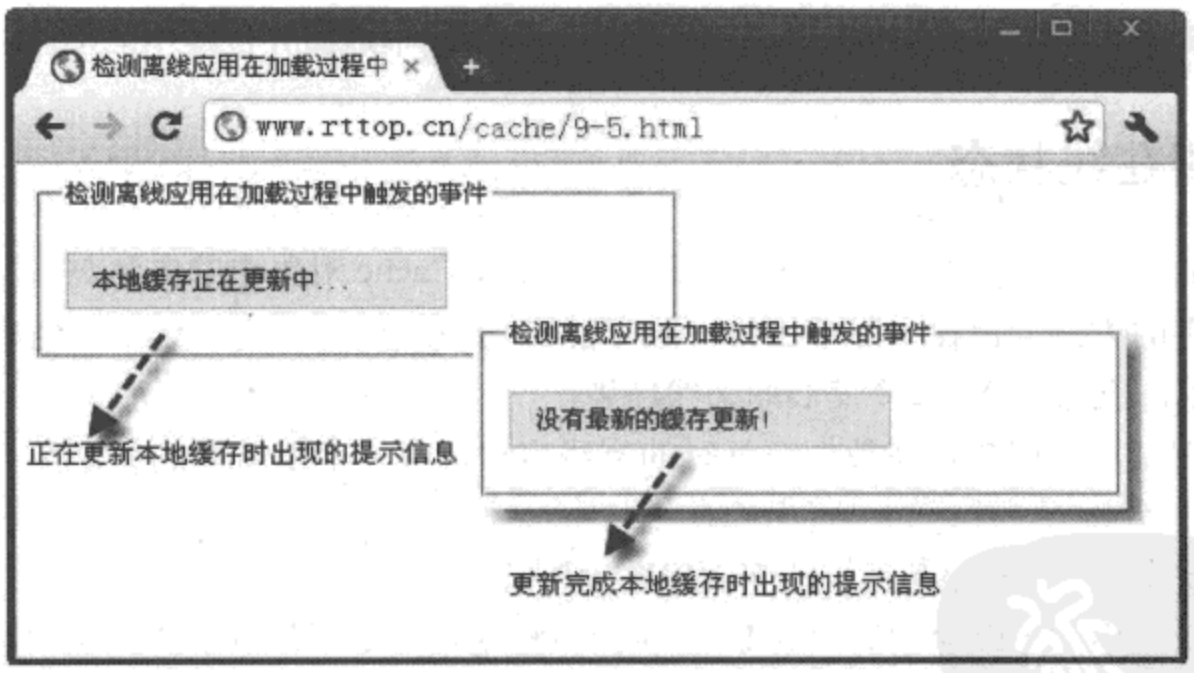


图 9-8　检测离线应用在加载过程中触发的事件

4. 源码分析

在本实例中，为 applicationCache 对象添加各种可能触发的事件。首次在浏览器中加载一个离线应用时所触发的整个事件过程如下：

1）浏览器：请求访问页面 9-5.html。

2）服务器：返回页面 9-5.html。

3）浏览器：解析页面头部时，发现 manifest 属性，触发 checking 事件，检测属性对应的 manifest 类型文件是否存在，如果不存在，则触发 error 事件。

4）浏览器：解析返回的页面 9-5.html，请求服务器返回页面中所有的资源文件，包括 manifest 文件。

5）服务器：返回请求的所有资源文件。

6）浏览器：处理 manifest 文件，请求服务器返回所有 manifest 文件中所要求缓存在本地的文件，即使是第一次请求过的文件，也要重新请求一次。

7）服务器：返回所请求的需要缓存至本地的资源文件。

8）浏览器：下载资源文件时触发 downloading 事件，如果文件很多，将间歇性地触发 progress 事件，表示正在下载过程中；下载完成后，触发 cached 事件，表示下载完成并存入缓存中。

如果没有修改 manifest 文件，而再次通过浏览器加载页面 9-5.html 时，将重复执行上述过程中的第 1 ～ 5 步。而在第 6 步时，将检测是否有可更新的本地缓存，如果无，则触发 noupdate 事件，表示没有最新的缓存可更新；如果有，则触发 updateready 事件，表示更新已下载完成，刷新页面或手动更新就可以展示本地缓存更新后的效果。

因此，当本地缓存更新的资源文件很多时，可以调用文件在下载时的 progress 事件，动态显示已更新的总量与未更新数量，从而优化用户在更新本地缓存过程中的 UI 体验。

9.3 检测在线状态

在前面的章节中，详细介绍了通过调用 applicationCache 对象中的方法与事件，实现手动更新本地缓存的方法。但在实际应用中，除了静态页面的离线应用，更多的是离线时用户数据的交互应用，这也是 Web 2.0 发展的必然趋势。

要开发浏览器与服务器在数据交互时的离线应用，有一个很重要的标志必须获取，那就是应用的在线状态。只有检测出页面的在线状态，才能在离线后，将数据保存在本地，上线时，再将本地数据同步至服务器，从而实现离线数据交互功能。

在 HTML 5 中，可以通过访问 onLine 属性与触发 online 事件分别检测应用是否在线，本节将分别进行详细的介绍。

9.3.1 onLine 属性

onLine 属性是一个布尔值，当该值为 true 时，表示在线，否则，表示离线；如果当前的网络状态发生了变化，该属性值就会随之发生变动。因此，可以通过获取该值来检测当前网络的状态，确定应用是否在线或离线，从而编写不同的代码。

下面通过实例 9-6 介绍通过 onLine 属性检测网络的当前状态的过程。

实例 9-6　通过 onLine 属性检测网络的当前状态

1. 功能描述

创建一个 HTML 页面，在页面加载时调用 onLine 属性。如果该属性的值为 true，则在页面中显示“在线”字样，否则显示“离线”字样。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-6.html，加入代码如代码清单 9-6 所示。

代码清单 9-6　通过 onLine 属性检测网络的当前状态

```
<!DOCTYPE html>
<html manifest="ce6.manifest">
<head>
<meta charset="utf-8" />
<title>通过 onLine 属性检测网络的当前状态 </title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js6.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>通过 onLine 属性检测网络的当前状态 </legend>
     <p id="pStatus"></p>
 </fieldset>
</body>
</html>
```

在代码清单 9-6 中，页面导入一个 JavaScript 文件 js6.js，在该文件的自定义函数 pageload() 中，根据“onLine”属性检测当前网络的状态，在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
    if (navigator.onLine) {
        Status_Handle("在线");
    } else {
        Status_Handle("离线");
    }
}
// 自定义显示执行过程中状态的函数
```

```
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

在代码清单 9-6 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的文件 ce6.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.6.0
CACHE:
Js/js6.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-9 所示。

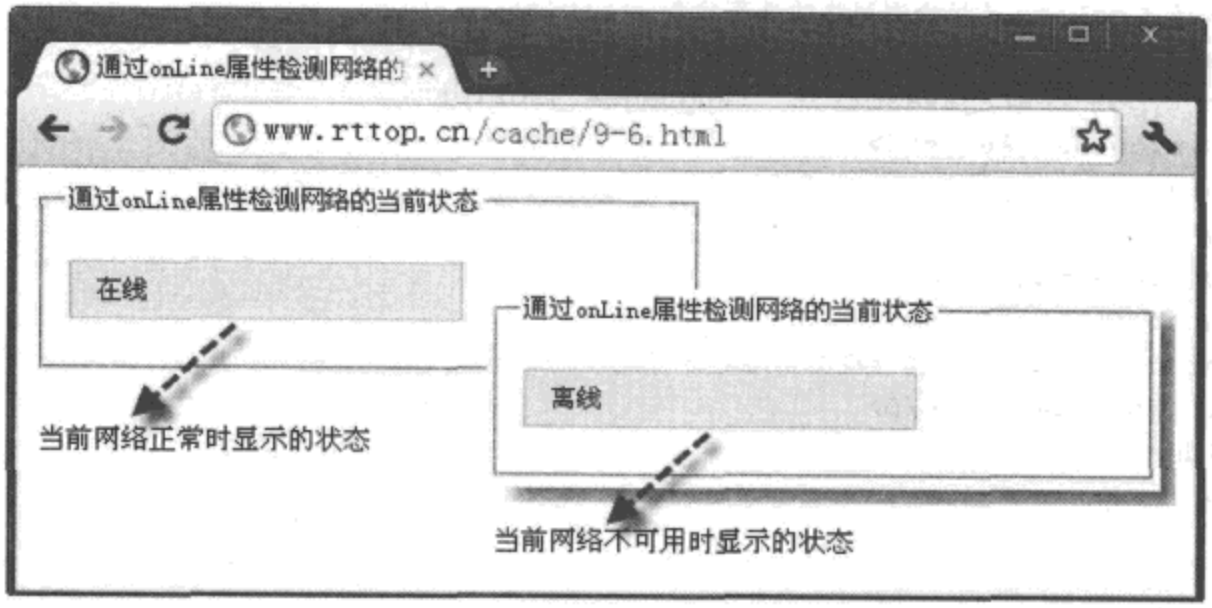


图 9-9　通过 onLine 属性检测网络当前状态

4. 源码分析

在本实例中，通过调用 navigator 对象的“onLine”属性来检测当前浏览器的在线模式，如果为“true”，则表示在线，否则表示离线。

navigator 是一个独立的对象，用于向用户返回浏览器和操作系统的相关信息，所有信息都是以 navigator 对象的属性方式进行调用的。在日常的代码开发过程中，除使用“onLine”属性检测当前浏览器的在线模式外，其他 navigator 对象常用的属性如表 9-2 所示。

9.3.2　online 与 offline 事件

使用 navigator 对象中的“onLine”属性来检测当前网络的状态还远远不够，因为该属性有滞后性，不能及时反馈当前网络的变化状态。在 HTML 5 中，还可以调用 online 与 offline 事件及时侦测网络在线与离线的状态。该事件是基于 body 对象触发的，以冒泡的方式传递给

document、window 对象，因此，这两个事件可以准确及时地捕获当前浏览器的在线状态。

表 9-2　navigator 对象常用属性

属性名称	描　　述
appCodeName	返回当前浏览器的内部代码名称
appName	返回当前浏览器的正式名称
appVersion	返回当前浏览器的版本号
language	返回当前浏览器所支持的语言
systemLanguage	返回用户操作系统支持的默认语言
vender	返回浏览器厂商的名称

下面通过实例 9-7 介绍利用 online 与 offline 事件检测网络的当前状态的过程。

实例 9-7　通过 online 与 offline 事件检测网络的当前状态

1. 功能描述

创建一个 HTML 页面，在页面浏览的过程中，如果用户有切换网络连接状态的操作，那么，页面将自动触发 online 或 offline 事件。如果是连接网络，则在页面中显示“在线”字样，否则显示“离线”字样。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-7.html，加入代码如代码清单 9-7 所示。

代码清单 9-7　通过 online 与 offline 事件检测网络的当前状态

```
<!DOCTYPE html>
<html manifest="ce7.manifest">
<head>
<meta charset="utf-8" />
<title>通过 online 与 offline 事件检测网络的当前状态 </title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js7.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>通过 online 与 offline 事件检测网络的当前状态 </legend>
     <p id="pStatus"></p>
 </fieldset>
</body>
</html>
```

在代码清单 9-7 中，页面导入一个 JavaScript 文件 js7.js，在该文件中，当页面加载时，调用函数 pageload() 为 window 对象添加“online”与“offline”事件。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
    window.addEventListener("online",function() {
        Status_Handle("在线");
    },false);
    window.addEventListener("offline",function() {
        Status_Handle("离线");
    },false);
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

在代码清单 9-7 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的文件 ce7.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.7.0
CACHE:
Js/js7.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-10 所示。

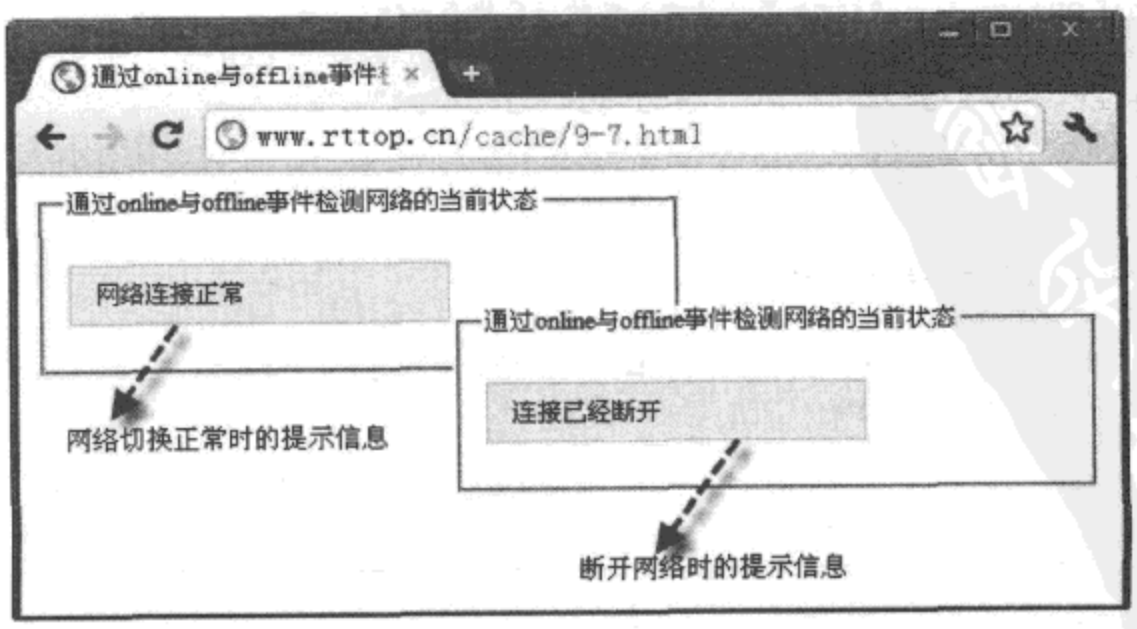


图 9-10　通过 online 与 offline 事件检测网络的当前状态

4. 源码分析

在本实例中，由于是通过事件的机制捕获网络的状态，因此，只有在触发 online 与 offline 事件时，才能在页面中展现提示信息。一般情况下，在手动或异常使网络断开后，将触发 offline 事件，打开的页面将自动侦测，无需刷新，并在页面中显示“连接已断开”的字样；在手动或自动重试的方式使网络连接成功后，打开的页面也无需刷新，自动在页面中显示“网络连接正常”的字样；如果不触发网络断开与连接的事件，将没有任何提示信息在页面中展现。

9.3.3　离线数据交互应用开发过程

如果仅仅是简单的静态页面，使用 Cache Manifest 处理方法就可以实现离线页面的访问；但如果是数据交互型的离线应用，不仅需要在离线时访问页面，而且还要支持数据在离线时的传输功能。那么，如何实现数据在离线时也能与服务器发生交互呢？通常的处理方式是：使用上一节介绍的 onLine 属性检测当前网络的状态；如果是离线，则先将交互的数据暂时存储至本地，例如保存到第 8 章介绍的 localStorage 对象或 Web SQL 数据库中；上线时，再将存储的数据同步至服务器，从而实现数据在离线时的交互功能。

下面通过实例 9-8 介绍开发一个离线留言数据交互应用的过程。

实例 9-8　开发一个离线留言数据交互应用

1. 功能描述

创建一个 HTML 页面，无论是网络在线或离线时，用户都可以访问页面，并在文本框中输入留言内容。单击“发表”按钮时，如果是在线状态，将向服务器与本地存储对象同时写入数据；如果是离线，则将数据保存在本地，等到上线时，再将存储的数据同步至服务器中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 9-8.html，加入代码如代码清单 9-8 所示。

代码清单 9-8　开发一个离线留言数据交互应用

```
<!DOCTYPE html>
<html manifest="ce8.manifest">
<head>
<meta charset="utf-8" />
<title>开发一个离线留言数据交互应用</title>
<link href="Css/css9.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js8.js"/>
</script>
</head>
<body onLoad="SynclocalData();">
```

```
    <ul id="ulMessage">
        正在读取数据 ...
    </ul>
    <p class="p4">
       <textarea id="txtContent" class="inputtxt"
                 cols="37" rows="5">
       </textarea><br>
       <input id="btnAdd" type="button" value="发表"
              class="inputbtn" onClick="btnAdd_Click();">
     </p>
</body>
</html>
```

在代码清单 9-8 中，页面导入一个 JavaScript 文件 js8.js，其中自定义了两个重要的函数：一个是 SynclocalData()，用于页面加载时的数据同步；另一个是 btnAdd_Click()，在单击“发表”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 单击"发表"按钮时调用
function btnAdd_Click() {
    // 获取文本框中的内容
    var strContent = $$("txtContent").value;
    // 如果不为空，则保存
    if (strContent.length > 0) {
        var strKey = RetRndNum(4);
        var strVal = strContent;
        if (navigator.onLine) {
            // 如果在线向服务器端增加数据
            AddServerData(strKey, strVal);
        }
        localStorage.setItem(strKey, strVal);
    }
    // 重新加载
    SynclocalData();
    // 清空原先内容
    $$("txtContent").value = "";
}

// 获取保存数据并显示在页面中
function SynclocalData() {
    // 标题部分
    var strHTML = "<li class='li_h'>";
    strHTML += "<span class='spn_a'>ID</span>";
    strHTML += "<span class='spn_b'>内容</span>";
```

```
        strHTML += "</li>";
        // 内容部分
        for (var intI = 0; intI < localStorage.length; intI++) {
            // 获取 Key 值
            var strKey = localStorage.key(intI);
            // 过滤键名内容
            var strVal = localStorage.getItem(strKey);
            strHTML += "<li class='li_c'>";
            strHTML += "<span class='spn_a'>" + strKey + "</span>";
            strHTML += "<span class='spn_b'>" + strVal + "</span>";
            strHTML += "</li>";
            if (navigator.onLine) {
                // 如果在线向服务端增加数据
                AddServerData(strKey, strVal);
            }
        }
        $$("ulMessage").innerHTML = strHTML;
    }
    // 生成指定长度的随机数
    function RetRndNum(n) {
        var strRnd = "";
        for (var intI = 0; intI < n; intI++) {
            strRnd += Math.floor(Math.random() * 10);
        }
        return strRnd;
    }
    // 向服务器同步点评数据
    function AddServerData(id, val) {
        // 根据 ID 号与内容，向服务器端数据库增加记录
        // 例如 "/Cache/Data/?id="+id+"&val="+val;
    }
```

在代码清单 9-8 中，页面导入的 CSS 文件 css9.css，在实例 9-3 的基础上，又添加了用于控制实例 9-8 页面的部分样式代码。其实现的代码如下所示：

```
@charset "utf-8";
/* CSS Document */
...省略实例 9-1、实例 9-2、实例 9-3 中已列出的代码
/* 示例 8*/
.inputtxt {
    border:solid 1px #ccc;
    line-height:18px;
    font-size:12px;
    padding-left:3px
}
ul{
    list-style:none;
    padding:0px;
```

