清华社“视频大讲堂”大系
网络开发视频大讲堂

1DVD

HTML5 + CSS3 精通

NLIC2970904634

从入门到精通

配套视频讲解227节

中小典型实例225个 综合实战案例31个 学习参考手册11部 实用网页模版83类

李东博◎编著

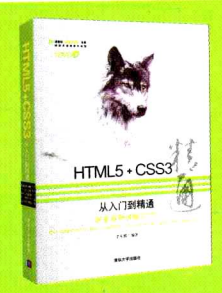
清华大学出版社

清华社“视频大讲堂”大系
网络开发视频大讲堂

执着于专业，精细于品质

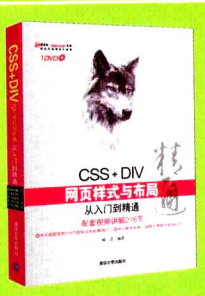
ISBN 978-7-302-30667-2

定价：79.80元



ISBN 978-7-302-30881-2

定价：69.80元



ISBN 978-7-302-30671-9

定价：69.80元

ISBN 978-7-302-30666-5

定价：79.80元

清华社“视频大讲堂”大系简介

“视频大讲堂”大系是清华社第六事业部重点打造的精品大系，该大系自2008年出版“软件开发视频大讲堂”以来，先后出版了多个子系列，本着“宁缺勿滥，件件精品”的原则，该大系取得了如下业绩：

- 4个品种荣获“全行业优秀畅销品种”
- 1个品种荣获2012年清华大学出版社“专业畅销书”一等奖
- 绝大多数品种在“全国计算机零售图书排行榜”同品种排行中名列前茅
- 截至目前该大系累计销售超过55万册
- 该大系已成为近年来清华社计算机专业基础类零售图书最畅销的品牌之一

清华大学出版社数字出版网站
WQBook 书文同泉
www.wqbook.com

ISBN 978-7-302-30881-2

定价：69.80元

（1DVD，含配套视频、参考手册、网页模板、教材源程序等）

清华社“视频大讲堂”大系

网络开发视频大讲堂

HTML5+CSS3 从入门到精通

李东博　编著

清华大学出版社

北　京

内 容 简 介

《HTML5+CSS3 从入门到精通》（清华社“视频大讲堂”大系）通过基础知识+中小实例+综合案例的方式，讲述了用 HTML5+ CSS3 设计构建网站的必备知识，相对于权威指南、高级程序设计、开发指南同类图书，本书是一本适合快速入手的自学教程。内容有：创建 HTML5 文档，实战 HTML5 表单，实战 HTML5 绘画，HTML5 音频与视频，Web 存储，离线应用，Workers 多线程处理，Geolocation 地理位置等技术。CSS3 部分主要介绍了 CSS3 概述，CSS 选择器，文本、字体与颜色，背景和边框，2D 变形，设计动画，网页布局、用户界面以及 CSS3 的其他新特性。

《HTML5+CSS3 从入门到精通》内容涵盖了 HTML5+CSS3 的所有重要特性，通过大量实际案例对 HTML5+CSS3 的重要特性进行了详细讲解，内容全面丰富，易于理解，能够帮助读者提升实际应用技能。

本书内容翔实、结构清晰、循序渐进，基础知识与案例实战紧密结合，既可作为 HTML5+CSS3 初学者的入门教材，也适合作为中高级用户对新技术作进一步学习的参考用书。本书显著特色有：

1. 同步视频讲解，让学习更为直观高效。227 节大型高清同步视频讲解，先看视频再学习效率更高。
2. 海量精彩实例，用实例学更轻松快捷。225 个精彩实例，模仿练习是最快捷的学习方式。
3. 精选实战案例，为高薪就业牵线搭桥。31 个实战案例展示可为以后就业积累经验。
4. 完整学习套餐，为读者提供贴心服务。参考手册 11 部、网页模版 83 类、素材源程序，让学习更加方便。
5. 讲解通俗翔实，看得懂学得会才是硬道理。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

HTML5+CSS3 从入门到精通/李东博编著. —北京：清华大学出版社，2013
（清华社“视频大讲堂”大系　网络开发视频大讲堂）

ISBN 978-7-302-30881-2

I. ①H…　II. ①李…　III. ①超文本标记语言-程序设计　②网页制作工具　IV. ①TP312　②TP393.092

中国版本图书馆 CIP 数据核字（2012）第 291406 号

责任编辑：赵洛育
封面设计：李志伟
版式设计：文森时代
责任校对：张兴旺
责任印制：王静怡

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：203mm×260mm　　**印　　张**：35　　**字　　数**：983 千字
（附 1DVD，含配套视频、参考手册、网页模板、素材源程序等）
版　　次：2013 年 6 月第 1 版　　**印　　次**：2013 年 6 月第 1 次印刷
印　　数：1～5000
定　　价：69.80 元

产品编号：044322-01

前　言

Preface

互联网技术日新月异。2011 年以前，HTML5 和 CSS3 看起来还遥不可及，如今很多公司都已经开始运用这些技术了，Chrome、Safari、Firefox 和 Opera 等主流浏览器已经开始逐步实现对它们的支持。从前端开发技术看，互联网发展经历了三个阶段：第一阶段是 Web 1.0 的以内容为主的网络，主流技术是 HTML 和 CSS；第二阶段是 Web 2.0 的 Ajax 应用，热门技术是 JavaScript/DOM/异步数据请求；第三阶段是即将迎来的 HTML5+CSS3 技术，这两者相辅相成，使互联网又进入了一个崭新的时代。

HTML5+CSS3 奠定了打造下一代 Web 应用的基础，这两项技术可以让网站更易开发、更易维护、更具用户友好性。HTML5 被设计为跨平台的技术，只需要一个所有主流操作系统上都有的免费现代浏览器。最新版本的 Apple Safari、Google Chrome、Mozilla Firefox、Opera 以及 Microsoft Internet Explorer 都支持 HTML5 的许多特性。在 iPhone、iPad 及 Android 移动设备上预装的浏览器也都对 HTML5 提供了极好的支持。

本书系统地讲解了 HTML5 和 CSS3 的基础理论和实际运用技术，通过大量实例对 HTML5 和 CSS3 进行深入浅出的分析，着重讲解如何用 HTML5+CSS3 进行 Web 应用和网页布局。全书注重实际操作，使读者在学习技术的同时，掌握 Web 开发和设计的精髓，提高综合应用的能力。

本书特色

☑ **系统的基础知识**

本书系统地讲解了 HTML5+CSS3 技术在网页设计中各种应用的知识，从为什么要用 HTML5 开始讲解，循序渐进，配合大量实例帮助读者奠定坚实的理论基础，做到知其所以然。

☑ **大量的案例实战**

本书设置大量应用实例，重点强调具体技术的灵活应用，并且全书结合了作者长期的网页设计制作和教学经验，使读者真正做到学以致用。

☑ **深入解剖 HTML5+CSS3 应用开发和布局**

本书用相当多的篇幅重点介绍了用 HTML5+CSS3 进行应用开发和布局的方法和技巧，配合经典的布局案例，帮助读者掌握 HTML5+CSS3 最核心的应用技术。

☑ **精选综合实例**

本书每章都会精选多个常见 Web 开发和设计的综合实例，帮助读者总结前面所学知识，综合应用各种技术、方法和技巧，提高读者综合应用的能力。

本书内容

本书分为两大部分，共 19 章，具体结构划分如下。

第一部分：HTML5 部分，包括第 1 章～第 10 章。这部分主要介绍了 HTML5 相关基础知识，包括 HTML5 概述、Web 开发历史、HTML5 文档结构异同与创建 HTML5 文档、设计 HTML5 表单、使

用 HTML5 绘画、HTML5 音频与视频、如何实现 Web 本地存储、如何实现 Web 离线应用、如何设计多线程应用、如何实现移动互联网中地图定位等技术。

第二部分：CSS3 部分，包括第 11 章～第 19 章。这部分主要讲解了 CSS3 的新特性和新用法，以实现在简单的代码中能够设计更加精彩的网页效果。主要内容包括 CSS3 概述，CSS 选择器，定义文本、字体与颜色，设计背景和边框，使用 2D 变形，设计动画，设计多列和流动网页布局，优化用户界面以及 CSS3 的其他新特性。

Note

本书读者

- ☑ 希望系统学习网页设计、网站制作的初学者、进阶者
- ☑ 从事网页设计制作和网站建设的专业人士
- ☑ 各大中专院校相关专业的老师、学生
- ☑ 相关培训机构的学员

本书约定

本书主要面向熟悉 HTML 和 CSS 的 Web 开发人员。初学者也可以从本书获益，读者还应该具备基本的 HTML、CSS、JavaScript 知识，我们会用这些知识来创建各种应用方案。

本书中上机练习的示例要用到 Firefox、Chrome 和 Opera 浏览器，因此，为了测试所有内容，读者需要安装上述所有类型的最新版本浏览器，因为各种浏览器的实现都稍有差异。

读者还需要安装 Internet Explorer 来测试自己的站点，以便确保示例的兼容性。如需针对不同版本的 Internet Explorer 测试示例，可以下载 IE Tester Windows 版，因为它可同时支持 IE6、IE7 和 IE8。对于非 Windows 用户，可以考虑使用 VirtualBox 或者 VMware 等虚拟机，或者使用 CrossBrowserTesting 和 MogoTest 等服务。

为了给读者提供更多的学习资源，同时弥补本书篇幅有限的遗憾，本书提供了很多参考链接，许多本书中无法详细介绍的问题都可以通过这些链接找到答案。因为这些链接地址会因时间而有所变动或调整，所以在此说明，这些链接地址仅供参考，无法保证这些地址都是长期有效的。

本书所列的插图可能会与读者实际环境中的操作界面有所差别，这可能是由于操作系统平台、浏览器版本等不同而引起的，在此特别说明，读者应该以实际情况为准。

为了帮助读者快速上手，在一般情况下，读者可以在程序和文档中自由使用本书中的示例代码。

关于我们

参与本书编写的人员包括咸建勋、奚晶、文菁、李静、钟世礼、李增辉、甘桂萍、刘燕、杨凡、李爱芝、余乐、孙宝良、余洪萍、谭贞军、孙爱荣、何子夜、赵美青、牛金鑫、孙玉静、左超红、蒋学军、邓才兵、袁江和李东博等。由于作者水平有限，书中疏漏和不足之处在所难免，欢迎读者朋友不吝赐教。广大读者如有好的建议、意见，或在学习本书时遇到疑难问题，可以联系我们，我们会尽快为您解答，服务邮箱为：design1993@163.com，liulm75@163.com。

编　者

目 录

Contents

第 1 章 Web 开发新时代.....1
1.1 HTML5 概述.....2
1.1.1 HTML5 新特性.....2
1.1.2 HTML5 组织.....3
1.1.3 HTML5 构成.....4
1.2 HTML5 设计原理.....4
1.2.1 HTML 开发历程.....5
1.2.2 HTML5 开发动力.....6
1.2.3 HTML5 设计理念.....7
1.3 编写第一个 HTML5 页面.....14
1.3.1 搭建上机练习环境.....14
1.3.2 检测浏览器是否支持.....14
1.3.3 使用 HTML5 编写简单的 Web 页面.....15
1.4 HTML5 页面的特征.....17
1.4.1 使用 HTML5 结构化元素.....17
1.4.2 使用 CSS 美化 HTML5 文档.....19

第 2 章 从 HTML、XHTML 到 HTML5.....21
视频讲解：2 小时
2.1 HTML 基础.....22
2.1.1 HTML 简介.....22
2.1.2 HTML 特性.....22
2.1.3 HTML 结构.....23
2.1.4 HTML 语法.....23
2.1.5 HTML 标签.....25
2.1.6 HTML 属性.....29
2.2 XHTML 基础.....32
2.2.1 XHTML 结构.....32
2.2.2 XHTML 语法.....33
2.2.3 XHTML 类型.....34
2.2.4 DTD 解析.....35
2.2.5 命名空间.....37
2.3 HTML5 基础.....38
2.3.1 HTML5 语法.....38
2.3.2 HTML5 元素.....40
2.3.3 HTML5 增加及废除的属性.....46
2.3.4 HTML5 全局属性.....48
2.3.5 HTML5 其他功能.....50

第 3 章 创建 HTML5 文档.....54
视频讲解：1 小时 13 分钟
3.1 认识 HTML5 文档结构.....55
3.2 HTML5 元素分类.....58
3.3 构建主体内容.....59
3.3.1 标识文章.....59
3.3.2 给内容分段.....61
3.3.3 设计导航信息.....64
3.3.4 设计辅助信息.....66
3.3.5 设计微格式.....67
3.3.6 添加发布日期.....68
3.4 添加语义模块.....69
3.4.1 添加标题块.....69
3.4.2 给标题分组.....70
3.4.3 添加脚注块.....70
3.4.4 添加联系信息.....71
3.5 综合实战：使用 HTML5 设计博客主页.....72
3.5.1 设计大纲.....73
3.5.2 设计样式.....76

第 4 章 实战 HTML5 表单.....79
视频讲解：1 小时 08 分钟
4.1 新增的 input 输入类型.....80
4.1.1 email 类型的应用.....80
4.1.2 url 类型的应用.....82

4.1.3 number 类型的应用 83
4.1.4 range 类型的应用 84
4.1.5 日期检出器类型的应用 85
4.1.6 search 类型的应用 90
4.1.7 tel 类型的应用 91
4.1.8 color 类型的应用 92
4.2 新增的 input 属性 93
4.2.1 新增的 autocomplete 属性 93
4.2.2 新增的 autofocus 属性 95
4.2.3 新增的 form 属性 96
4.2.4 新增的表单重写属性 97
4.2.5 新增的 height 与 width 属性 98
4.2.6 新增的 list 属性 98
4.2.7 新增的 min、max 和 step 属性 99
4.2.8 新增的 multiple 属性 100
4.2.9 新增的 pattern 属性 101
4.2.10 新增的 placeholder 属性 101
4.2.11 新增的 required 属性 102
4.3 新增的 form 元素 103
4.3.1 新增的 datalist 元素 103
4.3.2 新增的 keygen 元素 103
4.3.3 新增的 output 元素 104
4.4 新增的 form 属性 105
4.4.1 新增的 autocomplete 属性 105
4.4.2 新增的 novalidate 属性 105

第 5 章 实战 HTML5 画布 106

视频讲解：2 小时

5.1 认识 HTML5 canvas 元素 107
5.1.1 在页面中添加 canvas 元素 107
5.1.2 Canvas 如何绘制图形 108
5.1.3 认识 Canvas 坐标 109
5.1.4 何时不用 Canvas 109
5.1.5 如果浏览器不支持 Canvas 110
5.1.6 检测浏览器支持 110
5.2 绘制简单图形 111
5.2.1 绘制直线 111
5.2.2 绘制矩形 112
5.2.3 绘制圆形 113
5.2.4 绘制三角形 115
5.2.5 清空画布 116
5.3 绘制贝塞尔曲线 117
5.3.1 绘制二次方贝塞尔曲线 118
5.3.2 绘制三次方贝塞尔曲线 119
5.4 图形的变换 120
5.4.1 保存与恢复 Canvas 状态 120
5.4.2 移动坐标空间 121
5.4.3 旋转坐标空间 123
5.4.4 缩放图形 125
5.4.5 矩阵变换 126
5.5 图形的组合与裁切 129
5.5.1 图形的组合 129
5.5.2 裁切路径 132
5.6 更多的颜色和样式选项 133
5.6.1 应用不同的线型 133
5.6.2 绘制线性渐变 138
5.6.3 绘制径向渐变 139
5.6.4 绘制图案 140
5.6.5 设置图形的透明度 141
5.6.6 创建阴影 142
5.7 绘制文字 144
5.7.1 绘制填充文字 144
5.7.2 文字相关属性 145
5.7.3 绘制轮廓文字 145
5.7.4 测量文字宽度 146
5.8 操作与使用图像 147
5.8.1 向 Canvas 中引入图像 147
5.8.2 改变图像大小 149
5.8.3 创建图像切片 150

第 6 章 HTML5 音频与视频 152

视频讲解：50 分钟

6.1 HTML5 多媒体技术概述 153
6.1.1 关于编解码器 153
6.1.2 音频编解码器 153
6.1.3 视频编解码器 154
6.2 浏览器支持概述 156
6.2.1 用 JavaScript 检测音频格式支持情况 157
6.2.2 用 JavaScript 检测视频格式支持情况 158
6.3 在 HTML5 中播放音频 159

6.3.1 认识 audio 元素 159
6.3.2 播放音频 160
6.4 在 HTML5 中播放视频 161
6.4.1 认识 video 元素 161
6.4.2 播放视频 162
6.5 音频与视频相关属性、方法与事件 164
6.5.1 音频与视频相关属性 164
6.5.2 音频与视频相关方法 167
6.5.3 音频与视频相关事件 168
6.6 综合实战 169
6.6.1 用脚本控制音乐播放 169
6.6.2 查看视频帧画面 170

第 7 章 Web 存储 175
视频讲解：1 小时 20 分钟
7.1 认识 Web Storage 176
7.1.1 Cookie 存储机制的优缺点 176
7.1.2 为什么要用 Web Storage 176
7.1.3 Web Storage 的优缺点 177
7.1.4 浏览器支持概述 177
7.2 使用 Web Storage 178
7.2.1 检查浏览器的支持性 178
7.2.2 设置和获取数据 180
7.2.3 防止数据泄露 181
7.2.4 Web Storage 的其他用法 181
7.2.5 Web Storage 事件监测 182
7.2.6 实例 1：设计网页皮肤 183
7.2.7 实例 2：跟踪 localStorage 数据 184
7.2.8 实例 3：设计计数器 186
7.2.9 综合应用：Web 应用项目实时跟踪 187
7.3 Web SQL 数据库 192
7.3.1 Web SQL 数据库概述 192
7.3.2 使用 Web SQL 数据库 193
7.3.3 实例 1：创建简单的本地数据库 195
7.3.4 实例 2：批量存储本地数据 198
7.3.5 综合应用：Web Storage 和 Web SQL 混合开发 199

第 8 章 离线应用 207
视频讲解：23 分钟
8.1 HTML5 离线应用概述 208
8.1.1 为什么要学习 HTML5 离线应用 208
8.1.2 浏览器支持概述 209
8.2 HTML5 离线应用详解 210
8.2.1 解析 manifest 文件 210
8.2.2 搭建离线应用程序 213
8.2.3 检查浏览器是否支持 213
8.2.4 离线缓存更新实现 213
8.2.5 JavaScript 接口实现 214
8.2.6 离线存储事件监听 217
8.3 实战 1：缓存首页 218
8.4 实战 2：离线编辑内容 221
8.5 实战 3：离线跟踪 225

第 9 章 Workers 多线程处理 231
视频讲解：1 小时 01 分钟
9.1 认识 Web Workers 232
9.1.1 Web Workers 概述 232
9.1.2 浏览器支持概述 233
9.1.3 熟悉 Web Workers 成员 233
9.2 使用 Web Workers 234
9.2.1 检查浏览器支持性 234
9.2.2 创建 Web Workers 234
9.2.3 与 Web Workers 通信 235
9.2.4 使用 Web Workers 上机练习 237
9.3 案例实战 240
9.3.1 使用多线程实现后台运算 240
9.3.2 在后台过滤值 242
9.3.3 多任务并发处理 243
9.3.4 在多线程之间通信 246
9.3.5 使用线程技术计算 Fibonacci 数列 248
9.3.6 使用多线程绘图 249
9.4 综合应用：模拟退火算法 253
9.4.1 认识模拟退火算法 253
9.4.2 编写应用主页面 254
9.4.3 编写 worker.js 256

9.4.4 与 Web Workers 通信 257

第 10 章 Geolocation 地理位置 261

10.1 位置信息概述 262

10.1.1 为什么要学习 Geolocation....262

10.1.2 位置信息表示方式 262

10.1.3 位置信息来源 262

10.1.4 IP 定位 263

10.1.5 GPS 定位 263

10.1.6 Wi-Fi 定位 263

10.1.7 手机定位 264

10.1.8 自定义定位 264

10.2 使用 Geolocation API 264

10.2.1 检查浏览器支持性 264

10.2.2 获取当前地理位置 265

10.2.3 监视位置信息 267

10.2.4 停止获取位置信息 267

10.2.5 隐私保护 267

10.2.6 处理位置信息 267

10.2.7 使用 position 对象 268

10.3 案例实战 269

10.3.1 使用 Google 地图 269

10.3.2 跟踪行走速度 271

第 11 章 CSS3 概述 277

视频讲解：1 小时 01 分钟

11.1 回顾 CSS 278

11.1.1 CSS 发展简史 278

11.1.2 CSS 1.0 和 CSS 2.0 概述 278

11.1.3 CSS 与 DIV 标记之缘 285

11.1.4 CSS 编码规范 288

11.2 了解 CSS3 新增特性 289

11.2.1 属性选择器 289

11.2.2 RBGA 透明度 291

11.2.3 多栏布局 292

11.2.4 多背景图片 294

11.2.5 字符串溢出 295

11.2.6 块阴影与圆角阴影 296

11.2.7 圆角 297

11.2.8 边框图片 298

11.2.9 形变 299

11.3 CSS3 前景展望 301

11.3.1 CSS3 的应用范围 302

11.3.2 当前支持 CSS3 的浏览器 303

11.4 案例实战：设计漂亮的表单 305

第 12 章 CSS 选择器 310

视频讲解：34 分钟

12.1 属性选择器 311

12.1.1 认识属性选择器 311

12.1.2 案例实战 312

12.2 结构伪类选择器 314

12.2.1 认识结构伪类选择器 314

12.2.2 案例实战 315

12.3 UI 伪类选择器 321

12.3.1 认识常用 UI 伪类选择器 321

12.3.2 案例实战 322

12.4 其他选择器 324

第 13 章 文本、字体与颜色 330

视频讲解：49 分钟

13.1 设计文本阴影 331

13.1.1 定义 text-shadow 属性 331

13.1.2 应用阴影效果 333

13.1.3 综合实战：设计黑客网站首页 339

13.2 定义文本样式 341

13.2.1 文本样式简介 341

13.2.2 溢出文本 345

13.2.3 文本换行 347

13.3 设计颜色样式 353

13.3.1 使用 RGBA 颜色值 353

13.3.2 使用 HSL 颜色值 355

13.3.3 使用 HSLA 颜色值 359

13.3.4 定义 opacity 属性 360

13.3.5 定义 transparent 颜色值 363

第 14 章 背景和边框 365

视频讲解：1 小时 09 分钟

14.1 设计多色边框 366

14.1.1 用法详解 366

14.1.2　案例实战.....368
14.2　设计边框背景.....368
14.2.1　用法详解.....369
14.2.2　案例实战.....372
14.3　设计圆角.....375
14.3.1　用法详解.....376
14.3.2　案例实战：设计椭圆图形.....379
14.4　设计阴影.....380
14.4.1　用法详解.....380
14.4.2　案例实战：设计 Windows7 界面效果.....385
14.5　设计背景.....390
14.5.1　定义背景坐标.....390
14.5.2　定义背景裁剪区域.....392
14.5.3　定义背景图像大小.....395
14.5.4　定义背景图像循环方式.....396
14.5.5　定义多背景图像.....398

第 15 章　2D 变形.....400

视频讲解：31 分钟

15.1　认识 transform.....401
15.2　2D 变形.....402
15.2.1　旋转动画.....403
15.2.2　缩放动画.....404
15.2.3　移动动画.....406
15.2.4　倾斜动画.....408
15.2.5　变形动画.....410
15.2.6　案例实战：设计涂鸦墙.....412
15.3　自定义变形.....414
15.4　定义复杂变形.....416

第 16 章　设计动画.....420

视频讲解：1 小时 20 分钟

16.1　平滑过渡.....421
16.1.1　定义过渡属性.....421
16.1.2　定义过渡时间.....422
16.1.3　定义过渡延迟时间.....423
16.1.4　定义过渡效果.....424
16.1.5　案例实战：设计 Mac OS 导航器.....426
16.2　3D 动画.....428
16.2.1　定义动画名称.....429
16.2.2　定义动画时间.....429
16.2.3　定义动画播放方式.....429
16.2.4　定义动画延迟时间.....429
16.2.5　定义动画播放次数.....430
16.2.6　定义动画播放方向.....430
16.2.7　案例实战：设计图片翻转特效.....430
16.3　渐变效果.....431
16.3.1　设计 Webkit 渐变.....432
16.3.2　Webkit 案例实战.....437
16.3.3　设计 Gecko 渐变.....440
16.3.4　Gecko 案例实战.....446
16.3.5　设计 IE 渐变.....447
16.3.6　设计 W3C 渐变.....449
16.4　案例综合实战.....449
16.4.1　设计礼品盒.....450
16.4.2　设计折叠面板.....452
16.4.3　设计易拉罐.....454
16.4.4　设计光盘滑动动画.....457
16.4.5　设计下拉菜单.....461
16.4.6　设计精致按钮.....465

第 17 章　网页布局.....468

视频讲解：40 分钟

17.1　设计多列布局.....469
17.2　设置多列显示样式.....471
17.2.1　定义列宽.....471
17.2.2　定义列数.....472
17.2.3　定义列间距.....473
17.2.4　定义列边框样式.....475
17.2.5　定义跨列显示.....476
17.2.6　定义列高度.....478
17.2.7　定义打印列.....480
17.3　设计盒布局.....481
17.4　设置盒布局格式.....485
17.4.1　定义自适应宽度.....485
17.4.2　定义列显示顺序.....487
17.4.3　定义列排列方向.....489
17.4.4　定义模块大小自适应.....491

17.4.5 消除空白 493
17.4.6 定义对齐方式 497
17.5 综合实战：设计多列网页 498

第 18 章 用户界面 506

视频讲解：35 分钟

18.1 改变盒模型组成方式 507
18.2 调节元素尺寸 507
18.3 设计轮廓 509
18.3.1 定义轮廓线 509
18.3.2 定义轮廓线宽度 512
18.3.3 定义轮廓线样式 512
18.3.4 定义轮廓线颜色 513
18.3.5 定义轮廓线位移 513
18.4 设计导航 516
18.4.1 定义导航顺序 516
18.4.2 定义方向键控制顺序 519
18.5 添加显示内容 521
18.6 恢复默认样式 522
18.6.1 取消元素样式 522
18.6.2 慎用 initial 的情况 524

第 19 章 CSS3 其他新特性 527

视频讲解：40 分钟

19.1 溢出处理 528
19.2 自定义字体类型 530
19.2.1 使用@font-face 规则 530
19.2.2 开放字体 532
19.3 定义设备类型 532
19.3.1 认识 Media Queries 模块 532
19.3.2 认识@media 规则 533
19.3.3 使用@media 规则 535
19.3.4 在网站中应用@media 规则 537
19.4 添加语音功能 543
19.5 设计倒影 545