```
    margin:15px 0px 15px 0px;
    text-align:center
}
ul .li_bot{
    padding-top:10px
}
ul .li_top{
    padding-bottom:10px
}
#ulMessage{
    width:360px
}
#ulMessage .spn_a{
    width:60px;
    float:left;
    text-align:left
}
#ulMessage .spn_b{
    width:180px;
    text-align:left;
    float:left
}
#ulMessage .li_h{
    border-bottom:solid 1px #666;
    float:left;
    background-color:#eee;
    padding:5px;
    font-weight:bold
}
#ulMessage .li_c{
    border-bottom:dashed 1px #ccc;
    float:left;
    padding:5px
}
.p4{
    clear:both;
    padding-top:10px
}
```

在代码清单 9-8 中，页面通过 <html> 元素的“manifest”属性绑定了一个“manifest”类型的文件 ce8.manifest，列举服务器需要缓存至本地的文件清单。其实现的代码如下所示：

```
CACHE MANIFEST
#version 0.8.0
CACHE:
Js/js8.js
Css/css9.css
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的页面效果如图 9-11 所示。

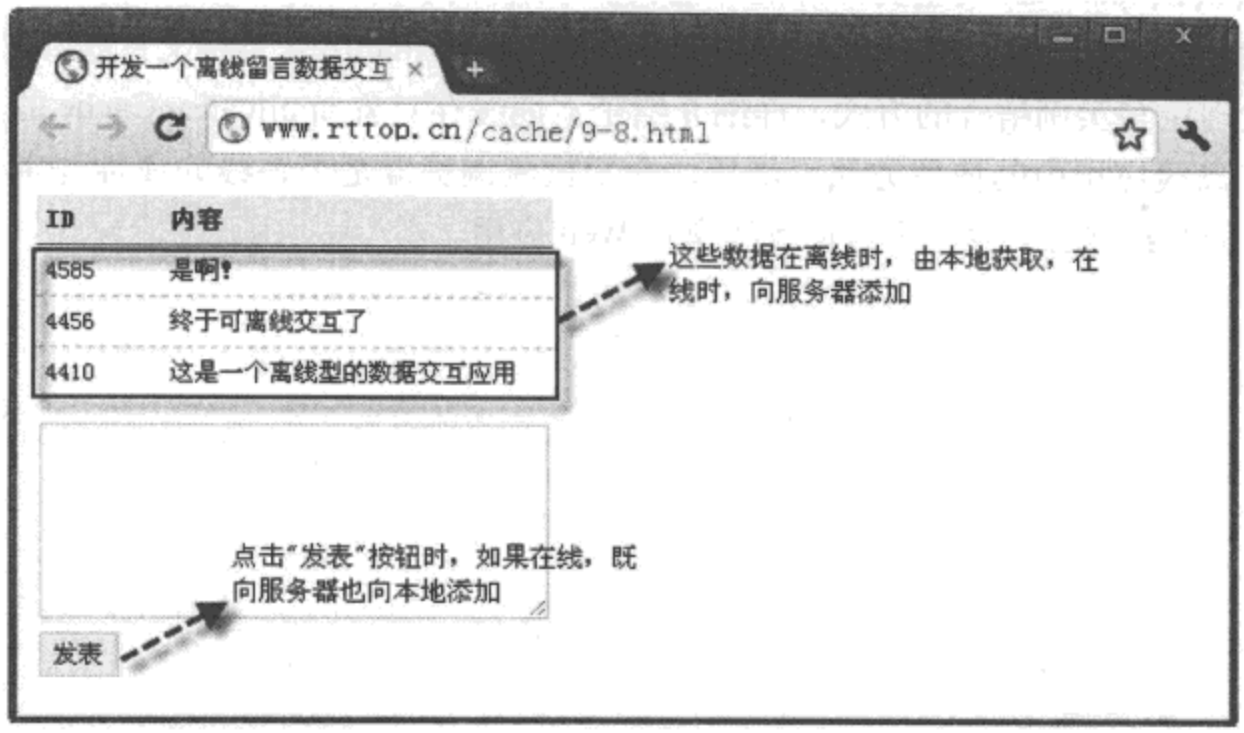


图 9-11　离线留言数据交互应用的效果

4. 源码分析

在本实例中，当页面加载时调用 SynclocalData() 函数，该函数的功能是调用本地存储的数据，显示在页面中；同时，在遍历数据的过程中，如果网络是在线状态，将通过 AddServerData() 函数向服务器同步数据，以确定本地存储的数据与服务器完全一致。

当用户单击“发表”按钮提交数据时，如果网络是在线状态，除向本地添加数据外，还要调用 AddServerData() 函数，向服务器添加一条记录，以确定数据的实时同步。

AddServerData() 函数用于根据获取的 ID 号与内容，在线时向服务器添加数据。这一功能可以借助 ajax 异步的方式，根据服务端提供的 URL 向服务器发送添加数据的请求，然后，根据请求反馈的信息，确定本次增加是否成功。由于 URL 提供的文件涉及服务端代码的开发，具体实现方法与服务端开发语言相关，在此不赘述。

实例 9-8 说明，如果要开发一个离线的 Web 应用，开发人员需要实现以下三个方面的功能：

1）通过 Cache Manifest 缓存资源文件，用于当浏览器在线时，将文件缓存至本地，离线时，调用缓存的文件进行访问，并能够实现自动或手动更新的功能。

2）通过 onLine 属性获取在线状态，只有获取了当前浏览器的状态，才能确定数据是从本地读取，还是与服务器同步。

3）通过本地存储对象 localStorage 或 Web SQL Database，保存 Web 应用在离线时产生的数据，用于在线时与服务器的数据同步。

9.4 本章小结

本章从认识 manifest 类型文件开始，使我们了解如何配置服务器端才能使用浏览器支持 manifest 类型的文件，并通过一个简单的实例，介绍使用 manifest 类型文件缓存资源的方法。然后，通过理论与实例结合的方式，详细介绍了本地缓存对象 applicationCache 的方法、事件及其在静态离线应用中的使用方法。最后，介绍检测浏览器是否在线的属性与事件，并通过获取浏览器的在线状态开发一个数据离线交互 Web 应用。

第 10 章

HTML 5 中的其他应用型 API

本章内容

- Web Sockets API
- Geolocation API
- Web Workers API
- 本章小结

目前，HTML 5 越来越受到人们的重视与青睐，一个很重要的原因就是：在 HTML 5 中可以方便地调用或访问众多的 API。开发人员可以通过编写一些预定义的函数或接口，实现与应用程序的快速访问或数据交互，而无需访问程序源码本身与内部的逻辑机制。

HTML 5 通过调用与数据通信相关的 Web Sockets API，实现从服务器中推送信息到客户端。另外，HTML 5 还可以通过使用 Geolocation API 获取地理位置相关的数据信息，以及使用 Web Workers API 实现单线程与多线程处理数据。本章将详细介绍 HTML 5 中各类应用型 API 的使用方法。

10.1 Web Sockets API

Socket 又称套接字或插座，是基于 W3C 标准开发在一个 TCP 接口中进行双向通信的技术。通常情况下，Socket 用于描述 IP 地址与端口，是通信过程中的一个字符句柄。当服务器端有多个应用服务绑定一个 Socket 时，通过通信中的字符句柄，实现不同端口对应不同应用服务功能。目前，大部分的浏览器都支持 HTML 5 中 Sockets API 的运行。

为了使大家对数据的通信或传输有一个基本的认识，我们首先介绍在 HTML 5 中如何使用 postMessage 方法实现在同一文档中不同区域或跨文档传输数据。

10.1.1 postMessage 方法

在 JavaScript 中，出于代码安全性的考虑，不允许跨域访问其他页面中的元素，这给不同区域的页面数据互访带来障碍。而在 HTML 5 中，可以利用对象的 postMessage 方法，在两个不同域名与端口的页面之间，实现数据的接收与发送功能。

要实现跨域页面间的数据互访，需要调用对象的 postMessage() 方法，其调用格式如下：

```
otherWindow.postMessage(message, targetOrigin)
```

其中，otherWindow 为接收数据页面的引用对象，可以是 window.open 的返回值，也可以是 iframe 的 contentWindow 属性，或通过下标返回的 window.frames 单个实体对象；message 表示所有发送的数据、字符类型，也可以是 JSON 对象转换后的字符内容；targetOrigin 表示发送数据的 URL 来源，用于限制 otherWindow 对象的接收范围，如果该值为通配符号（*），则表示不限制发送来源，指向全部的地址。

下面通过实例 10-1 介绍使用 postMessage() 方法实现跨文档传输数据的过程。

实例 10-1　使用 postMessage 方法实现跨文档传输数据

1. 功能描述

新建一个 HTML 页面，并在页面中添加一个 <iframe> 标记作为子页面。当在主页面的文本框中输入生成随机数的位数，并单击“请求”按钮后，子页面将接收该位数信息，并向主

页面返回根据该位数生成的随机数。主页面接收指定位数的随机数，并显示在页面，从而完成在不同文档间数据的互访功能。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-1.html，加入代码如代码清单 10-1 所示。

代码清单 10-1　使用 postMessage 方法实现跨文档传输数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 postMessage() 方法实现跨文档传输数据 </title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js1.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>跨文档请求数据 </legend>
   <p id="pStatus"></p>
   <input id="txtNum" type="text" class="inputtxt">
   <input id="btnAdd" type="button" value=" 请求 "
          class="inputbtn" onClick="btnSend_Click();">
   <iframe id="ifrA" src="Message.html"
          width="0px" height="0px" frameborder="0"/>
 </fieldset>
</body>
</html>
```

在代码清单 10-1 中，页面导入一个 JavaScript 文件 js1.js，其中编写了多个自定义函数，分别在主、子页面加载与单击“请求”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var strOrigin = "http://localhost";
// 自定义页面加载函数
function pageload() {
    window.addEventListener('message',
    function(event) {
        if (event.origin == strOrigin) {
            $$("pStatus").style.display = "block";
            $$("pStatus").innerHTML += event.data;
        }
```

```
        },
        false);
    }
    // 单击“请求”按钮时调用的函数
    function btnSend_Click() {
        // 获取发送内容
        var strTxtValue = $$("txtNum").value;
        if (strTxtValue.length > 0) {
            var targetOrigin = strOrigin;
            $$("ifrA").contentWindow.postMessage(strTxtValue, targetOrigin);
            $$("txtNum").value = "";
        }
    }
    //iframe 中子页面加载时调用的函数
    function PageLoadForMessage() {
        window.addEventListener('message',
        function(event) {
            if (event.origin == strOrigin) {
                var strRetHTML = "<span><b> ";
                strRetHTML += event.data + " </b>位随机数为：<b> ";
                strRetHTML += RetRndNum(event.data);
                strRetHTML += " </b></span><br>";
                event.source.postMessage(strRetHTML, event.origin);
            }
        },
        false);
    }
    // 生成指定长度的随机数
    function RetRndNum(n) {
        var strRnd = "";
        for (var intI = 0; intI < n; intI++) {
            strRnd += Math.floor(Math.random() * 10);
        }
        return strRnd;
    }
```

在代码清单 10-1 的主页面中，通过 <iframe> 元素的“src”属性导入一个名称为 Message.html 的子页面，用于接收主页面请求生成随机数长度的值，并返回根据该值生成的随机数。其完整的页面代码如下所示：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title></title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js1.js"/>
```

```
</script>
</head>
<body onLoad="PageLoadForMessage();">
</body>
</html>
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 10-1 所示。

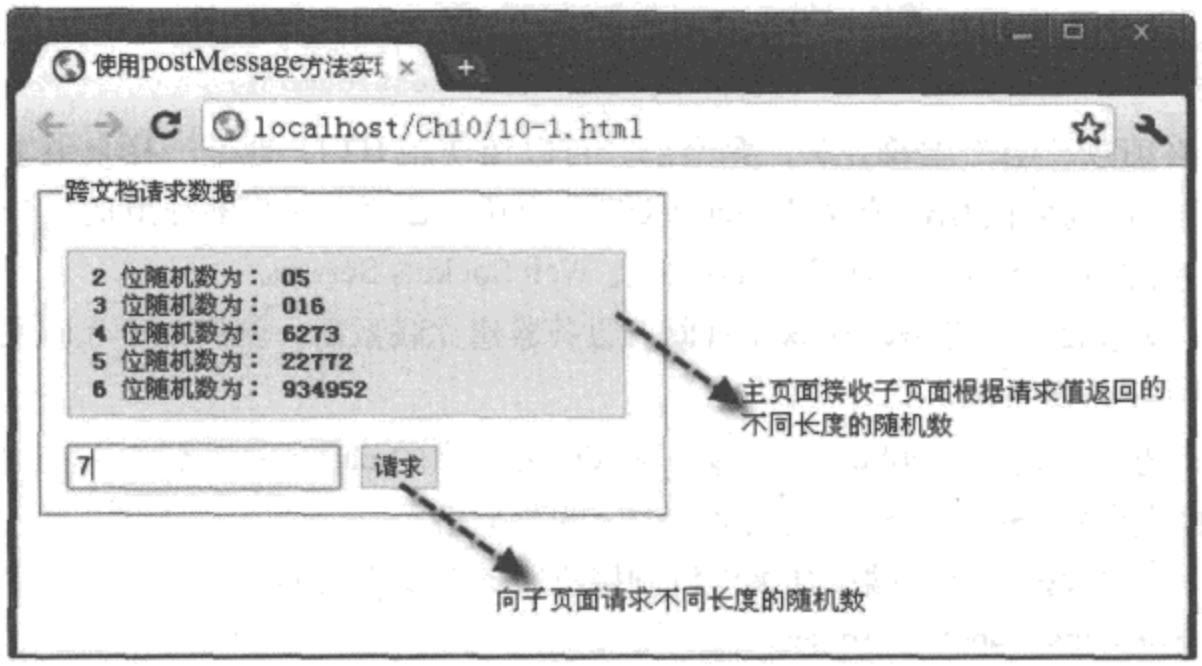


图 10-1　使用 PostMessage 方法实现跨文档传输数据

4. 源码分析

在本实例中，首先，为了接收页面间传输的数据，主、子页面在页面加载时都为页面添加了 message 事件，其添加方式如下：

```
window.addEventListener('message', function(event) {...}, false);
```

如果页面添加“message”事件成功，那么，通过 postMessage() 方法向页面发送数据请求时，将触发该事件，并通过事件回调函数中 event 对象的“data”属性捕获发送来的数据。本实例中，将捕获的数据 event.data 传递给另外一个自定义函数 RetRndNum()，用于生成随机数。

另外，event 对象中还包含“source”与“origin”属性，分别代表发送数据对象与发送来源，可以使用“source”属性向发送数据页面返回数据；同时，还可以通过“origin”属性检测互通数据的域名是否正确，以规避因域名不正确产生的恶意代码来源，确保数据交互的安全性。在本实例中，主、子页面通过“event.origin == strOrigin”代码，判断各自请求来源是否为约定的 strOrigin 值。如果是，则进行下面的操作；否则，不进行任何的数据交互操作，详细实现过程如代码中加粗部分所示。

10.1.2 使用 WebSocket 传送数据

在 HTML 5 中，WebSocket 为客户端与服务器端搭起了一座双向通信的桥梁，实现服务器端信息的推送功能。该桥梁是一个实时、永久性的连接，服务器端一旦与客户端建立了这种双向连接，就可以将数据推送至 Socket 中；而客户端只要有一个 Socket 绑定的地址和端口与服务器建立联系，就可以接收推送来的数据。

Web Sockets API 的使用方法十分简单，分为以下几步操作：

步骤 1 创建连接。新建一个 WebSocket 对象十分方便，代码如下：

```
var objWs = new WebSocket("ws://localhost:3131/test/demo");
```

其中，URL 必须以“ws”字符开头，剩余部分可以像使用 HTTP 地址一样来编写。该地址没有使用 HTTP，因为它的属性为 WebSocket URL；URL 必须由 4 个部分组成，分别是通信标记（ws）、主机名称（host）、端口号（port）及 Web Sockets Server。

步骤 2 发送数据。当 WebSocket 对象与服务器建立联系后，使用如下代码发送数据：

```
objWs.send(dataInfo);
```

其中，objWs 为新创建的 WebSocket 对象，send() 方法中的 dataInfo 参数为字符类型，即只能使用文本数据或者将 JSON 对象转换成文本内容的数据格式。

步骤 3 接收数据。客户端添加事件机制接收服务器发送来的数据，代码如下：

```
objWs.onmessage=function(event){
    alert(event.data)
    ...
}
```

其中，通过回调函数中 event 对象的“data”属性来获取服务器端发送的数据内容，该内容可以是一个字符串或者 JSON 对象。

步骤 4 状态标志。通过 WebSocket 对象的“readyState”属性记录连接过程中的状态值。

“readyState”属性是一个连接的状态标志，用于获取 WebSocket 对象在连接、打开、关闭中和关闭时的状态。该状态标志共有 4 种属性值，如表 10-1 所示。

表 10-1 WebSocket 对象的 readyState 属性

属性值	属性常量	描　述
0	CONNECTING	连接尚未建立
1	OPEN	WebSocket 的连接已建立，可以进行通信
2	CLOSING	连接正在关闭
3	CLOSED	连接已经关闭或不可用

WebSocket 对象在连接过程中，通过侦测这个状态标志的变化，可以获取服务器端与客户端连接的进度，并将反馈的信息实时返回客户端。

下面通过实例 10-2 介绍使用 WebSocket 对象传送数据的过程。

实例 10-2　使用 WebSocket 对象传送数据

1. 功能描述

新建一个 HTML 页面，在页面中，当用户在文本框中输入发送内容并单击“发送”按钮后，通过创建的 WebSocket 对象，将内容发送至服务器端，同时，页面也将接收服务器端返回来的数据，并展示在页面的 <textarea> 元素中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-2.html，加入代码如代码清单 10-2 所示。

代码清单 10-2　使用 WebSocket 对象传送数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 WebSocket 对象传送数据</title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js2.js"/>
</script>
</head>
<body onLoad="pageload();">
   <textarea id="txtaList" cols="26" rows="12"
             readonly="true"></textarea><br>
   <input id="txtMessage" type="text" class="inputtxt">
   <input id="btnAdd" type="button" value="发送"
          class="inputbtn" onClick="btnSend_Click();">
</body>
</html>
```

在代码清单 10-2 中，页面导入一个 JavaScript 文件 js2.js，其中编写了多个自定义的函数，在页面加载与单击“发送”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var strTip = "";
var objWs = null;
var conUrl = "ws://localhost:3131/test/demo";
var SocketCreated = false;
var arrState = new Array("正在建立连接 ...", "连接成功 !",
                         "正在关闭连接 ...", "连接已关闭 !",
                         "正在初始化值 ...", "连接出错 !");
// 自定义页面加载时函数
```

```
function pageload() {
    if (SocketCreated && (objWs.readyState == 0 || objWs.readyState == 1)) {
        objWs.close();
    } else {
        Handle_List(arrState[4]);
        try {
            objWs = new WebSocket(conUrl);
            SocketCreated = true;
        } catch(ex) {
            Handle_List(ex);
            return;
        }
    }
    // 添加 Socket 对象的打开事件
    objWs.onopen = function() {
        Handle_List(arrState[objWs.readyState]);
    }
    // 添加 Socket 对象的接收服务器数据事件
    objWs.onmessage = function(event) {
        Handle_List("系统消息:" +event.data);
    }
    // 添加 Socket 对象的关闭事件
    objWs.onclose = function() {
        Handle_List(arrState[objWs.readyState]);
    }
    // 添加 Socket 对象的出错事件
    objWs.onerror = function() {
        Handle_List(arrState[5]);
    }
}
// 自定义单击“发送”按钮时调用的函数
function btnSend_Click() {
    var strTxtMessage = $$("txtMessage").value;
    if (strTxtMessage.length > 0) {
        objWs.send(strTxtMessage);
        Handle_List("我说:" + strTxtMessage);
        $$("txtMessage").value = "";
    }
}
// 自定义显示与服务器交流内容的函数
function Handle_List(message) {
    strTip += message + "\n";
    $$("txtaList").innerHTML = strTip;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 10-2 所示。

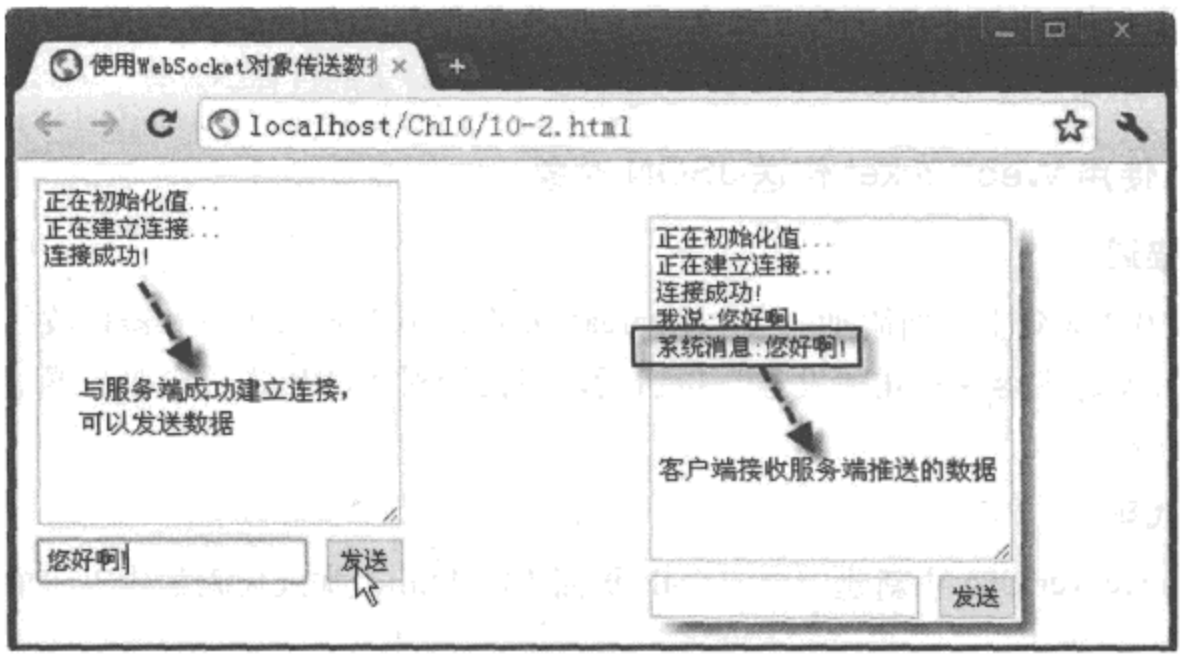


图 10-2　使用 WebSocket 对象传送数据

4. 源码分析

在本实例页面加载的 onLoad 事件中，调用自定义函数 pageload()。在该函数中，首先根据变量 SocketCreated 与 readyState 属性的值，检测是否还存在没有关闭的连接，如果存在，则调用 WebSocket 对象的 close() 方法进行关闭。然后，使用 try 语句通过新创建的 WebSocket 对象与服务器请求连接，如果连接成功，则将变量 SocketCreated 赋值为 true，否则，执行 catch 部分代码，将错误显示在页面的 <textarea> 元素中。为了能实时捕捉与服务器端连接的各种状态，在函数 pageload() 中自定义了 WebSocket 对象的打开（open）、接收数据（message）、关闭连接（close）、连接出错（error）事件，一旦触发这些事件，都将获取的数据显示在 <textarea> 元素中。

当用户单击“发送”按钮时，先检测发送的内容是否为空，再调用 WebSocket 对象的 send() 方法，将获取的数据发送至服务器端，其详细的实现过程如代码中的加粗部分所示。

需要说明的是，要实现客户端与服务器端的连接并且双方互通数据，首要条件需要在服务器端进行一些系统的配置，并使用服务器端代码编写程序支持客户端的请求。由于这些功能的开发涉及服务器端语言，在此不再赘述，本实例假设这些配置都已在服务器端完成。

10.1.3　使用 WebSocket 传送 JSON 对象

实例 10-2 介绍了使用 WebSocket 对象实现客户端与服务器端双向发送与接收数据的方法，那么，客户端能够发送与接收 JSON 对象吗？答案是肯定的，但是，在发送与接收过程中，需要借助 JavaScript 中的 JSON.parse 与 JSON.stringify 这两个方法，前者是将文本数据转换成 JSON 对象，后者是将 JSON 对象转换成文本数据。由于 WebScoket 对象的 send() 方法只能接收字符型的数据，因此，在发送时需要将 JSON 对象转换成文本数据，在接收过程中再将

服务器推送的文本数据转换成 JSON 对象。

下面通过实例 10-3 介绍使用 WebSocket 传送 JSON 对象的过程。

实例 10-3　使用 WebSocket 传送 JSON 对象

1. 功能描述

在实例 10-2 基础上，新添加一个 <textarea> 元素，用于显示从服务器接收的在线人员数据。用户输入发送内容并单击“发送”按钮后，将使用对象的形式，向服务器端发送输入的发送内容与时间。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-3.html，加入代码如代码清单 10-3 所示。

代码清单 10-3　使用 WebSocket 传送 JSON 对象

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 WebSocket 传送对象 </title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js3.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>使用 JSON 对象传输 </legend>
      <div>
          <span><b>对话记录 </b></span>
          <span class="pl140">
                <b>在线人员 </b>
          </span>
      </div>
   <textarea id="txtaList" cols="26" rows="12"
             readonly="true"></textarea>
   <textarea id="txtaUser" cols="10" rows="12"
             readonly="true"></textarea>
      <div class="pl2">
   <input id="txtMessage" type="text" class="inputtxt w176">
   <input id="btnAdd" type="button" value=" 发送 "
          class="inputbtn w85 ml4" onClick="btnSend_Click();">
      </div>
 </fieldset>
</body>
</html>
```

在代码清单 10-3 中，页面导入一个 JavaScript 文件 js3.js，其中编写了多个自定义函数，在页面加载与单击“发送”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var strList = "";
var strUser = "";
var objWs = null;
var conUrl = "ws://localhost:3131/test/JSON";
var SocketCreated = false;
var arrState = new Array( "正在建立连接 ...", "连接成功 !",
                          "正在关闭连接 ...", "连接已关闭 !",
                          "正在初始化值 ...", "连接出错 !");
// 自定义页面加载时调用的函数
function pageload() {
    if (SocketCreated && (objWs.readyState == 0 || objWs.readyState == 1)) {
        objWs.close();
    } else {
        Handle_List(arrState[4]);
        try {
            objWs = new WebSocket(conUrl);
            SocketCreated = true;
        } catch(ex) {
            Handle_List(ex);
            return;
        }
}
// 添加 Socket 对象的打开事件
objWs.onopen = function() {
    Handle_List(arrState[objWs.readyState]);
}
// 添加 Socket 对象的接收服务器数据事件
objWs.onmessage = function(event) {
    var objJSON =JSON.parse(event.data);
    for (var intI = 0; intI < objJSON.length; i++) {
        Handle_User(objJSON[intI].UserName);
        Handle_User(objJSON[intI].Stauts);
    }
}
// 添加 Socket 对象的关闭事件
objWs.onclose = function() {
    Handle_List(arrState[objWs.readyState]);
}
// 添加 Socket 对象的出错事件
objWs.onerror = function() {
    Handle_List(arrState[5]);
```

```
        }
    }
    // 自定义单击 " 发送 " 按钮时调用的函数
    function btnSend_Click() {
        var strTxtMessage = $$("txtMessage").value;
        // 定义一个日期型对象
        var strTime = new Date();
        if (strTxtMessage.length > 0) {
            objWs.send(JSON.stringify({
                content: strTxtMessage,
                datetime: strTime.toLocaleTimeString()
            }));
            Handle_List(strTime.toLocaleTimeString());
            Handle_List(" 我说 :" + strTxtMessage);
            $$("txtMessage").value = "";
        }
    }
    // 自定义显示对话记录内容的函数
    function Handle_List(message) {
        strList += message + "\n";
        $$("txtaList").innerHTML = strList;
    }
    // 自定义显示在线人员内容的函数
    function Handle_User(message) {
        strUser += message + "\n";
        $$("txtaUser").innerHTML = strUser;
    }
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 10-3 所示。

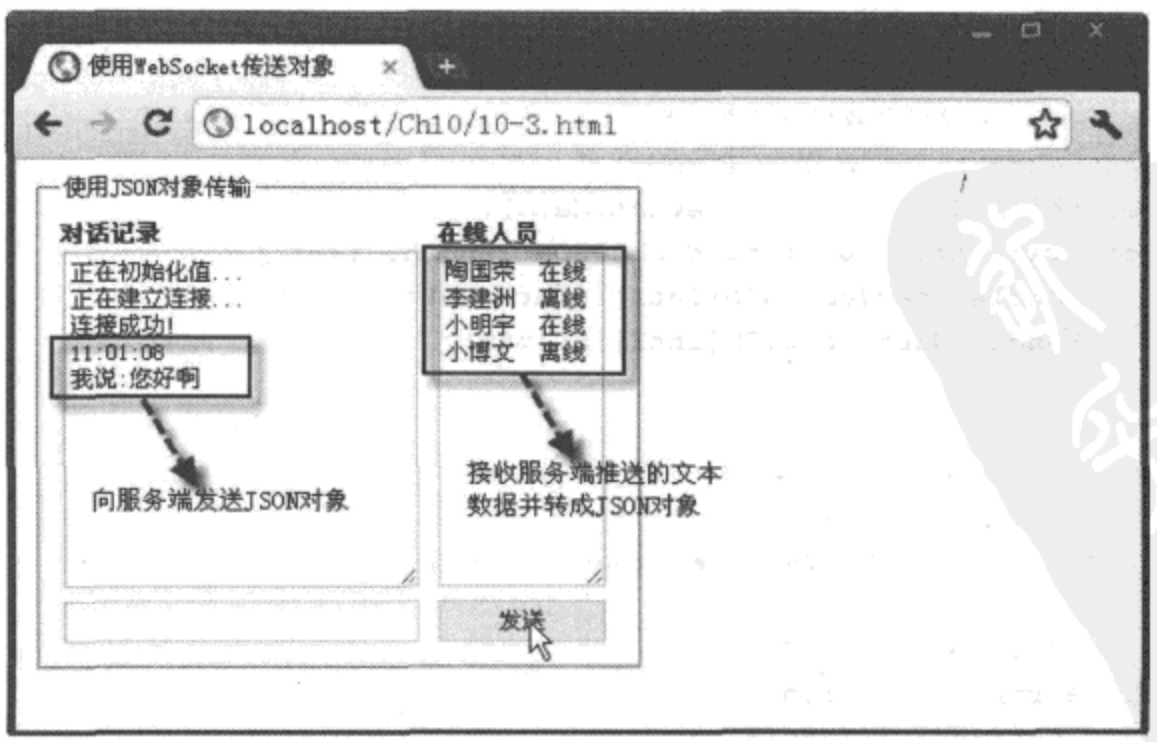


图 10-3　使用 WebSocket 对象传送 JSON 对象

4. 源码分析

在本实例的 JavaScript 代码中，结构与实例 10-2 基本相同，但有两个地方存在明显差别，分别是发送客户端数据与接收服务器推送来的数据处理方式。

在本实例中，为了能够向服务器端发送输入内容与对应时间，需要将获取的内容变量 strTxtMessage 与当前时间 strTime.toLocaleTimeString() 这两项内容，通过调用 JSON.stringify 方法转成文本数据，再调用 send() 方法向服务器端发送数据。

在本实例的 message 事件中，为了更好地接收服务器端推送来的数据，先调用 JSON.parse 方法将获取的 event.data 数据转成 JSON 对象，再通过遍历对象元素的方法，将接收的全部数据信息展示在对应的 <textarea> 元素中。

10.2　Geolocation API

HTML 5 中提供了一组 Geolocation API，用来获取用户的地理位置信息。在移动设备中，如果浏览器支持且设置有定位的功能，就可以使用这组 API 定位用户的地理位置。本节介绍使用 getCurrentPosition() 方法获取用户地理位置信息。

10.2.1　使用 getCurrentPosition 方法获取当前地理位置

在 HTML 5 中，如果浏览器需要获取用户当前的地理位置信息，需要通过 API 访问 window.navigator 对象中新添加的 geolocation 属性，并调用该属性中的 getCurrentPositiont() 方法获取用户当前地理位置信息，其调用的代码格式如下所示：

```
navigator.geolocation.getCurrentPosition(
        successCallback,errorCallback,[Options])
```

其中，参数 successCallback 是一个函数，用于成功获取用户当前地理位置信息时的回调操作；该回调函数中有一个形参 position，该参数是一个对象，用于描述位置的详细数据信息。参数 errorCallback 是获取地理位置失败时回调的函数，该函数中通过一个 error 对象作为形参，根据该对象的“code”属性获取定位失败的原因。该属性包括 4 个值，详细描述如表 10-2 所示。

表 10-2　error 对象的 code 属性

属性值	描　述
0	未知错误信息
1	用户拒绝了定位服务的请求
2	没有获取正确的地理位置信息
3	获取位置的操作超时

在 error 对象中，除“code”属性表示出错数字外，还可以通过“message”属性获取出错的详细文字信息。该属性是一个字符串，包含与“code”属性值相对应的错误说明信息。

参数 Options 是一个可选择的对象，如果设置，则可以为对象添加一些属性内容，如表 10-3 所示。

表 10-3 Options 属性包含的值

属性值	描　述
timeout	设置获取地理位置信息操作的超时限制（单位：毫秒），超过时触发 errorCallback 回调函数，error.code 的属性值为 3，表示已超时
maximumAge	设置缓存获取的地理位置数据信息的有效时间（单位：毫秒），超过设置的时间时重新获取，否则调用缓存中的数据信息
enableHighAccuracy	布尔值，表示是否要精确地获取地理位置信息，在移动设备上，如果为 true，则需要消耗更多的设置电量，默认值为 false

下面通过实例 10-4 介绍使用 getCurrentPosition() 方法获取出错数据信息的过程。

实例 10-4　使用 getCurrentPosition 方法获取出错数据信息

1. 功能描述

在新建的 HTML 页面中，当使用 getCurrentPosition 方法获取当前用户的浏览器地理位置信息时，在弹出的是否共享窗口中，如果用户选择了“拒绝”，则将捕获的错误信息通过回调函数 errorCallback() 中的 error.code 与 errormessage 显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-4.html，加入代码如代码清单 10-4 所示。

代码清单 10-4　使用 getCurrentPosition 方法获取出错数据信息

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 getCurrentPosition 方法获取出错数据信息 </title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js4.js"/>
</script>
<script type="text/javascript" language="jscript"
        src="http://maps.google.com/maps/api/js?sensor=false"/>
</script>
</head>
<body onLoad="pageload();">
   <p id="pStatus"></p>
</body>
</html>
```

在代码清单 10-4 中，页面导入一个 JavaScript 文件 js4.js，其中自定义了一个函数 pageload()，在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(function(ObjPos) {
            Status_Handle("获取成功!");
        },
        function(objError) {
            Status_Handle(objError.code + ":" + objError.message);
        },
        {
            maximumAge: 3 * 1000 * 60,
            timeout: 3000
        });
    }
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 10-4 所示。

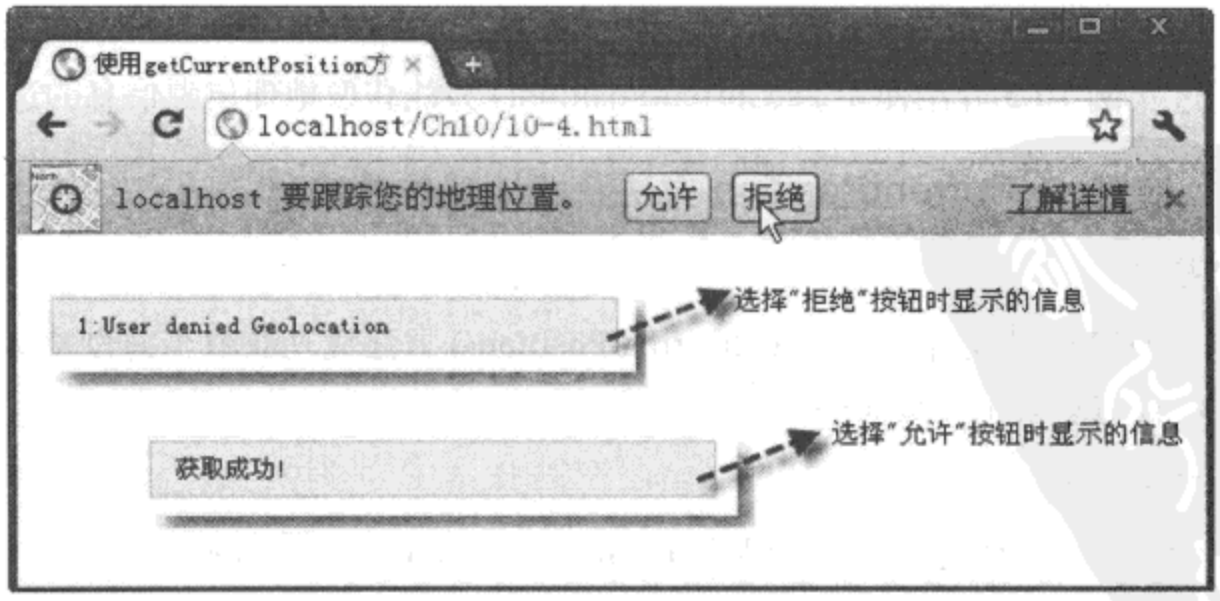


图 10-4　使用 getCurrentPosition 方法获取出错数据信息

4. 源码分析

在本实例中，如果浏览器第一次调用 getCurrentPosition() 方法，出于安全的考虑，浏览器会询问用户是否共享位置数据信息。如果用户拒绝，该方法将出现错误，并无法获取用户的地理位置数据，只有当用户允许共享地理位置时，getCurrentPosition() 方法才能生效。

目前，各浏览器厂商对该 Geolocation API 的支持情况不完全相同，因此，在调用 getCurrentPosition() 方法之前，需要先用 navigator.geolocation() 方法检测当前浏览器是否支持定位功能，然后才开始调用 getCurrentPosition() 方法获取用户的地理位置信息。完整的实现过程如代码中加粗部分所示。

如果在使用 getCurrentPosition() 方法获取当前浏览器地理位置信息时，用户允许了位置共享，并且浏览器也支持定位功能，那么，该方法就可以正确地获取当前地理位置数据。

使用 getCurrentPosition() 方法时，如果获取位置成功，将回调 successCallback() 函数。该函数通过一个对象参数 position 返回所有的地理位置详细数据信息，这些信息以对象的属性形式进行展示。position 对象包含两个重要的属性，分别为“timestamp”和“coords”，其中“timestamp”属性表示获取地理位置时的时间，而“coords”属性则包含多个值，详细描述如表 10-4 所示。

表 10-4　coords 属性包含的值

属性值	描　述
accuracy	当前地理位置的精确度
latitude	当前地理位置的纬度
longitude	当前地理位置的经度
altitude	当前地理位置的海拔高度
altitudeAcuracy	当前地理位置的海拔精确度（单位：米）
heading	当前设置的前进方向，无法获取时，返回 null
speed	当前设置的前进速度（单位：米 / 秒），无法获取时，返回 null

下面通过实例 10-5 介绍使用 getCurrentPosition() 方法获取地理位置信息的过程。

实例 10-5　使用 getCurrentPosition 方法获取地理位置信息

1. 功能描述

在新建的 HTML 页面中，调用 getCurrentPosition() 方法成功获取当前浏览器的地理位置，并将获取的位置信息展示在页面的 <p> 元素中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-5.html，加入代码如代码清单 10-5 所示。

代码清单 10-5　使用 getCurrentPosition 方法获取地理位置信息