Web 开发新时代

从 2010 年开始，HTML5 和 CSS3 就一直是互联网技术中最受关注的两个话题。2010 年 MIX 10 大会上微软的工程师在介绍 IE9 时，从前端技术的角度把互联网的发展分为 3 个阶段：第一阶段是以 Web 1.0 为主的网络阶段，前端主流技术是 HTML 和 CSS；第二阶段是 Web 2.0 的 Ajax 应用阶段，热门技术是 JavaScript/DOM/异步数据请求；第三阶段是即将迎来的 HTML5+CSS3 阶段，这两者相辅相成，使互联网又进入了一个崭新的时代。本章将重点介绍 HTML5 语言基础，为将来系统地学习 HTML5+CSS3 编程奠定扎实的基础。

1.1 HTML5 概述

Note

2004 年成立的 Web 超文本应用技术工作组（WHATWG）创立了 HTML5 规范，同时开始专门针对 Web 应用开发新的功能。2006 年，W3C 介入 HTML5 的开发，并于 2008 年发布了 HTML5 的工作草案。2009 年，W3C 停止了对 XHTML2 的更新。2010 年 HTML5 开始用于解决实际问题。这时各大浏览器厂商开始对旗下产品进行升级以支持 HTML5 的新功能，因此，HTML5 规范也得到了持续性的完善。

1.1.1 HTML5 新特性

HTML5 是基于各种全新的理念进行设计的，这些设计理念体现了对 Web 应用的可能性和可行性的新认识，下面简单介绍 HTML5 语言的特征和优势，以便提高读者自学 HTML5 的动力，明确学习目标。

1. 兼容性

考虑到互联网上 HTML 文档已经存在二十多年了，因此支持所有现存 HTML 文档是非常重要的。HTML5 不是颠覆性的革新，它的核心理念就是要保持与过去技术的兼容和过渡。一旦浏览器不支持 HTML5 的某项功能，针对该功能的备选行为就会悄悄运行。

2. 合理性

HTML5 新增加的元素都是对现有网页和用户习惯进行跟踪、分析和概括而推出的。例如，Google 分析了上百万的页面，从中分析出了 DIV 标签的通用 ID 名称，并且发现其重复量很大，如很多开发人员使用 来标记页眉区域，为了解决实际问题，HTML5 就直接添加一个<header>标签。也就是说，HTML5 新增的很多元素、属性或者功能都是根据现实互联网中已经存在的各种应用进行技术精炼，而不是在实验室中理想化地虚构新功能。

3. 效率

HTML5 规范是基于用户优先准则编写的，其宗旨是用户即上帝，这意味着在遇到无法解决的冲突时，规范会把用户放到第一位，其次是页面作者，再次是实现者（或浏览器），接着是规范制定者（W3C/WHATWG），最后才考虑理论的纯粹性。因此，HTML5 的绝大部分是实用的，只是在有些情况下还不够完美。例如，下面的几种代码写法在 HTML5 中都能被识别。

```
id="prohtml5"
id=prohtml5
ID="prohtml5"
```

当然，上面几种写法比较混乱，不够严谨，但是从用户开发角度考虑，用户不在乎代码怎么写，根据个人习惯书写反而提高了代码编写效率。当然，我们并不提倡初学者一开始写代码就这样随意、不严谨。

4. 安全性

为保证安全性，HTML5 规范中引入了一种新的基于来源的安全模型，该模型不仅易用，而且各

种不同的 API 都可通用。这个安全模型可以不需要借助于任何所谓聪明、有创意却不安全的 hack 就能跨域进行安全对话。

5. 分离

在清晰分离表现与内容方面，HTML5 迈出了很大一步。HTML5 在所有可能的地方都努力进行了分离，包括 HTML 和 CSS。实际上，HTML5 规范已经不支持老版本 HTML 的大部分表现功能了。

Note

6. 简化

HTML5 要的就是简单，避免不必要的复杂性。为了尽可能简化，HTML5 做了以下改进：

- ☑ 以浏览器原生能力替代复杂的 JavaScript 代码。
- ☑ 简化的 DOCTYPE。
- ☑ 简化的字符集声明。
- ☑ 简单而强大的 HTML5 API。

7. 通用性

通用访问的原则可以分成 3 个概念。

- ☑ 可访问性：出于对残障用户的考虑，HTML5 与 WAI（Web 可访问性倡议）和 ARIA（可访问的富 Internet 应用）做到了紧密结合，WAI-ARIA 中以屏幕阅读器为基础的元素已经被添加到 HTML 中。
- ☑ 媒体中立：如果可能的话，HTML5 的功能在所有不同的设备和平台上应该都能正常运行。
- ☑ 支持所有语种：如新的<ruby>元素支持在东亚页面排版中会用到的 Ruby 注释。

8. 无插件

在传统 Web 应用中，很多功能只能通过插件或者复杂的 hack 来实现，但在 HTML5 中提供了对这些功能的原生支持。插件的方式存在很多问题：

- ☑ 插件安装可能失败。
- ☑ 插件可以被禁用或屏蔽（如 Flash 插件）。
- ☑ 插件自身会成为被攻击的对象。
- ☑ 插件不容易与 HTML 文档的其他部分集成，因为存在插件边界、剪裁和透明度问题。

以 HTML5 中的 canvas 元素为例，以前在 HTML4 的页面中较难画出对角线，而有了 canvas 元素就可以很轻易地实现了。基于 HTML5 的各类 API 的优秀设计，可以轻松地对它们进行组合应用。例如，从 video 元素中抓取的帧可以显示在 canvas 中，用户单击 canvas 即可播放这帧对应的视频文件。

1.1.2　HTML5 组织

HTML5 开发主要由下面 3 个组织负责和实施。

- ☑ WHATWG：由来自 Apple、Mozilla、Google 和 Opera 等浏览器厂商的人员组成，成立于 2004 年。WHATWG 开发 HTML 和 Web 应用 API，同时为各浏览器厂商以及其他有意向的组织提供开放式合作。
- ☑ W3C：W3C 下辖的 HTML 工作组，目前负责发布 HTML5 规范。
- ☑ IETF（因特网工程任务组）：这个任务组下辖 HTTP 等，是负责开发 Internet 协议的团队。HTML5 定义的一种新 API（WebSocket API）所依赖的 WebSocket 协议，正由 IETF 工作组负责开发。

1.1.3 HTML5 构成

Note

HTML5 主要包括下面这些功能。

- ☑ Canvas（2D 和 3D）
- ☑ Channel 消息传送
- ☑ Cross-document 消息传送
- ☑ Geolocation
- ☑ MathML
- ☑ Microdata
- ☑ Server-Sent Events
- ☑ Scalable Vector Graphics（SVG）
- ☑ WebSocket API 及协议
- ☑ Web Origin Concept
- ☑ Web Storage
- ☑ Web SQL database
- ☑ Web Workers
- ☑ XMLHttpRequest Level 2

HTML5 发展的速度非常快，因此不用担心浏览器的支持问题。读者可以访问 www.caniuse.com，该网站按照浏览器的版本提供了详尽的 HTML5 的功能支持情况。如果通过浏览器访问 www.html5test.com，该网站会直接显示用户浏览器对 HTML5 规范的支持情况。另外，还可以使用 Modernizr（JavaScript 库）进行特性检测，它提供了非常先进的 HTML5 和 CSS3 检测功能。建议读者使用 Modernizr 检测当前浏览器是否支持某些特性。

1.2 HTML5 设计原理

设计原理是 Web 发展背后的驱动力，也是通过 HTML5 反映出来的某种思维方式。软件就像所有技术一样，具有天然的独裁性。代码必然会反映作者的选择、偏见和期望。任何开放的标准，都应该追求以下几点：

- ☑ 简化最常见的任务，让不常见的任务不至于太麻烦。
- ☑ 只为 80%设计。
- ☑ 给内容创建者最大的权力。
- ☑ 默认设置智能化。

在制定设计原理时，很多人用了很多时间却抓不住重点，因为他们想取悦所有人。关键在于我们不是要取悦所有人，而是要明确哪些人是最重要的。

意大利经济学家帕累托提出“世界上 20%的人口拥有 80%的财富”。这个比例又暗合了自然界各个领域的幂律分布现象。总之，无论是编写软件，还是制造什么东西，都是一样的，即 20%的努力可以触及 80%的用例，最后 20%的用例则需要付出 80%甚至更多的努力。因此，有时候据此确定只为 80%设计是很合理的，因为我们知道为此只要付出 20%的努力即可。

1.2.1 HTML 开发历程

Note

HTML 最早是从 2.0 版开始的，实际上并没有 HTML 1.0 版官方规范。HTML Tags 文档可以算作 HTML 的第一个版本，但它却不是一个正式的版本。第一个正式版本 HTML 2.0 也不是出自 W3C 之手，而是由 IETF（Internet Engineering Task Force，因特网工程任务组）制定的，从第三个版本开始，W3C（World Wide Web Consortium，万维网联盟）开始接手并负责后续版本的制定工作。

20 世纪 90 年代，HTML 有过几次快速的发展。众所周知，那时构建网站是一项十分复杂的工程，浏览器大战曾令人头疼不已，市场竞争的结果就是各家浏览器里都塞满了各种专有的特性，都试图在专有特性上胜人一筹。当时的混乱程度不堪回首，HTML 还重不重要，或者它作为 Web 格式的前景如何，谁都说不清楚。

从 1997 年到 1999 年，HTML 的版本从 3.2 到 4.0，再到 4.01，经历了非常快的发展。问题是到了 4.01 的时候，W3C 的认识发生了倒退，W3C 并没有停止开发这门语言，只不过他们对 HTML 不再感兴趣了。在 HTML 4.01 之后，W3C 提出了 XHTML 1.0 的概念。虽然听起来完全不同，但 XHTML 1.0 与 HTML 4.01 其实是一样的。

这两个规范的内容是一样的，词汇表是一样的，所有的元素是一样，所有的属性也都是一样的，唯一不同的就是 XHTML 1.0 要求使用 XML 语法。也就是说，所有属性都必须使用小写字母，所有元素也必须使用小写字母，所有属性值都必须加引号，所有的标签都必须有结束标签，对 img 和 br 等孤标签，要使用自结束标签。

从规范本身的内容来看，本质是相同的，不同之处就是编码风格，因为浏览器读取符合 HTML 4.01、HTML 3.2 或者 XHTML 1.0 规范的网页都没有问题，对浏览器来说这些网页都是一样的，都会生成相同的 DOM 树，只不过用户更喜欢 XHTML 1.0，因为不少人认同它比较严格的编码风格。

到了 2000 年，Web 标准项目（Web Standards Project）的活动开展得如火如荼，开发人员对浏览器里包含的那些乱七八糟的专有特性已经忍无可忍了。当时 CSS 有了长足的发展，而且与 XHTML 1.0 的结合也很紧密，CSS+XHTML 1.0 可以算是最佳实践了。虽然 HTML 4.01 与 XHTML 1.0 没有本质上的区别，但是大部分开发人员接受了这种组合。专业的开发人员能做到元素全部小写，属性全部小写，属性值也全部加引号。由于专业人员起到了模范带头作用，越来越多的人也都开始支持这种语法。

XHTML 1.0 之后是 XHTML 1.1，只是小数点后面的数字变成了 1，而且从词汇表的角度看，规范本身没有什么新内容，元素、属性也都相同，唯一的变化就是必须把文档标记为 XML 文档。而在使用 XHTML 1.0 的时候，还可以把文档标记为 HTML。

但是，这样做就带来很多问题：首先，把文档标记为 XML 后，IE 浏览器不能处理。当然，IE9 及其以上版本是可以处理的。作为全球领先的浏览器，IE 无法处理接收到的 XML 文档类型的文档，而规范又要求以 XML 文档类型来发送文档，这对于广大用户来说，是一件很痛苦的事情。

所以说 XHTML 1.1 有点脱离现实，而用户不想把文档以 XML 格式发送给那些能够理解 XML 的浏览器，则是因为 XML 的错误处理模型。XML 的语法，无论是属性小写，元素小写，还是始终要给属性值加引号，这些都没有问题，但 XML 的错误处理模型却是这样的：如果解析器遇到错误，停止解析。如果把 XHTML 1.1 标记为 XML 文档类型，假设用 Firefox 打开这个文档，而文档中有一个符号没有正确编码，就算整个页面中只有这一处错误，浏览器也会死掉，用户将看不到任何网页内容。根据 XML 规范，这样处理是正确的，对 Firefox 而言，遇到错误就停止解析，并且不呈现其他任何内容是严格按照 XML 规范处理的。因为它不是 HTML，HTML 根本没有错误处理模型，但根据 XML 规范，这样做没错。这就是为什么人们不会把文档标记为 XML 的另一个原因。

Note

接下来，新的版本是 XHTML2，但是这个版本并没有完成。从理论的角度来说，XHTML2 是一个非常好的规范。如果所有人都同意使用的话，也一定是一个非常好的格式。只不过，它还不够实际。

首先，XHTML2 仍然使用 XML 错误处理模型，用户必须保证以 XML 文档类型发送文档；其次，XHTML2 中有意不再向后兼容已有的 HTML 的各个版本，甚至曾经讨论过废除 img 元素，这对每天都在做 Web 开发的人来说确实有点难以接受，理论上分析，使用 object 元素可能会更好。

因此，无论 XHTML2 在理论上是多么完美的一种格式，却从未有机会付诸实践。之所以难以将其付诸实践，就是因为开发人员永远不会支持它，它不向后兼容。同样，浏览器厂商也不会支持它，必须要保证浏览器向后兼容。

为什么 XHTML 1.1 没有像 XML 那样得到真正广泛地应用？为什么 XHTML2 从未落到实处？因为它违反了一条设计原理，这条设计原理就是著名的伯斯塔尔法则：发送时要保守，接收时要开放。

接收的时候要开放，而这也正是 Web 得以构建的基础。开发浏览器的人必须敞开胸怀，接收所有发送给浏览器的东西，因为它们过去一直都在接收那些不够标准的东西，Web 上的很多文档都不规范，但那正是 Web 发展的动力。从某种角度讲，Web 走的正是一条混沌发展之路，虽然混沌，却非常美丽诱人。在 Web 上，格式不规范的文档随处可见，如果所有人都能够写出精准的 XML，所有文档的格式都十分正确，那当然好，可是那不现实。

作为专业人士，在发送文档的时候应该保守一些，尽量采用最佳实践，尽量确保文档格式良好，但从浏览器的角度说，它们必须以开放的姿态去接收任何文档。

XHTML 1.1 和 XHTML2 都使用 XML 错误处理模型，但这个错误处理模型太苛刻了，它不符合“接收时开放”这个法则，遇到一个错误就停止解析怎么能叫开放呢？我们只能说它与伯斯塔尔法则是对立的。

1.2.2 HTML5 开发动力

在 20 世纪末期，W3C 琢磨着改良 HTML 语言，“HTML 也许还可以更长寿一点，只要把我们放在 XHTML 上的时间和精力拿出一部分来，就可以提升一下 HTML 中的表单，可以让 HTML 更接近编程语言，就可以让它更上一层楼。”

于是，在 2004 年 W3C 成员内部的一次研讨会上，Opera 公司的代表伊恩·希克森（Ian Hickson）提出了一个扩展和改进 HTML 的建议。他建议新任务组可以跟 XHTML2 并行，但是在已有 HTML 的基础上开展工作，目标是对 HTML 进行扩展。但是 W3C 投票表示反对，因为 HTML 已经死了，XHTML2 才是未来的方向。然后，Opera、Apple 等浏览器厂商以及其他一些成员脱离了 W3C，他们成立了 WHATWG （Web Hypertext Applications Technology Working Group，Web 超文本应用技术工作组），这就为 HTML5 将来的命运埋下了伏笔。

WHATWG 决定完全脱离 W3C，在 HTML 的基础上开展工作，向其中添加一些新东西。这个工作组的成员里有浏览器厂商，因此他们可以保证实现各种新奇、实用的点子。结果，大家不断提出一些好点子，并且逐一做到了浏览器中。

WHATWG 的工作效率很高，不久就初见成效。在此期间，W3C 的 XHTML2 没有什么实质性的进展。在 2006 年，蒂姆·伯纳斯·李（Tim Berners-Lee）写了一篇博客反思 HTML 的发展历史，“你们知道吗？我们错了。我们错在企图一夜之间就让 Web 跨入 XML 时代，我们的想法太不切实际了，是的，也许我们应该重新组建 HTML 工作组了。”

W3C 在 2007 年组建了 HTML5 工作组。这个工作组面临的第一个问题，毫无疑问就是“我们是从头开始做起呢，还是在 2004 年成立的那个叫 WHATWG 的工作组既有成果的基础上开始工作呢？”

Note

答案是显而易见的，他们当然希望从已经取得的成果着手，以之为基础展开工作。于是他们又投了一次票，同意在 WHATWG 工作成果的基础上继续开展工作。

第二个问题就是如何理顺两个工作组之间的关系。W3C 这个工作组的编辑应该由谁担任？是不是还让 WHATWG 的编辑，也就是现在 Google 的伊恩 • 希克森来兼任？于是他们又投了一次票，赞成让伊恩 • 希克森担任 W3C HTML5 规范的编辑，同时兼任 WHATWG 的编辑，更有助于新工作组开展工作。

这就是他们投票的结果，也就是我们今天看到的局面：一种格式，两个版本。WHATWG 的网站上有这个规范，而 W3C 的站点上同样也有一份。

如果不了解内情，你很可能会产生这样的疑问："哪个版本才是真正的规范？"当然，这两个版本内容是一样的，基本上相同，但这两个版本将来还会分道扬镳。现在已经有分道扬镳的迹象了：W3C 最终要制定一个具体的规范，这个规范会成为一个工作草案，定格在某个历史时刻，而 WHATWG 还在不断地迭代，即使是目前的 HTML5 也不能完全涵盖 WHATWG 正在从事的工作。最准确的理解就是 WHATWG 正在开发一项简单的 HTML 或 Web 技术，因为这才是他们工作的核心目标。然而，同时存在两个这样的工作组，这两个工作组同时开发一个基本相同的规范，这无论如何也容易让人产生误解，误解就可能造成麻烦。

其实这两个工作组背后各自有各自的流程，因为它们的理念完全不同。在 WHATWG，可以说是一种独裁的工作机制。伊恩 • 希克森是编辑，他会听取各方意见，在所有成员各抒己见，充分陈述自己的观点之后，他批准自己认为正确的意见。而 W3C 则截然相反，可以说是一种民主的工作机制。所有成员都可以发表意见，而且每个人都有投票表决的权利。这个流程的关键在于投票表决。从表面上看，WHATWG 的工作机制让人难以接受，W3C 的工作机制听起来让人很舒服，至少体现了人人平等的精神。但在实践中，WHATWG 的工作机制运行得非常好，这主要归功于伊恩 • 希克森。他在听取各方意见时，始终可以做到丝毫不带个人感情色彩。

从原理上讲，W3C 的工作机制很公平，而实际上却非常容易在某些流程或环节上卡壳，造成工作停滞不前，一件事情要达成决议往往需要花费很长时间。那到底哪种工作机制最好呢？最好的工作机制是将二者结合起来。而事实也是两个规范制定主体在共同制定一份相同的规范，这倒是非常有利于两种工作机制相互取长补短。

两个工作组之所以能够同心同德，主要原因是 HTML5 的设计思想。因为他们从一开始就确定了设计 HTML5 所要坚持的原则。结果，我们不仅看到了 HTML5 语言规范，也就是 W3C 站点上公布的那份文档，还在 W3C 站点上看到了另一份文档，也就是 HTML5 设计原理。

1.2.3 HTML5 设计理念

HTML5 是一个里程碑式的规范，它为下一代 Web 发展指明了方向，下面我们就来探析 HTML5 语言的设计理念。这个设计理念相当于共同行动纲领，无论 W3C 与 WHATWG 之间有多大的分歧，他们都将遵循这个设计理念。

☑ 避免不必要的复杂性

例如，使用 HTML 4.01 规范时，如果定义 doctype，需要输入很长的字符串：

```
<!DOCTYPE html PUBLIC "-//W3C/DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

很少有人能够记住这行代码，本来它要告诉浏览器的是"这个文档是 XHTML 1.0 的文档"。那么在 HTML5 中，省掉不必要的复杂性，doctype 就简化成了：

```
<!DOCTYPE html>
```

第一次看到这个 doctype 的时候，或许很难想到，这个 doctype 要告诉浏览器什么呢？这个文档是 HTML 吗？难道这是有史以来唯一一个 HTML 版本吗？HTML 今后永远不会再有新版本了吗？

Note

实际上这个 doctype 并没有 HTML 版本区分的意思。我们先搞清楚为什么文档一开头就要写 doctype。这不是写给浏览器看的，而是写给验证器看的。也就是说，之所以要在文档一开头写那行 XHTML 1.0 的 doctype，是为了让验证器按照该 doctype 来验证我的文档。

浏览器反倒无所谓了。假设写的是 HTML 3.2 文档，文档开头写的是 HTML 3.2 的 doctype，而在文档中某个地方使用了 HTML 4.01 中才出现的一个元素，浏览器会怎么处理这种情况？它会因为这个元素出现在比 doctype 声明的 HTML 版本更晚的规范中，就不解释呈现该元素吗？不会，它照样会解释呈现该元素，别忘了伯斯塔尔法则，浏览器在接收的时候必须要开放。因此，它不会检查任何格式类型，而验证器会，验证器才关心格式类型。这才是存在 doctype 的真正原因。

而按照 HTML5 的另一个设计原理，它必须向前向后兼容，兼容未来的 HTML 版本，不管是 HTML6、HTML7，还是其他版本，都要与当前的 HTML5 兼容。因此，把一个版本号放在 doctype 里面没有多大的意义，对验证器也一样。

但是有一种情况下，使用 doctype 会影响到浏览器，这也是为了达到某种特殊的目的才使用 doctype。当初微软在引入 CSS 的时候，走在了标准的前头，他们率先在浏览器中支持 CSS，也推出了自己的盒模型。后来标准发布了，但标准中使用了不一样的盒模型。微软想支持标准，但也想向后兼容自己过去推出的编码方式。于是，他们想出了一个非常巧妙的主意，那就是利用有效的 doctype 来触发标准模式，而不是兼容模型（quiks mode）。这个想法非常巧妙。我们今天也都是在这样做，在向文档中加入 doctype 时，就相当于声明了使用标准模式，但这并不是发明 doctype 的本意。这只是为了达到特殊的目的在利用 doctype。

HTML5 规范的本质是不追求理论上的完美。HTML5 所体现的不是给用户一个简短好记的 doctype，好记的 doctype 也无法适应现有的浏览器。因此，doctype 不仅从理论上看简短好记，而且在实践中仍然可以触发标准模式。

还有一个例子，同样可以说明规范是如何省略不必要的复杂性，避免不必要的复杂性的。如果前面的文档使用的是 HTML 4.01，假设要指定文档的字符编码，理想的方式是通过服务器在头部信息中发送字符编码，不过也可以在文档这个级别上指定：

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

如果想指定文档使用 UTF-8 编码，只能添加上面一行代码。在 HTML 4.01 中需要这样做。要是在 XHTML 1.0 指定同样的编码，就得在一个开始的 XML 标签中声明 meta 元素。

```
<?xml version="1.0" encoding="UTF-8" ?>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

在 HTML5 中，只要输入下面代码即可。

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

同样，这样写也是有效的。它不仅适用于最新版本的浏览器，仍在使用的以往版本的浏览器中也同样有效，因为在把这些 meta 元素输入浏览器时，浏览器都会正确解释。

关于省略不必要的复杂性的例子还有很多，但关键是既能避免不必要的复杂性，又不会妨碍在现有浏览器中使用。在 HTML5 中，如果使用 link 元素链接到一个样式表，先定义 rel="stylesheet"，然后再定义 type="text/css"，这样就重复了。对浏览器而言，不需要同时看到这两个属性，只要看到 rel="stylesheet"就够了，因为它可以猜出来要链接的是一个 CSS 样式表，所以就不用再指定 type 属

性了。

同样，如果使用 script 元素，定义 type="text/javascript"，浏览器也可以识别。对 Web 开发而言，谁还使用其他的脚本语言吗？愿意的话，可以添加一个 type 属性，也可以什么都不写，浏览器自然会假设在使用 JavaScript。

Note

☑　支持已有的内容

支持已有的内容。这一点非常重要，因为很多人都认为 HTML5 很新，它应该代表着未来发展的方向，应该把 Web 推向一个新的发展阶段。当然我们都会考虑让 Web 的未来发展得更好，但工作组则必须考虑过去。别忘了 WHATWG 这个工作组中有很多人代表的是浏览器厂商，他们肯定是要考虑支持已有内容的。只要想构建一款浏览器，就必须记住一个原则：必须支持已有的内容。

【示例 1】本示例展示了编写同样内容的 4 种不同方式。上面是一个 img 元素，下面是带一个属性的段落元素，4 种写法唯一的不同点就是语法。把其中任何一段代码交给浏览器，浏览器都会生成相同的 DOM 树，没有任何问题。从浏览器的角度看，这 4 种写法没有区别，因此在 HTML5 中可以随意使用下列语法。

```
<img src="foo" alt="bar" />
<p class="foo">Hello world</p>
<img src="foo" alt="bar">
<p class="foo">Hello world
<IMG SRC="foo" ALT="bar">
<P CLASS="foo">Hello world</P>
<img src=foo alt=bar>
<p class=foo>Hello world</p>
```

HTML5 同时允许这些写法，对于那些写了很多年 XHTML 1.0 代码，已经非常适应严格的语法的开发人员来说，是难以适应的，但站在浏览器的角度上，这些写法实际上都是一样的，确实没有什么问题。

HTML5 必须支持已经存在的内容，而已有的内容就是这个样子的。根据伯斯塔尔法则，浏览器没有别的选择。

有人可能会说“这样不行。我觉得语言本身应该提供一种开关，让作者能够表明自己想做什么。”比如说，想使用某种特定的语法，像 XHTML，而不是使用其他语法。这些人的想法可以理解，但没必要在语言里设置开关。因为我们讨论的只是编码风格或者写作风格，跟哪种语法正确与否无关。专业人士可以使用 lint 工具（一种软件质量保证工具，或者说是一种更加严格的编译器。它不仅可以像普通编译器那样检查出一般的语法错误，还可以检查出那些虽然完全合乎语法要求，但很可能是潜在的、不易发现的错误）。

对其他技术我们也在使用 lint 工具，比如对 JavaScript 使用 lint 工具。JavaScript 同样也是比较混乱、不严谨的典型，但它非常强大，恰恰是因为它混乱、不严谨，而且有很多不同的编码方式。在 JavaScript 中可以在每条语句末尾加上分号，但不是必需的，因为 JavaScript 会自动插入分号。