```
<!DOCTYPE html>
<html>
```

```
<head>
<meta charset="utf-8" />
<title>使用 getCurrentPosition 方法获取地理位置信息 </title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js5.js"/>
</script>
<script type="text/javascript" language="jscript"
        src="http://maps.google.com/maps/api/js?sensor=false"/>
</script>
</head>
<body onLoad="pageload();">
   <p id="pStatus"></p>
</body>
</html>
```

在代码清单 10-5 中，页面导入一个 JavaScript 文件 js5.js，其中自定义一个函数 pageload()，在页面加载时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var objNav = null;
var strHTML = "";
function pageload() {
    if (objNav == null) {
        objNav = window.navigator;
    }
    if (objNav != null) {
        var objGeoLoc = objNav.geolocation;
        if (objGeoLoc != null) {
            objGeoLoc.getCurrentPosition(function(objPos) {
                var objCrd = objPos.coords;
                strHTML += "纬度值：<b>" + objCrd.latitude + "</b><br>";
                strHTML += "精准度：<b>" + objCrd.accuracy + "</b><br>";
                strHTML += "精度值：<b>" + objCrd.longitude + "</b><br>";
                strHTML += "时间戳：<b>" + objPos.timestamp + "</b><br>";
                var objAdd = objPos.address;
                strHTML +="------------------------------<br>";
                strHTML += "国家：<b>" + objAdd.country + "</b><br>";
                strHTML += "省份：<b>" + objAdd.region + "</b><br>";
                strHTML += "城市：<b>" + objAdd.city + "</b><br>";
                Status_Handle(strHTML);
            },
            function(objError) {
                Status_Handle(objError.code + ":" + objError.message);
            },
```

```
                {
                    maximumAge: 3 * 1000 * 60,
                    timeout: 3000
                });
            }
        }
    }
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = message;
}
```

3. 页面效果

该页面在不同浏览器中执行的效果如图 10-5 所示。

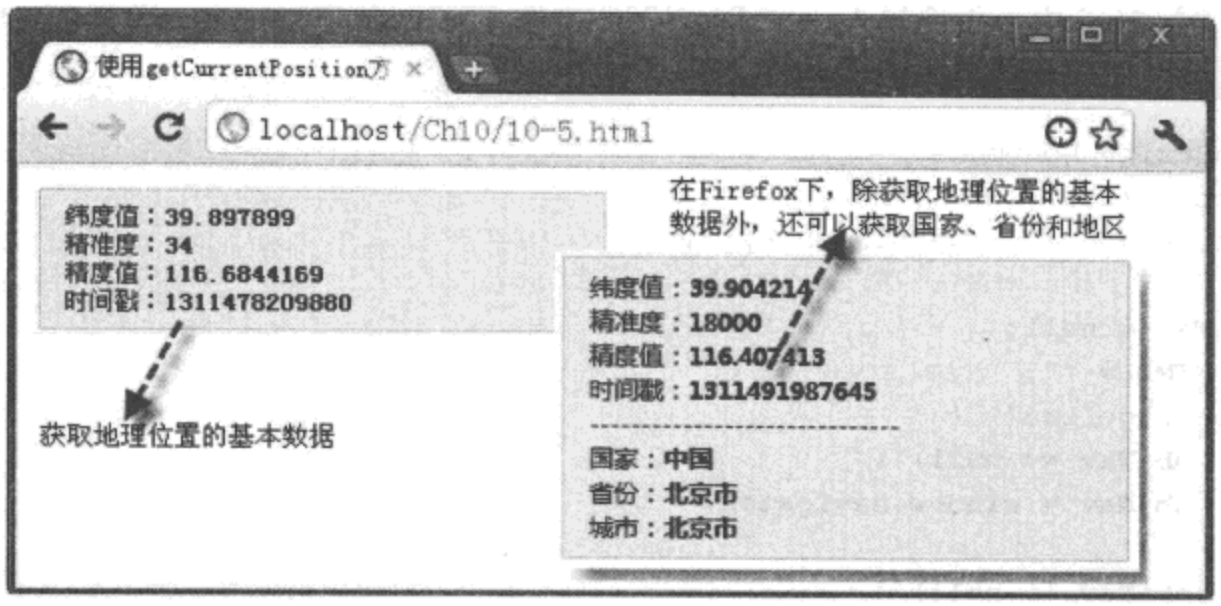


图 10-5　使用 getCurrentPosition 方法获取地理位置信息

4. 源码分析

在本实例中，当使用 getCurrentPosition() 方法成功获取地理位置数据后，将可以回调函数 successCallback()。解析对象参数 objPos，如果需要展示获取时间，则调用该对象的“timestamp”属性；如果需要展示地理位置数据，则通过对象的“coords”各个属性值来显示。

由于各浏览器对 Geolocation API 支持的情况不同，因此，同一代码在两个不同浏览器中执行，其返回的结果会出现一些偏差或对某些属性不支持，如 Firefox 5.0 中支持显示地理位置所在的国家、省份、城市等信息，而 Chrome 10 浏览器则不支持。

此外，如果需要持续监测当前的地理位置，可以调用以下方法：

```
var intWatchID = navigator.geolocation.watchCurrentPosition(
                    successCallback,errorCallback,[Options])
```

其中的参数与 getCurrentPosition() 方法一样，但该方法还返回一个“intWatchID”值，用于停

止持续监测的操作。如果需要停止持续监测，则调用下列方法：

```
clearWatch(intWatchID)
```

该方法通过清除持续监测时返回的 intWatchID 值，实现停止持续监测的功能。

10.2.2　使用 Google 地图锁定位置

在前面的章节中，详细介绍了使用 getCurrentPosition() 方法获取用户地理位置信息。既然可以正确获取这些地理位置数据，那么，就可以通过使用 Google 地图中的 Google Map API，将获取的位置信息标记在地图中，从而实现在 Google 地图中锁定位置的功能。

下面通过实例 10-6 介绍使用 Google 地图锁定位置的过程。

实例 10-6　使用 Google 地图锁定位置

1. 功能描述

在新建的 HTML 页面中，通过 元素显示一幅 Google 地图，并将 Google Map API 中的对象与 getCurrentPosition() 方法相结合，在地图中标注当前地理位置，当该位置发生变化时，地图中的标注信息也随之发生变化。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-6.html，加入代码如代码清单 10-6 所示。

代码清单 10-6　使用 Google 地图锁定位置

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 Google 地图锁定位置 </title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js6.js"/>
</script>
<script type="text/javascript" language="jscript"
        src="http://maps.google.com/maps/api/js?sensor=false"/>
</script>
</head>
<body onLoad="pageload();">
   <div id="divMap"></div>
</body>
</html>
```

在代码清单 10-6 中，页面导入一个 JavaScript 文件 js6.js，其中自定义一个函数 pageload()，在页面加载时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var objNav = null;
var strHTML = "";
// 自定义页面加载时调用的函数
function pageload() {
    if (objNav == null) {
        objNav = window.navigator;
    }
    if (objNav != null) {
        var objGeoLoc = objNav.geolocation;
        if (objGeoLoc != null) {
            objGeoLoc.getCurrentPosition(function(objPos) {
                var objCrd = objPos.coords;
                var lat = objCrd.latitude;
                var lng = objCrd.longitude;
                // 根据获取的经度与纬度创建一个地图中心坐标
                var latlng = new google.maps.LatLng(lat, lng);
                // 将中心点设置为页面打开时 Google 地图的中心点
                var objOpt = {
                    zoom: 16,
                    center: latlng,
                    mapTypeId: google.maps.MapTypeId.ROADMAP
                };
                // 创建地图，并与页面中 ID 号为“divMap”的元素相绑定
                var objMap = new google.maps.Map($$("divMap"), objOpt);
                // 创建一个地图标记
                var objMrk = new google.maps.Marker({
                    position: latlng,
                    map: objMap
                });
                // 创建一个地图标记窗口并设置注释内容
                var objInf=new google.maps.InfoWindow({
                  content:"我在这里"
                });
                // 在地图中打开标记窗口
                objInf.open(objMap,objMrk);
            },
            function(objError) {
                Status_Handle(objError.code + ":" + objError.message);
            },
            {
                maximumAge: 3 * 1000 * 60,
                timeout: 3000
            });
        }
    }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 10-6 所示。

图 10-6　使用 Google 地图锁定位置

4. 源码分析

在本实例中，首先，为了能够使用 Google 地图及 Google Map API，需要使用 <script> 元素导入对应的脚本文件，文件的 URL 为“http://maps.google.com/maps/api/js?sensor=false”。

通过 getCurrentPosition() 方法获取经度与纬度，创建一个地图中心坐标 latlng，并将该中心点设置为页面打开时 Google 地图的中心点；同时，将设置好的地图与页面中 ID 号为“divMap”的元素绑定，将地图显示在页面中。最后，在地图中创建一个锁定标记 objMrk，并在创建的标记窗口 objInf 中设定标记在地图中显示的注释中文，通过调用地图的 open() 方法，在地图中打开带有注释中文的标记窗口。完整的实现过程如代码中加粗部分所示。

10.3　Web Workers API

在 HTML 5 中，Web Workers 是构造 Web 应用程序的重要功能，通过使用 Worker 方法，可以将前台中的 JavaScript 代码分割成若干个分散的代码块，分别由不同的后台线程负责执行，从而避免由于前台单线程执行缓慢出现用户等待的局面。

后台的单个独立线程不仅可以被前台所调用，实现数据间的互访，而且在后台线程中还可以调用新的子线程，分割父线程的功能，实现线程的嵌套调用。

本节详细介绍使用 Worker 线程的方式实现前、后台数据交互的过程。

10.3.1 Worker 对象处理线程

在页面中，如果是执行时间较长、可能需要用户等待的操作，就可以交给后台线程 Worker 去处理。因为它与前台的线程分离，互不影响，但可以通过 postMessage() 方法与 onmessage 事件进行数据的交互。postMessage() 方法用于 worker 对象发送数据，其调用的格式如下：

```
var objWorker = new Worker("脚本文件 URL");
objWorker.postMessage(data);
```

其中，第一行通过实例化一个 Worker 类对象，创建了一个名为 objWorker 的后台线程；第二行代码通过 objWorker 调用 postMessage() 方法，向后台线程发送文本格式的 data 数据。

另外，为了在前台接收后台线程返回的数据，需要在定义 objWorker 对象后添加一个 message 事件，用于捕捉后台线程返回的数据，其调用的格式如下：

```
objWorker.addEventListener('message',
    function(event) {
       alert(event.data);
       ...
    },
    false);
```

其中，event.data 就是后台线程处理完成后返回给前台的数据。

下面通过实例 10-7 介绍使用 Worker 对象处理线程的过程。

实例 10-7　使用 Worker 对象处理线程

1. 功能描述

在新建的 HTML 页面中，页面在加载时创建一个 Worker 后台线程。当用户在文本框中输入生成随机数的位数并单击“请求”按钮时，向该后台线程发送文本框中的输入值，后台线程将根据接收的数据生成指定位数的随机数，返回给前台调用代码，并显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-7.html，加入代码如代码清单 10-7 所示。

代码清单 10-7　使用 Worker 对象处理线程

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 Worker 对象处理线程的简单示例</title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js7.js"/>
```

```
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend> 线程脚本处理数据 </legend>
   <p id="pStatus"></p>
   <input id="txtNum" type="text" class="inputtxt">
   <input id="btnAdd" type="button" value=" 请求 "
          class="inputbtn" onClick="btnSend_Click();">
 </fieldset>
</body>
</html>
```

在代码清单 10-7 中，页面导入一个 JavaScript 文件 js7.js，其中自定义两个函数，分别在页面加载与单击“请求”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var objWorker = new Worker("Js/js7_1.js");
// 自定义页面加载时调用的函数
function pageload() {
    objWorker.addEventListener('message',
    function(event) {
        $$("pStatus").style.display = "block";
        $$("pStatus").innerHTML += event.data;
    },
    false);
}
// 自定义单击 " 请求 " 按钮时调用的函数
function btnSend_Click() {
    // 获取发送内容
    var strTxtValue = $$("txtNum").value;
    if (strTxtValue.length > 0) {
        objWorker.postMessage(strTxtValue);
        $$("txtNum").value = "";
    }
}
```

在代码清单 10-7 的 JavaScript 文件 js7.js 代码中，通过 Worker 对象调用了一个后台线程脚本文件 js7_1.js，在该文件中，根据获取的位数生成随机数并将该数值返回前台。其实现的代码如下所示：

```
// JavaScript Document
self.onmessage = function(event) {
    var strRetHTML = "<span><b> ";
    strRetHTML += event.data + " </b> 位随机数为：<b> ";
```

```
    strRetHTML += RetRndNum(event.data);
    strRetHTML += " </b></span><br>";
    self.postMessage(strRetHTML);
}
// 生成指定长度的随机数
function RetRndNum(n) {
    var strRnd = "";
    for (var intI = 0; intI < n; intI++) {
        strRnd += Math.floor(Math.random() * 10);
    }
    return strRnd;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 10-7 所示。

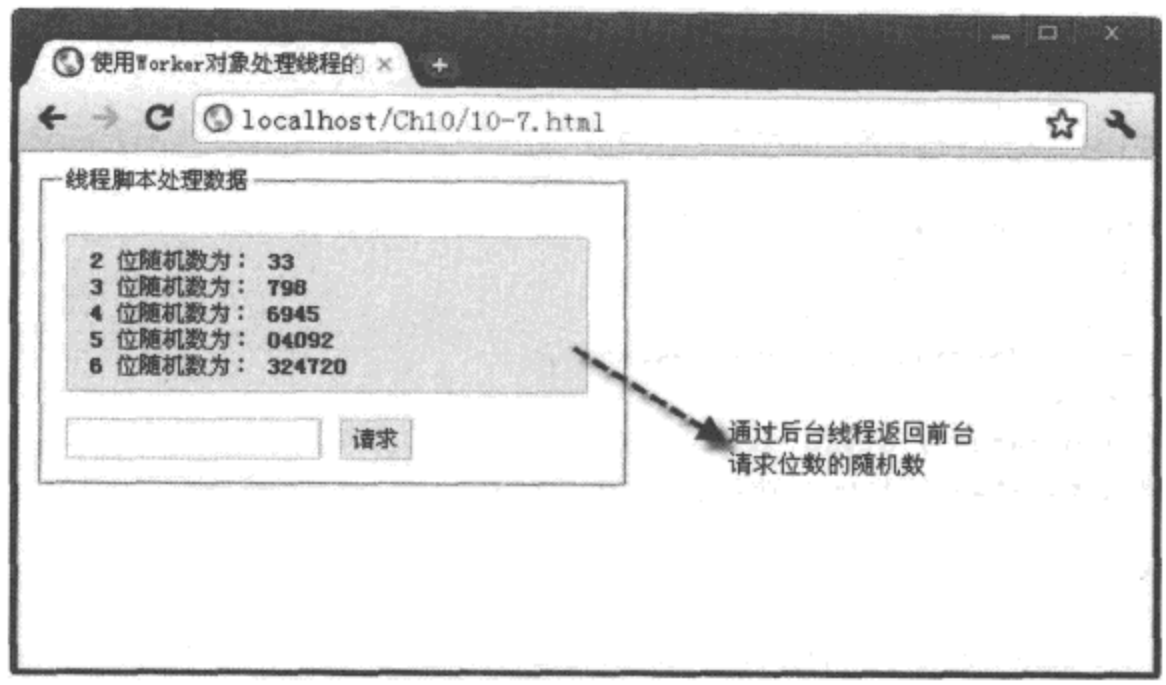


图 10-7　使用 Worker 对象处理线程

4. 源码分析

在本实例中，首先定义一个后台线程 objWorker，其脚本文件指向 js7_1.js，表示由该文件实现前台请求的操作。当用户在文本框中输入随机数长度并单击“请求”按钮时，该输入的内容通过调用线程 objWorker 对象的 postMessage() 方法，发送至脚本文件 js7_1.js。

在脚本文件 js7_1.js 中，通过添加 message 事件获取前台传回的数据，并将该数据值 event.data 作为自定义函数 RetRndNum() 的实参，生成指定位数的随机数，并将该随机数通过 self.postMessage() 方法发送至调用后台线程的前台程序。在前台代码中，通过添加 message 事件获取后台线程处理完成后传回的数据，并将数据的信息展示在页面中。

说明　虽然后台线程可以处理前台的代码，但是不允许后台线程中访问前台页面的对象或元素，如果访问，后台线程将报错，它们只限于进行数据上的交互。

10.3.2　使用线程传递 JSON 对象

由于 JSON 对象在 HTML 5 中使用十分广泛，那么，可以使用后台线程传递 JSON 对象吗？答案是肯定的，即可以通过后台线程传递一个 JSON 对象给前台，前台接收并显示 JSON 对象内容的方法。

下面通过实例 10-8 介绍使用线程传递 JSON 对象的过程。

实例 10-8　使用线程传递 JSON 对象

1. 功能描述

在新建的 HTML 页面中，页面在加载时创建一个 Worker 后台线程，该线程将返回给前台页面一个 JSON 对象，前台获取该 JSON 对象，使用遍历的方式显示对象中的全部内容。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-8.html，加入代码如代码清单 10-8 所示。