正因为如此，才有了像 JSlint 这样的工具。在道格拉斯 • 克劳克福德（Douglas Crockford）的网站 jslint.org 上写着“JSlint 可能会伤害你的感情”，但这确实是个非常棒的工具，它可以把 JavaScript 代码变得完美无瑕。如果通过 JSlint 运行 JavaScript，它会告诉你你的 JavaScript 代码有效，但写法不妥。特别是对团队，对于要使用统一的编码风格的团队，JSlint 是非常方便的工具。

个人认为，不仅对团队来说，就算是自己写代码，也要坚持一种语法风格。从浏览器解析的角度讲，不存在哪种语法比另一种更好的问题，但作为专业人士，必须能够自信地讲“这就是我的编码风格”。然而，语言里并不需要内置这种开关，用户可以使用 lint 工具来统一编码风格。现在就来说说

lint 工具。大家可以登录 htmllint.com，在其中运行 HTML5 文档，它会帮用户检查属性值是否加了引号，元素是否小写，还可以通过选中复选框来设置其他检查项。

☑ 解决现实的问题

HTML5 的另一个设计原理是解决现实的问题。显而易见的是，解决各种问题的格式和规范已经比比皆是了，因此，这个原理其实是要解决理论问题，而非解决现实问题。这条设计原理是要从理论上承认人们普遍存在的问题，消除敏感问题。但是，那些格式和规范要解决的都是理论问题，而非现实问题。这条设计原理才是真正要解决今天人们所面临的现实问题、令人头疼的问题。

【示例 2】假设使用 HTML 4 或 XHTML 1，页面中已经有了一块内容，想给整块内容添加一个链接，怎么办？问题是这块内容里包含一个标题，一个段落，也许还有一张图片。如果想使它们全部都可以生成链接，必须使用 3 个链接元素。于是，得先把光标放在标题（比如说 h2 元素）中，写一个链接标签，然后再选中所有要包含到链接里的文本。接着，把光标放在段落里，写一个链接标签，然后把段落中的文本放在链接里。

```
<h2><a href="#">标题文本</a></h2>
<p><a href="#">段落文本</a></p>
```

在 HTML5 中，只要简单地把所有内容都写在一个链接元素中就可以了。

```
<a href="#">
    <h2>标题文本</h2>
    <p>段落文本</p>
</a>
```

链接包含的都是块级元素，但现在可以用一个元素包含它们。这样，碰到类似的情形，就必须给几个块级元素加上相同的链接，因为这样做解决了一个现实的问题：不必因此重新写代码来支持这种写法。这种写法其实早就存在于浏览器中了，因为早就有人这样写了，当然以前这样写是不合乎规范的。所以说 HTML5 解决了现实的问题，允许这样写了。

☑ 求真务实

求真务实对于 HTML 的含义是：在解决那些令人头痛的问题之前，先看看人们为应对这些问题都想出了哪些办法。集中精力去理解这些解决方案才是当务之急。

HTML5 中新的语义元素就是遵循求真务实原理的反映。新增的元素不算多，谈不上无限的扩展性，但却不失为一件好事，尽管数量屈指可数，但意义却非同一般。这些新元素涉及头部（header）、脚部（footer）、分区（section）、文章（article）等，相信大家都不会觉得陌生。即便不使用 HTML5，也应该熟悉这些名称，它们都是曾经使用过的类名，如 class="header"/"head"/"heading"，或 class="footer"/"foot"。当然，也可能是 ID，如 id="header"或 id="footer"。

```
<body>
    <div id="header">...</div>
    <div id="navigation">...</div>
    <div id="main">...</div>
    <div id="sidebar">...</div>
    <div id="footer">...</div>
</body>
```

在 HTML5 中，这些元素都可以换掉。说起新增的语义元素，它们价值的一方面就是可以用 HTML5 新增的元素把这些 div 都替换掉。

```
<body>
```

```
    <header>...</header>
    <nav>...</nav>
    <div id="main">...</div>
    <aside>...</aside>
    <footer>...</footer>
</body>
```

Note

在文档级别上使用这些元素没有问题，但是，假如新增这些元素仅仅是为了取代原来的 div，那就没有必要了。

虽然在这个文档中用这些新元素来替换 ID，但在个人看来，将它们作为类的替代品更有价值，因为这些元素在一个页面中不止可以使用一次，而是可以使用多次。没错，也可以为文档添加一个头部（header），再添加一个脚部（footer），但文档中的每个分区（section）照样也都可以有一个头部和一个脚部，而每个分区里还可以嵌套另一个分区，被嵌套的分区仍然可以有自己的头部和脚部。

这 4 个新元素是 section、article、aside 和 nav，之所以说它们强大，原因在于它们代表了一种新的内容模型，一种 HTML 中前所未有的内容模型——给内容分区。迄今为止，我们一直都在用 div 来组织页面中的内容，但与其他类似的元素一样，div 本身并没有语义；但 section、article、aside 和 nav 实际上是在明确地告诉用户，这一块就像文档中的另一个文档一样。位于这些元素中的任何内容，都可以拥有自己的概要、标题和脚部。

其中最为通用的 section，可以说是与内容最相关的一个，而 article 则是一种特殊的 section。aside 和 nav 也是特殊的 section。

当然，现在仍然可以使用 div 和类来描述页面中不同的部分：

```
<div class="item">
    <h2>...</h2>
    <div class="meta">...</div>
    <div class="content">
    </div>
    <div class="links">...</div>
</div>
```

其中包含的可能是有关内容作者的元数据，而下面会给出一些链接。在 HTML5 中，完全可以说这块内容就是一个文档，通过对内容分区，使用 section、article 或 aside，这一块完全是可以独立存在的。因此，可以使用 header 和 footer。

```
<section class="item">
    <header><h1>...</h1></header>
    <footer class="meta">...</footer>
    <div class="content">
    </div>
    <nav class="links">...</nav>
</section>
```

注意： 即便是 footer，也不一定非要出现在下面。header、footer、aside 和 nav 元素，最重要的是它们的语义，与位置没有关系。如果阅读 HTML5 规范，就会发现这些元素只跟内容有关。因此，放在 footer 中的内容也可以是署名、文章作者之类的信息，它只是所使用的一个元素，并没有说这个元素必须放在文档或者分区的下面。

在 HTML5 中，只要建立一个新的内容块，不管用 section、article、aside、nav，还是用别的元素，

Note

都可以在其中使用 H1，而不必担心这个块里的标题在整个页面中应该排在什么级别；使用 H2、H3 也都没有问题，因为现在可以把每个内容分区想象成一个独立的、能够从页面中拿出来的部分。此时，根据上下文的不同，这个独立部分中的 H1 在整个页面中也许会扮演 H2 或 H3 的角色，这取决于它在文档中出现的位置。面对这个突如其来的变化，也许有的人暂时不能理解，但这才是 HTML5 中这些新语义标记的真正价值所在。换句话说，现在有了独立的元素，这些元素中的标题级别可以重新定义。

文档中可能会包含一个分区，这个分区中可能会嵌套另一个分区，或者一篇文章，然后文章再嵌套分区，分区再嵌套文章、嵌套分区，文章再嵌套文章。而且每个分区和文章都可以拥有自己的 H1~H6。从这个意义上讲，H 元素真可谓层层嵌套。但是，在编写内容或者内容管理系统的时候，它们又都是完全独立的内容块。

这个点子并不是 HTML5 工作组想出来的，也不是 W3C 最近才提出的，实际上蒂姆·伯纳斯·李在 1991 年的一封邮件中就提到这种想法。他在邮件中解释了对 HTML 的理解，他说“我认为 H1、H2 这样单调地排下去不好，我希望它成为一种可以嵌套的元素，或者说一个通用的 H 元素，我们可以在其中嵌套不同的层次。”但后来，我们没有看到通用的 H 元素，而是一直在使用 H1 和 H2，那是因为我们一直在支持已有的内容。今天，这个理想终于实现了。

☑ 平稳退化

渐进增强的另一面就是平稳退化。HTML5 遵循这条原理的例子，就是使用 type 属性增强表单。下面列出了可以为 type 属性指定的新值，有 number、search 和 range 等。

```
input type="number"
input type="search"
input type="range"
input type="email"
input type="date"
input type="url"
```

最关键的问题在于浏览器在看到这些新 type 值时会如何处理。现有的浏览器是无法理解这些新 type 值的，但它们会将 type 的值解释为 text。

无论你写的是 input type="foo"，还是 input type="bar"，现有的任何浏览器都会把它理解为"text"。因而，从现在开始就可以使用这些新值，而且也可以放心，那些不理解它们的浏览器会把新值看成 type="text"，而这正是一个浏览器实践平稳退化原理的好例子。

例如，现在输入了 type="number"。假设需要一个输入数值的文本框，那么可以把这个 input 的 type 属性设置为 number，然后理解它的浏览器就会呈现一个可爱的小控件，如带小箭头图标的微调控件之类的，而在不理解它的浏览器中，则会看到一个文本框，那么为什么不能说输入 type="number"就会得到一个带小箭头图标的微调控件呢？

当然，还可以设置最小值和最大值属性，它们同样可以平稳退化。

再看 input type="search"。你也可以考虑一下这种输入框，因为这种输入框在 Safari 中会被呈现为一个系统级的搜索控件，右边还有一个单击即可清除搜索关键词的 X。而在其他浏览器中，得到的则是一个文本框，就像所写的是 input type="text"一样。那为什么还不使用 input type="search"呢？它不会有什么副作用。

HTML5 还为输入元素增加了新的属性，如 placeholder（占位符），它表示在文本框中预先放一些文本。占位符就是文本框可以接受的示例内容，一般颜色是灰色的。只要一单击文本框，它就消失了。如果把已经输入的内容全部删除，然后单击文本框外部，它又会出现。使用 JavaScript 编写一些代码

当然也可以实现这个功能，但 HTML5 只用一个 placeholder 属性就解决了问题。

当然，对于不支持这个属性的浏览器，还是可以使用 JavaScript 来实现占位符功能的。通过 JavaScript 来测试浏览器是否支持该属性也非常简单。如果不支持，可以让 JavaScript 来模拟这个功能。

下面就来看看这个新的 video 元素，它设计得既简单又实用。一个开始的 video 元素，加一个结束的 video 元素，中间可以放后备内容。注意，是后备内容，而不是保证可访问性的内容。下面就是针对不支持 video 元素的浏览器写的代码。

Note

```
<video src="movie.mp4">
    <!-- 后备内容 -->
</video>
```

那么，在后备内容里面放些什么东西呢？可以放 Flash 影片，这样，HTML5 的视频与 Flash 的视频就可以协同起来了，用户不需要作出选择。

```
<video src="movie.mp4">
    <object data="movie.swf">
        <!-- 后备内容 -->
    </object>
</video>
```

因为这里使用了 H264，部分浏览器支持这种视频格式，但有的浏览器不支持，对于不支持的情况，仅需要把后备内容放在那儿而已，后备内容可以包含多种视频格式。如果愿意的话，可以使用 source 元素而非 src 属性来指定不同的视频格式。

```
<video>
    <source src="movie.mp4">
    <source src="movie.ogv">
    <object data="movie.swf">
        <a href="movie.mp4">download</a>
    </object>
</video>
```

上面的代码中包含了 4 个不同的层次。

☑ 如果浏览器支持 video 元素，也支持 H264，则直接用第一个视频。

☑ 如果浏览器支持 video 元素，也支持 Ogg，那么用第二个视频。

☑ 如果浏览器不支持 video 元素，就要试试 Flash 影片了。

☑ 如果浏览器不支持 video 元素，也不支持 Flash，代码中还给出了下载链接。

总之，无论是 HTML5 和 Flash 一个也不能少。如果只使用 video 元素提供视频，难免会遇到问题。而如果只提供 Flash 影片，性质是一样的。所以还是应该两者兼顾。

为什么要兼顾这两种技术呢？假设你需要面向某些不支持 Flash 的手持设备提供视频，当然希望手持设备的用户能够看到视频了。

至于为什么要使用不同的格式？当然，面对竞争性的视频格式和不同的编码方式，只向浏览器发送用一种方式编码的视频是行不通的，而这也正是 Flash 在视频和音频领域如此成功的原因。只要把 Flash 影片发送给浏览器，安装了插件的浏览器都能正常播放。

☑ 最终用户优先

这个设计原理本质上是一种解决冲突的机制。换句话说，当面临一个要解决的问题时，如果 W3C 和 WHATWG 给出了不同的解决方案，一旦遇到冲突，最终用户优先，然后是作者，其次是实现者，

再次是标准制定者，最后才是理论上的完满。

理论上的完满，大致是指尽可能创建出最完美的格式。标准制定者指的是 WHATWG、W3C 等。实现者指的是浏览器厂商。作者就是开发人员。用户是第一位的。

Note

根据最终用户优先的原理，开发人员在链条中的位置高于实现者，假如我们发现了规范中的某些地方有问题，又不支持实现这个特性，那么就等于把相应的特性给否定了，规范里就得删除，因为我们的声音具有更高的权重。本质上是我们拥有了更大的发言权，我认为开发人员就应该拥有更多的发言权。

这应该是最重要的一条设计原理了，因为它承认了用户的权利，无论是设计一种格式，还是设计软件，这条原理保证了用户的发言权。而这条原理也正道出了事物运行的本质，用户的权利大于作者，作者的权利大于实现者，实现者的权利大于标准制定者。然而，反观其他规范，如 XHTML2，就会发现完全相反的做法——把追求理论的完满放在第一位，而把用户放在了链条的最底端，需要忍受严格错误处理带来的各种麻烦。这是一种完全不同的思维方式。

因此，无论是构建像 HTML5 这样的格式，还是构建一个网站，亦或一个内容管理系统，明确设计原理都至关重要。

1.3 编写第一个 HTML5 页面

尽管各主流厂商的最新版浏览器都对 HTML5 提供了很好的支持，但毕竟 HTML5 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 HTML5 页面之前，必须先搭建支持 HTML5 的浏览器环境，并检查浏览器是否支持 HTML5 标记。

1.3.1 搭建上机练习环境

目前，Microsoft 的 IE 系列（IE9+）浏览器，以及 Mozlilla 的 Firefox 与 Google 的 Chrome 浏览器等都可以很好地支持 HTML5。

本书所有的应用示例，主要执行的浏览器为 Chrome，其对应的版本号是 15.0.874.120。如需运行本书中的示例，则要安装最新的 Chrome 浏览器，以浏览相应的示例页面效果。

1.3.2 检测浏览器是否支持

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（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 HTML5 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。在 HTML 页面中插入一段 HTML5 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 HTML5 的画布标记！”的提示。

【示例】在 Dreamweaver 中新建一个 HTML 页面，保存为 index.html，代码如下所示。

Note

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title></title>
<style type="text/css">
#myCanvas {
    background:red;
    width:200px;
    height:100px;
}
</style>
</head>
<body>
<canvas id="myCanvas">该浏览器不支持 HTML5 的画布标记！ </canvas>
</body>
</html>
```

将页面文件 index.html 在 IE8 浏览器中执行。由于 IE 8 浏览器暂不支持 HTML5 的画布标记，因此将显示如图 1.1 所示的页面效果。但是在 Chrome 浏览器中执行，由于该浏览器支持 HTML5 的画布标记，因此将显示如图 1.2 所示的页面效果。

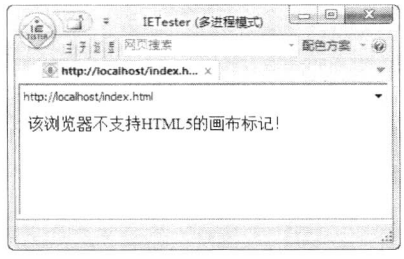


图 1.1　IE8 不支持 HTML5 的画布标记

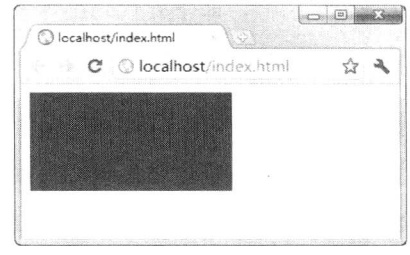


图 1.2　Chrome 支持 HTML5 的画布标记

虽然是同样一个页面，但由于不同的浏览器对 HTML5 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 HTML5 新标记时，有必要先检测浏览器是否支持该标记。

接下来将利用 HTML5 结构编写一个简单的程序，从中体会 HTML5 的代码简洁所在。

1.3.3　使用 HTML5 编写简单的 Web 页面

HTML5 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 HTML5 新结构来编写一个简单的页面，实现在页面中输出“Hello,World”的字样。

【示例】在 Dreamweaver 中新建一个 HTML 页面保存为 index.html，加入代码如下所示。

```
<!DOCTYPE html>
<META charset="utf-8">
<TITLE>第一个 HTML5 页面</TITLE>
<P>Hello,World</P>
```

该页面在 Chrome 浏览器中预览效果如图 1.3 所示。

图 1.3　第一个 HTML5 页面效果

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 HTML5 语法的简洁，同时，HTML5 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。

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。接下来需要说明文档的字符编码，否则浏览器将不能正确解析，导致安全隐患，为此加入如下一行代码：

```
<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，HTML5 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，可以省略主体标记，直接编写需要显示的内容，代码如下：

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

虽然在编写代码时省略了<html>、<head>和<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>、<head>与<body>标记后的源文件

考虑到代码的可维护性，在编写代码时，应该尽量增加这些在 HTML5 中可选的元素，从而最大限度地实现页面代码的简洁与完整。

1.4 HTML5 页面的特征

Note

上节简单介绍了一个 HTML5 页面的创建过程，而实际上，HTML5 页面的特征远不止这些，下面通过一个较为完整的页面来介绍 HTML5 的页面特征。

1.4.1 使用 HTML5 结构化元素

通过研究 Web 页面发现，如果使用一些带有语义性的标记，可以加快浏览器解释页面中元素的速度，如早期的<samp>、<var>元素；HTML5 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，因为这些标记能真正体现开发者真实意图所在。下面通过示例说明 HTML5 是如何使用这些全新的 HTML5 特征来结构化元素的。

【示例】本示例将页面分成上、中、下 3 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。演示效果如图 1.5 所示。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
#header, #siderLeft, #siderRight,#footer{
    border:solid 1px #666;
    padding:10px;
    margin:6px;
}
#header { width:500px }
#siderLeft {
    float:left;
    width:60px;
    height:100px
}
#siderRight {
    float:left;
    width:406px;
    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>
```

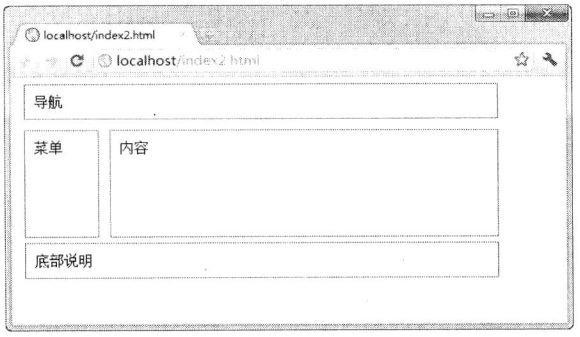


图 1.5　简单的网页布局

尽管上述代码不存在任何错误，还可以在 HTML5 环境中很好地运行，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，这会造成浏览器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 HTML5 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 HTML5 支持的页面代码，如下所示。演示效果如图 1.6 所示。

```
<!DOCTYPE html>
<html>
<head>
<meta charset=utf-8>
<title></title>
<style type="text/css">
header, nav, article,footer{
      border:solid 1px #666;
      padding:10px;
      margin:6px;
}
header { width:500px }
nav {
      float:left;
      width:60px;
      height:100px
}
article {
      float:left;
      width:406px;
      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>
```

图 1.6 简单的网页布局

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。从上述两段代码来看，使用 HTML5 新增元素创建的页面代码更加简单和高效。

可以看出，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 HTML5 中的新增元素<header>可以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>元素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，极大地提高了开发者的工作效率。

此外，有些新增的 HTML5 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 HTML5 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。

1.4.2 使用 CSS 美化 HTML5 文档

在支持 HTML5 新元素的浏览器中，样式化各新增元素变得十分简单，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 会默认元素的 display 属

性值为 inline，因此，为了正确地显示设置的页面效果，需要将元素的 display 属性设置为 block。下面通过一个简单示例来说明这一点。

【示例】在页面中设置相关样式，显示一段文章的内容。该页面在 Chrome 浏览器下预览效果如图 1.7 所示。

Note

```
<!DOCTYPE html >
<head>
<meta charset="utf-8" />
<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>
</head>
<body>
<article>
    <header>
        <h1><a>谷歌推出多项新搜索功能 避谈 Google+</a></h1>
        <p class="p-date">日期：2012-08-11</p>
        <p>网易科技讯 8 月 11 日消息，据国外媒体报道，谷歌在主打社交网络服务 Google+一年后终于重归“老本行”，推出一系列和社交网络没有关联的搜索引擎新功能。</p>
    </header>
</article>
</body>
```

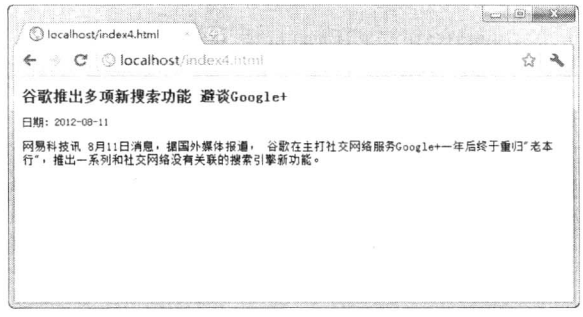


图 1.7　使用 CSS 美化 HTML5 页面

由于有些浏览器并不支持 HTML5 中的新增元素，如 IE8 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 HTML5 元素应用样式，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 HTML5 的情况下才执行这段脚本。

从 HTML、XHTML 到 HTML5

（视频讲解：2 小时）

1993 年，HTML 首次以草案的形式发布，20 世纪 90 年代是 HTML 发展速度最快的时期，直到 1999 年的 4.01 版。在这个过程中，W3C 主要负责 HTML 规范的制定。HTML 4.01 发布之后，业界普遍认为 HTML 已经到了穷途末路，对 Web 标准的焦点也开始转移到了 XML 和 XHTML 上，HTML 被放在了次要位置。HTML5 以 HTML 为基础，对 HTML 进行了大量的修改。本章将从总体上介绍 HTML5 与 HTML 前面版本的不同，以及 HTML5 与 HTML4 之间比较大的区别。

2.1 HTML 基础

Note

HTML 是目前在网络上应用最为广泛的语言，也是构成网页文档的主要语言。HTML 文档是由 HTML 标签组成的描述性文本，HTML 标签可以标识文字、图形、动画、声音、表格和链接等。

2.1.1 HTML 简介

1982 年，美国人蒂姆·伯纳斯·李为了方便世界各地的物理学家能够进行合作研究以及信息共享，创造了 HTML 语言。1990 年他又发明了世界上第一个浏览器。1991 年 3 月，他把这个发明介绍给了一起在 CERN 公司工作的朋友，当时网页浏览器被 CERN 公司在世界各地的成员用来浏览 CERN 庞大的电话薄。1993 年，NCSA 推出了 Mosaic 浏览器并迅速爆红，成为世界上第一个广泛应用的浏览器，推动着互联网迅猛发展。在随后的 5 年里，Netscape 和 Microsoft 两个软件巨头掀起了一场互联网浏览器大战。这场战争最后以 Microsoft 的 Internet Explorer 完胜告终，但它极大地推动了互联网的发展，把网络带到了千千万万普通用户面前。从 1993 年互联网工程工作小组（IETF）工作草案发布，到 1999 年 W3C 发布 HTML 4.01 标准，HTML 共经历过 5 个版本。如今的 HTML 不仅成为 Web 上最主要的文档格式，而且在个人及商业应用中都发挥着它的作用。

HTML 是 Hypertext Markup Language 的缩写，中文翻译为超文本标识语言。使用 HTML 标签编写的文档称为 HTML 文档，目前最新版本是 HTML 5.0，使用最广泛的是 HTML 4.01 版本。

早期版本的 HTML 语言不适合构建标准化网页，因为它把结构和表现混淆在一起，例如，HTML 把不同类型的元素，如描述性元素 color、i 等和结构性元素 div、table 等，以及元素属性放在一起，为以后的维护和管理埋下隐患。

XHTML 是 The Extensible HyperText Markup Language 的缩写，中文翻译为可扩展标识语言，实际上它是 HTML 语言的升级版本，目前遵循的是 W3C 于 2000 年 1 月推荐的 XHTML 1.0 标准（参考 http://www.w1.org/TR/xhtml1）。XHTML 和 HTML 在语法和标签使用方面差别不大。熟悉 HTML 语言，再稍加熟悉标准结构和规范，也就熟悉了 XHTML 语言。XHTML 具有如下特点：

- ☑ 用户可以扩展元素，从而扩展功能，但在目前 1.0 版本下，用户只能够使用固定的预定义元素，这些元素基本上与 HTML 4.01 版本元素相同，但删除了部分属性描述性的元素。
- ☑ 能够与 HTML 很好地沟通，可以兼容当前不同的网页浏览器，实现 XHTML 页面的正确浏览。

2.1.2 HTML 特性

HTML 作为一种网页内容标识语言，易学易懂，熟悉使用该语言可以制作功能强大、美观大方的网页。HTML 语言的主要作用说明如下。

- ☑ 使用 HTMl 语言标识文本。例如，定义标题文本、段落文本、列表文本和预定义文本等。
- ☑ 使用 HTML 语言建立超链接，通过超链接可以访问互联网上的所有信息，当使用鼠标单击超链接时，会自动跳转到链接页面。
- ☑ 使用 HTML 语言创建列表，把信息有序组织在一起，以方便浏览。
- ☑ 使用 HTMl 语言在网页中显示图像、声音、视频、动画等多媒体信息，把网页设计得更富冲击力。

- ☑ 使用 HTML 语言可以制作表格，以便显示大量数据。
- ☑ 使用 HTML 语言制作表单，允许在网页内输入文本信息，执行其他用户操作，方便信息互动。

Note

2.1.3 HTML 结构

HTML 文档一般都应包含两部分：头部区域和主体区域。HTML 文档基本结构由 3 个标签负责组织：<html>、<head>和<body>。其中<html>标签标识 HTML 文档，<head>标签标识头部区域，而<body>标签标识主体区域。一个完整的 HTML 文档基本结构如下所示。

```
<html> <!--语法开始-->
    <head>
        <!--头部信息，如<title>标签定义的网页标题-->
    </head>
    <body>
        <!--主体信息，包含网页显示的内容-->
    </body>
</html> <!--语法结束-->
```

可以看到，每个标签都是成对组成的，第一个标签（如<html>）表示标识的开始位置，而第二个标签（如</html>）表示标识的结束位置。<html>标签中包含<head>和<body>标签，而<head>和<body>标签则是并列排列的。

如果把上面的字符代码放置在文本文件中，然后另存为“test.html”，就可以在浏览器中浏览了。当然，由于这个简单的 HTML 文档还没有包含任何可显示信息，所以在浏览器中是看不到任何内容的。

2.1.4 HTML 语法

编写 HTML 文档时，必须遵循 HTML 语法规范。HTML 文档实际上就是一个文本文件，它由标签和信息混合组成，当然这些标签和信息必须遵循一定的组合规则，否则浏览器是无法解析的。

HTML 语言的规范条文不多，相信读者也很容易理解。从逻辑上分析，这些标签包含的内容就表示一类对象，也可以称为网页元素。从形式上分析，这些网页元素通过标签进行分隔，然后表达一定的语义。很多时候，我们把网页标签和网页元素混为一谈，而实际上，网页文档就是由元素和标签组成的容器。

- ☑ 所有标签都包含在“<”和“>”起止标识符中，构成一个标签如<style>、<head>、<body>和 等。
- ☑ 在 HTML 文档中，绝大多数元素都有起始标签和结束标签，在起始标签和结束标签之间包含的是元素主体，例如，<body>和</body>中间包含的就是网页内容主体。
- ☑ 起始标签包含元素的名称以及可选属性，也就是说元素的名称和属性都必须包含在起始标签中。结束标签以反斜杠开始，然后附加上元素名称。例如：

```
<tag>元素主体</tag>
```

- ☑ 元素的属性包含属性名称和属性值两部分，中间通过等号进行连接，多个属性之间通过空格进行分隔。属性与元素名称之间也通过空格进行分隔。例如：

```
<tag a1="v1" a2="v2" a3="v3" … an="vn">元素主体</tag>
```

☑ 少数元素的属性也可能不包含属性值，仅包含一个属性名称。例如：

```
<tag a1 a2 a3 … an>元素主体</tag>
```

Note

☑ 一般属性值应该包含在引号内，虽然不加引号浏览器也能够解析，但是读者应该养成良好的习惯。

☑ 属性是可选的，元素包含多少个属性也是不确定的，这主要根据不同元素而定。不同的元素会包含不同的属性。HTML 也为所有元素定义了公共属性，如 title、id、class 和 style 等。

虽然大部分标签都是成对出现，但是也有少数标签不是成对的，这些孤立的标签被称为空标签。空标签仅包含起始标签，没有结束标签。例如：

```
<tag>
```

同样，空标签也可以包含很多属性，用来标识特殊效果或者功能，例如：

```
<tag a1="v1" a2="v2" … an="vn">
```

☑ 标签可以相互嵌套，形成文档结构。嵌套必须匹配，不能交错嵌套，例如， 就是不合法的。合法的嵌套应该是包含或被包含的关系，例如， 或 。

☑ HTML 文档所有信息必须包含在<html>标签中，所有文档的元信息应包含在<head>子标签中，而 HTML 传递信息和网页显示内容应包含在<body>子标签中。

对于 HTML 文档来说，除了必须符合基本语法规范外，还必须保证文档结构信息的完整性。完整文档结构如下所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w1.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w1.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>文档标题</title>
</head>
<body></body>
</html>
```

HTML 文档应主要包括如下内容：

☑ 必须在首行定义文档的类型，过渡型文档可省略。

☑ <html>标签应该设置文档名字空间，过渡型文档可省略。

☑ 必须定义文档的字符编码，一般使用<meta>标签在头部定义，常用字符编码包括中文简体（gb2312）、中文繁体（big5）和通用字符编码（utf-8）。

☑ 应该设置文档的标题，可以使用<title>标签在头部定义。

HTML 文档扩展名为.htm 或.html，保存时必须正确使用扩展名，否则浏览器无法正确地解析。如果要在 HTML 文档中增加注释性文本，则可以添加在“<!--”和“-->”标识符之间，例如：

```
<!--单行注释-->
```

或

```
<!----------------
多行注释
---------------->
```

2.1.5　HTML 标签

HTML 定义的标签很多，下面对常用标签进行说明，随着读者学习不断深入，相信会完全掌握 HTML 所有标签的用法和使用技巧。

Note

1. 文档结构标签

此类标签主要用来标识文档的基本结构，主要包括以下几种。

☑ <html>...</html>：标识 HTML 文档的起始和终止。

☑ <head>...</head>：标识 HTML 文档的头部区域。

☑ <body>...</body>：标识 HTML 文档的主体区域。

【示例 1】设计最简单的网页文档。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>无标题文档</title>
</head>
<body> 网页正文写在这里……
</body>
</html>
```

2. 文本格式标签

这些标签主要用来标识文本区块，并附带一定的显示格式，主要标签说明如下。

☑ <title>...</title>：标识网页标题。

☑ <hi>...</hi>：标识标题文本，其中 i 表示 1、2、3、4、5、6，分别表示一级、二级、三级等标题。

☑ ... ：标识段落文本。

☑ <pre>...</pre>：标识预定义文本。

☑ <blockquote>...</blockquote>：标识引用文本。

【示例 2】分别使用<h1>和 标签标识网页标题和段落文本。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<h1>文本格式标签</h1>
<p>&lt;p&gt;标签标识段落文本</p>
</body>
</html>
```

3. 字符格式标签

字符格式标签主要用来标识部分文本字符的语义，很多字符格式标签可以呈现一定的显示效果。例如，加粗显示、斜体显示或者下划线显示等。主要标签说明如下。

☑ ...：标识强调文本，以加粗效果显示。

Note

☑ ...：标识引用文本，以斜体效果显示。

☑ <blink>...</blink>：标识闪烁文本，以闪烁效果显示（IE 浏览器不支持该标签）。

☑ <big>...</big>：标识放大文本，以放大效果显示。

☑ <small>...</small>：标识缩小文本，以缩小效果显示。

☑ ^{...}：标识上标文本，以上标效果显示。

☑ _{...}：标识下标文本，以下标效果显示。

☑ <cite>...</cite>：标识引用文本，以引用效果显示。

【示例 3】分别使用各种字符格式标签显示一个数学方程式的解法。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<p>例如，针对下面这个一元二次方程：</p>
<p><i>x</i><sup>2</sup>-<b>5</b><i>x</i>+<b>4</b>=0</p>
<p>我们使用<big><b>分解因式法</b></big>来演示解题思路如下：</p>
<p><small>由：</small>(<i>x</i>-1)(<i>x</i>-4)=0</p>
<p><small>得：</small><br /><i>x</i><sub>1</sub>=1<br />
    <i>x</i><sub>2</sub>=4</p>
</body>
</html>
```

4. 列表标签

在 HTML 文档中，列表结构可以分为两种类型：有序列表和无序列表。无序列表使用项目符号来标识列表，而有序列表则使用编号来标识列表的项目顺序。具体使用标签说明如下。

☑ ...：标识无序列表。

☑ ...：标识有序列表。

☑ ...：标识列表项目。

【示例 4】使用无序列表分别显示一元二次方程求解的 4 种方法。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<h1>解一元二次方程</h1>
<p>一元二次方程求解有 4 种方法：</p>
<ul>
    <li>直接开平方法 </li>
    <li>配方法 </li>
    <li>公式法 </li>
    <li>分解因式法</li>
</ul>
</body>
</html>
```

另外，还可以定义定义列表。定义列表是一种特殊的结构，它包括词条和解释两块内容。包含的标签说明如下。

☑ <dl>...</dl>：标识定义列表。

☑ <dt>...</dt>：标识词条。

☑ <dd>...</dd>：标识解释。

Note

【示例 5】使用定义列表显示两个成语的解释。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<h1>成语词条列表</h1>
<dl>
    <dt>知无不言，言无不尽</dt>
    <dd>知道的就说，要说就毫无保留。</dd>
    <dt>智者千虑，必有一失</dt>
    <dd>不管多聪明的人，在很多次的考虑中，也一定会出现个别错误。</dd>
</dl>
</body>
</html>
```

5. 链接标签

链接标签可以实现把多个网页联系在一起。主要结构如下。

☑ <a>...：标识超链接。

【示例 6】使用<a>标签定义一个超链接，单击该超链接可以跳转到百度首页。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<a href="http://www.baidu.com/">去百度搜索</a>
</body>
</html>
```

<a>标签还可以定义锚点。锚点是一类特殊的超链接，它可以定位到网页中某个具体的位置。例如，在下面示例中单击超链接文本，就可以跳转到网页的底部。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<a href="#btm">跳转到底部</a>
<div id="box" style="height:2000px; border:solid 1px red;">撑开浏览器滚动条</div>
<span id="btm">底部锚点位置</span>
```

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

6. 多媒体标签

多媒体标签主要用于引入外部多媒体文件并进行显示。多媒体标签主要包括以下几种。

☑ <img />：嵌入图像。

☑ <embed>...</embed>：嵌入多媒体。

☑ <object>...</object>：嵌入多媒体。

7. 表格标签

表格标签用来组织和管理数据。主要包括下面几种标签。

☑ <table>...</table>：定义表格结构。

☑ <caption>...</caption>：定义表格标题。

☑ <th>...</th>：定义表头。

☑ <tr>...</tr>：定义表格行。

☑ <td...</td>：定义表格单元格。

【示例 7】使用表格结构显示 5 行 3 列的数据集。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<table summary="ASCII 是英文 American Standard Code for Information Interchange 的缩写。ASCII 编码是目前计算机最通用的编码标准。因为计算机只能接受数字信息，ASCII 编码将字符转换为数字来表示，以便计算机能够接受和处理。">
    <caption>ASCII 字符集（节选）</caption>
    <tr>
        <th>十进制</th>
        <th>十六进制</th>
        <th>字符</th>
    </tr>
    <tr>
        <td>9</td>
        <td>9</td>
        <td>TAB（制表符）</td>
    </tr>
    <tr>
        <td>10</td>
        <td>A</td>
        <td>换行</td>
    </tr>
    <tr>
        <td>13</td>
        <td>D</td>
        <td>回车</td>
    </tr>
    <tr>
```

```
            <td>32</td>
            <td>20</td>
            <td>空格</td>
        </tr>
</table>
</body>
</html>
```

Note

8. 表单标签

表单标签主要用来制作交互式表单。主要包括下面几种标签。

☑ <form>...</form>：定义表单结构。

☑ <input />：定义文本域、按钮和复选。

☑ <textarea>...</textarea>：定义多行文本框。

☑ <select>...</select>：定义下拉列表。

☑ <option>...</option>：定义下拉列表中的选择项目。

【示例 8】分别定义单行文本框、多行文本框、复选框、单选按钮、下拉菜单和提交按钮的表单。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>示例代码</title>
</head>
<body>
<form id="form1" name="form1" method="post" action="">
    <p>单行文本域：<input type="text" name="textfield" id="textfield" /></p>
    <p>密码域：<input type="password" name="passwordfield" id="passwordfield" /></p>
    <p>多行文本域：<textarea name="textareafield" id="textareafield"> </textarea></p>
    <p>复选框：复选框 1<input name="checkbox1" type="checkbox" value="" />
              复选框 2<input name="checkbox2" type="checkbox" value="" />
    </p>
    <p>单选按钮：
        <input name="radio1" type="radio" value="" />按钮 1
        <input name="radio2" type="radio" value="" />按钮 2
    </p>
    <p>下拉菜单：
        <select name="selectlist">
            <option value="1">选项 1</option>
            <option value="2">选项 2</option>
            <option value="3">选项 3</option>
        </select>
    </p>
    <p><input type="submit" name="button" id="button" value="提交" /></p>
</form>
</body>
</html>
```

2.1.6　HTML 属性

HTML 元素包含的属性众多，这里无法列出所有元素的全部属性，当然也没有个必要，仅就公共

属性进行分析。公共属性大致可分为基本属性、语言属性、键盘属性、内容属性和延伸属性等类型。

1. 基本属性

Note

基本属性主要包括以下 3 个，这 3 个基本属性大部分元素都拥有。

class	定义类规则或样式规则
id	定义元素的唯一标识
style	定义元素的样式声明

下面这些元素不拥有基本属性。

html、head	文档和头部基本结构
title	网页标题
base	网页基准信息
meta	网页元信息
param	元素参数信息
script、style	网页的脚本和样式

这些元素一般位于文档头部区域，用来标识网页元信息。

2. 语言属性

语言属性主要用来定义元素的语言类型，包括两个属性。

lang	定义元素的语言代码或编码
dir	定义文本的方向，包括 ltr 和 rtl 取值，分别表示从左向右和从右向左

下面这些元素不拥有语言语义属性。

frameset、frame、iframe	网页框架结构
br	换行标识
hr	结构装饰线
base	网页基准信息
param	元素参数信息
script	网页的脚本

【示例 1】分别为网页代码定义简体中文的语言，字符对齐方式为从左到右。第二行代码为 body 定义了美式英语。

```
<html xmlns="http://www.w3.org/1999/xhtml" dir="ltr" xml:lang="zh-CN">
<body id="myid" lang="en-us">
```

3. 键盘属性

键盘属性定义元素的键盘访问方法，包括两个属性。

accesskey	定义访问某元素的键盘快捷键
tabindex	定义元素的 Tab 键索引编号

使用 accesskey 属性可以使用快捷键（Alt+字母）访问指定 URL，但是浏览器不能很好地支持，在 IE 中仅激活超链接，需要配合 Enter 键确定，而在 Firefox 中没有反应。

【示例 2】在导航菜单中设置快捷键。

```
<a href="http://www.html5.cn/" accesskey="a">按住 Alt 键，再按 A 键可以链接到样吧首页</a>
```

tabindex 属性用来定义元素的 Tab 键访问顺序，可以使用 Tab 键遍历页面中的所有链接和表单元素。遍历时会按照 tabindex 的大小决定顺序，当遍历到某个链接时，按 Enter 键即可打开链接页面。例如：

Note

```
<a href="#" tabindex="1">Tab 1</a>
<a href="#" tabindex="3">Tab 3</a>
<a href="#" tabindex="2">Tab 2</a>
```

4. 内容属性

内容属性定义元素包含内容的附加信息，这些信息对于元素来说具有重要补充作用，避免元素本身包含信息不全而被误解，包括 5 个属性。

alt	定义元素的替换文本
title	定义元素的提示文本
longdesc	定义元素包含内容的大段描述信息
cite	定义元素包含内容的引用信息
datetime	定义元素包含内容的日期和时间

alt 和 title 是两个常用的属性，分别定义元素的替换文本和提示文本，但是很多设计师习惯于混用这两个属性，没有刻意去区分它们的语义性。实际上，除了 IE 浏览器，其他标准浏览器都不会支持它们的混用，但是由于 IE 浏览器的纵容，才导致了很多设计师误以为 alt 属性就是设置提示文本的。

```
<a href="URL" title="提示文本">超链接</a>
<img src="URL" alt="替换文本" title="提示文本" />
```

替换文本（Alternate Text）并不是用来做提示（Tool Tip）的，或者更加确切地说，它并不是为图像提供额外说明信息的。相反，title 属性才负责为元素提供额外说明信息。

当图像无法显示时，必须准备替换的文本来替换无法显示的图像，这对于图像和图像热点是必须的，因此 alt 属性只能用在 img、area 和 input 元素中（包括 applet 元素）。对于 input 元素，alt 属性用来替换提交按钮的图片。

```
<input type="image" src="URL" alt="替换文本" />.
```

为什么要设置替换文本呢？这主要是因为浏览器被禁止显示、不支持或无法下载图像时，通过替换文本为那些不能看到图像的浏览者提供文本说明，这是一个很重要的预防和补救措施。另外，还应该考虑到网页对于视觉障碍者，或者使用其他用户代理，如屏幕阅读器、打印机等代理设备的影响。当然，从语义角度考虑，替换文本应该提供图像的简明信息，并保证在上下文中有意义，而对于那些修饰性的图片可以使用空值（alt=""）。

title 属性为元素提供提示性的参考信息，这些信息是一些额外的说明，具有非本质性，因此该属性也不是一个必须设置的属性。当鼠标指针移到元素上面时，即可看到这些提示信息，但是 title 属性不能够用在下面的元素上。

html、head	文档和头部基本结构
title	网页标题
base、basefont	网页基准信息
meta	网页元信息
param	元素参数信息
script	网页的脚本和样式

相对而言，title 属性可以比 alt 属性设置更长的文本，不过有些浏览器可能会限制提示文本的长

度，但是不管怎么规定，提示文本一定要简明、扼要，并用在恰当的地方，而不是为所有元素都定义一个提示文本，那样就显得画蛇添足了。提示文本一般多用在超链接上，特别是对图标按钮必须提供提示性说明信息，否则用户就会不明白这些图标按钮的作用。

Note

如果要为元素定义更长的描述信息，则应该使用 longdesc 属性。longdesc 属性可以用来提供链接到一个包含图片描述信息的单独页面或者长段描述信息。其用法如下。

```
<img src="URL" alt="人物照" title="张三于 2010-5-1 中国馆留念" longdesc="这是张三于 2010 年 5 月 1 日在中国馆前的留影，当时天很热，穿着短裤，手里拿着矿泉水，到处都是云集于此的世博会开幕式观众，场面热闹非凡" />
```

或

```
<img src="UTL" alt="替换文本" longdesc="详细描述图像的网页.html" />
```

这种方法意味着从当前页面链接到另一个页面，由此可能会造成理解上的困难。另外浏览器对于 longdesc 属性的支持也不一致，因此应该避免使用。如果感觉对图片的长描述信息很有用，那么不妨考虑把这些信息简单地显示在同一个文档里，而不是链接到其他页面或者藏起来，这样能够保证每个人都可以阅读。

cite 一般用来定义引用信息的 URL。例如，下面一段文字引自 http://www.html5.cn/csslayout/ndex.htm，所以可以这样来设置：

```
<blockquote cite="http://www.html5.cn/csslayout/index.htm">
    <p>CSS 的精髓是布局，而不是样式，布局是需要缜密的结构分析和设计的</p>
</blockquote>
```

datetime 属性定义包含文本的时间，这个时间表示信息的发布时间，也可能是更新时间，例如：

```
<ins datetime="2010-5-1 8:0:0">2010 年上海世博会</ins>
```

2.2 XHTML 基础

XHTML 语言是在 HTML 语言基础上发展而来的，但是为了兼容数以万计的现存网页和不同浏览器，XHTML 文档与 HTML 文档没有太大区别，只是添加了 XML 语言的基本规范和要求。

2.2.1 XHTML 结构

完整的 XHTML 文档结构如下。

```
<!--[XHTML 文档基本框架]-->
<!--定义 XHTML 文档类型-->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!--XHTML 文档根元素，其中 xmlns 属性声明文档命名空间-->
<html xmlns="http://www.w3.org/1999/xhtml">
<!--头部信息结构元素-->
<head>
<!--设置文档字符编码-->
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
```

```
<!--设置文档标题-->
<title>无标题文档</title>
</head>
<!--主体内容结构元素-->
<body>
</body>
</html>
```

Note

XHTML 代码不排斥 HTML 规则，在结构上也基本相似，但如果仔细比较，它们有两点不同。

1. 定义文档类型

在 XHTML 文档第一行新增了<!DOCTYPE>元素，该元素用来定义文档类型。DOCTYPE 是 document type（文档类型）的简写，用于设置 XHTML 文档的版本。使用时应注意该元素的名称和属性必须大写。

DTD（如 xhtml1-transitional.dtd）表示文档类型定义，里面包含了文档的规则，网页浏览器会根据预定义的 DTD 来解析页面元素，并把这些元素所组织的页面显示出来。要建立符合网页标准的文档，DOCTYPE 声明是必不可少的关键组成部分，除非所编写的 XHTML 确定了一个正确的 DOCTYPE，否则页面内的元素和 CSS 不能正确生效。

2. 声明命名空间

在 XHTML 文档根元素中必须使用 xmlns 属性声明文档的命名空间。xmlns 是 XHTML NameSpace 的缩写，中文翻译为命名空间（也有人翻译为名字空间、名称空间）。命名空间是收集元素类型和属性名字的一个详细 DTD，它允许通过一个 URL 地址指向来识别命名空间。

XHTML 是 HTML 向 XML 过渡的标识语言，它需要符合 XML 规则，因此也需要定义命名空间。又因为 XHTML 1.0 还不允许用户自定义元素，因此它的命名空间都相同，就是 http://www.w3.org/1999/xhtml，这也是每个 XHTML 文档的 xmlns 值都相同的原因。

2.2.2 XHTML 语法

XHTML 是根据 XML 语法简化而来的，因此它遵循 XML 文档规范。同时 XHTML 又大量继承 HTML 语言语法规范，因此与 HTML 语言非常相似，不过它对代码的要求更加严谨。遵循这些要求，对于培养良好的 XHTML 代码书写习惯是非常重要的。

- ☑ 在文档的开头必须定义文档类型。
- ☑ 在根元素中应声明命名空间，即设置 xmlns 属性。
- ☑ 所有标签都必须是闭合的。在 HTML 中，用户可能习惯书写独立的标签，如 、
，而不习惯写对应的 和
来关闭它们，但在 XHTML 中这是不合法的。XHTML 要求有严谨的结构，所有标签都必须关闭。如果是单独不成对的标签，应在标签的最后加一个“/”来关闭它，如
。
- ☑ 所有元素和属性都必须小写。这与 HTML 不同，XHTML 对大小写是敏感的，<title>和<TITLE>表示不同的标签。
- ☑ 所有的属性必须用引号（""）括起来。在 HTML 中，可以不给属性值加引号，但是在 XHTML 中必须加引号，如<table height="80"></table>。特殊情况下，可以在属性值里使用双引号或单引号。
- ☑ 所有标签都必须合理嵌套。这是因为 XHTML 要求有严谨的结构，因此所有的嵌套都必须

按顺序，详细讲解请参阅最后一章。

☑ 所有属性都必须被赋值，没有值的属性就用自身来赋值。例如：

错误写法：

Note

```
<td nowrap>
```

正确写法：

```
<td nowrap="nowrap">
```

☑ 所有特殊符号都用编码表示，例如，小于号（<）不是元素的一部分，必须被编码为“<”；大于号（>）不是元素的一部分，必须被编码为“>”。

☑ 不要在注释内容中使用“--”。“--”只能出现在 XHTML 注释的开头和结束，也就是说在内容中它们不再有效。例如：

错误写法：

```
<!--注释---------注释-->
```

正确写法：

```
<!--注释————————注释-->
```

☑ XHTML 规范废除了 name 属性，而使用 id 属性作为统一的名称。在 IE 4.0 及以下版本中应保留 name 属性，使用时可以同时使用 id 和 name 属性。

上面列举的几点是 XHTML 最基本的语法要求，习惯于 HTML 的读者，应克服代码书写中的随意性，相信好的习惯会影响你的一生。

2.2.3 XHTML 类型

XHTML 1.0 支持 3 种 DTD（文档类型定义）声明：过渡型（Transitional）、严格型（Strict）和框架型（Frameset）。

1. 过渡型

这种文档类型对于标签和属性的语法要求不是很严格，允许在页面中使用 HTML 4.01 的标签（符合 XHTML 语法标准）。过渡型 DTD 语句如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w1.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

2. 严格型

这类文档类型对于文档内的代码要求比较严格，不允许使用任何表现层的标签和属性。严格型 DTD 语句如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w1.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

在严格型文档类型中，以下元素将不被支持：