代码清单 10-8　使用线程传递 JSON 对象

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用线程传递 JSON 对象 </title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js8.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>线程传递 JSON 对象 </legend>
   <p id="pStatus"></p>
 </fieldset>
</body>
</html>
```

在代码清单 10-8 中，页面导入一个 JavaScript 文件 js8.js，其中自定义一个函数 pageload()，在页面加载时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var objWorker = new Worker("Js/js8_1.js");
// 自定义页面加载时调用的函数
function pageload() {
```

```
        objWorker.addEventListener('message',
        function(event) {
            var strHTML = "";
            var ev = event.data;
            for (var i in ev) {
                strHTML +="<span>"+ i + " :";
                strHTML +="<b> " + ev[i] + " </b></span><br>";
            }
            $$("pStatus").style.display = "block";
            $$("pStatus").innerHTML = strHTML;
        },
        false);
        objWorker.postMessage("");
    }
```

在代码清单 10-8 的 JavaScript 文件 js8.js 代码中，调用的后台线程脚本文件为 js8_1.js，在该文件中通过 postMessage() 方法向前台发送 JSON 对象，其实现的代码如下所示：

```
// JavaScript Document
var json = {
    姓名 : "陶国荣",
    性别 : "男",
    邮箱 : "tao_guo_rong@163.com"
};
self.onmessage = function(event) {
    self.postMessage(json);
    close();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 10-8 所示。

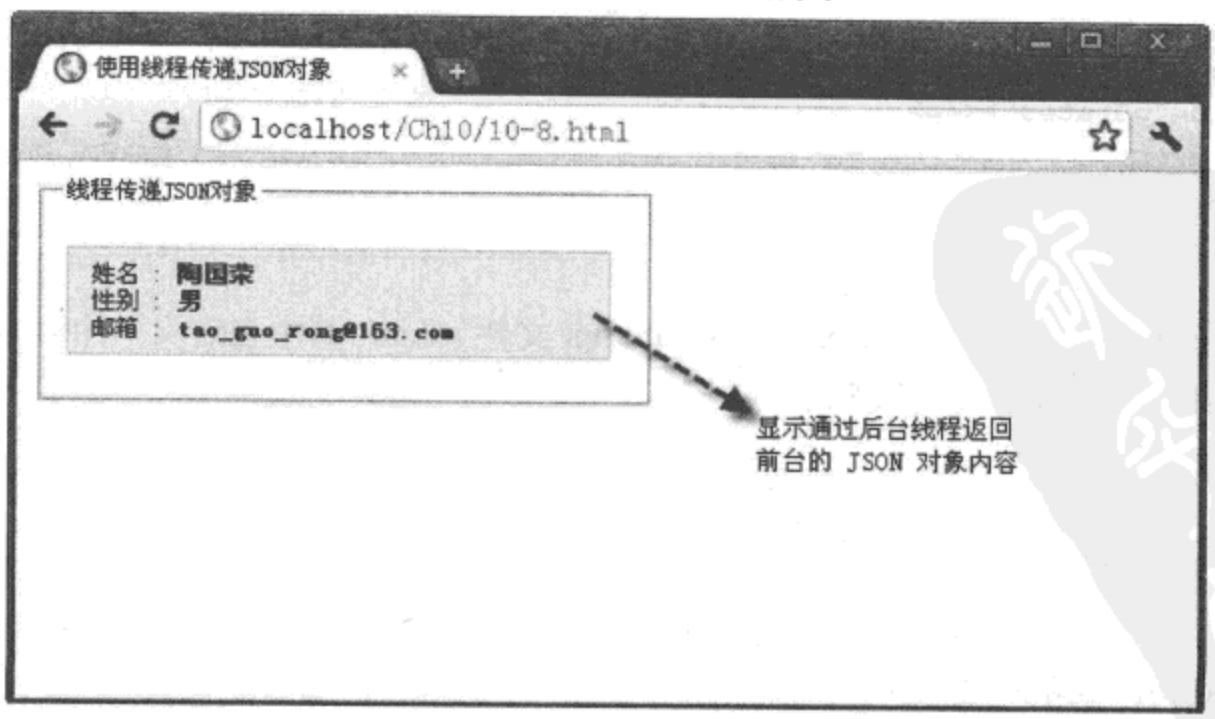


图 10-8　使用线程传递 JSON 对象

4. 源码分析

在本实例中，当页面加载时触发 onLoad 事件，在该事件中调用 pageload() 函数。该函数首先定义一个后台线程对象 objWorker，脚本文件指向 js8_1.js，并通过调用对象的 postMessage() 方法向后台线程发送一个空字符请求。

在后台线程指向文件 js8_1.js 中，先自定义一个 JSON 对象 json，当通过 message 事件监测前台页面请求后，调用 self.postMessage() 方法向前台代码传递 JSON 对象，并使用 close 语句关闭后台线程。前台为了在 message 事件中获取传递来的 JSON 对象内容，使用 for 语句的方式遍历整个 JSON 对象的内容，并显示在页面中。

10.3.3　使用线程嵌套交互数据

在前面的章节中，我们介绍了使用后台子线程分割处理前台 JavaScript 代码的方法，其实，在后台线程中还可以调用线程，分割主线程的功能，形成线程嵌套处理代码的格局。这种方式可以将各功能块分离，形成独立的子模块，有利于 Web 应用的开发。目前，只有 Firefox 5.0 浏览器支持这种后台子线程嵌套交互数据的方法。

下面通过实例 10-9 介绍使用线程嵌套交互数据的过程。

实例 10-9　使用线程嵌套交互数据

1. 功能描述

在实例 10-7 的基础上，添加一个显示随机数奇偶特征的功能。当用户在页面中输入生成随机数的位数并单击“请求”按钮后，不仅在页面中显示对应位数的随机数，而且将随机数的奇偶特征一起显示在页面中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 10-9.html，加入代码如代码清单 10-9 所示。

代码清单 10-9　使用线程嵌套交互数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用线程嵌套交互数据</title>
<link href="Css/css10.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js9.js"/>
</script>
</head>
<body onLoad="pageload();">
 <fieldset>
   <legend>线程嵌套请求交互数据</legend>
```

```
    <p id="pStatus"></p>
    <input id="txtNum" type="text" class="inputtxt">
    <input id="btnAdd" type="button" value="请求"
           class="inputbtn" onClick="btnSend_Click();">
  </fieldset>
</body>
</html>
```

在代码清单 10-9 中，页面导入一个 JavaScript 文件 js9.js，其中自定义了两个函数，分别在页面加载与单击“请求”按钮时调用。其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var objWorker = new Worker("Js/js9_1.js");
// 自定义页面加载时调用的函数
function pageload() {
    objWorker.addEventListener('message',
    function(event) {
        $$("pStatus").style.display = "block";
        $$("pStatus").innerHTML += event.data;
    },
    false);
}
// 自定义单击"请求"按钮时调用的函数
function btnSend_Click() {
    // 获取发送内容
    var strTxtValue = $$("txtNum").value;
    if (strTxtValue.length > 0) {
        objWorker.postMessage(strTxtValue);
        $$("txtNum").value = "";
    }
}
```

在代码清单 10-9 的 JavaScript 文件 js9.js 代码中，调用的后台线程脚本文件为 js9_1.js，在该文件中，按指定位数生成随机数，其实现的代码如下所示：

```
// JavaScript Document
self.onmessage = function(event) {
    var intLen = event.data;
    var LngRndNum = RetRndNum(intLen);
    var objWorker = new Worker("js9_1_1.js");
    objWorker.postMessage(LngRndNum);
    objWorker.onmessage = function(event) {
        var strRetHTML = "<span><b> ";
        strRetHTML += intLen + " </b>位随机数为：<b> ";
        strRetHTML += LngRndNum;
```

```
            strRetHTML += " </b> " + event.data + " </span><br>";
            self.postMessage(strRetHTML);
        }
    }
    // 生成指定长度的随机数
    function RetRndNum(n) {
        var strRnd = "";
        for (var intI = 0; intI < n; intI++) {
            strRnd += Math.floor(Math.random() * 10);
        }
        return strRnd;
    }
```

在代码清单 10-9 的 JavaScript 文件 js9_1.js 代码中，又调用了另外一个后台线程脚本文件 js9_1_1.js，在该文件中，检测随机数奇偶的特征，其实现的代码如下所示：

```
// JavaScript Document
self.onmessage = function(event) {
    if (event.data % 2 == 0) {
        self.postMessage("偶数");
    } else {
        self.postMessage("奇数");
    }
    self.close();
}
```

3. 页面效果

该页面在 Firefox 5.0 浏览器中执行的效果如图 10-9 所示。

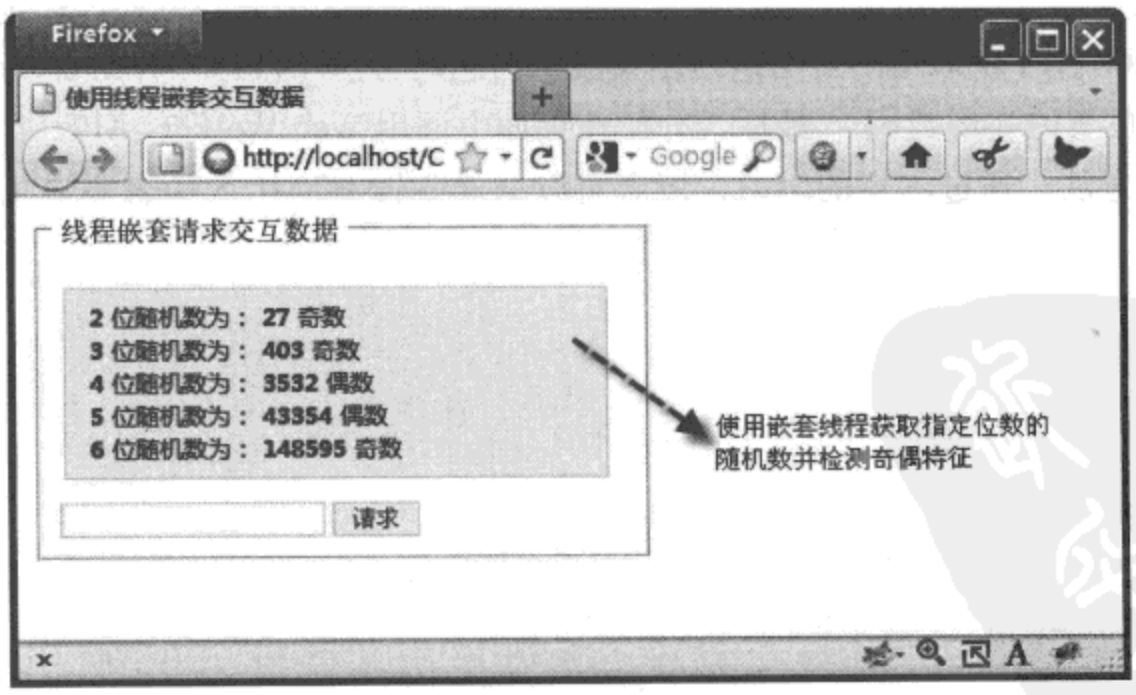


图 10-9　使用线程嵌套交互数据

4. 源码分析

本实例是在实例 10-7 的基础上进行的修改，为了在前台页面中既显示按指定位数生成的

随机数，又检测随机数奇偶特征，在调用的后台线程中使用了嵌套的方式来实现。

脚本文件 js9.js 中指定的后台线程文件 js9_1.js 为主线程。在该后台主线程文件中，首先在 message 事件中获取前台页面传来的生成随机数的长度值 event.data，并保存至变量 intLen 中；然后，根据该变量值调用 RetRndNum() 函数，生成一个指定长度的随机数，并保存至变量 LngRndNum 中；接下来，创建一个后台子线程对象 objWorker，并指定该对象的脚本文件为 js9_1_1.js，通过 postMessage() 方法将生成的随机数发送给 objWorker 对象对应的脚本文件。

在子线程 js9_1_1.js 文件中，通过监测 message 事件获取 event.data 值，得到主线程传回的随机数，并通过“event.data % 2”的方法检测随机数的奇偶性，通过 postMessage() 方法返回给主线程。主线程 js9_1.js 文件在监测的 message 事件中接收子线程传回的随机数奇偶特征，与生成的随机数一起组成一个字符串，通过调用 self.postMessage() 方法，将字符串传递给前台页面。前台页面在监测的 message 事件中，获取后台主线程传回的数据 event.data，即字符串内容，显示在页面中。

需要说明的是，主线程向子线程发送数据时，使用子线程对象的 postMessage() 方法，即 objWorker.postMessage(LngRndNum)；而在向前台页面发送数据时，则使用线程自身的 postMessage() 方法，即 self.postMessage(strRetHTML)，或省略 self 也可以。

10.4 本章小结

本章详细介绍了使用服务器端推送数据的 Web Sockets API，以及客户端与服务器端如何通过 Socket 建立连接后的数据交互过程；然后，以理论结合实例的方式，详细阐述了页面中使用 Geolocation API 获取当前页面地理位置数据信息的方法与技巧，并结合 Google 地图获取锁定页面所在位置的实现代码；最后，介绍在页面中调用 Web Workers API，以及利用后台线程交互数据的方法。通过本章的学习，读者可以全面掌握 HTML 5 中各类应用型 API 的使用方法。

第 11 章

HTML 5 中元素的拖放

本章内容

- 拖放基础
- dataTransfer 对象应用详解
- 拖放应用实战
- 本章小结

在第 5 章中，详细介绍过从其他应用窗口拖动文件到浏览器中来实现文件上传，这种跨平台拖动文件的方式，是 HTML 5 中拖放 API 的一个重要特征。

在 HTML 5 中，拖放元素变得十分简单，只要为元素添加一个“draggable”属性，并设置该属性的值为“true”即可实现元素的拖放。在拖动元素的过程中，不仅可以触发多个事件，还可以通过 dataTransfer 对象携带拖动元素的内容，并将其放入目标元素中；同时，当元素被拖动时，还可以控制鼠标的形状与移动时的效果，接下来逐一介绍各元素的拖放功能。

11.1 拖放基础

在 HTML 4 及以前的版本中，如果需要实现一个元素的拖放效果，需要结合该元素的 onmousedown、onmousemove、onmouseup 等多个事件来共同完成。这种方式代码相对复杂与冗余，而且也仅限于在浏览器内的元素间拖放，不能实现跨应用拖放。

在 HTML 5 中，一旦将某个元素的“draggable”属性值设置为“true”，该元素就可以实现拖放的效果。并且在拖放过程中，也能触发众多的事件。通过调用这些事件，更加准确、及时地反映元素从拖动到放下这一过程中的各种状态与数据值。

11.1.1 使用 JavaScript 代码实现拖放

在介绍使用 HTML 5 中的元素实现拖放之前，先来回顾在 HTML 4 及以前的版本中，如何通过 JavaScript 代码，实现某个元素的拖放。

下面通过实例 11-1 介绍使用 JavaScript 代码实现元素拖放的过程。

实例 11-1　使用 JavaScript 代码实现元素拖放

1. 功能描述

在新建的 HTML 页面中，添加两个 元素，并且使 ID 号为“divFrame”的元素包含 ID 号为“divTitle”的元素。当用户将鼠标移到 ID 号为“divTitle”的元素上时，按下鼠标左键并移动鼠标，可以拖动整个 ID 号为“divFrame”的元素。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 11-1.html，加入代码如代码清单 11-1 所示。

代码清单 11-1　使用 JavaScript 代码实现元素拖放

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 JavaScript 代码实现元素拖放 </title>
<link href="Css/css11.css" rel="stylesheet" type="text/css">
```

```
<script type="text/javascript" language="jscript"
        src="Js/js1.js"/>
</script>
</head>
<body onLoad="pageload();">
    <div id="divFrame" >
        <div id="divTitle">请拖动我</div>
    </div>
</body>
</html>
```

在代码清单 11-1 中，页面导入一个 JavaScript 文件 js1.js，其中自定义了一个加载页面时调用的函数 pageload()，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var started;
var initX,initY,offsetX,offsetY;
// 自定义加载页面时调用的函数
function pageload() {
    var divTitle = $$("divTitle");
    var divFrame = $$("divFrame");
    divFrame.style.left = 30 + "px";
    divFrame.style.top = 20 + "px";
//按下鼠标时触发的事件
    divTitle.onmousedown = function(e) {
        started = true;
        initX = parseInt(divFrame.style.left);
        initY = parseInt(divFrame.style.top);
        offsetX = e.clientX;
        offsetY = e.clientY;
    };
//鼠标移动时触发的事件
    divFrame.onmousemove = function(e) {
        if (started) {
            var x = e.clientX - offsetX + initX;
            var y = e.clientY - offsetY + initY;
            divFrame.style.left = x + "px";
            divFrame.style.top = y + "px";
            divTitle.innerHTML=" 已拖动 ";
        }
    };
//鼠标弹起时触发的事件
    divFrame.onmouseup = function() {
```

```
            started = false;
            document.onmousemove = null;
        }
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的效果如图 11-1 所示。

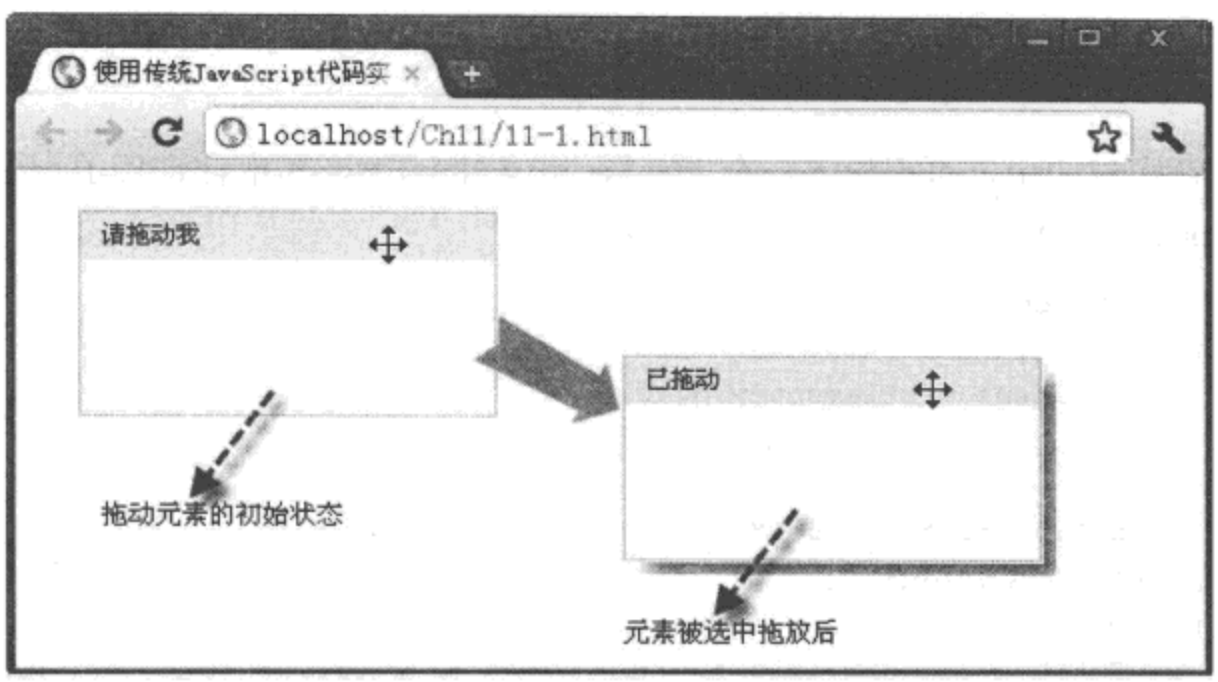


图 11-1　使用 JavaScript 代码实现元素拖放

4. 源码分析

在本实例的 JavaScript 代码中，为了拖放 ID 号为“divFrame”的元素，首先将为 ID 号为“divTitle”的被包含元素设为拖放标签，当单击该元素时，触发 mousedown 事件，在该事件中设置变量“started”的值为“true”，表示可以拖放；然后，获取被拖放元素“divFrame”与鼠标的坐标位置，分别保存到变量 initX，initY，offsetX，offsetY 中。

当移动被拖放元素“divFrame”时，触发 mousemove 事件。在该事件中，先检测“started”变量值是否为“true”，如果成立，则根据鼠标新旧坐标位置的对比及被拖放元素“divFrame”原有坐标的数据得到元素被移动后的新坐标数据，并将获取的数值赋给被拖放元素“divFrame”的横、纵坐标属性，从而实现元素被拖放的效果。

拖放停止时触发 onmouseup 事件，在该事件中，将“started”变量值设置为“false”，并将 DOM 的 mousemove 事件设置为“null”，阻止其他鼠标移动事件的发生。

11.1.2　在 HTML 5 中实现拖放时触发的事件

在 HTML 5 中，拖放元素变得十分简单，只要将元素的“draggable”属性设置为“true”，就可以实现元素的拖放功能。元素在拖放过程中，触发了多个事件，如表 11-1 所示。

表 11-1　元素拖放时触发的相关事件

事件主体	事件名称	描　述
被拖放元素	dragstart	在开始拖放被拖放元素时触发
被拖放元素	drag	在正在拖放被拖放元素时触发
经过 / 目标元素	dragenter	在被拖放元素进入某元素时触发
经过 / 目标元素	dragover	在被拖放元素在某元素内移动时触发
目标元素	dragleave	在被拖放元素移出目标元素时触发
目标元素	drop	在目标元素完全接收被拖放元素时触发
被拖放对象元素	dragend	在整个拖放操作结束时触发

下面通过实例 11-2 介绍元素在拖放过程中触发的事件。

实例 11-2　元素在拖放过程中触发的事件

1. 功能描述

在新建的 HTML 页面中，分别使用 元素添加两个区域，一个是 ID 号为“divDrag”的拖放元素，另一个是 ID 号为“divArea”的目标元素。当用户将“divDrag”元素拖放到目标元素“divArea”时，在页面中将显示所触发的重要事件状态。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 11-2.html，加入代码如代码清单 11-2 所示。

代码清单 11-2　元素在拖放过程中触发的事件

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>元素在拖放过程中触发的事件</title>
<link href="Css/css11.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js2.js"/>
</script>
</head>
<body onLoad="pageload();">
  <div class="wPub">
    <div class="wPub">
      <div id="divDrag" draggable="true"></div>
      <div id="divTips"></div>
    </div>
    <div id="divArea"></div>
  </div>
</body>
</html>
```

在代码清单 11-2 中，页面导入一个 JavaScript 文件 js2.js，其中自定义了一个函数 pageload()，在加载页面时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义加载页面时调用的函数
function pageload() {
    var Drag = $$("divDrag");
    var Area = $$("divArea");
    // 添加被拖放元素的 dragstart 事件
    Drag.addEventListener("dragstart",
    function(e) {
        Status_Handle("元素正在开始拖动 ...")
    });
   // 添加目标元素的 drop 事件
   Area.addEventListener("drop",
   function(e) {
        Status_Handle("元素拖入成功！")
    });
   // 添加目标元素的 dragleave 事件
   Area.addEventListener("dragleave",
   function(e) {
        Status_Handle("拖动元素正在离开 ...")
    });
}
// 自定义显示执行过程中状态的函数
function Status_Handle(message) {
    $$("divTips").innerHTML += message + "<br>";
}
// 添加页面的 dragover 事件
document.ondragover = function(e) {
   // 阻止默认方法，取消拒绝被拖放
   e.preventDefault();
}
// 添加页面 drop 事件
document.ondrop = function(e) {
   // 阻止默认方法，取消拒绝被拖放
   e.preventDefault();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的效果如图 11-2 所示。

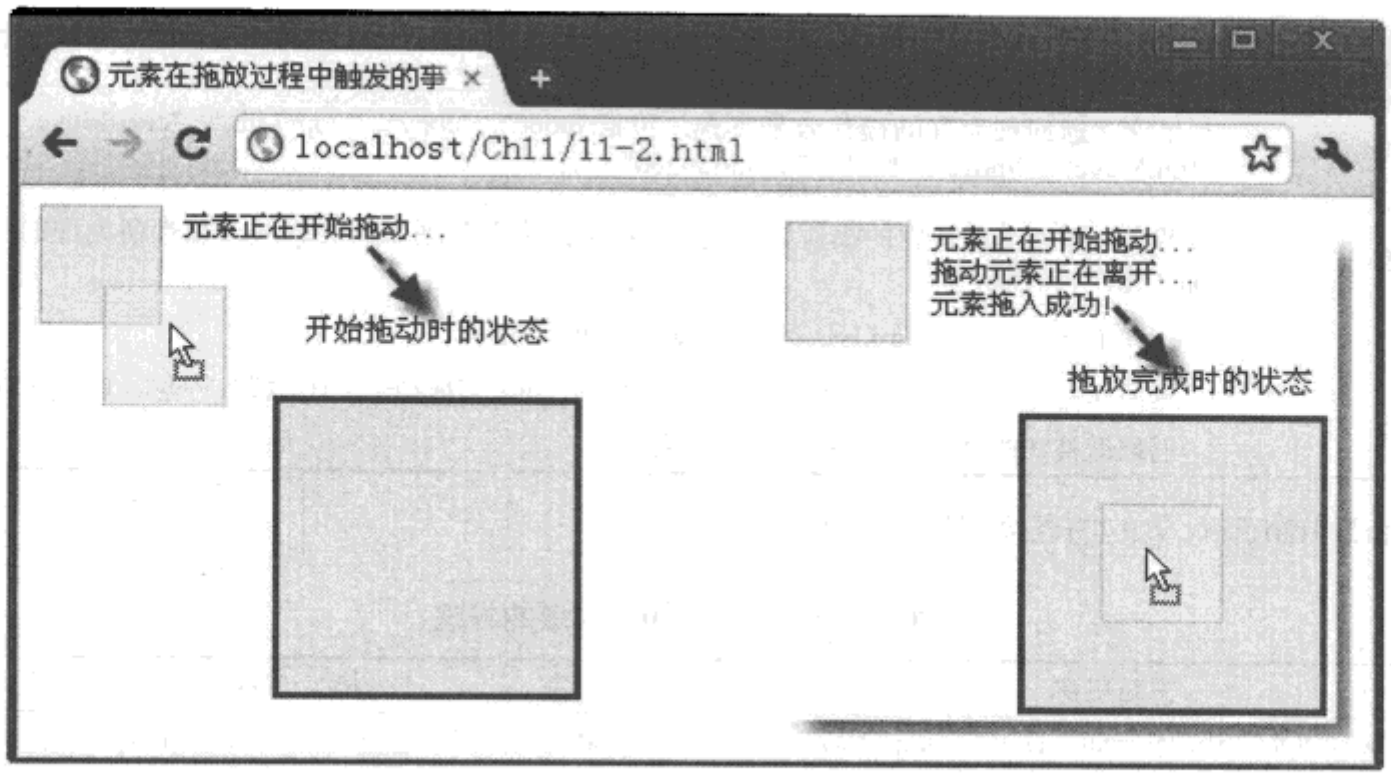


图 11-2　在页面中显示元素在拖放过程中触发的事件

4. 源码分析

在本实例中，首先，将拖放元素“divDrag”的“draggable”属性值设置为“true”，同时，添加页面的 dragover 与 drop 事件。在这两个事件中，都使用 e.preventDefault() 方法取消页面的默认值，允许拖放页面。由于在拖放元素的过程中，首先被拖放的是页面，如果页面都不可以拖放，那么页面中的元素也将不可被拖放。

接下来，分别为拖放元素“divDrag”和目标元素“divArea”添加 dragstart 事件和 drop、dragleave 事件。在添加的这些事件中，通过调用自定义的函数 Status_Handle()，在页面中显示事件触发时的各种状态值。

11.2　dataTransfer 对象应用详解

在实例 11-2 中，拖放元素还没有放入目标元素中，而要实现这一功能，需要调用 dataTransfer 对象。该对象专门用于携带拖放过程中的数据，它拥有许多实用的属性和方法，例如，“dropEffect”与“effectAllowed”属性结合使用，可以自定义在拖放过程中的拖放效果；使用 setData() 与 getData() 方法可以将拖放元素的数据放置于目标元素中。dataTransfer 对象的属性如表 11-2 所示。

表 11-2 dataTransfer 对象的属性

属性名称	描　述
effectAllowed	设置 / 返回被允许的操作效果类别，包括 "none"、"copy"、"copyLink"、"copyMove"、"link"、"linkMove"、"move"、"all"、"uninitialized"
dropEffect	设置 / 返回当前选定的操作效果类别，如果此类别不是 effectAllowed 属性所允许的范围值，那么操作将失败
items	返回 DataTransferItemList 对象，即拖动数据
types	返回 dragstart 事件中设置的数据格式。此外，如果被拖动是文件，那么将返回 Files 型字符串
files	返回被拖动的文件 FileList 清单（如果有）

dataTransfer 对象的方法如表 11-3 所示。

表 11-3 dataTransfer 对象的方法

方法名称	描　述
setData(DOMString format，DOMString data)	将拖放元素中数据存入 dataTransfer 对象中
getData(DOMString format)	读取存入 dataTransfer 对象中的元素
setDragImage(Element img，long x，long y)	设置拖放过程中的图标
clearData(DOMString format)	清空 dataTransfer 对象中指定格式的数据

在 dataTransfer 对象的方法中，都使用了“format”作为形参，表示读取 / 存入 / 清空时的数据格式，该参数的格式包含如下几种：

- text/plain （文本文字格式）
- text/html （HTML 页面代码格式）
- text/xml （XML 字符格式）
- text/url-list （URL 格式列表）

11.2.1 使用 setData 与 getData 方法存入与读取拖放数据

在 HTML 5 中，可以通过访问 dataTransfer 对象携带拖放元素的数据，携带的过程就是在拖放元素时，调用 setData() 方法将数据存入 dataTransfer 对象中；在放入目标元素时调用 getData() 方法读取存入的数据。当然，存入与读取时，指定的数据格式都是相同的。

下面通过实例 11-3 介绍使用 setData() 与 getData() 方法存入与读取拖放数据的过程。

实例 11-3 使用 setData 与 getData 方法存入与读取拖放数据

1. 功能描述

在实例 11-2 的基础上修改代码，当拖放元素触发 dragstart 事件时，将元素的相关数据通过 setData() 方法存入 dataTransfer 对象中。在目标元素接收拖放元素时，触发 drop 事件，在该事件中，读取 dataTransfer 对象中存入的数据，并放入目标元素中。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 11-3.html，加入代码如代码清单 11-3 所示。

代码清单 11-3　使用 setData 与 getData 方法存入与读取拖放数据

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用 setData 与 getData 方法存入与读取拖放数据 </title>
<link href="Css/css11.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js3.js"/>
</script>
</head>
<body onLoad="pageload();">
  <div class="wPub">
    <div class="wPub">
      <div id="divDrag" draggable="true"></div>
      <div id="divTips"></div>
    </div>
    <div id="divArea"></div>
  </div>
</body>
</html>
```

在代码清单 11-3 中，页面导入一个 JavaScript 文件 js3.js，其中自定义一个函数 pageload()，在页面加载时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义返回 HTML 内容函数
function RetDragHTMLById(Id) {
    var strHTML = "<div id=" + Id + "></div>";
    return strHTML;
}
// 自定义加载页面时调用的函数
function pageload() {
    var Drag = $$("divDrag");
    var Area = $$("divArea");
    // 添加被拖放元素的 dragstart 事件
    Drag.addEventListener("dragstart",
    function(e) {
        var objDtf = e.dataTransfer;
        objDtf.setData("text/html", RetDragHTMLById(this.id));
```

```
        },
        false);
        // 添加目标元素的 drop 事件
        Area.addEventListener("drop",
        function(e) {
            var objDtf = e.dataTransfer;
            var strHTML = objDtf.getData("text/html");
            Area.innerHTML += strHTML;
            e.preventDefault();
            e.stopPropagation();
        },
        false);
}
// 添加页面的 dragover 事件
document.ondragover = function(e) {
        // 阻止默认方法，取消拒绝被拖放
        e.preventDefault();
}
// 添加页面 drop 事件
document.ondrop = function(e) {
        // 阻止默认方法，取消拒绝被拖放
        e.preventDefault();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行后的效果如图 11-3 所示。

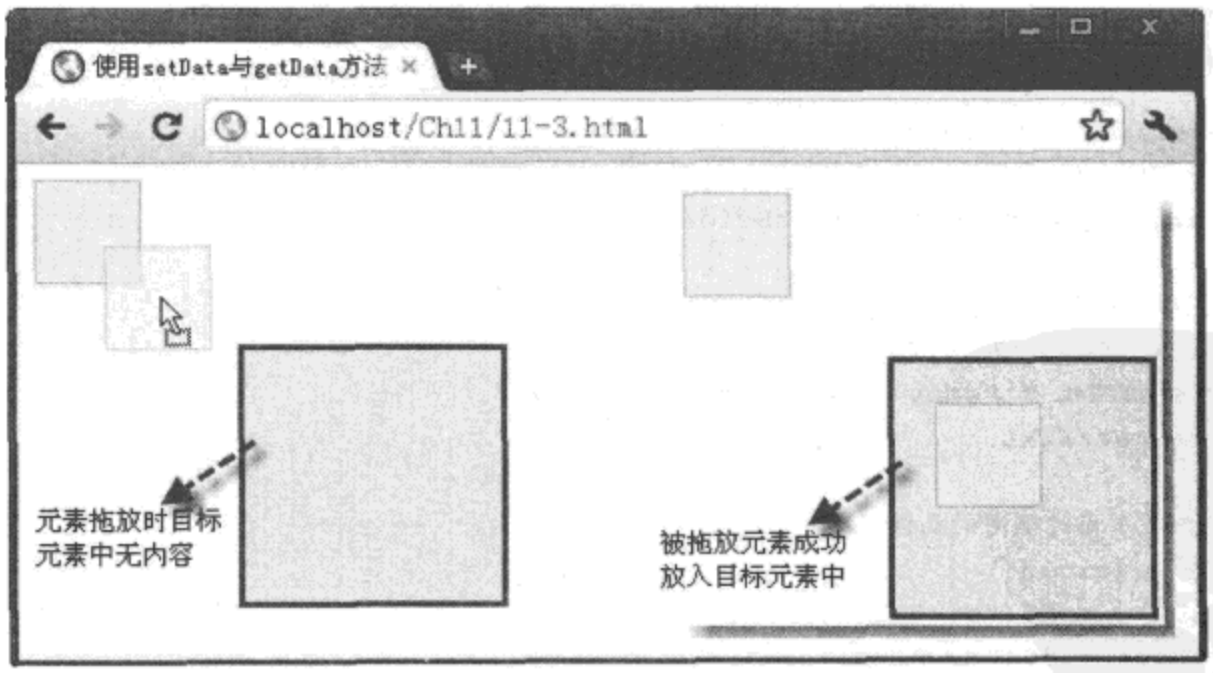


图 11-3　使用 setData 与 getData 方法存入与读取拖放数据

4. 源码分析

在本实例中，首先，在拖放元素开始被拖动时触发 dragstart 事件。在该事件中，this 表

示触发事件的元素本身，而“this.id”则表示触发事件元素的 ID 号，即“divDrag”。在调用 setData() 方法向 dataTransfer 对象存入对象时，数据的格式为“text/html”，内容为“this.id”作为实参的自定义函数 RetDragHTMLById() 的返回值，即一段 HTML 格式的字符串数据。

当目标元素“divArea”完全接收拖放元素时，便触发 drop 事件。在该事件中，通过调用 getData() 方法从 dataTransfer 对象中读取携带的数据，即 HTML 格式的字符串，并作为目标元素“divArea”包含的内容一起显示在页面中。

在 drop 事件中，同样需要调用 e.preventDefault() 方法阻止默认方法，取消拒绝被拖放的设置；同时，调用 e.stopPropagation() 方法停止其他事件的进程，否则，目标元素不能正常接收拖放来的数据。

11.2.2　使用 setDragImage 方法设置拖放图标

在 HTML 5 中，一个元素在被拖放时，还可以改变鼠标的图标，即可以自定义拖放元素时的鼠标图标。要实现这一功能，需要调用 dataTransfer 对象的 setDragImage() 方法，调用格式为：

```
setDragImage(Element img, long x, long y)
```

其中，参数 img 是一个元素，表示拖放时显示的 <img> 元素图标，x 表示图标距离鼠标指针的 x 轴方向偏移值，y 表示图标距离鼠标指针的 y 轴方向偏移值。

下面通过实例 11-4 介绍使用 setDragImage() 方法设置拖放图标的过程。

实例 11-4　使用 setDragImage 方法设置拖放图标

1. 功能描述

在新创建的 HTML 页面中，添加两个 区域，一个作为拖放元素，另一个作为目标元素，用于接收拖放来的元素或数据。当选中拖放元素，并按住鼠标开始拖放时，鼠标的图标将发生变化；拖放完成时，在目标元素中显示“拖动时改变图标”的字样。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 11-4.html，加入代码如代码清单 11-4 所示。

代码清单 11-4　使用 setDragImage 方法设置拖放图标

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>setDragImage 方法设置拖放图标 </title>
<link href="Css/css11.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js4.js"/>
</script>
```