```
center                    居中（属于表现层）
font                      字体样式，如大小、颜色和样式（属于表现层）
strike                    删除线（属于表现层）
s                         删除线（属于表现层）
```

Note

u	文本下划线（属于表现层）
iframe	嵌入式框架窗口（专用于框架文档类型或过渡型文档）
isindex	提示用户输入单行文本（与 input 元素语义重复）
dir	定义目录列表（与 dl 元素语义重复）
menu	定义菜单列表（与 ul 元素语义重复）
basefont	定义文档默认字体属性（属于表现层）
applet	定义插件（与 object 元素语义重复）

在严格型文档类型中，以下属性将不被支持：

align（支持 table 包含的相关元素：tr、td、th、col、colgroup、thead、tbody、tfoot）
language
background
bgcolor
border（table 元素支持）
height（img 和 object 元素支持）
hspace
name（在 HTML 4.01 Strict 中支持，在 XHTML 1.0 Strict 中的 form 和 img 元素不支持）
noshade
nowrap
target
text、link、vlink 和 alink
vspace
width（img、object、table、col 和 colgroup 元素支持）

3. 框架型

这是一种专门针对框架页面所使用的 DTD，当页面中含有框架元素时，就应该采用这种 DTD。框架型 DTD 语句如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w1.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

使用严格的 DTD 来制作页面，当然是最理想的方式，但是，对于没有深入了解 Web 标准的网页设计者来说，比较适合使用过渡型 DTD。因为过渡型 DTD 还允许使用表现层元素和属性，比较适合大多数网页制作人员使用。

对于大多数标准网页设计师来说，过渡型 DTD（XHTML 1.0 Transitional）是比较理想的选择。因为这种 DTD 允许使用描述性的元素和属性，也比较容易通过 W3C 的代码校验。

2.2.4　DTD 解析

在 XHTML 文档中，DOCTYPE 是一个必要元素，它决定了网页文档的显示规则。DOCTYPE 是 Document Type 的简写，中文翻译为文档类型。在网页中通过在首行代码中定义文档类型，来指定页面所使用的 HTML 的版本类型。在构建符合标准的网页时，只有使用正确的 DOCTYPE（文档类型），HTML 文档的结构和样式才能被正常解析和呈现。

实际上，DTD 是一套关于标签的语法规则。DTD 文件是一个 ASCII 的文本文件，后缀名为.dtd。利用 DOCTYPE 声明中的 URL 可以访问指定类型的 DTD 详细信息。例如，对于 XHTML 1.0 过渡型 DTD 的 URL 为 http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd，在 Dreamweaver 中打开 XHTML 1.0 过渡型 DTD 文档，代码显示如图 2.1 所示。

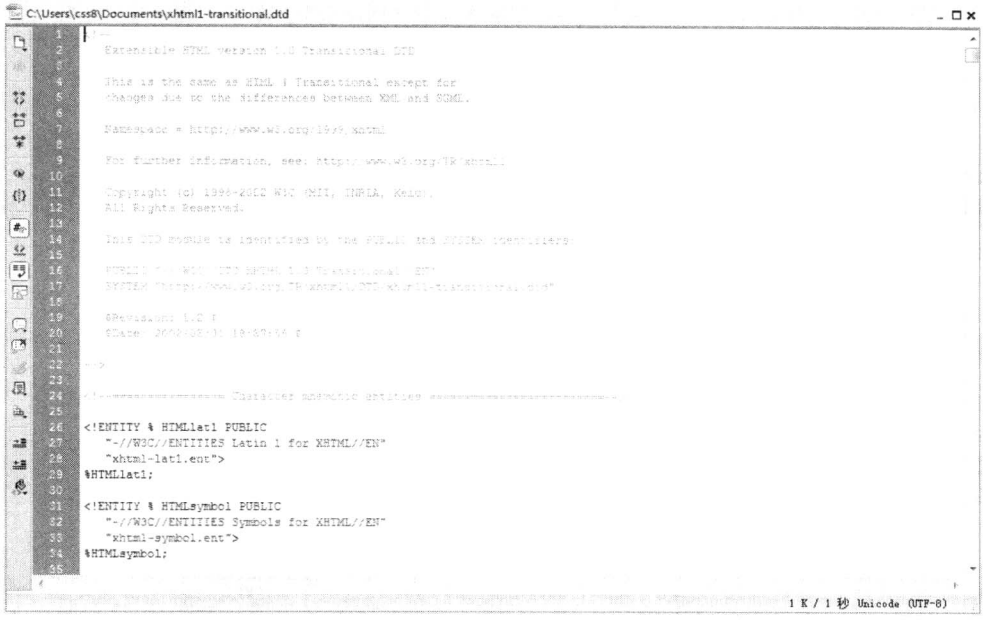


图 2.1　XHTML 1.0 过渡型 DTD 文档

一个 DTD 文档包含元素的定义规则，元素间关系的定义规则，元素可使用的属性、实体或符号规则。这些规则用于标识 Web 文档的内容。此外还包括了一些其他规则，它们规定了哪些标签能出现在其他标签中。文档类型不同，它们对应的 DTD 也不相同。

如果页面中没有显示声明 DOCTYPE，则不同浏览器就会自动采用各自默认的 DOCTYPE 规则来解析文档中的各种标签和 CSS 样式码。因此，从浏览器兼容性来考虑，声明 DOCTYPE 是必须的。DOCTYPE 声明必须放在（X）HTML 文档的顶部，在文档类型声明语句的上面不能够包含任何 HTML 代码，也不能包括 HTML 注释标签。DOCTYPE 声明语句的说明如图 2.2 所示。

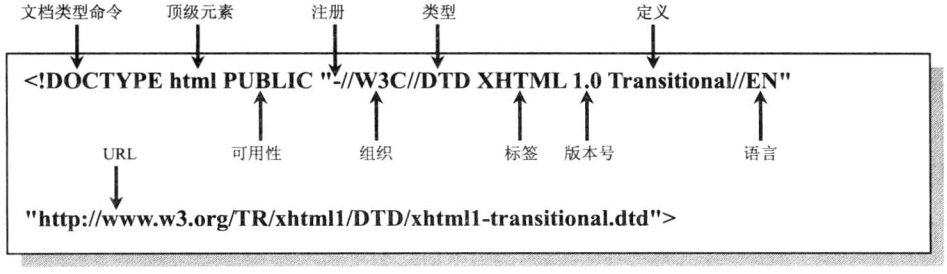

文档类型命令
顶级元素
注册
类型
定义
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
URL
可用性
组织
标签
版本号
语言
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

图 2.2　DOCTYPE 结构图

DOCTYPE 声明中各个部分说明如下。

- ☑ 顶级元素：指定 DTD 中声明的顶级元素类型，这与声明的 SGML 文档类型相对应。HTML 文档默认顶级元素为 html。
- ☑ 可用性：指定正式公开标识符（FPI）是可公开访问的对象（PUBLIC）还是系统资源（SYSTEM）。默认为 PUBLIC，SYSTEM 系统资源包括本地文件或 URL。
- ☑ 注册：指定组织是否由国际标准化组织（ISO）注册。“+”（默认）表示组织名称已注册，“-”表示组织名称未注册。W3C 是属于未注册 ISO 的组织，以显示为“-”符号。

Note

- ☑ 组织：指定在 DOCTYPE 声明引用的 DTD（文档类型定义）的创建和维护的团体或组织的名称。XHTML 语言规范的创建和维护组织为 W3C。
- ☑ 类型：指定公开文本的类，即所引用的对象类型。XHTML 默认为 DTD。
- ☑ 标签：指定公开文本的描述，即对所引用的公开文本的唯一描述性名称，后面可附带版本号。HTML 默认为 HTML，XHTML 默认为 XHTML，后面跟随的是语言版本号。
- ☑ 定义：指定文档类型定义，包含 Frameset（框架集文档）、Strict（严格型文档）和 Transitional（过渡型文档）。Strict（严格型文档）禁止使用 W3C 规范中指定将逐步淘汰的元素和属性，而 Transitional（过渡型文档）可以包含除 frameset 元素以外的全部内容。
- ☑ 语言：指定公开文本的语言，即用于创建所引用对象的自然语言编码系统。该语言定义已编写为 ISO 639 语言代码（两个字母要大写），默认为 EN（英语）。
- ☑ URL：指定所引用对象的位置。

从上面的结果分析中可以看到 DOCTYPE 声明语句的写法是严格遵循一定的规则的，只有这样，浏览器才能够调用对应文档类型的规则集来解释文档中的标签。所谓的文档类型规则集也就是 W3C 公开发布的一个文档类型定义（DTD）中包含的规则。

2.2.5　命名空间

在 XHTML 文档中，读者还需要注意另一个容易忽略的问题，即给<html>标签定义命名空间。例如：

```
<html xmlns="http://www.w1.org/1999/xhtml">
```

xmlns 是 html 元素的一个特殊属性。这个 xmlns 属性是 XHTML Name Space 的缩写，中文翻译为命名空间，该属性声明了 html 顶级元素的命名空间。那么命名空间在文档中是必须的吗？它有什么作用呢？

在标准设计中，命名空间是必须设置的一个属性，用来定义该顶级元素以及其包含的各级子元素的唯一性。命名空间声明允许用户通过一个网址指向来识别文档内标签的唯一性。

由于 XML 语言允许用户自定义标签，这样就可能存在所定义的标签与别人定义的标签名称发生冲突的情况。虽然标签名称不同，但是标签所表示的语义可能相同。当这些文档在网上自由传播或者相互交换文件时，由于名称相同可能会发生语义冲突。为此需要为各自的文档指定其语义的限制空间，于是 xmlns 属性就派上了用场。

【示例】分别定义张三和李四两个人的文档。

```
<!--张三：自定义文档 -->
<document>
    <name>书名</name>
    <author>作者</author>
    <content>目录</content>
</document>
<!--李四：自定义文档 -->
<document>
    <title>论文题目</title>
    <author>作者</author>
    <content>论文内容</content>
</document>
```

文档的根元素都是 document，同时文档中包含有很多相同的元素名。如果文档都在网上共享就会发生语义冲突。如果使用 xmlns 分别为它们定义一个命名空间，就不会发生冲突。例如：

Note

```
<!--张三：自定义文档 -->
<document xmlns="http://www.html5.cn/zhangsan">
    <name>书名</name>
    <author>作者</author>
    <content>目录</content>
</document>
<!--李四：自定义文档 -->
<document xmlns="http://www.html5.cn/lisi">
    <title>论文题目</title>
    <author>作者</author>
    <content>论文内容</content>
</document>
```

在上面的代码中，张三的文档命名空间为 http://www.html5.cn/zhangsan，而李四的命名空间为 http://www.html5.cn/lisi，虽然他们的文档存在相同的标签，但是借助顶级元素中定义的命名空间，相互之间就不会发生语义冲突。通俗地说，命名空间就是给文档做一个标签，标明该文档是属于哪个网站的。对于 HTML 文档来说，由于它的元素是固定的，不允许用户进行定义，所以指定的命名空间永远为 http://www.w1.org/1999/xhtml。

2.3 HTML5 基础

HTML5 以 HTML4 为基础，对 HTML4 进行了大量的修改。下面简单介绍 HTML5 对 HTML4 进行了哪些修改，以及 HTML5 与 HTML4 之间较大的区别是什么。

2.3.1 HTML5 语法

1. 内容类型

HTML5 的文件扩展名与内容类型保持不变。也就是说，扩展名仍然为.html 或.htm，内容类型仍然为 text/html。

2. 文档类型声明

根据 HTML5 设计化繁为简的准则，文档类型和字符说明都进行了简化。DOCTYPE 声明是 HTML 文件中必不可少的，位于文件第一行。在 HTML4 中，它的声明方法如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

在 HTML5 中，刻意不使用版本声明，一份文档将会适用于所有版本的 HTML。HTML5 中的 DOCTYPE 声明方法（不区分大小写）如下。

```
<!DOCTYPE html>
```

另外，当使用工具时，也可以在 DOCTYPE 声明方式中加入 SYSTEM 识别符，声明方法如下面

的代码所示。

```
<!DOCTYPE HTML SYSTEM "about:legacy-compat">
```

在 HTML5 中，像这样的 DOCTYPE 声明方式是允许的，不区分大小写，引号不区分单引号和双引号。

Note

提示：使用 HTML5 的 DOCTYPE 会触发浏览器以标准兼容模式显示页面。众所周知，网页都有多种显示模式，如怪异模式（Quirks）、近标准模式（Almost Standards）和标准模式（Standards）。其中标准模式也被称为非怪异模式（no-quirks）。浏览器会根据 DOCTYPE 来识别该使用哪种模式，以及使用什么规则来验证页面。

3. 字符编码

在 HTML4 中，使用<meta>元素的形式指定文件中的字符编码，如下所示。

```
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
```

在 HTML5 中，可以使用对<meta>元素直接追加 charset 属性的方式来指定字符编码，如下所示。

```
<meta charset="UTF-8">
```

两种方法都有效，可以继续使用前面一种方式，即通过 content 元素的属性来指定。但是不能同时混合使用两种方式。在以前的网站代码中可能会存在下面代码所示的标记方式，但在 HTML5 中，下面这种字符编码方式将被认为是错误的。

```
<meta charset="UTF-8" http-equiv="Content-Type" content="text/html;charset=UTF-8">
```

从 HTML5 开始，对于文件的字符编码推荐使用 UTF-8。

4. 版本兼容性

HTML5 的语法是为了保证与之前的 HTML 语法达到最大程度的兼容而设计的。简单说明如下。

☑　可以省略标记的元素

在 HTML5 中，元素的标记可以省略。具体来说，元素的标记分为 3 种类型：不允许写结束标记、可以省略结束标记、开始标记和结束标记全部可以省略。下面简单介绍这 3 种类型各包括哪些 HTML5 新元素。

第一，不允许写结束标记的元素有：area、base、br、col、command、embed、hr、img、input、keygen、link、meta、param、source、track 和 wbr。

第二，可以省略结束标记的元素有：li、dt、dd、p、rt、rp、optgroup、option、colgroup、thead、tbody、tfoot、tr、td 和 th。

第三，可以省略全部标记的元素有：html、head、body、colgroup 和 tbody。

提示：不允许写结束标记的元素是指不允许使用开始标记与结束标记将元素括起来的形式，只允许使用<元素/>的形式进行书写。例如，
...</br>的书写方式是错误的，正确的书写方式为
。当然，HTML5 之前的版本中
这种写法可以被沿用。

可以省略全部标记的元素是指该元素可以完全被省略。注意，即使标记被省略了，该元素还是以隐式的方式存在的。例如，将 body 元素省略不写时，它在文档结构中还是存在的，可以使用 document.body 进行访问。

☑ 具有布尔值的属性

对于具有 boolean 值的属性，如 disabled 与 readonly 等，当只写属性而不指定属性值时，表示属性值为 true；如果想要将属性值设为 false，可以不使用该属性。另外，要想将属性值设定为 true，也可以将属性名设定为属性值，或将空字符串设定为属性值。例如：

Note

```
<!--只写属性，不写属性值，代表属性为 true-->
<input type="checkbox" checked>
<!--不写属性，代表属性为 false-->
<input type="checkbox">
<!--属性值=属性名，代表属性为 true-->
<input type="checkbox" checked="checked">
<!--属性值=空字符串，代表属性为 true-->
<input type="checkbox" checked="">
```

☑ 省略引号

属性值两边既可以用双引号，也可以用单引号。HTML5 在此基础上做了一些改进，当属性值不包括空字符串、<、>、=、单引号、双引号等字符时，属性值两边的引号可以省略。例如，下面的写法都是合法的。

```
<input type="text">
<input type='text'>
<input type=text>
```

【示例】通过上面介绍的 HTML5 语法知识，下面完全用 HTML5 编写一个文档，在该文档中省略了<html>、<head>、<body>等元素。可以通过本示例复习一下 HTML5 的 DOCTYPE 声明、用<meta>元素的 charset 属性指定字符编码、<p>元素的结束标记的省略、使用<元素/>的方式来结束<meta>元素，以及
元素等本节中所介绍到的知识要点。

```
<!DOCTYPE html>
<meta charset="UTF-8">
<title>HTML5 基本语法</title>
<h1>HTML5 的目标</h1>
<p>HTML5 的目标是为了能够创建更简单的 Web 程序，书写出更简洁的 HTML 代码。
<br/>例如，为了使 Web 应用程序的开发变得更容易，提供了很多 API；为了使 HTML 变得更简洁，开发出了新的属性、新的元素等。总体来说，为下一代 Web 平台提供了许许多多新的功能。
```

这段代码在 IE 浏览器中的运行结果如图 2.3 所示。

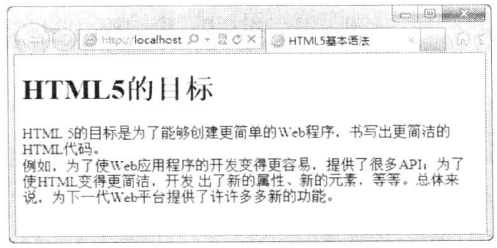


图 2.3 第一个 HTML5 文档

2.3.2 HTML5 元素

HTML5 中引入了很多新的标记元素，根据内容类型的不同，这些元素被分成了 7 大类，如表 2.1

所示。

表 2.1　HTML5 的内容类型

内 容 类 型	说　　明
内嵌	在文档中添加其他类型的内容，如 audio、video、canvas 和 iframe 等
流	在文档和应用的 body 中使用的元素，如 form、h1 和 small 等
标题	段落标题，如 h1、h2 和 hgroup 等
交互	与用户交互的内容，如音频和视频的控件、button 和 textarea 等
元数据	通常出现在页面的 head 中，设置页面其他部分的表现和行为，如 script、style 和 title 等
短语	文本和文本标记元素，如 mark、kbd、sub 和 sup 等

Note

上表中所有类型的元素都可以通过 CSS 来设定样式。虽然 canvas、audio 和 video 元素在使用时往往需要其他 API 来配合，以实现细粒度控制，但它们同样可以直接使用。

1. 新增的结构元素

HTML5 定义了一组新的语义化标记来描述元素的内容。虽然语义化标记也可以使用 HTML 标记进行替换，但是它可以简化 HTML 页面设计，并且搜索引擎在抓取和索引网页的时候，也会利用这些元素。在目前主流的浏览器中已经可以用这些元素了，新增的语义化标记元素如表 2.2 所示。

表 2.2　HTML5 新增的语义化标记元素

元 素 名 称	说　　明
header	标记头部区域的内容（用于整个页面或页面中的一块区域）
footer	标记脚部区域的内容（用于整个页面或页面中的一块区域）
section	Web 页面中的一块区域
article	独立的文章内容
aside	相关内容或者引文
nav	导航类辅助内容

根据 HTML5 效率优先的设计理念，它推崇表现和内容的分离，所以在 HTML5 的实际编程中，开发人员必须使用 CSS 来定义样式。

【示例】在本示例中分别使用 HTML5 提供的各种语义化结构标记重新设计一个网页，效果如图 2.4 所示。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" >
<title>HTML5 结构元素</title>
<link rel="stylesheet" href="html5.css">
</head>
<body>
<header>
    <h1>网页标题</h1>
    <h2>次级标题</h2>
    <h4>提示信息</h4>
```

```
</header>
<div id="container">
    <nav>
        <h3>导航</h3>
        <a href="#">链接 1</a> <a href="#">链接 2</a> <a href="#">链接 3</a> </nav>
    <section>
        <article>
            <header>
                <h1>文章标题</h1>
            </header>
            <p>文章内容.....</p>
            <footer>
                <h2>文章注脚</h2>
            </footer>
        </article>
    </section>
    <aside>
        <h3>相关内容</h3>
        <p>相关辅助信息或者服务.....</p>
    </aside>
    <footer>
        <h2>页脚</h2>
    </footer>
</div>
</body>
</html>
```

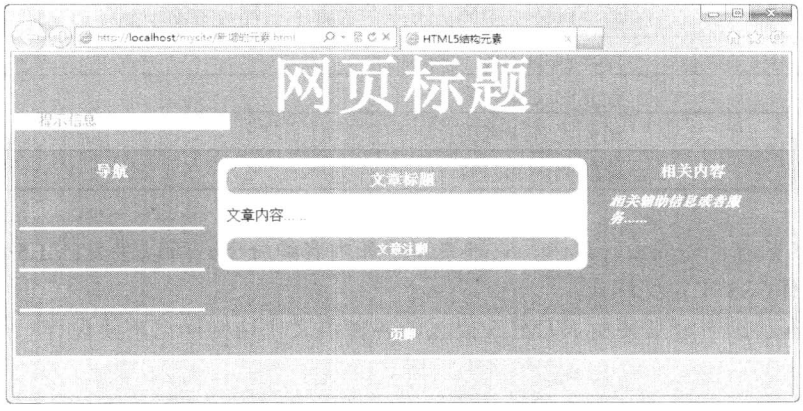


图 2.4　HTML5 语义化结构网页

上面示例中使用了 CSS3 的一些新特性，如圆角（border-radius）和旋转变换（transform:rotate()）等，相关介绍请参阅后面章节的内容。

2. HTML5 新增功能元素

☑ hgroup 元素：用于对整个页面或页面中一个内容区块的标题进行组合。例如：

```
<hgroup>...</hgroup>
```

在 HTML4 中表示为：

```
<div>...</div>
```

☑ figure 元素：表示一段独立的流内容，一般表示文档主体流内容中的一个独立单元。使用 figcaption 元素可为 figure 元素组添加标题。例如：

```
<figure>
    <figcaption>标题</figcaption>
    <p>内容...</p>
</figure>
```

Note

在 HTML4 中表示为：

```
<dl>
    <h1>标题</h1>
    <p>内容...</p>
</dl>
```

☑ video 元素：定义视频，比如电影片段或其他视频流。例如：

```
<video src="movie.ogg" controls="controls">video 元素</video>
```

在 HTML4 中表示为：

```
<object type="video/ogg" data="movie.ogv">
    <param name="src" value="movie.ogv">
</object>
```

☑ audio 元素：定义音频，比如音乐或其他音频流。例如：

```
<audio src="someaudio.wav">audio 元素</audio>
```

在 HTML4 中表示为：

```
<object type="application/ogg" data="someaudio.wav">
    <param name="src" value="someaudio.wav">
</object>
```

☑ embed 元素：用来插入各种多媒体，格式可以是 MIDI、WAV、AIFF、AU、MP3 等。例如：

```
<embed src="horse.wav" />
```

在 HTML4 中表示为：

```
<object data="flash.swf" type="application/x-shockwave-flash"></object>
```

☑ mark 元素：主要用来在视觉上向用户呈现那些需要突出显示或高亮显示的文字。mark 元素的一个比较典型的应用就是在搜索结果中向用户高亮显示搜索关键词。例如：

```
<mark></mark>
```

在 HTML4 中表示为：

```
<span></span>
```

☑ time 元素：表示日期或时间，也可以同时表示两者。例如：

```
<time></time>
```

在 HTML4 中表示为：

```
<span></span>
```

☑ canvas 元素：表示图形，如图表和其他图像。这个元素本身没有行为，仅提供一块画布，但它把一个绘图 API 展现给客户端 JavaScript，以使脚本能够把想绘制的东西绘制到这块画布上。例如：

Note

```
<canvas id="myCanvas" width="200" height="200"></canvas>
```

在 HTML4 中表示为：

```
<object data="inc/hdr.svg" type="image/svg+xml" width="200" height="200">
</object>
```

☑ output 元素：表示不同类型的输出，比如脚本的输出。例如：

```
<output></output>
```

在 HTML4 中表示为：

```
<span></span>
```

☑ source 元素：为媒介元素（比如<video>和<audio>）定义媒介资源。例如：

```
<source>
```

在 HTML4 中表示为：

```
<param>
```

☑ menu 元素：表示菜单列表。当希望列出表单控件时使用该标签。例如：

```
<menu>
    <li><input type="checkbox" />Red</li>
    <li><input type="checkbox" />blue</li>
</menu>
```

在 HTML4 中，menu 元素不被推荐使用。

☑ ruby 元素：表示 ruby 注释（中文注音或字符）。例如：

```
<ruby>汉<rt><rp>(</rp>ㄏㄢˋ<rp>)</rp></rt></ruby>
```

☑ rt 元素：表示字符（中文注音或字符）的解释或发音。例如：

```
<ruby>汉<rt> ㄏㄢˋ</rt></ruby>
```

☑ rp 元素：在 ruby 注释中使用，以定义不支持 ruby 元素的浏览器所显示的内容。例如：

```
<ruby>汉<rt><rp>(</rp>ㄏㄢˋ<rp>)</rp></rt></ruby>
```

☑ wbr 元素：表示软换行。wbr 元素与 br 元素的区别是 br 元素表示此处必须换行；而 wbr 元素的意思是浏览器窗口或父级元素的宽度足够宽时（没必要换行时）不进行换行，而当宽度不够时，主动在此处进行换行。

```
<p> TW3C invites media, analysts, and other attendees of Mobile World Congress (MWC) <wbr> 2012 to meet
with W3C and learn how the Open Web Platform <wbr>is transforming industry. From 27 February through 1 March
W3C will </p>\
```

☑ command 元素：表示命令按钮，如单选按钮、复选框或按钮。例如：

```
<command onclick=ucut()" label="cut">
```

☑ details 元素：表示用户要求得到并且可以得到的细节信息，可以与 summary 元素配合使用。summary 元素提供标题或图例。标题是可见的，用户单击标题时，会显示出细节信息。summary 元素应该是 details 元素的第一个子元素。例如：

Note

```
<details>
    <summary>HTML5</summary>
    For the latest updates from the HTML WG, possibly including important bug fixes, please look at the editor's draft instead. There may also be a more up-to-date Working Draft with changes based on resolution of Last Call issues..
</details>
```

☑ datalist 元素

datalist 元素表示可选数据的列表，与 input 元素配合使用，可以制作出输入值的下拉列表。例如：

```
<datalist></datalist>
```

☑ datagrid 元素：表示可选数据的列表，它以树形列表的形式来显示。例如：

```
<datagrid></datagrid>
```

☑ keygen 元素：表示生成密钥。例如：

```
<keygen>
```

☑ progress 元素：表示运行中的进程，可以使用 progress 元素来显示 JavaScript 中耗费时间的函数的进程。例如：

```
<meter></meter>
```

☑ email：表示必须输入 E-mail 地址的文本输入框。
☑ url：表示必须输入 URL 地址的文本输入框。
☑ number：表示必须输入数值的文本输入框。
☑ range：表示必须输入一定范围内数字值的文本输入框。
☑ Date Pickers：HTML5 拥有多个可供选取日期和时间的新型输入文本框：
- date—选取日、月、年。
- month—选取月和年。
- week—选取周和年。
- time—选取时间（小时和分钟）。
- datetime—选取时间、日、月、年（UTC 时间）。
- datetime-local—选取时间、日、月、年（本地时间）。

3. HTML5 中废除的元素

在 HTML5 中废除了 HTML4 中过时的元素，简单介绍如下。

☑ 能使用 CSS 替代的元素

对于 basefont、big、center、font、s、strike、tt 和 u 这些元素，由于它们的功能都是表现文本效果，而 HTML5 中提倡把呈现性功能放在 CSS 样式表中统一编辑，所以将这些元素废除了，并使用编辑 CSS、添加 CSS 样式表的方式进行替代。其中 font 元素允许由“所见即所得”的编辑器来插入，s 元素、strike 元素可以由 del 元素替代，tt 元素可以由 CSS 的 font-family 属性替代。

☑ 不再使用 frame 框架

对于 frameset 元素、frame 元素与 noframes 元素，由于 frame 框架对网页可用性存在负面影响，在 HTML5 中已不支持 frame 框架，只支持 iframe 框架，或者用服务器方创建的由多个页面组成的复

合页面的形式，同时将以上这 3 个元素废除。

☑ 只有部分浏览器支持的元素

对于 applet、bgsound、blink 和 marquee 等元素，由于只有部分浏览器支持这些元素，特别是 bgsound 元素以及 marquee 元素，只被 IE 所支持，所以在 HTML5 中被废除。其中 applet 元素可由 embed 元素或 object 元素替代，bgsound 元素可由 audio 元素替代，marquee 可以由 JavaScript 编程的方式替代。

Note

其他被废除元素还有：

☑ 使用 ruby 元素替代 rb 元素。
☑ 使用 abbr 元素替代 acronym 元素。
☑ 使用 ul 元素替代 dir 元素。
☑ 使用 form 元素与 input 元素相结合的方式替代 isindex 元素。
☑ 使用 pre 元素替代 listing 元素。
☑ 使用 code 元素替代 xmp 元素。
☑ 使用 GUIDS 替代 nextid 元素。
☑ 使用“text/plian”MIME 类型替代 plaintext 元素。

2.3.3 HTML5 增加及废除的属性

HTML5 同时增加和废除了很多属性。简单说明如下。

1. 增加的属性

☑ 表单属性

➢ 为 input（type=text）、select、textarea 与 button 元素新增加 autofocus 属性。它以指定属性的方式让元素在画面打开时自动获得焦点。
➢ 为 input 元素（type=text）与 textarea 元素新增加 placeholder 属性，它会对用户的输入进行提示，提示用户可以输入的内容。
➢ 为 input、output、select、textarea、button 与 fieldset 新增加 form 属性，声明它属于哪个表单，然后将其放置在页面上任何位置，而不是表单之内。
➢ 为 input 元素（type=text）与 textarea 元素新增加 required 属性。该属性表示在用户提交的时候进行检查，检查该元素内一定要有输入内容。
➢ 为 input 元素增加 autocomplete、min、max、multiple、pattern 和 step 属性。同时还有一个新的 list 元素与 datalist 元素配合使用。datalist 元素与 autocomplete 属性配合使用。multiple 属性允许在上传文件时一次上传多个文件。
➢ 为 input 元素与 button 元素增加了新属性 formaction、formenctype、formmethod、formnovalidate 与 formtarget，它们可以重载 form 元素的 action、enctype、method、novalidate 与 target 属性。为 fieldset 元素增加了 disabled 属性，可以把它的子元素设为 disabled（无效）状态。
➢ 为 input 元素、button 元素和 form 元素增加了 novalidate 属性，该属性可以取消提交时进行的有关检查，表单可以被无条件地提交。

☑ 链接属性

➢ 为 a 元素与 area 元素增加了 media 属性，该属性规定目标 URL 是为哪种类型的媒介或设备进行优化的，只能在 href 属性存在时使用。
➢ 为 area 元素增加了 hreflang 属性与 rel 属性，以保持与 a 元素、link 元素的一致。

- ➢ 为 link 元素增加了新属性 sizes。该属性可以与 icon 元素结合使用（通过 rel 属性），用于指定关联图标（icon 元素）的大小。
- ➢ 为 base 元素增加了 target 属性，主要目的是保持与 a 元素的一致性。

☑ 其他属性

- ➢ 为 ol 元素增加 reversed 属性，用于指定列表倒序显示。
- ➢ 为 meta 元素增加 charset 属性，因为这个属性已经被广泛支持了，而且为文档的字符编码的指定提供了一种比较良好的方式。
- ➢ 为 menu 元素增加了两个新的属性——type 与 label。label 属性为菜单定义一个可见的标注，type 属性让菜单可以以上下文菜单、工具条或列表菜单这 3 种形式出现。
- ➢ 为 style 元素增加 scoped 属性，用来规定样式的作用范围，譬如只对页面上某个树起作用。
- ➢ 为 script 元素增加 async 属性，用于定义脚本是否异步执行。
- ➢ 为 html 元素增加 manifest 属性，开发离线 Web 应用程序时，它与 API 结合使用，定义一个 URL，在这个 URL 上描述文档的缓存信息。
- ➢ 为 iframe 元素增加 3 个属性 sandbox、seamless 与 srcdoc，用来提高页面安全性，防止不信任的 Web 页面执行某些操作。'

2. 废除的属性

HTML5 废除了 HTML4 中过时的属性，而采用其他属性或其他方案进行替代，具体说明如表 2.3 所示。

表 2.3　HTML5 废除的属性

HTML 4 属性	适 应 元 素	HTML5 替代方案
rev	link、a	rel
charset	link、a	在被链接的资源中使用 HTTP Content-type 头元素
shape、coords	a	使用 area 元素代替 a 元素
longdesc	img、iframe	使用 a 元素链接到较长描述
target	link	多余属性，被省略
nohref	area	多余属性，被省略
profile	head	多余属性，被省略
version	html	多余属性，被省略
name	img	id
scheme	meta	只为某个表单域使用 scheme
archive、classid、codebase、codetype、declare、standby	object	使用 data 与 type 属性类调用插件。需要使用这些属性来设置参数时，使用 param 属性
valuetype、type	param	使用 name 与 value 属性，不声明值的 MIME 类型
axis、abbr	td、th	使用以明确简洁的文字开头，后跟详述文字的形式。可以对更详细内容使用 title 属性，来使单元格的内容变得简短
scope	td	在被链接的资源中使用 HTTP Content-type 头元素
align	caption、input、legend、div、h1、h2、h3、h4、h5、h6、p	使用 CSS 样式表替代

续表

HTML 4 属性	适 应 元 素	HTML5 替代方案
alink、link、text、vlink、background、bgcolor	body	使用 CSS 样式表替代
align、bgcolor、border、cellpadding、cellspacing、frame、rules、width	table	使用 CSS 样式表替代
align、char、charoff、height、nowrap、valign	tbody、thead、tfoot	使用 CSS 样式表替代
align、bgcolor、char、charoff、height、nowrap、valign、width	td、th	使用 CSS 样式表替代
align、bgcolor、char、charoff、valign	tr	使用 CSS 样式表替代
align、char、charoff、valign、width	col、colgroup	使用 CSS 样式表替代
align、border、hspace、vspace	object	使用 CSS 样式表替代
clear	br	使用 CSS 样式表替代
compact、type	ol、ul、li	使用 CSS 样式表替代
compact	dl	使用 CSS 样式表替代
compact	menu	使用 CSS 样式表替代
width	pre	使用 CSS 样式表替代
align、hspace、vspace	img	使用 CSS 样式表替代
align、noshade、size、width	hr	使用 CSS 样式表替代
align、frameborder、scrolling、marginheight、marginwidth	iframe	使用 CSS 样式表替代
autosubmit	menu	

2.3.4 HTML5 全局属性

在 HTML5 中新增了全局属性的概念。所谓全局属性是指可以对任何元素都使用的属性。

1. contentEditable 属性

contentEditable 属性的主要功能是允许用户在线编辑元素中的内容。contentEditable 是一个布尔值属性，可以被指定为 true 或 false。此外，该属性还有个隐藏的 inherit（继承）状态，属性值为 true 时，元素被指定为允许编辑；属性值为 false 时，元素被指定为不允许编辑；未指定 true 或 false 时，则由 inherit 状态来决定，如果元素的父元素是可编辑的，则该元素就是可编辑的。

【示例】在本示例中为列表元素加上 contentEditable 属性后，该元素就变成可编辑的了，读者可自行在浏览器中修改列表内容。

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>conentEditalbe 属性示例</title>
</head>
<h2>可编辑列表</h2>
<ul contentEditable="true">
    <li>列表元素 1</li>
    <li>列表元素 2</li>
```

```
    <li>列表元素 3</li>
</ul>
```

这段代码运行后的结果如图 2.5 所示。

原始列表　　编辑列表项项目

图 2.5　可编辑列表

在编辑完元素中的内容后，如果想要保存这些内容，只能把该元素的 innerHTML 发送到服务器端进行保存，因为改变元素内容后该元素的 innerHTML 内容也会随之改变，目前还没有特别的 API 来保存编辑后元素中的内容。

contentEditable 属性支持的元素包括：defaults、A、ABBR、ACRONYM、ADDRESS、B、BDO、BIG、BLOCKQUOTE、BODY、BUTTON、CENTER、CITE、CODE、CUSTOM、DD、DEL、DFN、DIR、DIV、DL、DT、EM、FIELDSET、FONT、FORM、hn、I、INPUT type=button、INPUT type=password、INPUT type=radio、INPUT type=reset、INPUT type=submit、INPUT type=text、INS、ISINDEX、KBD 和 LABEL。

2. designMode 属性

designMode 属性用来指定整个页面是否可编辑，当页面可编辑时，页面中任何支持上文所述的 contentEditable 属性的元素都变成了可编辑状态。designMode 属性只能在 JavaScript 脚本里被编辑修改。该属性有两个值：on 与 off。属性被指定为 on 时，页面可编辑；被指定为 off 时，页面不可编辑。使用 JavaScript 脚本来指定 designMode 属性的用法如下所示。

```
document.designMode="on"
```

针对 designMode 属性，各浏览器的支持情况也各不相同。

- ☑ IE8：出于安全考虑，不允许使用 designMode 属性让页面进入编辑状态。
- ☑ IE9：允许使用 designMode 属性让页面进入编辑状态。
- ☑ Chrome 3 和 Safari：使用内嵌 frame 的方式，该内嵌 frame 是可编辑的。
- ☑ Firefox 和 Opera：允许使用 designMode 属性让页面进入编辑状态。

3. hidden 属性

在 HTML5 中，所有的元素都允许使用一个 hidden 属性。该属性类似于 input 元素中的 hidden 元素，功能是通知浏览器不渲染该元素，使该元素处于不可见状态。但是元素中的内容还是浏览器创建的，也就是说页面装载后允许使用 JavaScript 脚本将该属性取消，取消后该元素变为可见状态，同时元素中的内容也即时显示出来。hidden 属性是一个布尔值的属性，当设为 true 时，元素处于不可见状态；当设为 false 时，元素处于可见状态。

4. spellcheck 属性

spellcheck 属性是 HTML5 针对 input 元素（type=text）与 textarea 这两个文本输入框提供的一个

新属性，它的功能为对用户输入的文本内容进行拼写和语法检查。spellcheck 属性是一个布尔值的属性，具有 true 和 false 两种值，但是书写时有一个特殊的地方，就是必须明确声明属性值为 true 或 false。基本用法如下所示。

Note

```
<!--以下两种书写方法正确-->
<textarea spellcheck="true" >
<input type=text spellcheck=false>
<!--以下书写方法错误-->
<textarea spellcheck >
```

需要注意的是，如果元素的 readOnly 属性或 disabled 属性设为 true，则不执行拼写检查。目前除了 IE 之外，Firefox、Chrome、Safari 和 Opera 等浏览器都对该属性提供了支持。

5. tabindex 属性

tabindex 是开发中的一个基本概念，当不断按 Tab 键让窗口或页面中的控件获得焦点，对窗口或页面中的所有控件进行遍历的时候，每一个控件的 tabindex 表示该控件是第几个被访问到的。

2.3.5 HTML5 其他功能

1. Selectors API

HTML5 引入了一种用于查找页面 DOM 元素的快捷方式。在传统方法中主要使用 JavaScript 脚本来实现。例如，使用 getElementById()函数根据指定的 ID 值查找并返回元素，使用 getElementsByName()函数返回所有 name 指定值的元素，getElementsByTagName()函数返回所有标签名称与指定值相匹配的元素。

有了新的 Selectors API 之后，可以用更精确的方式来指定希望获取的元素，而不必再用标准 DOM 方式循环遍历。Selectors API 与现在 CSS 中使用的选择规则一样，通过它可以查找页面中的一个或多个元素。例如，CSS 已经可以基于嵌套、兄弟和子模式等关系进行元素选择。CSS 的最新版除添加了更多对伪类的支持，如判断一个对象是否被启用、禁用或者被选择等，还支持对属性和层次的随意组合叠加。使用如表 2.4 所示的函数就能按照 CSS 规则来选取 DOM 中的元素。

表 2.4 QuerySelector 新方法

函　数	说　明	示　例	返 回 值
querySelector()	根据指定的选择规则，返回在页面中找到的第一个匹配元素	querySelector ("input.error");	返回第一个 CSS 类名为“error”的文本输入框
querySelectorAll()	根据指定规则返回页面中所有相匹配的元素	querySelectorAll ("#results td");	返回 id 值为 results 的元素下所有的单元格

可以为 Selectors API 函数同时指定多个选择规则，例如：

```
//选择文档中类名为 highClass 或 lowClass 的第一个元素
var x = document.querySelector(".highClass", ".lowClass");
```

对于 querySelector()来说，选择的是满足规则中任意条件的第一个元素。对于 querySelectorAll()来说，页面中的元素只要满足规则中的任何一个条件，都会被返回，多条规则是用逗号分隔的。以前在页面上跟踪用户操作很困难，但新的 Selectors API 提供了更为便捷的方法。

【示例】在页面上有一个表格，如果想获取鼠标当前位置，使用 Selectors API 就可以很简单地实现。演示效果如图 2.6 所示。

Note

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>上机练习</title>
<style type="text/css">
td { border-style: solid; border-width: 1px; font-size: 200%; }
td:hover { background-color: cyan; }
#hoverResult { color: green; font-size: 200%; }
</style>
</head>
<body>
<section>
    <table>
        <tr>
            <td>1</td>
            <td>一个人生活</td>
            <td>温岚</td>
        </tr>
        <tr>
            <td>2</td>
            <td>让我爱你</td>
            <td>胡夏</td>
        </tr>
    </table>
    <button type="button" id="findHover" autofocus>查看鼠标焦点目标位置</button>
    <div id="hoverResult"></div>
    <script type="text/javascript">
    document.getElementById("findHover").onclick = function() {
    // 找到鼠标当前悬停的单元格
        var hovered = document.querySelector("td:hover");
        if (hovered)
            document.getElementById("hoverResult").innerHTML = hovered.innerHTML;
        }
    </script>
</section>
</body>
</html>
```

图 2.6　Selectors API 应用

从以上示例可以看到，仅用一行代码即可找到用户鼠标下面的元素：

```
var hovered = document.querySelector("td:hover");
```

Note

提示：Selectors API 不仅仅是方便，在遍历 DOM 的时候，Selectors API 通常会比以前的子节点搜索 API 更快。为了实现快速样式表，浏览器对选择器匹配进行了高度优化。

2. JavaScript 日志和调试

从技术上讲 JavaScript 日志和浏览器内调试虽然不属于 HTML5 的功能，但在过去的几年里，相关工具的发展出现了质的飞跃。第一个可以用来分析 Web 页面及其所运行脚本的强大工具是一款名为 Firebug 的 Firefox 插件。现在，相同的功能在其他浏览器的内嵌开发工具中也可以找到。例如，Safari 的 Web Inspector、Google 的 Chrome 开发者工具（Developer Tools）、IE 的开发者工具（Developer Tools），以及 Opera 的 Dragonfly。很多调试工具支持设置断点来暂停代码执行、分析程序状态以及查看变量的当前值。

console.log API 已经成为 JavaScript 开发人员记录日志的事实标准。为了便于开发人员查看记录到控制台的信息，很多浏览器提供了分栏窗格的视图。console.log API 要比 alert()好用很多，因为它不会阻塞脚本的执行。

3. window.JSON

JSON 是一种相对来说比较新并且正在日益流行的数据交换格式。作为 JavaScript 语法的一个子集，它将数据表示为对象字面量。由于其语法简单和在 JavaScript 编程中与生俱来的兼容性，JSON 变成了 HTML5 应用内部数据交换的事实标准。典型的 JSON API 包含两个函数：parse()和 stringify()（分别用于将字符串序列化成 DOM 对象和将 DOM 对象转换成字符串）。

如果在旧的浏览器中使用 JSON，需要 JavaScript 库（有些可以从 http://json.org 上找到）。在 JavaScript 中执行解析和序列化效率往往不高，所以为了提高执行速度，新的浏览器原生扩展了对 JSON 的支持，可以直接通过 JavaScript 来调用 JSON 了。这种本地化的 JSON 对象被纳入了 ECMAScript 5 标准，成为了下一代 JavaScript 语言的一部分。它也是 ECMAScript 5 标准中首批被浏览器支持的功能之一。所有新的浏览器都支持 window.JSON，将来 JSON 必将大量应用于 HTML5 中。

4. DOM Level 3

事件处理是目前 Web 应用开发中最麻烦的部分。除了 IE 以外，绝大多数浏览器都支持处理事件和元素的标准 API。早期 IE 实现的是与最终标准不同的事件模型，而 IE9 开始支持 DOM Level 2 和 DOM Level 3 的特性。如此，在所有支持 HTML5 的浏览器中，用户终于可以使用相同的代码来实现 DOM 操作和事件处理了，包括非常重要的 addEventListener()和 dispatchEvent()方法。

5. Monkeys、Squirrelfish 和其他 JavaScript 引擎

最新版本的主流浏览器不仅大量增加了新的 HTML5 标签和 API。同时主流浏览器中 JavaScript/ECMAScript 引擎升级幅度也非常大。新的 API 提供了很多上一代浏览器无法实现的功能，因而脚本引擎整体执行效率的提升，不论对现有的，还是使用了最新 HTML5 特性的 Web 应用都有好处。

开发出更快的 JavaScript 引擎是目前主流浏览器竞争的核心。过去的 JavaScript 纯粹是被解释执行，而最新的引擎则直接将脚本编译成原生机器代码，相比 2005 年前后的浏览器，速度的提升已经不在一个数量级上了。

2006 年 Adobe 将其 JIT 编译引擎和代号为 Tamarin 的 ECMAScript 虚拟机捐赠给 Mozilla 基金会，

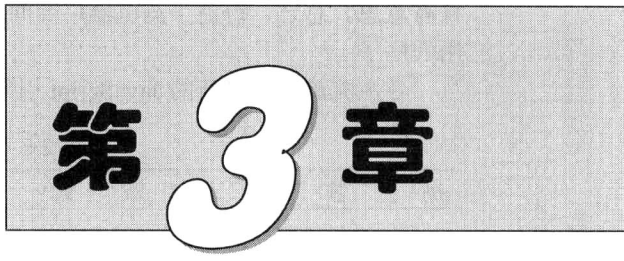
从此 JavaScript 引擎的竞争序幕就拉开了。尽管新版的 Mozilla 中 Tamarin 技术已经所剩无几，但 Tamarin 的捐赠促进了各家浏览器对新脚本引擎的研发，而这些引擎的名字就如同他们声称的性能一样有意思。总之，得益于浏览器厂商间的良性竞争，JavaScript 的执行性能越来越接近于本地桌面应用程序了。

Note

各主流浏览器最新的 JavaScript 引擎说明如表 2.5 所示。

表 2.5　Web 浏览器的 JavaScript 引擎

浏　览　器	引 擎 名 称	说　　明
Safari	Nitro（也称 Squirrel Fish Extreme）	Safari 4 中发布，在 Safari 5 中提升性能，包括字节码优化和上下文线程的本地编译器
Chrome	V8	自从 Chrome 2 开始，使用了新一代垃圾回收机制，可确保内存高度可扩展而不会发生中断
IE	Chakra	注重于后台编译和高效的类型系统，速度比 IE8 快 10 倍
Firefox	JägerMonkey	从 3.5 版本优化而来，结合了快速解释和源自追踪树（trace tree）的本地编译
Opera	Carakan	它采用了基于寄存器的字节码和选择性本地编译的方式，声称效率比 10.50 版本提升了 75%
nohref	area	多余属性，被省略



创建 HTML5 文档

（视频讲解：1 小时 13 分钟）

毋庸置疑，HTML5 在音频、视频、动画、应用、页面效果和开发效率等方面给网页结构带来了巨大的变化，对传统网页设计风格及相关理念带来了冲击。为了增强 Web 应用的实用性，HTML5 扩展了很多新技术，同时对传统 HTML 文档进行了修改，使文档结构更加清晰明确，容易阅读，增加了很多新的结构元素，减少了复杂性，这样既方便了浏览者的访问，也提高了 Web 设计人员的开发速度。本章将详细介绍 HTML5 中新增的主体结构元素的定义、使用方法以及使用场合，掌握 HTML5 中应该怎样结合这些新增结构元素来合理编排页面总体布局，掌握什么是显式编排，什么是隐式编排，HTML5 分析器是按什么原则来分析页面结构的，以及怎样对这些新增元素使用 CSS 样式。

3.1　认识 HTML5 文档结构

Note

在没有接触 HTML5 文档之前，相信很多读者对于 XHTML 文档结构比较熟悉，由于 XHTML 文档是 HTML 向 XML 规范的过渡版本，其文档格式也基本按 XML 规范进行要求。

- ☑ 必须为文档定义命名空间，其值为 http://www.w3.org/1999/xhtml。
- ☑ MIME type 不能是 text/html，而是 text/xml、application/xml 或者 application/xml+html。
- ☑ 必须有根元素，根元素为<html>，即<html>的开始和结束标签不能省略。
- ☑ 所有元素只要有了开始标签，就不能没有结束标签，或者自闭合。
- ☑ 所有元素都得严格遵守大小写，元素名称必须为小写。

因此 XHTML 文档格式变得严格了很多，也因为是 XML，其可读性和规范性提高了不少。但最终，我们要在 HTML 的宽容性和 XML 的规范性之间找到最佳的平衡点，一味地追求极端始终是一个错误。

但是当问到一个 HTML5 文档必须有哪些内容时，恐怕很少有人能正确地做出回答。为了帮助读者更好地对 HTML5 的网页有一个简单的理解与认识，也为了让读者能够顺利读懂 HTML5 网页代码的准确意思，下面给出一个详细的、符合标准的 HTML5 文档结构代码，并进行详细注释。

```
<!DOCTYPE html>
<!-- 声明文档结构类型 -->
<html lang=zh-cn>
<!-- 声明文档文字区域-->
<head>
<!-- 文档的头部区域 -->
<meta charset=utf-8>
<!-- 文档的头部区域中元数据区的字符集定义，utf-8 表示国际通用的字符集编码格式 -->
<!--[if IE]><![endif]-->
<!-- 文档的头部区域的兼容性写法 -->
<title>文档标题</title>
<!-- 文档的头部区域的标题。title 内容对于 SEO 来说极其重要-->
<!--[if IE 9]><meta name=ie content=9><![endif]-->
<!-- 文档的头部区域的兼容性写法 -->
<!--[if IE 8]><meta name=ie content=8 ><![endif]-->
<!-- 文档的头部区域的兼容性写法 -->
<meta name=description content=文档描述信息>
<!-- 文档的头部区域元数据区关于文档描述的定义 -->
<meta name=author content=文档作者>
<!-- 文档的头部区域元数据区关于开发人员姓名的定义 -->
<meta name=copyright content=版权信息>
<!-- 文档的头部区域元数据区关于版权的定义 -->
<link rel=shortcut icon href=favicon.ico>
<!-- 文档的头部区域的兼容性写法 -->
<link rel=apple-touch-icon href=custom_icon.png>
<!-- 文档的头部区域的 apple 设备的图标的引用 -->
<meta name=viewport content=width=device-width, user-scalable=no >
<!-- 文档的头部区域对于不同接口设备的特殊声明。宽=设备宽，用户不能自行缩放 -->
<link rel=stylesheet href=main.css>
```

```
<!-- 文档的头部区域的样式文件引用 -->
<!--[if IE]><link rel=stylesheet href=win-ie-all.css><![endif]-->
<!-- 文档的头部区域的兼容性样式文件引用写法 --><!--[if IE 7]>
<link rel=stylesheet type=text/css href=win-ie7.css><![endif]-->
<!-- 文档的头部区域的 IE7 浏览器的兼容性写法 -->
<!--[if lt IE 8]><script src=http://ie7-js.googlecode.com/svn/version/2.0(beta3)/IE8. js></script><![endif]-->
<!-- 文档的头部区域的关于让 IE8 也兼容 HTML5 的 JavaScript 脚本-->
<script src=script.js></script>
<!-- 文档的头部区域的 JavaScript 脚本文件调用 -->
</head>
<body>
<header>HTML5 文档的头部区域</header>
<nav>HTML5 文档的导航区域</nav>
<section>HTML5 文档的主要内容区域
    <aside> HTML5 文档的主要内容区域的侧边导航或菜单区 </aside>
    <article> HTML5 文档的主要内容区域的内容区
        <section>以下是一个 section 和 article 的嵌套。
            <aside> </aside>
            <article>
                <header>
                HTML5 文档的嵌套区域，可以对某个 article 区域进行头部和脚部的定义。这样做，可以有非常清晰和严谨的文档目录结构关系。
                <footer>
            </article>
        </section>
    </article>
</section>
<footer>HTML5 文档的脚部区域</footer>
</body>
</html>
```

当然，并不是每个 HTML5 文档都需要包含上面每部分，一个最简单的 HTML5 文档需要的内容如下。

```
<!DOCTYPE html>
```

该句声明了文档类型，除了大小写可任意变化外，其他的任何内容都是不能变动的。那么究竟是怎样的规则，导致一个最简的源码文件必须有 doctype 声明呢？根据标准，一个 HTML 文档由如下内容组成（严格按照顺序）：

- ☑ 一个 BOM 标记，且这个 BOM 标记必须为 U+FEFF。
- ☑ n 个空格或注释。
- ☑ DOCTYPE 声明。
- ☑ n 个空格或注释。
- ☑ 一个 HTML 元素。
- ☑ n 个空格或注释。

这里存在着一些和 HTML4 的区别，一个 HTML4 的最简源码文件是这样的：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<title>标题</title>
```

两者的区别是显而易见的：

☑ HTML5 把 DOCTYPE 修改为更简单的<!DOCTYPE html>，这个已经众所周知了。在 HTML4 中多了一个<title>标签。这里的重点就是<title>标签了，关于这个标签，在 HTML4.01 标准中是这么说的：Every HTML document must have a TITLE element in the HEAD section。也就是说，HTML4 要求<title>标签是必须存在的。

☑ 在 HTML5 的标准中，又是这么说的：There must be no more than one title element per document。HTML5 中只设定了<title>标签数量的上限，却没有指明下限，也就是说，没有<title>的文档已经被视为一个合法的文档了。

对于 DOCTYPE，HTML4 中设定了 6 种 DOCTYPE，HTML5 中将 DOCTYPE 分为 3 种，这个在以后的章节中再详细说明。

再回过来看一下文档组成，除去 0~n 个空格或注释这些并没有多大意义的元素之外，组成的列表中还说明有一个 HTML 元素，但是最简的源码中却没有。这是因为在 HTML 的规范中，一直存在“隐式标签”这样的概念，关于隐式标签，大致可以这么解释：

一部分元素，当满足特定的前提条件时，其开始标签或结束标签可以在源码中省略。在这种情况下，被省略的标签称为“隐式标签”。

需要注意的是，此处的省略指的是在源码中省略，而在最终成型的 DOM 树中，这个标签是存在的，因此才称为隐式标签。因此上面最简的源码结构，在生成 DOM 树后，其真正的结构是这样的：

```
<!DOCTYPE html>
<html>
    <head></head>
    <body></body>
</html>
```

【示例】HTML5 文档扩展名为.htm 或者.html。现在主流浏览器都能够正确解析 HTML5 文档，如 Chrome、Firefox、Safari 和 IE9+。下面是一个简单的 HTML5 文档代码。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Hello HTML5</title>
</head>
<body>
</body>
</html>
```

HTML5 文档以<!DOCTYPE html>开头，这是一个文档类型声明，并且必须位于 HTML5 文档的第一行，它可以用来告诉浏览器或任何其他分析程序它们所查看的文件类型。

html 标签是 HTML5 文档的根标签，紧跟在<!DOCTYPE html>下面。html 标签支持 HTML5 全局属性和 manifest 属性。manifest 属性主要在创建 HTML5 离线应用的时候使用。

head 标签是所有头部元素的容器。位于 <head> 内部的元素可以包含脚本、样式表、元信息等。head 标签支持 HTML5 全局属性。

meta 标签位于文档的头部，不包含任何内容。标签的属性定义了与文档相关联的名称/值对。该标签提供页面的元信息（meta-information），如针对搜索引擎和更新频度的描述和关键词。

<meta charset="utf-8"/>定义了文档的字符编码是 utf-8。这里 charset 是 meta 标签的属性，而 utf-8

是该属性的值。HTML5 中的很多标签都有属性，从而扩展了标签的功能。

title 标签位于 head 标签内，定义了文档的标题。该标签定义了浏览器工具栏中的标题、提供页面被添加到收藏夹时的标题和显示在搜索引擎结果中的页面标题。所以该标签非常重要，在写 HTML5 文档的时候一定要记得写这个标签。title 标签支持 HTML5 全局属性。