```
</head>
<body onLoad="pageload();">
  <div class="wPub">
    <div class="wPub">
      <div id="divDrag" draggable="true"></div>
      <div id="divTips"></div>
    </div>
    <div id="divArea"></div>
  </div>
</body>
</html>
```

在代码清单 11-4 中，页面导入一个 JavaScript 文件 js4.js，其中自定义一个函数 pageload()，在页面加载时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
    var Drag = $$("divDrag");
    var Area = $$("divArea");
    // 创建一个 img 元素，并设置图片来源
    var objImg = document.createElement("img");
    objImg.src = "Images/img01.jpg";
    // 添加被拖放元素的 dragstart 事件
    Drag.addEventListener("dragstart",
    function(e) {
        var objDtf = e.dataTransfer;
        objDtf.setDragImage(objImg, 0, 0);
        objDtf.setData("text/plain", "拖动时改变图标");
    },
    false);
    // 添加目标元素的 drop 事件
    Area.addEventListener("drop",
    function(e) {
        var objDtf = e.dataTransfer;
        var strText = objDtf.getData("text/plain");
        Area.textContent += strText;
        e.preventDefault();
        e.stopPropagation();
    },
    false);
}
// 添加页面的 dragover 事件
document.ondragover = function(e) {
    // 阻止默认方法，取消拒绝被拖放
```

```
    e.preventDefault();
}
// 添加页面 drop 事件
document.ondrop = function(e) {
    // 阻止默认方法，取消拒绝被拖放
    e.preventDefault();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 11-4 所示。

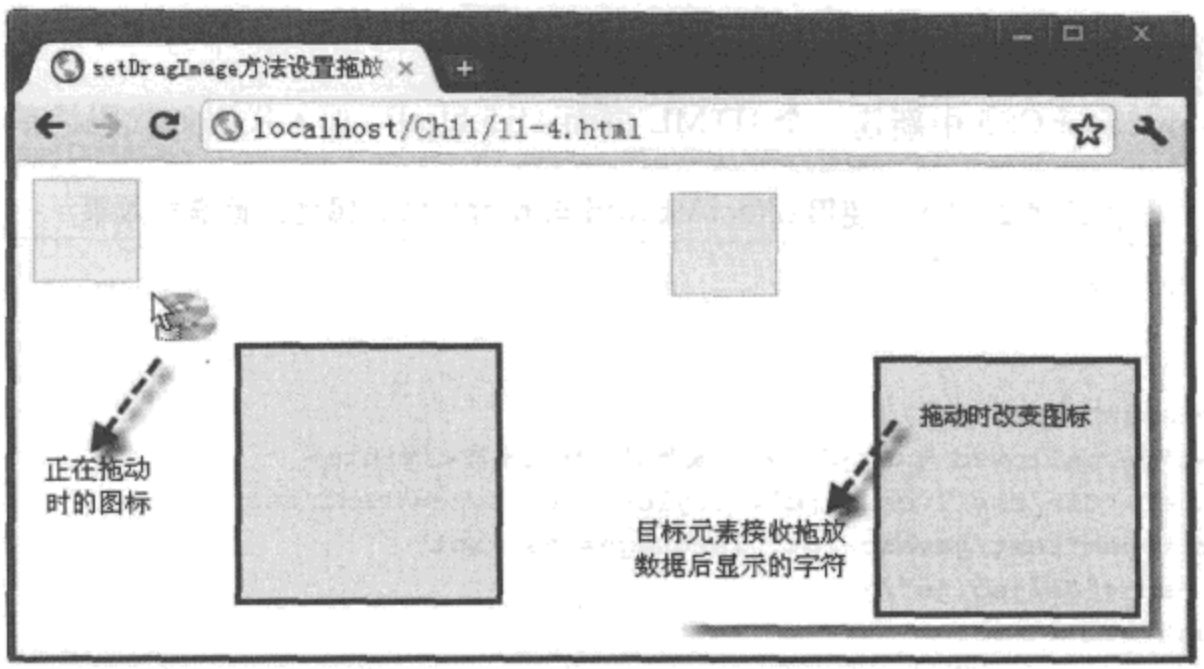


图 11-4　使用 setDragImage 方法设置拖放图标

4. 源码分析

在本实例中，首先，在页面加载时创建一个 <img> 元素，并为该元素设置 src 属性，指定图片的来源；然后，在拖放元素的 dragstart 事件中，调用 setDragImage() 方法自定义拖放图标，在调用过程中，将新创建的 <img> 元素作为第一个实参，第二个和第三个参数的值分别为 0，表示鼠标与图标在同一个坐标位置上。

为了确定在拖放元素完全被目标元素所接收，在元素拖放时，通过 dataTransfer 对象携带一个内容为“拖动时改变图标”的文字数据；当目标元素触发 drop 事件时，将从 dataTransfer 对象中读取该数据，显示在目标元素中。

11.2.3　使用 effectAllowed 与 dropEffect 属性设置拖放效果

结合“effectAllowed”与“dropEffect”这两个属性，可以自定义拖放过程中的效果。两个属性虽然都是为了实现同一功能，但绑定的元素不同：“effectAllowed”属性用于 dragstart 事件中，绑定被拖放元素；而 dropEffect() 属性用于绑定目标元素。注意，“dropEffect”属性中指定的效果，必须在“effectAllowed”属性中存在，否则，也不能实现自定义的拖放效果。

下面通过实例 11-5 介绍使用“effectAllowed”与“dropEffect”属性设置拖放效果的过程。

实例 11-5 使用 effectAllowed 与 dropEffect 属性设置拖放效果

1. 功能描述

在实例 11-4 的基础上修改代码，当拖放元素在触发 dragstart 事件时，将 dataTransfer 对象的“effectAllowed”属性值设置为“move”；同时，为目标元素添加一个 dragover 事件，当通过 dataTransfer 对象接收数据时，将对象的“dropEffect”属性值也设置为“move”，从而实现自定义元素拖放效果的功能。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 11-5.html，加入代码如代码清单 11-5 所示。

代码清单 11-5 使用 effectAllowed 与 dropEffect 属性设置拖放效果

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>effectAllowed 与 dropEffect 属性设置拖放效果 </title>
<link href="Css/css11.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js5.js"/>
</script>
</head>
<body onLoad="pageload();">
  <div class="wPub">
    <div class="wPub">
      <div id="divDrag" draggable="true"></div>
      <div id="divTips"></div>
    </div>
    <div id="divArea"></div>
  </div>
</body>
</html>
```

在代码清单 11-5 中，页面导入一个 JavaScript 文件 js5.js，其中自定义一个函数 pageload()，在页面加载时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
    var Drag = $$("divDrag");
```

```
var Area = $$("divArea");
// 添加被拖放元素的 dragstart 事件
Drag.addEventListener("dragstart",
function(e) {
    var objDtf = e.dataTransfer;
    objDtf.effectAllowed = "copy";
    objDtf.setData("text/plain", "拖动时动画效果");
},
false);
// 添加目标元素的 dragover 事件
Area.addEventListener("dragover",
function(e) {
    var objDtf = e.dataTransfer;
    objDtf.dropEffect = "copy";
    e.preventDefault();
},
false);
// 添加目标元素的 drop 事件
Area.addEventListener("drop",
function(e) {
    var objDtf = e.dataTransfer;
    var strText = objDtf.getData("text/plain");
    Area.textContent += strText;
    e.preventDefault();
    e.stopPropagation();
    },
    false);
}
// 添加页面的 dragover 事件
document.ondragover = function(e) {
    // 阻止默认方法，取消拒绝被拖放
    e.preventDefault();
}
// 添加页面 drop 事件
document.ondrop = function(e) {
    // 阻止默认方法，取消拒绝被拖放
    e.preventDefault();
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 11-5 所示。

4. 源码分析

在本实例中，首先，在拖放元素的 dragstart 事件中，将 dataTransfer 对象的“effectAllowed”属性值设为“copy”，表示被拖放的元素以复制的方式放置在目标元素中。

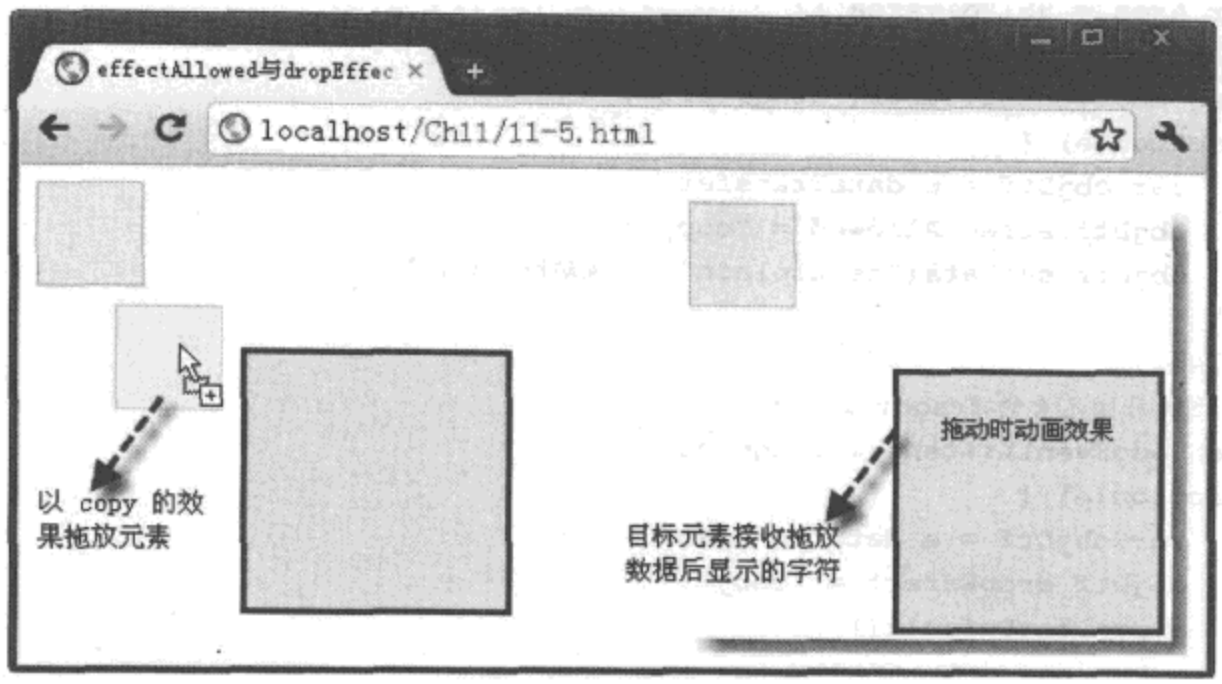


图 11-5　使用 effectAllowed 与 dropEffect 属性设置元素拖放效果

在目标元素的 dragover 事件中，为了响应拖放元素设置的效果，将 dataTransfer 对象的“dropEffect”属性值设置为“copy”，使“effectAllowed”与“dropEffect”属性值保持一致，从而最终实现以复制的方式将拖放元素放入目标元素中的功能。

“effectAllowed”属性值除“copy”之外，还包括“move”，表示被拖动的元素以移动的方式放置在目标元素中；“link”表示被拖动的元素以浏览的方式放置中目标元素中，该属性只对拖放 URL 有效；“none”表示不允许拖放元素；“all”表示允许“dropEffect”属性所有支持的效果。

需要说明的是，除非“effectAllowed”属性值为“all”，“dropEffect”与“effectAllowed”属性值必须保持一致，否则，将不能显示设置的拖放效果。

11.3　拖放应用实战

在 HTML 5 中，拖放是一个十分重要的功能，借助拖放操作的 API，可以在应用程序中实现相对复杂的功能与特效。例如，购物网中的购物车，用户可以将选中的商品以拖放的方式放入购物车中；同时，购物车接收拖来的数据，检测重复性并自动计价，实现商品入购物车的功能。在相册管理时，用户可以将需要删除的图片以拖放的方式放入回收站；回收站在接收图片时，改变回收站的图标样式，并在相册中删除被拖放的图片。

本节以购物车和相册的管理为例，详细介绍拖放功能在应用程序中的使用方法与技巧。

11.3.1　购物车的实现

通常情况下，用户在购买商品时，单击该商品的“购买”链接，便可以将该商品放入购物车中。但在 HTML 5 中，也可以利用元素的拖放 API，直接将需要购买的商品拖放至购物车

中，实现购买的功能。

下面通过实例 11-6 介绍使用拖放 API 将商品拖入购物车的过程。

实例 11-6　使用拖放 API 将商品拖入购物车

1. 功能描述

在新创建的 HTML 页面中展示 4 件图书商品，用户可以选中其中的任意一件，按住鼠标，以拖放的方式将选择的商品放入购物车中；同时，购物车接收拖来的商品数据，自动增加一条选择记录，并显示商品的基本信息。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 11-6.html，加入代码如代码清单 11-6 所示。

代码清单 11-6　使用拖放 API 将商品拖入购物车

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用拖放API将商品拖入购物车</title>
<link href="Css/css11.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js6.js"/>
</script>
</head>
<body onLoad="pageload();">
  <ul>
    <li class="liF">
        <img src="Images/img02.jpg" id="img02"
             alt="42" title="2006作品" draggable="true">
    </li>
    <li class="liF">
        <img src="Images/img03.jpg" id="img03"
             alt="56" title="2008作品" draggable="true">
    </li>
    <li class="liF">
        <img src="Images/img04.jpg" id="img04"
             alt="52" title="2010作品" draggable="true">
    </li>
    <li class="liF">
        <img src="Images/img05.jpg" id="img05"
             alt="59" title="2011作品" draggable="true">
    </li>
  </ul>
  <ul id="ulCart">
    <li class="liT">
```

```
            <span> 书名 </span>
            <span> 定价 </span>
            <span> 数量 </span>
            <span> 总价 </span>
        </li>
    </ul>
</body>
</html>
```

在代码清单 11-6 中，页面导入一个 JavaScript 文件 js6.js，其中自定义一个函数 pageload()，在页面加载时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
// 自定义页面加载时调用的函数
function pageload() {
    // 获取全部的图书商品
    var Drag = document.getElementsByTagName("img");
    // 遍历每一个图书商品
    for (var intI = 0; intI < Drag.length; intI++) {
        // 为每一个商品添加被拖放元素的 dragstart 事件
        Drag[intI].addEventListener("dragstart",
        function(e) {
            var objDtf = e.dataTransfer;
            objDtf.setData("text/html", addCart(this.title, this.alt, 1));
        },
        false);
    }
    var Cart = $$("ulCart");
    // 添加目标元素的 drop 事件
    Cart.addEventListener("drop",
    function(e) {
        var objDtf = e.dataTransfer;
        var strHTML = objDtf.getData("text/html");
        Cart.innerHTML += strHTML;
        e.preventDefault();
        e.stopPropagation();
    },
    false);
}
// 添加页面的 dragover 事件
document.ondragover = function(e) {
    // 阻止默认方法，取消拒绝被拖放
    e.preventDefault();
}
// 添加页面 drop 事件
```

```
document.ondrop = function(e) {
    // 阻止默认方法，取消拒绝被拖放
    e.preventDefault();
}
// 自定义向购物车中添加记录的函数
function addCart(a, b, c) {
    var strHTML = "<li class='liC'>";
    strHTML += "<span>" + a + "</span>";
    strHTML += "<span>" + b + "</span>";
    strHTML += "<span>" + c + "</span>";
    strHTML += "<span>" + b * c + "</span>";
    strHTML += "</li>";
    return strHTML;
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 11-6 所示。

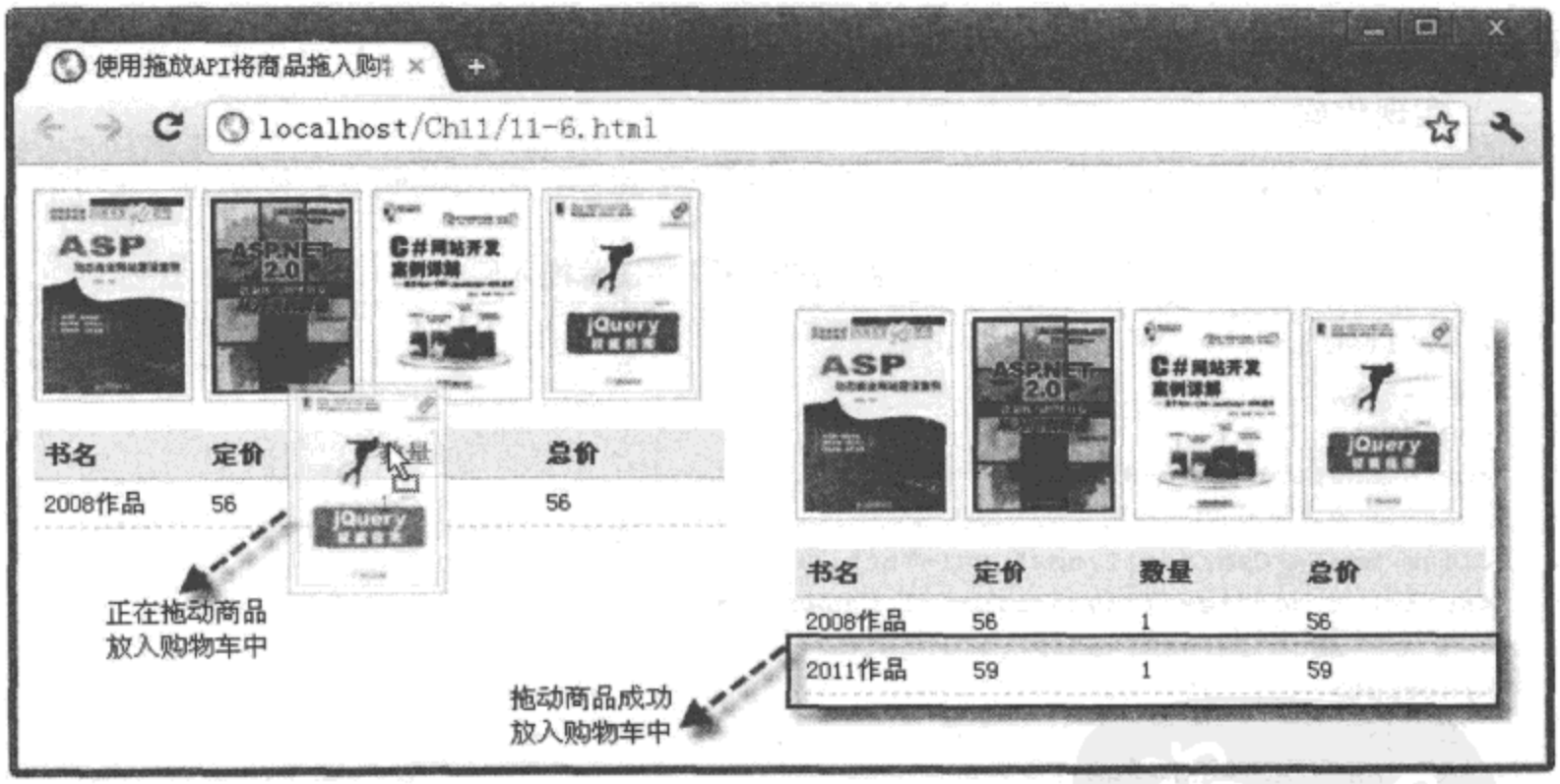


图 11-6　使用拖放 API 将商品拖入购物车

4. 源码分析

在本实例中，为了使所有的图书商品都具有可拖放的功能，首先在 HTML 页面中，为每个商品元素添加一个“draggable”属性，并将该属性的值设为“true”，表示允许拖放。

为了在 JavaScript 代码中，给每一个商品元素添加“dragstart”事件，先使用 getElementsByTagName() 方法获取全部的商品元素；然后，采用遍历的方式，为每个商品元素添加一个“dragstart”事件。在该事件中，调用自定义的函数 addCart()，将商品的基本信息组成一段 HTML 代码格式数据，存入 dataTransfer 对象中，用于购物车中的接收。

在购物车中，通过访问 dataTransfer 对象，读取存入的某个商品的数据信息，即一段 HTML 代码，将该代码追加至购物车元素的 innerHTML 属性中，从而实现购物车中新增一条选择记录的效果。完整的实现过程如代码中加粗部分所示。

11.3.2 相册的管理

除了上述应用，还可以通过拖放 API，将相册中的某张图片，以拖放的方式放入回收站，从而实现管理相册的功能。

下面通过实例 11-7 介绍使用拖放 API 将图片拖入回收站的过程。

实例 11-7 使用拖放 API 将图片拖入回收站

1. 功能描述

在新建的 HTML 页面中，以列表的方式展示相册中的 3 张图片，用户可以选中任意一张，按住鼠标，以拖动的方式放入右下角的回收站中。拖放成功后，将在页面中显示删除图片的记录数；同时，回收站的样式也发生了相应的变化。

2. 实现代码

在 Dreamweaver CS5 中新建一个 HTML 页面 11-7.html，加入代码如代码清单 11-7 所示。

代码清单 11-7 使用拖放 API 将图片拖入回收站

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>使用拖放API将图片拖入回收站</title>
<link href="Css/css11.css" rel="stylesheet" type="text/css">
<script type="text/javascript" language="jscript"
        src="Js/js7.js"/>
</script>
</head>
<body onLoad="pageload();">
 <div class="wPub">
  <ul>
    <li class="liF" id="li01">
        <img src="Images/img08.jpg" class="img95"
             draggable="true">
    </li>
    <li class="liF" id="li02">
        <img src="Images/img09.jpg" class="img95"
             draggable="true">
    </li>
    <li class="liF" id="li03">
        <img src="Images/img10.jpg" class="img95"
```

```
                draggable="true">
        </li>
      </ul>
      <p id="pStatus"></p>
      <div id="divRecycle" class="EmptRyl"></div>
    </div>
  </body>
  </html>
```

在代码清单 11-7 中，页面导入一个 JavaScript 文件 js7.js，其中自定义一个函数 pageload()，在页面加载时调用，其实现的代码如下所示：

```
// JavaScript Document
function $$(id) {
    return document.getElementById(id);
}
var intDeleNum = 0;
// 自定义页面加载时调用的函数
function pageload() {
    // 获取全部的图片信息
    var Drag = document.getElementsByTagName("li");
    // 遍历每一个图片元素
    for (var intI = 0; intI < Drag.length; intI++) {
        // 为每一个图片元素添加被拖放元素的 dragstart 事件
        Drag[intI].addEventListener("dragstart",
        function(e) {
            var objDtf = e.dataTransfer;
            objDtf.setData("text/plain", this.id);
        },
        false);
    }
    var Recy = $$("divRecycle");
    // 添加目标元素的 drop 事件
    Recy.addEventListener("drop",
    function(e) {
        var objDtf = e.dataTransfer;
        var intVal = objDtf.getData("text/plain");
        Drop_Event(intVal);
        Recy.className = "HaveRyl";
    },
    false);
}
// 添加页面的 dragover 事件
document.ondragover = function(e) {
    // 阻止默认方法，取消拒绝被拖放
    e.preventDefault();
}
// 添加页面 drop 事件
```

```
document.ondrop = function(e) {
    // 阻止默认方法，取消拒绝被拖放
    e.preventDefault();
}
// 自定义图片成功被拖入回收站时调用的函数
function Drop_Event(Id) {
    var Node = $$(Id);
    Node.parentNode.removeChild(Node);
    intDeleNum++;
    $$("pStatus").style.display = "block";
    $$("pStatus").innerHTML = " 已成功删除 " + intDeleNum + " 张图片 ";
}
```

3. 页面效果

该页面在 Chrome 10 浏览器中执行的效果如图 11-7 所示。

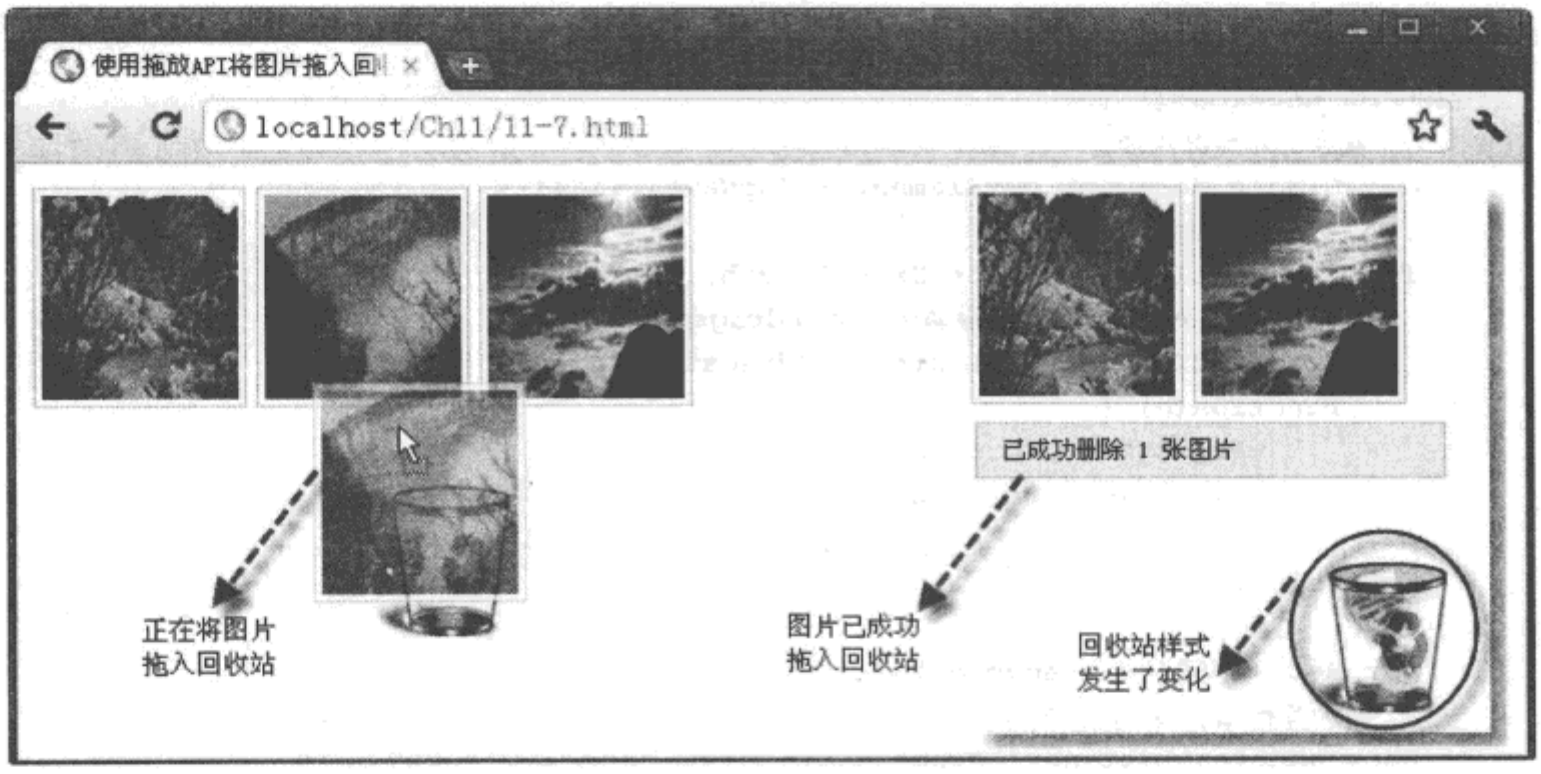


图 11-7　使用拖放 API 将图片拖入回收站

4. 源码分析

在本实例中，首先，使用 getElementsByTagName() 方法获取全部相册中的元素，然后，遍历全部元素，为每一个图片元素添加拖动时触发的“dragstart”事件。在该事件中，通过调用 dataTransfer 对象存入每一个图片元素对应的 ID 号，即 this.id 值。

在回收站元素的接收事件“drop”中，调用 dataTransfer 对象读取存入的单个图片元素的 ID 号，将该 ID 号作为实参，调用自定义的函数 Drop_Event()。在该函数中，先根据传回的 ID 号值，通过 removeChild() 方法移除相册中的指定图片，形成删除的效果；同时，通过全局变量 intDeleNum 累计删除图片的总量，并将该总量值显示在页面中。通过传回的 ID 号完成

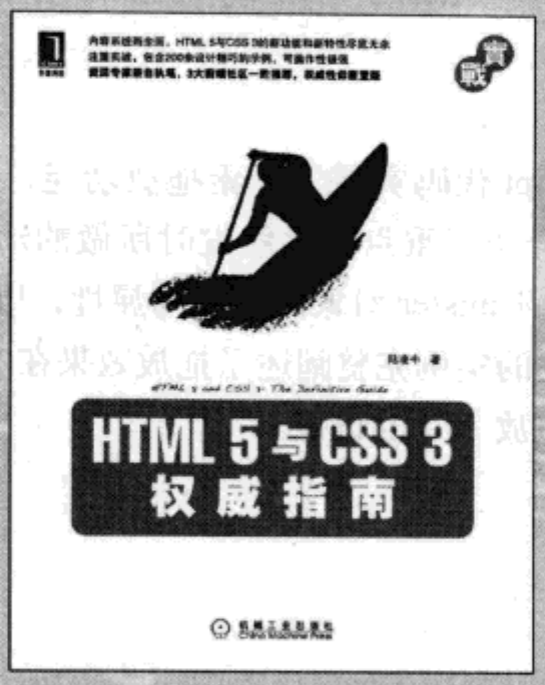
对函数 Drop_Event() 的调用，还通过“Recy.className”代码修改了拖放成功后的回收站的样式，完整的实现过程如代码中加粗部分所示。

11.4　本章小结

本章先使用传统的 JavaScript 代码实现了元素拖放功能，描述了元素在拖放过程中可能会触发的全部事件，结合实例介绍了重点事件触发时所做的动作。然后，由浅入深，结合有代表性的实例，详细介绍了 dataTransfer 对象的方法与属性，以及该对象在拖放元素过程中的使用技巧。最后，通过两个常见的实例完整阐述了拖放效果在实际的 Web 应用中的实现方法，使读者进一步加深对 HTML 5 拖放 API 的认识。

内容系统而全面，HTML 5与CSS 3的新功能和新特性尽览无余
注重实战，包含200余设计精巧的示例，可操作性极强
资深专家亲自执笔，3大前端社区一致推荐，权威性毋庸置疑

作者：陆凌牛 著　　ISBN 978-7-111-33624-2　　定价：69.00 元



软件开发领域从来不缺少新技术和新产品，Web前端领域尤其如此。2005年，Ajax技术的正式出现给沉寂多年的Web前端领域注入了新的活力；紧接着，各种Ajax框架如雨后春笋般涌现出来，又为Web前端领域打了一针兴奋剂；如今，HTML 5和CSS 3的技术和应用环境已经日趋成熟，它们将书写Web前端领域的新历史，一场新的革命即将爆发。

HTML 5将成为新一代的Web技术标准，必定会改变整个Web应用领域的游戏规则（比如实现富Web应用的本地化，摆脱对Flash和Silverlight等浏览器插件的依赖），它在给新的Web应用带来无限可能性的同时，还能带来更快、更好、更炫的用户体验。CSS 3也将为Web开发带来革命性的影响，很多以前需要JavaScript和Ajax框架才能实现的复杂效果（如多背景、圆角、3D动画，等等），现在使用CSS 3就能简单地实现，极大地提高了程序的开发效率。

HTML 5和CSS 3已成为全球各大互联网巨头的必争之地，Microsoft、Google、Apple、Mozilla、Opera 等浏览器厂商对它们的支持犹如一场竞赛，似乎已经成为衡量它们的浏览器性能优劣的一个重要指标，Adobe和Facebook等公司也纷纷开始发力，似乎谁也不想在这一次变革中失去话语权。

作为前端工作者的你，你甘于在这场变革中落后于人吗？答案一定是否定的，那就请你翻开这本书吧！

.....

作为一位前端工作者，如果你想在这场即将到来的Web技术革命中立于不败之地，建议你从现在开始学习和使用HTML 5和CSS 3。如果你想更好、更快、更系统地学习HTML 5和CSS 3，强烈推荐你阅读本书。无论从内容的权威性和全面性，还是案例的实战性和可操作性，这本书都是你当下的最佳选择。

——jQuery中文社区（jquery.org.cn）

HTML 5有两个使命：第一，弥补上一代HTML的不足；第二，实现富Web应用的本地化，使浏览器逃离Flash和Silverlight等富客户端插件的羁绊。它必定成为新一代的Web技术标准。本书基于HTML 5技术的最新发展现状，详尽地介绍了HTML 5的所有新功能和新特性，不仅内容全面，而且实战性强，强烈推荐！

——HTML51（http://www.html51.com/）

CSS 2.1发布至今已有12年的历史，但它已经无法满足当前Web应用日益增长的高性能、高用户体验的需求。CSS 3极大地简化了CSS的编程模型，它不仅对已有的功能进行了扩展和延伸，而且更多的是对Web UI的设计理念的和方法进行了革新。在未来，CSS 3配合HTML 5标准，将掀起一场新的Web应用变革，甚至是整个互联网产业的变革。本书全面而系统、极具针对性地对CSS 3进行了深入浅出地讲解，完整地将CSS 3的所有新功能和新特性呈现给了我们，是学习CSS 3的不二之选。

——CSS 3中文开发者社区

Web前端开发者的内功修炼秘笈
4大社区鼎力推荐！

作者：曹刘阳 著　　ISBN：978-7-111-30595-8　　定价：49.00 元

如果你在思考下面这些问题，也许本书就是你想要的！

- 作为一名合格的Web前端开发工程师，究竟需要具备哪些技能和素质？为什么说如果要精通Web前端开发这一行，必须先精通十行？
- 在Web应用的实现代码中，有哪些技术因素会导致应用难以维护？
- 高质量的Web前端代码应该满足哪些条件？如何才能提高Web前端代码的可读性和可重用性？
- 在HTML代码中，为何要使用语义化标签？如何检查你使用的标签是否语义良好？语义化标签时应该注意哪些问题？
- 如何编写CSS代码和JavaScript代码可以避免团队合作时产生冲突？
- 如何组织CSS文件才能让它们更易于管理？如何让CSS模块化，从而提高代码的重用率？CSS的命名应该注意哪些问题？何谓优良的CSS编码风格？
- 如何在CSS编码中引入面向对象的编程思想？这样做有哪些好处？
- 原生JavaScript和JavaScript类库之间有何关系？如何编写自己的JavaScript类库？
- JavaScript有哪些常见的跨浏览器兼容问题？如何解决这些问题？
- 如何组织JavaScript才能让代码的结构更清晰有序，从而更易于维护？如何才能编写出弹性良好的JavaScript代码？编写过程中应该注意哪些问题？
- JavaScript的面向对象编程是如何实现的？如何用面向对象的方式重写原有的代码？
- 编写高质量的JavaScript代码有哪些实用的技巧？又有哪些常见的问题需要注意？
- 为了提高Web前端代码的可维护性，我们应该遵循哪些规范？

全面探讨Web前端设计的方法、原则、技巧和最佳实践
5大专业社区一致鼎力推荐！

作者：赖定清 林坚 著　　ISBN：978-7-111-31245-1　　定价：59.00 元

如果你在思考下面这些问题，也许本书就是你想要的！

- 作为一名合格的Web前端开发工程师，究竟需要具备哪些技能和素质？为什么说如果要精通Web前端开发这一行，必须先精通十行？
- 前端设计者如何才能正确地理解自己的用户？如何理解并实践以用户为中心的设计原则？
- 原型设计应该注意哪些问题？如何更好地利用工具快速地进行原型设计？
- 可用性设计的关键要素是什么？如何设计高可用性的页面元素（导航、表单、链接等）？
- “可用性”的首要原则是“别让我思考”，你的网站如何才能做到不让用户思考呢？
- 可用性测试的5项目标是什么？如何通过可用性测试发现问题现象背后的本质？
- 如何保持设计的一致性？一致性设计的三项原则是什么？
- 如何理解“样式就是设计”这句话？有哪些样式技术是前端开发者和设计者必须掌握的？样式究竟有哪些功能？
- 如何编写易于管理、维护和复用的JavaScript代码？JavaScript有哪些最佳实践？
- 如何理解HTML文件、CSS文件和JavaScript文件之间的关系？如何良好地组织这些文件从而让它们更易于管理、复用和维护？
- 如何平衡网站的色彩？如何让你的网站设计简洁而美观？页面排版的艺术你知多少？
- Web前端设计领域有哪些经典的设计思维？如何才能掌握这些设计思维的本质？
- 如何测试前端的性能？前端性能优化的基本原则是什么？如何进行页面内容的优化和服务器端的优化？如何利用SEO技术让你的网站更容易被发现？
- CSS 3与HTML 5将带来哪些全新的设计方式？
- Web 3.0真的来了吗？Web 3.0的先驱者们有哪些杰出的表现？Web前端开发与设计的未来会怎样？

专业成就人生
立体服务大众

www.hzbook.com

填写读者调查表　加入华章书友会
获赠精彩技术书　参与活动和抽奖

尊敬的读者：

感谢您选择华章图书。为了聆听您的意见，以便我们能够为您提供更优秀的图书产品，敬请您抽出宝贵的时间填写本表，并按底部的地址邮寄给我们（您也可通过www.hzbook.com填写本表）。您将加入我们的“华章书友会”，及时获得新书资讯，免费参加书友会活动。我们将定期选出若干名热心读者，免费赠送我们出版的图书。请一定填写书名书号并留全您的联系信息，以便我们联络您，谢谢！

书名：　　书号：7-111-(　　)

姓名：	性别：☐男　☐女	年龄：	职业：
通信地址：		E-mail：	
电话：	手机：	邮编：	

1. 您是如何获知本书的：

☐朋友推荐　☐书店　☐图书目录　☐杂志、报纸、网络等　☐其他

2. 您从哪里购买本书：

☐新华书店　☐计算机专业书店　☐网上书店　☐其他

3. 您对本书的评价是：

技术内容	☐很好	☐一般	☐较差	☐理由_____
文字质量	☐很好	☐一般	☐较差	☐理由_____
版式封面	☐很好	☐一般	☐较差	☐理由_____
印装质量	☐很好	☐一般	☐较差	☐理由_____
图书定价	☐太高	☐合适	☐较低	☐理由_____

4. 您希望我们的图书在哪些方面进行改进？

_____

_____

5. 您最希望我们出版哪方面的图书？如果有英文版请写出书名。

_____

_____

6. 您有没有写作或翻译技术图书的想法？

☐是，我的计划是_____　☐否

7. 您希望获取图书信息的形式：

☐邮件　☐信函　☐短信　☐其他_____

请寄：北京市西城区百万庄南街1号　机械工业出版社　华章公司　计算机图书策划部收

邮编：100037　电话：（010）88379512　传真：（010）68311602　E-mail: hzjsj@hzbook.com