Note

body 标签定义文档的主体和所有内容，如文本、超链接、图像、表格和列表等都包含在该标签中。

3.2 HTML5 元素分类

HTML5 新增了 27 个元素，废弃了 16 个元素，根据现有的标准规范，把 HTML5 的元素按优先等级定义为结构性元素、级块性元素、行内语义性元素和交互性元素 4 大类。

1. 结构性元素

结构性元素主要负责 Web 的上下文结构的定义，确保 HTML 文档的完整性，这类元素包括以下几个。

- ☑ section：在 Web 页面应用中，该元素也可以用于区域的章节表述。
- ☑ header：页面主体上的头部，注意区别于 head 元素。这里可以给初学者提供一个判断的小技巧：head 元素中的内容往往是不可见的，而 header 元素往往在一对 body 元素之中。
- ☑ footer：页面的底部（页脚）。通常会在这里标出网站的一些相关信息，例如，关于我们、法律申明、邮件信息和管理入口等。
- ☑ nav：是专门用于菜单导航、链接导航的元素，是 navigator 的缩写。
- ☑ article：用于表示一篇文章的主体内容，一般为文字集中显示的区域。

2. 级块性元素

级块性元素主要完成 Web 页面区域的划分，确保内容的有效分隔，这类元素包括以下几个。

- ☑ aside：用以表达注记、贴士、侧栏、摘要、插入的引用等作为补充主体的内容。从一个简单页面显示上看，就是侧边栏，可以在左边，也可以在右边。从一个页面的局部看，就是摘要。
- ☑ figure：是对多个元素进行组合并展示的元素，通常与 figcaption 联合使用。
- ☑ code：表示一段代码块。
- ☑ dialog：用于表达人与人之间的对话。该元素还包括 dt 和 dd 这两个组合元素，它们常常同时使用。dt 用于表示说话者，而 dd 则用来表示说话者说的内容。

3. 行内语义性元素

行内语义性元素主要完成 Web 页面具体内容的引用和表述，是丰富内容展示的基础，这类元素包括以下几个。

- ☑ meter：表示特定范围内的数值，可用于工资、数量、百分比等。
- ☑ time：表示时间值。
- ☑ progress：用来表示进度条，可通过对其 max、min、step 等属性进行控制，完成对进度的表示和监视。
- ☑ video：视频元素，用于支持和实现视频（含视频流）文件的直接播放，支持缓冲预载和多

种视频媒体格式，如 MPEG-4、OggV 和 WebM 等。

☑ audio：音频元素，用于支持和实现音频（音频流）文件的直接播放，支持缓冲预载和多种音频媒体格式。

Note

4. 交互性元素

交互性元素主要用于功能性的内容表达，会有一定的内容和数据的关联，是各种事件的基础，这类元素包括以下几个。

☑ details：用来表示一段具体的内容，但是内容默认可能不显示，通过某种手段（如单击）与 legend 交互才会显示出来。

☑ datagrid：用来控制客户端数据与显示，可以由动态脚本及时更新。

☑ menu：主要用于交互菜单（这是一个曾被废弃现在又被重新启用的元素）。

☑ command：用来处理命令按钮。

3.3　构建主体内容

在 HTML5 中，为了使文档的结构更加清晰明确，追加了几个与页眉、页脚、内容区块等文档结构相关联的结构元素。需要说明的是，本章所讲的内容区块是指将 HTML 页面按逻辑进行分割后的单位。例如，对于书籍来说，章、节都可以称为内容区块；对于博客网站来说，导航菜单、文章正文、文章的评论等每一个部分都可称为内容区块。接下来将详细讲解 HTML5 中在页面的主体结构方面新增加的结构元素。

3.3.1　标识文章

article 元素用来表示文档、页面中独立的、完整的、可以独自被外部引用的内容。它可以是一篇博客或报刊中的文章、一篇论坛帖子、一段用户评论或独立的插件等。除了内容部分，一个 article 元素通常有它自己的标题，一般放在一个 header 元素里面，有时还有自己的脚注。当 article 元素嵌套使用的时候，内部的 article 元素内容必须和外部 article 元素内容相关。article 元素支持 HTML5 全局属性。

【示例 1】本段代码演示了如何使用 article 元素设计网络新闻展示。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>新闻</title>
</head>
<body>
<article>
    <header>
        <h1>谷歌董事长施密特：每天把手机电脑关机 1 小时</h1>
        <time pubdate="pubdate">2012 年 05 月 21 日  09:04</time>
    </header>
    <p>新浪科技讯 北京时间 5 月 21 日早间消息，谷歌（微博）执行董事长埃里克·施密特（Eric Schmidt）周日在波士顿大学发表演讲时表示，大学生应当将目光从智能手机和电脑屏幕上移开。
```

```
</p>
    <footer>
        <p>http://www.sina.com.cn</p>
    </footer>
</article>
</body>
</html>
```

Note

这个示例是一篇讲述科技新闻的文章，在 header 元素中嵌入了文章的标题部分，在这部分中，文章的标题被镶嵌在 h1 元素中，文章的发表日期镶嵌在 time 元素中。在标题下部的 p 元素中，嵌入了一大段该博客文章的正文，在结尾处的 footer 元素中，嵌入了文章的著作权，作为脚注。整个示例的内容相对比较独立、完整，因此，对这部分内容使用了 article 元素。

article 元素是可以嵌套使用的，内层的内容在原则上需要与外层的内容相关联。例如，一篇科技新闻中，针对该新闻的相关评论就可以使用嵌套 article 元素的方式，用来呈现评论的 article 元素被包含在表示整体内容的 article 元素里面。

【示例 2】本示例是在上面代码的基础上演示如何实现 article 元素嵌套使用。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>新闻</title>
</head>
<body>
<article>
    <header>
        <h1>谷歌董事长施密特：每天把手机电脑关机 1 小时</h1>
        <time pubdate="pubdate">2012 年 05 月 21 日  09:04</time>
    </header>
    <p>新浪科技讯 北京时间 5 月 21 日早间消息，谷歌（微博）执行董事长埃里克·施密特（Eric Schmidt）周日在波士顿大学发表演讲时表示，大学生应当将目光从智能手机和电脑屏幕上移开。</p>
    <footer>
        <p>http://www.sina.com.cn</p>
    </footer>
    <section>
        <h2>评论</h2>
        <article>
            <header>
                <h3>张三</h3>
                <p>
                    <time pubdate datetime="2012-6-1 19:10-08:00"> 1 小时前 </time>
                </p>
            </header>
            <p>ok</p>
        </article>
        <article>
            <header>
                <h3>李四</h3>
                <p>
```

Note

```
                <time pubdate datetime="2012-6-2 19:10-08:00"> 1 小时前 </time>
            </p>
        </header>
        <p>well</p>
    </article>
</section>
</article>
</body>
</html>
```

示例 2 中的内容比示例 1 中的内容更加完整，它添加了评论内容。整个内容比较独立、完整，因此对其使用 article 元素。具体来说，示例内容又分为几部分，文章标题放在了 header 元素中，文章正文放在了 header 元素后面的 p 元素中，然后 section 元素把正文与评论部分进行了区分，在 section 元素中嵌入了评论的内容，评论中每一个人的评论相对来说又是比较独立、完整的，因此对它们都使用一个 article 元素，在评论的 article 元素中，又可以分为标题与评论内容部分，分别放在 header 元素与 p 元素中。

另外，article 元素也可以用来表示插件，它的作用是使插件看起来好像内嵌在页面中一样。下面代码使用 article 元素表示插件使用。

```
<article>
    <h1>使用插件</h1>
    <object>
        <param name="allowFullScreen" value="true">
        <embed src="#" width="600" height="395"></embed>
    </object>
</article>
```

3.3.2　给内容分段

section 元素用于对网站或应用程序中页面上的内容进行分区。一个 section 元素通常由内容及其标题组成。div 元素也可以用来对页面进行分区，但 section 元素并非一个普通的容器元素，当一个容器需要被直接定义样式或通过脚本定义行为时，推荐使用 div，而非 section 元素。

提示： div 元素关注结构的独立性，而 section 元素关注内容的独立性，section 元素包含的内容可以单独存储到数据库中或输出到 Word 文档中。

【示例 1】本示例使用 section 元素把新歌排行榜的内容进行单独分隔，在 HTML5 之前，通常使用 div 元素来分隔该块内容。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title></title>
</head>
<body>
<section>
    <h1>新歌 TOP10</h1>
    <ol>
```

Note

```
            <li>心术 张宇</li>
            <li>最亲爱的你 范玮琪</li>
            <li>珍惜 李宇春</li>
            <li>思凡 林宥嘉 </li>
            <li>错过 王铮亮</li>
            <li>好难得 丁当</li>
            <li>抱着你的感... 费玉清</li>
            <li>好想你也在 郁可唯</li>
            <li>不难 徐佳莹 </li>
            <li>我不能哭 莫艳琳</li>
        </ol>
    </section>
    </body>
    </html>
```

article 元素与 section 元素都是 HTML5 新增的元素，它们的功能与 div 类似，都是用来区分不同区域，它们的使用方法也相似，因此很多初学者会将其混用。HTML5 之所以新增这两种元素，就是为了更好地描述文档的内容，所以它们之间肯定是有区别的。

article 元素代表文档、页面或者应用程序中独立完整的可以被外部引用的内容。例如，博客中的一篇文章，论坛中的一个帖子或者一段浏览者的评论等。因为 article 元素是一段独立的内容，所以 article 元素通常包含头部（header 元素）和底部（footer 元素）。

section 元素用于对网站或者应用程序中页面上的内容进行分块。一个 section 元素通常由内容以及标题组成。

section 元素需要包含一个<hn>标题元素，一般不用包含头部（header 元素）或者底部（footer 元素）。通常用 section 元素为那些有标题的内容进行分段。

section 元素的作用是对页面上的内容分块处理，如对文章分段等，相邻的 section 元素的内容，应当是相关的，而不是像 article 那样独立。

```
<article>
    <header>
        <h1>潜行者 m 的个人介绍</h1>
    </header>
    <p>潜行者 m 是一个中国男人，是一个帅哥……</p>
    <section>
        <h2>评论</h2>
        <article>
            <h3>评论者：潜行者 n</h3>
            <p>确实，m 同学真的很帅</p>
        </article>
        <article>
            <h3>评论者：潜行者 a</h3>
            <p>m 今天吃药了没？ </p>
        </article>
    </section>
</article>
```

在上面示例中，读者能够观察到 article 元素与 section 元素的区别。事实上 article 元素可以看做是特殊的 section 元素。article 元素更强调独立性、完整性，section 更强调相关性。

既然 article、section 是用来划分区域的，又是 HTML5 的新元素，那么是否可以用 article 和 section 取代 div 来布局网页呢？答案是否定的，div 的用处就是用来布局网页，划分大的区域，HTML4 中只有 div、span 用来划分区域，所以我们习惯性地把 div 当成了一个容器。而 HTML5 改变了这种用法，它让 div 的工作更纯正。div 就是用来布局大块，在不同的内容块中，我们按照需求添加 article、section 等内容块，并且显示其中的内容，这样才是合理地使用这些元素。

Note

因此，在使用 section 元素时应该注意以下几个问题。

☑ 不要将 section 元素当做设置样式的页面容器，对于此类操作应该使用 div 元素实现。

☑ 如果 article 元素、aside 元素或 nav 元素更符合使用条件，不要使用 section 元素。

☑ 不要为没有标题的内容区块使用 section 元素。

通常不推荐为那些没有标题的内容使用 section 元素，可以使用 HTML5 轮廓工具（http://gsnedders.html5.org/outliner/）来检查页面中是否有没标题的 section，如果使用该工具进行检查后，发现某个 section 的说明中有“untitiled section”（没有标题的 section）文字，这个 section 就有可能使用不当，但是 nav 元素和 aside 元素没有标题是合理的。

【示例 2】section 元素的作用是对页面上的内容进行分块，类似对文章进行分段，与具有完整、独立的内容模块 article 元素不同。下面我们来看 article 元素与 section 元素混合使用的示例。

```
<article>
    <h1>W3C</h1>
    <p>万维网联盟（World Wide Web Consortium，W3C），又称 W3C 理事会。1994 年 10 月在麻省理工学院计算机科学实验室成立。建立者是万维网的发明者蒂姆&middot;伯纳斯-李。</p>
    <section>
        <h2>CSS</h2>
        <p>全称 Cascading Style Sheet，级联样式表，通常又称为“风格样式表（Style Sheet）”，它是用来进行网页风格设计的。</p>
    </section>
    <section>
        <h2>HTML</h2>
        <p>全称 Hypertext Markup Language，超文本标记语言，用于描述网页文档的一种标记语言。</p>
    </section>
</article>
```

在上面代码中，首先可以看到整个版块是一段独立、完整的内容，因此使用 article 元素。该内容是一篇关于 W3C 的简介，该文章分为 3 段，每一段都有一个独立的标题，因此使用了两个 section 元素。

注意：对文章分段的工作是使用 section 元素完成的。为什么没有对第一段使用 section 元素，其实是可以使用的，但是由于其结构比较清晰，分析器可以识别第一段内容在一个 section 元素里，所以也可以将第一个 section 元素省略，但是如果第一个 section 元素里还要包含子 section 元素或子 article 元素，那么就必须写明第一个 section 元素。

接着来看一个包含 article 元素的 section 元素示例。

```
<section>
    <h1>W3C</h1>
    <article>
        <h2>CSS</h2>
        <p>全称 Cascading Style Sheet，级联样式表，通常又称为“风格样式表（Style Sheet）”，它是用来进行网页风格设计的。</p>
```

```
    </article>
        <h2>HTML</h2>
        <p>全称 Hypertext Markup Language，超文本标记语言，用于描述网页文档的一种标记语言。</p>
</section>
```

这个示例比第一个示例复杂了一些。首先，它是一篇文章中的一段，因此没有使用 article 元素。但是，在这一段中有几块独立的内容，所以嵌入了几个独立的 article 元素。

在 HTML5 中，article 元素可以看成是一种特殊种类的 section 元素，它比 section 元素更强调独立性。即 section 元素强调分段或分块，而 article 强调独立性。具体来说，如果一块内容相对来说比较独立、完整的时候，应该使用 article 元素，但是如果想将一块内容分成几段的时候，应该使用 section 元素。另外，在 HTML5 中，div 元素变成了一种容器，当使用 CSS 样式的时候，可以对这个容器进行一个总体的 CSS 样式的套用。

在 HTML5 中，可以将所有页面的从属部分，如导航条、菜单、版权说明等，包含在一个统一的页面中，以便统一使用 CSS 样式来进行装饰。

3.3.3 设计导航信息

nav 元素是一个可以用作页面导航的链接组，其中的导航元素链接到其他页面或当前页面的其他部分。并不是所有的链接组都要被放进 nav 元素，只需要将主要的、基本的链接组放进 nav 元素即可。

例如，在页脚中通常会有一组链接，包括服务条款、首页和版权声明等，这时使用 footer 元素最恰当。一个页面中可以拥有多个 nav 元素，作为页面整体或不同部分的导航。具体来说，nav 元素可以用于以下场合：

- ☑ 传统导航条。常规网站都设置有不同层级的导航条，其作用是将当前画面跳转到网站的其他主要页面上去。
- ☑ 侧边栏导航。现在主流博客网站及商品网站上都有侧边栏导航，其作用是将页面从当前文章或当前商品跳转到其他文章或其他商品页面上去。
- ☑ 页内导航。页内导航的作用是在本页面几个主要的组成部分之间进行跳转。
- ☑ 翻页操作。翻页操作是指在多个页面的前后页或博客网站的前后篇文章滚动。

【示例 1】在 HTML5 中，只要是导航性质的链接，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。

```
<!DOCTYPE html>
<html>
<body>
<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>
</body>
</html>
```

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 3 个用于导航的超链接，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。

【示例 2】在本示例中，页面由几部分组成，每个部分都带有链接，但只将最主要的链接放入了 nav 元素中。

Note

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
</head>
<body>
<h1>技术资料</h1>
<nav>
    <ul>
        <li><a href="/">主页</a></li>
        <li><a href="/blog">博客</a></li>
    </ul>
</nav>
<article>
    <header>
        <h1>HTML5+CSS3</h1>
        <nav>
            <ul>
                <li><a href="#HTML5">HTML5</a></li>
                <li><a href="#CSS3">CSS3</a></li>
            </ul>
        </nav>
    </header>
    <section id="HTML5">
        <h1>HTML5</h1>
        <p>HTML5 特性说明</p>
    </section>
    <section id="CSS3">
        <h1>CSS3</h1>
        <p>CSS3 特性说明</p>
    </section>
    <footer>
        <p> <a href="?edit">编辑</a> | <a href="?delete">删除</a> | <a href="?add">添加</a> </p>
    </footer>
</article>
<footer>
    <p><small>版权信息</small></p>
</footer>
</body>
</html>
```

在这个例子中，第一个 nav 元素用于页面导航，将页面跳转到其他页面上去，如跳转到网站主页或博客页面；第二个 nav 元素放置在 article 元素中，表示在文章中进行导航。除此之外，nav 元素也可以用于其他所有你认为是重要的、基本的导航链接组中。

提示：在 HTML5 中不要用 menu 元素代替 nav 元素。很多用户喜欢用 menu 元素进行导航，menu 元素主要用在一系列交互命令的菜单上的，如使用在 Web 应用程序中的。

3.3.4 设计辅助信息

Note

aside 元素用来表示当前页面或文章的附属信息部分，它可以包含与当前页面或主要内容相关的引用、侧边栏、广告、导航条以及其他类似的有别于主要内容的部分。aside 元素主要有以下两种使用方法。

☑ 作为主要内容的附属信息部分，包含在 article 元素中，其中的内容可以是与当前文章有关的参考资料、名词解释等。

【示例 1】本代码使用 aside 元素解释在 HTML5 发展过程中出现的两个名词。这是一篇文章，网页的标题放在了 header 元素中，在 header 元素的后面将所有关于文章的部分放在了一个 article 元素中，将文章的正文部分放在了一个 p 元素中，但是该文章还有一个名词解释的附属部分，用来解释该文章中的一些名词，因此，在 p 元素的下部又放置了一个 aside 元素，用来存放名词解释部分的内容。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<header>
    <h1>HTML5</h1>
</header>
<article>
    <h1>HTML5 历史</h1>
    <p>HTML5 草案的前身名为 Web Applications 1.0，于 2004 年被 WHATWG 提出，于 2007 年被 W3C 接纳，并成立了新的 HTML 工作团队。HTML5 的第一份正式草案已于 2008 年 1 月 22 日公布。HTML5 仍处于完善之中。然而，大部分现代浏览器已经具备了某些 HTML5 支持。</p>
    <aside>
        <h1>名词解释</h1>
        <dl>
            <dt>WHATWG</dt>
            <dd>Web Hypertext Application Technology Working Group，HTML 工作开发组的简称，目前与W3C 组织同时研发 HTML5。</dd>
        </dl>
        <dl>
            <dt>W3C</dt>
            <dd>World Wide Web Consortium，万维网联盟，万维网联盟是国际著名的标准化组织。1994年成立后，至今已发布近百项相关万维网的标准，对万维网发展做出了杰出的贡献。</dd>
        </dl>
    </aside>
</article>
</body>
```

因为这个 aside 元素被放置在一个 article 元素内部，因此引擎将该 aside 元素的内容理解成是和 article 元素的内容相关联的。

☑ 作为页面或站点全局的附属信息部分，在 article 元素之外使用。最典型的形式是侧边栏，其中的内容可以是友情链接，博客中其他文章列表、广告单元等。

Note

【示例 2】本代码使用 aside 元素为个人网页添加一个友情链接版块。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<aside>
    <nav>
        <h2>友情链接</h2>
        <ul>
            <li> <a href="#">网站 1</a></li>
            <li> <a href="#">网站 2</a></li>
            <li> <a href="#">网站 3</a></li>
        </ul>
    </nav>
</aside>
</body>
```

友情链接在博客网站中比较典型，一般放在左右两侧的边栏中，因此可以使用 aside 元素来实现，但是该侧边栏又是具有导航作用的，因此嵌套了一个 nav 元素，该侧边栏的标题是“友情链接”，放在了 h2 元素中，在标题之后使用了一个 ul 列表，用来存放具体的导航链接。

3.3.5　设计微格式

HTML5 微格式的提出主要目的是简化 Web 开发的数据提取。曾从网页中提取过数据的 Web 开发人员都知道，现有的 HTML 结构除了告诉浏览器这些信息在哪里之外，几乎不能再提供任何有意义的信息。开发人员需要了解与数据本身有关的信息，这些信息能帮助程序员了解这些数据的真正含义。HTML5 中所谓的微格式（Microformat）引入了一种新的机制，它在 HTML 中新增了一些专门的标签，可以帮助程序员分析标签之中的数据的真实含义。

没有人能够预测微格式到底将带给网络多少改变，但很容易看出，这种新的机制将给程序员带来很大方便，帮助程序员开发出更有效率的 Web 应用。例如，如果有一个好的、标准的方式来表示日期和时间，那么程序员在为网站开发与时间有关的 Web 程序时，就无需另外编写专门的代码来分析或者猜测别人可能用的什么时间格式。这样，日历、时间表、日程安排等需要从多个数据源收集时间信息的应用也就变成非常简单的工作了。

微格式是一种利用 HTML 的 class 属性来对网页添加附加信息的方法，附加信息如新闻事件发生的日期和时间、个人电话号码、企业邮箱等。微格式并不是在 HTML5 之后才有的，在 HTML5 之前它就和 HTML 结合使用了，但是在使用过程中发现在日期和时间的机器编码上出现了一些问题，编码过程中会产生一些歧义。HTML5 增加了一种新的元素来无歧义地、明确地对机器的日期和时间进行编码，并且以让人易读的方式来展现它。这个元素就是 time 元素。

time 元素代表 24 小时中的某个时刻或某个日期，表示时刻时允许带时差。它可以定义很多格式的日期和时间，如下所示。

```
<time datetime="2012-11-13">2012 年 11 月 13 日</time>
<time datetime="2012-11-13">11 月 13 日</time>
```

```
<time datetime="2012-11-13">我的生日</time>
<time datetime="2012-11-13T20:00">我生日的晚上 8 点</time>
<time datetime="2012-11-13T20:00Z">我生日的晚上 8 点</time>
<time datetime="2012-11-13T20:00+09:00">我生日的晚上 8 点的美国时间</time>
```

Note

编码时引擎读到的部分在 datetime 属性里，而元素的开始标记与结束标记中间的部分是显示在网页上的。datetime 属性中日期与时间之间要用“T”字母分隔，“T”表示时间。

注意倒数第 2 行，时间加上“Z”字母表示给机器编码时使用 UTC 标准时间，倒数第一行则加上了时差，表示向机器编码另一地区时间，如果是编码本地时间，则不需要添加时差。

3.3.6 添加发布日期

pubdate 属性是一个可选的布尔值属性，可用在 article 元素中的 time 元素上，意思是 time 元素代表了文章（artilce 元素的内容）或整个网页的发布日期。

【示例】本示例使用 pubdate 属性为文档添加引擎检索的发布日期。

```
<article>
    <header>
        <h1>谷歌董事长施密特：每天把手机电脑关机 1 小时</h1>
        <p>发布日期<time datetime="2012-5-22" pubdate>2012 年 5 月 22 日</time></p>
    </header>
    <p>新浪科技讯 北京时间 5 月 21 日早间消息，谷歌（微博）执行董事长埃里克·施密特（Eric Schmidt）周日在波士顿大学发表演讲时表示，大学生应当将目光从智能手机和电脑屏幕上移开。
    </p>
    <footer>
        <p>http://www.sina.com.cn</p>
    </footer>
</article>
```

由于 time 元素不仅仅表示发布时间，而且还可以表示其他用途的时间，如通知、约会等。为了避免引擎误解发布日期，使用 pubdate 属性可以显式地告诉引擎文章中哪个是真正的发布时间。

```
<article>
    <header>
        <h1>谷歌董事长施密特：每天把手机电脑关机 1 小时</h1>
        <p>发布日期<time datetime="2012-5-22" pubdate>2012 年 5 月 22 日</time></p>
        <p>关于<time datetime=2012-5-23>5 月 23 日</time>更正通知</p>
    </header>
    <p>新浪科技讯 北京时间 5 月 21 日早间消息，谷歌（微博）执行董事长埃里克·施密特（Eric Schmidt）周日在波士顿大学发表演讲时表示，大学生应当将目光从智能手机和电脑屏幕上移开。
    </p>
    <footer>
        <p>http://www.sina.com.cn</p>
    </footer>
</article>
```

在这个例子中，有两个 time 元素，分别定义了两个日期：更正日期和发布日期。由于都使用了 time 元素，所以需要使用 pubdate 属性表明哪个 time 元素代表了新闻的发布日期。

3.4　添加语义模块

Note

除了以上几个主要的结构元素之外，HTML5 内还增加了一些表示逻辑结构或附加信息的非主体结构元素。

3.4.1　添加标题块

header 元素是一种具有引导和导航作用的结构元素，通常用来放置整个页面或页面内的一个内容区块的标题，但也可以包含其他内容，如数据表格、搜索表单或相关的 logo 图片，因此整个页面的标题应该放在页面的开头。

【示例 1】在一个网页内可以多次使用 header 元素，本示例显示为每个内容区块添加一个 header 元素。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<header>
    <h1>网页标题</h1>
</header>
<article>
    <header>
        <h1>文章标题</h1>
    </header>
    <p>文章正文</p>
</article>
</body>
```

在 HTML5 中，header 元素通常包含 h1~h6 元素，也可以包含 hgroup、table、form、nav 等元素，只要应该显示在头部区域的语义标签，都可以包含在 header 元素中。

【示例 2】下面是个人博客首页的头部区域代码示例，整个头部内容都放在 header 元素中。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<header>
    <hgroup>
        <h1>我的博客</h1>
        <a href="#">[URL]</a> <a href="#">[订阅]</a> <a href="#">[手机订阅]</a> </hgroup>
    <nav>
        <ul>
            <li>首页</li>
```

```
            <li><a href="#">目录</a></li>
            <li><a href="#">社区</a></li>
            <li><a href="#">微博我</a></li>
        </ul>
    </nav>
</header>
</body>
```

Note

3.4.2 给标题分组

hgroup 元素可以为标题或者子标题进行分组，通常它与 h1～h6 元素组合使用，一个内容块中的标题及其子标题可以通过 hrgoup 元素组成一组。但是，如果文章只有一个主标题，则不需要 hgroup 元素。

【示例】本示例显示如何使用 hgroup 元素把主标题、副标题和标题说明进行分组，以便让搜索引擎更容易识别标题块。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<article>
    <header>
        <hgroup>
            <h1>主标题</h1>
            <h2>副标题</h2>
            <h3>标题说明</h3>
        </hgroup>
        <p>
            <time datetime="2012-6-20">发布时间：2012 年 6 月 20 日</time>
        </p>
    </header>
    <p>新闻正文</p>
</article>
</body>
```

3.4.3 添加脚注块

footer 元素可以作为内容块的注脚，如在父级内容块中添加注释，或者在网页中添加版权信息等。脚注信息有很多种形式，如作者、相关阅读链接及版权信息等。

【示例 1】在 HTML5 之前，要描述注脚信息，一般使用 标签定义包含框。自从 HTML5 新增了 footer 元素，这种方式将不再使用，而是使用更加语义化的 footer 元素来替代。在下面的代码中使用了 footer 元素为页面添加版权信息栏。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
```

Note

```
<body>
<article>
    <header>
        <hgroup>
            <h1>主标题</h1>
            <h2>副标题</h2>
            <h3>标题说明</h3>
        </hgroup>
        <p>
            <time datetime="2012-03-20">发布时间：2012 年 10 月 29 日</time>
        </p>
    </header>
    <p>新闻正文</p>
</article>
<footer>
    <ul>
        <li>关于</li>
        <li>导航</li>
        <li>联系</li>
    </ul>
</footer>
</body>
```

【示例 2】与 header 元素一样，页面中也可以重复使用 footer 元素。同时，可以为 article 元素或 section 元素添加 footer 元素。在下面代码中分别在 article、section 和 body 元素中添加 footer 元素。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<header>
    <h1>网页标题</h1>
</header>
<article> 文章内容
    <h2>文章标题</h2>
    <p>正文</p>
    <footer>注释</footer>
</article>
<section>
    <h2>段落标题</h2>
    <p>正文</p>
    <footer>段落标记</footer>
</section>
<footer>网页版权信息</footer>
</body>
```

3.4.4　添加联系信息

address 元素用来在文档中定义联系信息，包括文档作者或文档编辑者名称、电子邮箱、真实地

址、电话号码等。

【示例 1】address 元素的用途不仅仅用来描述电子邮箱或真实地址，还可以描述与文档相关的联系人的所有联系信息。下面代码展示了博客侧边栏中的一些技术参考网站的网址链接。

Note

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<address>
    <a href="http://www.w3.org/">W3C</a>
    <a href="http://www.whatwg.org/">WHATWG</a>
    <a href="http://www.mhtml5.com/">HTML5 研究小组</a>
</address>
</body>
```

【示例 2】也可以把 footer 元素、time 元素与 address 元素结合起来使用，以实现设计一个比较复杂的版块结构。

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body>
<footer>
    <section>
        <address>
        <a title="作者：html5" href="http://www.whatwg.org/">HTML5+CSS3 技术趋势</a>
        </address>
        <p> 发布于：
            <time datetime="2012-6-1">2012 年 6 月 1 日</time>
        </p>
    </section>
</footer>
</body>
```

在这个示例中，把博客文章的作者、博客的主页链接作为作者信息放在了 address 元素中，把文章发表日期放在了 time 元素中，把这个 address 元素与 time 元素中的总体内容作为脚注信息放在了 footer 元素中。

3.5 综合实战：使用 HTML5 设计博客主页

通过使用新的结构元素，HTML5 的文档结构比大量使用 div 元素的 HTML4 的文档结构清晰、明确了很多。如果再规划好文档结构的大纲，就可以创建出对于阅读者和屏幕阅读程序来说，都很清晰易读的文档结构。下面通过对一个博客主页的设计介绍如何综合运用 HTML5 结构元素设计一个

页面。

3.5.1　设计大纲

Note

所谓大纲，简单来说就是文档中各内容区块的结构编排。内容区块可以使用标题元素（h1～h6）来展示各级内容区块的标题。综合运用各级内容区块的标题创建好文档的目录后，该目录就是一个大纲了。

关于内容区块的编排，可以分为显示编排和隐式编排两种方式。

1. 显式编排

显式编排是指明确使用 section 等元素创建文档结构，在每个内容区块内使用标题（h1～h6、hgroup 等）。例如：

```
<section id="work" class="clearfix">
    <header>
        <h2>My Work</h2>
    </header>
    <ul class="projects list"></ul>
</section>
```

2. 隐式编排

隐式编排是指不明确使用 section 等元素，而是根据页面中所书写的各级标题（h1～h6、hgroup）等把内容区块自动创建出来。因为 HTML5 引擎只要看到书写了某个级别的标题，就会判断存在相对应的内容区块。例如：

```
<div id="work" class="clearfix">
    <h2>My Work</h2>
    <ul class="projects list"></ul>
</div>
```

将这两种编排方式进行对比，很明显，显式编排更加清晰、易读。

不同的标题有不同的级别，h1 的级别最高，h6 的级别最低。隐式编排时按如下规则自动生成内容区块。

☑　如果新出现的标题比上一个标题级别低，生成下级内容区块。

☑　如果新出现的标题比上一个标题级别高或级别相等，生成新的内容区块。

例如，针对下面内容块，第二个标题比第一个标题级别高。

```
<section>
<h2> </h2>
<p> </p>
<h1> </h1>
<p> </p>
</section>
```

因此，可以把它改成显式编排。

```
<section>
    <h2> </h2>
    <p> </p>
<section>
```

```
</section>
    <h1> </h1>
    <p> </p>
</section>
```

Note

因为隐式编排容易让自动生成的整个文档结构与想要的文档结构不一样，而且也容易引起文档结构的混乱，所以尽量使用显式编排。

另外，不同的内容区块可以使用相同级别的标题。例如，父内容区块与子内容区块可以使用相同级别的标题 h1。这样每个级别的标题都可以单独设计，如果既需要整个网页的标题，又需要文章的标题，下面这样处理就符合语义结构化设计原则。

```
<h1> </h1>
<article>
    <header>
        <hgroup>
            <h1> </h1>
            <h2> </h2>
        </hgroup>
        <p> </p>
    </header>
</article>
```

基于以上设计原则，下面就来设计本案例的博客主页基本框架结构。在本示例中，具备了一个标准博客网页所需具备的基本要素。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title></title>
<link href="css/reset.css" rel="stylesheet" media="screen">
<link href="css/style.css" rel="stylesheet" media="screen">
<link rel="stylesheet" type="text/css" href="fancybox/jquery.fancybox-1.3.4.css" media="screen" />
</head>
<body>
<div id="page">
    <aside id="sidebar">
        <nav>
            <ul>
                <li class="active" id="nav-1"><a href="#home">Home</a></li>
                <li id="nav-2"><a href="#work">Work</a></li>
                <li id="nav-3"><a href="#about">About</a></li>
                <li id="nav-4"><a href="#contact">Contact</a></li>
            </ul>
            <div class="bg_bottom"></div>
        </nav>
    </aside>
    <div id="main-content">
        <section id="top"></section>
        <section id="home">
            <div id="loader" class="loader"></div>
```

Note

```
            <div id="ps_container" class="ps_container"></div>
            <header class="divider intro-text">
                <h2>I Make Beautiful Websites </h2>
            </header>
            <div class="recent-work columns">
                <h3>My Recent Work</h3>
                <div class="two-column">
                    <figure></figure>
                </div>
                <div class="two-column last">
                    <figure></figure>
                </div>
            </div>
        </section>
        <section id="work" class="clearfix">
            <header>
                <h2>My Work</h2>
            </header>
            <ul class="projects list"></ul>
        </section>
        <section id="about" class="clearfix">
            <header>
                <h2>Who is this Guy?</h2>
            </header>
            <figure class="marginRight"><img src="images/me.gif" alt="Image" /></figure>
            <h3>Nerdy Skills</h3>
            <ul class="skills"></ul>
        </section>
        <section id="contact" class="clearfix">
            <header>
                <h2>Get in touch</h2>
            </header>
            <form action="#" method="post"></form>
            <div class="social_wrapper">
                <h3>Where to find me?</h3>
                <ul class="social"> </ul>
            </div>
            <div class="copyright">
                <p></p>
            </div>
        </section>
    </div>
</div>
</body>
</html>
```

在上面代码中使用了嵌套 article 元素的方式，将关于评论的 article 元素嵌套在了主 article 元素中，在 HTML5 中推荐使用这种方式。最后，在浏览器中预览，显示效果如图 3.1 所示。

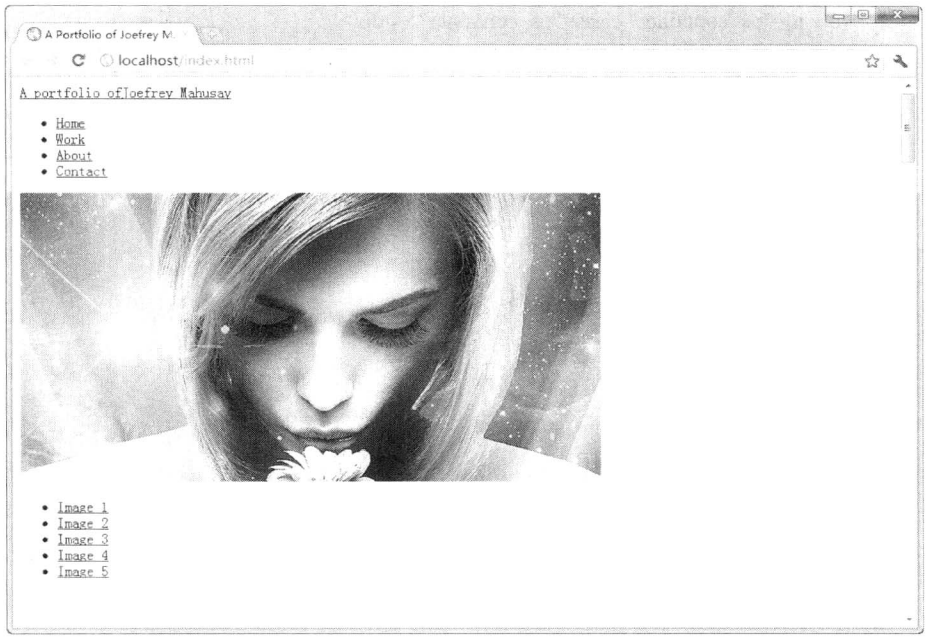


图 3.1　设计的 HTML5 博客主页结构

3.5.2　设计样式

因为很多浏览器尚未对 HTML5 中新增的结构元素提供支持，因此我们无法知道客户端使用的浏览器是否支持这些元素，所以需要使用 CSS 追加如下声明，目的是通知浏览器页面中使用的 HTML5 中新增元素都是以块方式显示的，代码如下所示。

```
//追加 block 声明
html, body, div, span, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre,
abbr, address, cite, code, del, dfn, em, img, ins, kbd, q, samp,
small, strong, sub, sup, var, b, i, dl, dt, dd, ol, ul, li,
fieldset, form, label, legend, table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, figcaption, figure,  footer, header, hgroup,
menu, nav, section, summary, time, mark, audio, video {
   margin:0;
   padding:0;
   border:0;
   outline:0;
   font-size:100%;
   vertical-align:baseline;
   background:transparent;
}
article, aside, details, figcaption, figure, footer, header, hgroup, menu, nav, section {
      display:block;
}
```

另外，IE8 及之前的浏览器是不支持用 CSS 的方法来使用这些尚未支持的结构元素的，为了在 IE 浏览器中也能正常使用这些结构元素，需要使用 JavaScript 脚本，如下所示。

Note

```
//在脚本中创建元素
<script>
document.createElement("header");
document.createElement("nav");
document.createElement("article");
document.createElement("footer");
</script>
<style>
//正常使用样式
nav{float;left;;}
article{float:right;;}
</style>
```

尽管这段 JavaScript 脚本在其他浏览器中是不需要的，但它不会对这些浏览器造成不良影响。另外，到了 IE9 之后，这段脚本就不需要了。

读者也可以导入外部 IE 兼容解析方案，该方案通过 html5.js 实现，导入代码如下，该代码只在低于 IE9 版本的浏览器中有效。

```
<!--[if lt IE 9]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

然后规范化标签的默认样式，这些默认样式在不同浏览器中有不同的规定，为了方便统一管理，不妨通过显式声明的方式实现一致效果。

```
nav ul { list-style:none; }
blockquote, q { quotes:none; }
blockquote:before, blockquote:after,
q:before, q:after { content:''; content:none; }
a { margin:0; padding:0; font-size:100%; vertical-align:baseline; background:transparent; }
ins { background-color:#ff9; color:#000; text-decoration:none; }
mark { background-color:#ff9; color:#000; font-style:italic; font-weight:bold; }
del { text-decoration: line-through; }
abbr[title], dfn[title] { border-bottom:1px dotted; cursor:help; }
table { border-collapse:collapse; border-spacing:0; }
hr { display:block; height:1px; border:0; border-top:1px solid #ccc; margin:1em 0; padding:0; }
input, select { vertical-align:middle; }
```

一个网页中可能有多个独立的 article 元素，每一个 article 元素都允许有自己的标题与脚注等从属元素，并允许对自己的从属元素单独使用样式。譬如一个网页中的样式可能如下所示：

```
/*----aside----*/
aside{
    float:left;
    width:248px;
    position:fixed;
    padding-top:10px;
}

/*----nav----*/
nav{
```

```
    text-align:center;
    margin-bottom:20px;
}
```

Note

最后，在浏览器中预览，显示效果如图 3.2 所示。

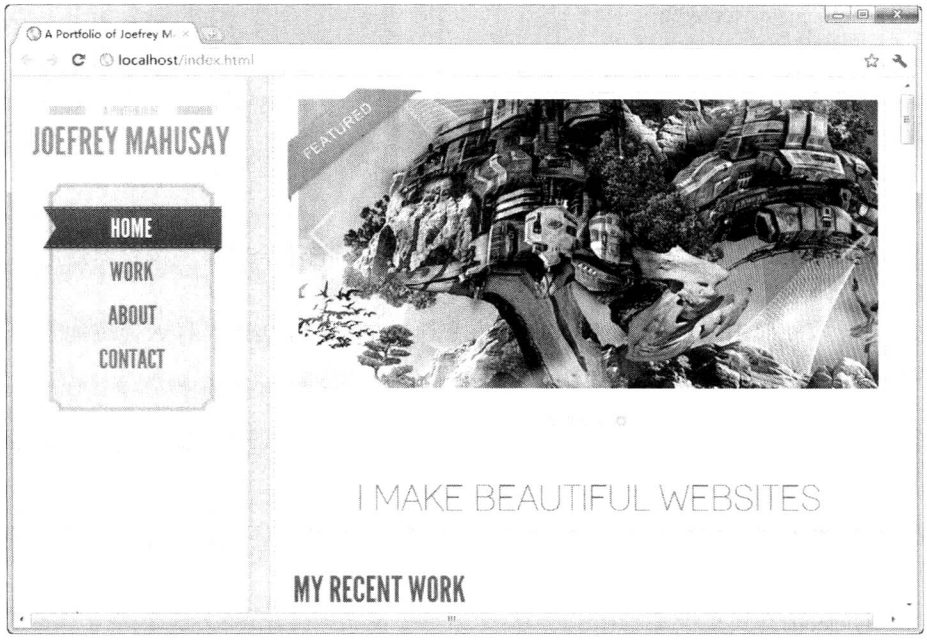


图 3.2　设计的 HTML5 博客主页效果

实战 HTML5 表单

（视频讲解：1 小时 08 分钟）

表单在网页设计中的作用非常重要，网页如果要实现采集数据的功能，表单是不可或缺的。通过使用表单，可以采集访问者的信息，如姓名、性别、年龄、职业、联系方式等，也可以制作调查表、留言簿等，从而获取重要的数据以满足各种需要。访问者与后台的交互是通过单击表单中的按钮来实现的，而与前台的交互则通过输入数据或选择选项来实现。如果以前经常使用表单，相信读者已经对这些表单基本功能的实现了如指掌。

HTML5 中增加了表单方面的诸多功能，包括增加 input 输入类型、表单元素、form 属性和 input 属性等。使用这些新的元素，前端设计人员可以更加省力和高效地制作出标准的 Web 表单。

4.1 新增的 input 输入类型

Note

HTML5 出现之前，HTML 表单仅支持少数的 input 输入类型，如表 4.1 所示。

表 4.1 HTML5 之前版本支持的少数 input 输入类型

输 入 类 型	HTML 代码	功能与说明
文本域	<input type="text">	定义单行输入字段，用于在表单中输入字母、数字等内容。默认宽度为 20 个字符
单选按钮	<input type="radio">	定义单选按钮，用于从若干给定的选项中选取其一，常与其他类型的输入框构成一组使用
复选框	<input type="checkbox">	定义复选框，用于从若干给定的选择中选取一项或若干选项
下拉列表	<select><option>	定义下拉列表，提供多个可选的项，select 元素必须与 option 元素配合使用
密码域	<input type="password">	定义密码字段，用于输入密码，输入的内容会以“点”或星号的形式出现，即被“掩码”
提交按钮	<input type="submit">	定义提交按钮，用于将表单数据发送到服务器
可单击按钮	<input type="button">	定义普通可单击按钮，多数情况下，用于通过 JavaScript 启动脚本
图像按钮	<input type="image">	定义图像形式的提交按钮。用户可以通过选择不同的图像来自定义这种按钮的样式
隐藏域	<input type="hidden">	定义隐藏的输入字段
重置按钮	<input type="reset">	定义重置按钮。用户可以通过单击重置按钮以清除表单中的所有数据
文件域	<input type="file">	定义输入字段和“浏览”按钮，用于上传文件

在 HTML5 中，增加了多个新的表单 input 输入类型，通过使用这些新增的元素，可以实现更好的输入控制和验证。下面通过示例来一一介绍这些新的 input 输入类型。

4.1.1 email 类型的应用

email 类型的 input 元素是一种专门用于输入 E-mail 地址的文本输入框，在提交表单的时候，会自动验证 email 输入框的值。如果不是一个有效的 E-mail 地址，则该输入框不允许提交该表单。在以前版本的 Web 表单中，采用的是<input type="text">这种纯文本输入框来输入 E-mail 地址。从用户的角度来说，很难看出这种输入框有什么变化，因为多数新版的支持 HTML5 的浏览器只是简单地将 E-mail 地址输入框显示为与纯文本框完全相同。

email 类型的 input 元素用法如下：

```
<input type="email" name="user_email"/>
```

【示例】下面是 email 类型的一个应用示例。

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

```
<body>
<form action="demo_form.php" method="get">
请输入您的 E-mail 地址：<input type="email" name="user_email" /><br />
<input type="submit" />
</form>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行结果如图 4.1 所示。如果输入了错误的 E-mail 地址格式，单击“提交”按钮时会出现如图 4.2 所示的“请输入电子邮件地址。”的提示。

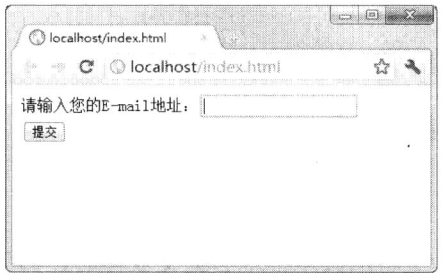


图 4.1　email 类型的 input 元素示例

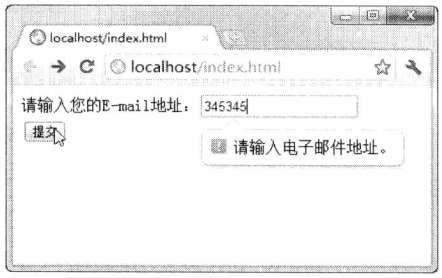


图 4.2　检测到不是有效的 E-mail 地址

其中 demo_form.php 表示提交给服务器端的处理文件。对于不支持 type="email"的浏览器来说，将会以 type="text"来处理，所以并不妨碍旧版浏览器浏览采用 HTML5 中 type="email"输入框的网页。

如果使用 IE9 测试该网页，则当输入错误的 e-mail 地址格式并单击“提交”按钮时，会首先访问后台服务器网页 demo_form.asp，如果找不到链接会出现如图 4.3 所示的“无法找到该网页”的错误提示，而不会出现如 Chrome 浏览器中这样的“请输入电子邮件地址”的提示，可见不同的浏览器处理 HTML5 网页的方式也会略有不同，在没有统一标准面世之前，各浏览器对 HTML5 的支持程度是不一样的。

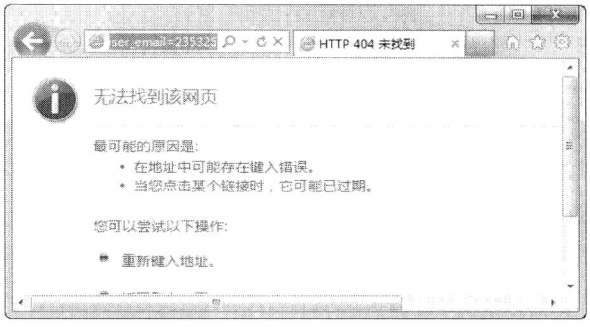


图 4.3　使用 IE9 测试该网页时出现的错误提示

如果将 email 类型的 input 元素用在手机浏览器中，则会更加突显其优势。例如，如果使用 iPhone 或 iPod touch 中的 Safari 浏览器浏览包含 E-mail 输入框的网页，Safari 浏览器会通过改变触摸屏键盘来配合该输入框，在触摸屏键盘中添加“@”和“.”键以方便用户输入，如图 4.4 所示，而当浏览普通内容时则不会出现这两个键。email 类型的 input 元素这一新增功能虽然用户不易察觉，但屏幕键盘的变化无疑会带来更好的用户体验。

图 4.4　iPod touch 中的 Safari 浏览器触摸屏键盘随输入域改变而改变

4.1.2　url 类型的应用

url 类型的 input 元素提供用于输入 url 地址这类特殊文本的文本框。当提交表单时，如果所输入的内容是 url 地址格式的文本，则会提交数据到服务器，如果不是 url 地址格式的文本，则不允许提交。

url 类型的 input 元素用法如下：

```
<input type="url" name="user_url" />
```

【示例】下面是 url 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请输入网址：<input type="url" name="user_url" /><br/>
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.5 所示。如果输入了错误的 url 地址格式，单击“提交”按钮时会出现如图 4.6 所示的“请输入网址。”的提示，本例中网址前面漏掉了协议类型，如 http://。

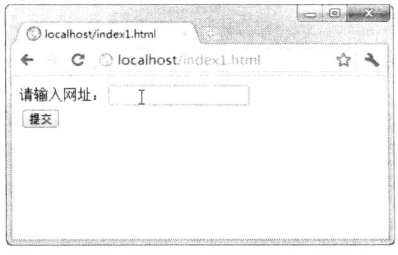


图 4.5　url 类型的 input 元素示例

图 4.6　检测到不是有效的 url 地址

与前面介绍的 email 类型输入框相同，对于不支持 type="url"的浏览器，将会以 type="text"来处理，

所以并不妨碍旧版浏览器浏览采用 HTML5 中 type="url"输入框的网页。

如果使用 iPhone 或 iPod touch 中的 Safari 浏览器浏览包含 url 输入域的网页，Safari 浏览器会通过改变触摸屏键盘来配合该输入框，在触摸屏键盘中添加“.”、“/”和“.com”键以方便用户输入，如图 4.7 所示，而当浏览普通内容时则不会出现这 3 个键。

图 4.7　iPod touch 中的 Safari 浏览器触摸屏键盘随输入域改变而改变

4.1.3　number 类型的应用

number 类型的 input 元素提供用于输入数值的文本框。我们还可以设定对所接受的数字的限制，包括规定允许的最大值和最小值、合法的数字间隔或默认值等。如果所输入的数字不在限定范围之内，则会出现错误提示。

【示例】下面是 number 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请输入数值：<input type="number" name="number1" min="1" max="20" step="4">
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.8 所示。如果输入了不在限定范围之内的数字，单击“提交”按钮时会出现如图 4.9 所示的提示。

图 4.8　number 类型的 input 元素示例

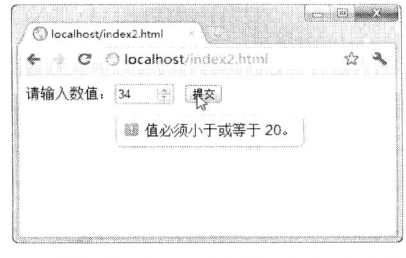


图 4.9　检测到输入了不在限定范围之内的数字

图 4.9 所示为输入了大于规定的最大值时所出现的提示。同样，如果违反了其他限定，也会出现相关提示。例如，如果输入数值 15，单击“提交”按钮时会出现“值无效。”的提示，如图 4.10 所示。这是因为限定了合法的数字间隔为 4，在输入时只能输入 4 的倍数，如 4、8、16 等。又如，如果输入数值-12，则会提示“值必须大于或等于 1。”，如图 4.11 所示。

Note

图 4.10　出现“值无效。”的提示

图 4.11　提示“值必须大于或等于 1。”

number 类型使用如表 4.2 所示的属性来规定对数字类型的限定。

表 4.2　number 类型的属性

属　性	值	描　述
max	number	规定允许的最大值
min	number	规定允许的最小值
step	number	规定合法的数字间隔（如果 step="4"，则合法的数是-4、0、4、8 等）
value	number	规定默认值

对于不同的浏览器，number 类型的输入框外观也可能会有所不同。而如果使用 iPhone 或 iPod touch 中的 Safari 浏览器浏览包含 number 输入框的网页，则 Safari 浏览器同样会通过改变触摸屏键盘来配合该输入框，触摸屏键盘会优先显示数字以方便用户输入，如图 4.12 所示。

图 4.12　iPod touch 中的 Safari 浏览器触摸屏键盘显示出数字与符号

4.1.4　range 类型的应用

range 类型的 input 元素提供用于输入包含一定范围内数字值的文本框，在网页中显示为滑动条。

我们还可以设定对所接受的数字的限制，包括规定允许的最大值和最小值、合法的数字间隔或默认值等。如果所输入的数字不在限定范围之内，则会出现错误提示。

【示例】下面是 range 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请输入数值： <input type="range" name="range1" min="1" max="30" />
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.13 所示。range 类型的 input 元素在不同浏览器中的外观也不同，例如，在 Opera 浏览器中的外观如图 4.14 所示，会在滑块下方显示出额外的数字间隔短线。

图 4.13　range 类型的 input 元素示例

图 4.14　range 类型的 input 元素在 Opera 浏览器中的外观

range 类型使用如表 4.3 所示的属性来规定对数字类型的限定。

表 4.3　range 类型的属性

属　性	值	描　述
max	number	规定允许的最大值
min	number	规定允许的最小值
step	number	规定合法的数字间隔（如果 step="4"，则合法的数是-4、0、4、8 等）
value	number	规定默认值

从上表可以看出，range 类型与 number 类型的属性是完全相同的，这两种类型的不同在于外观表现上，支持 range 类型的浏览器都会将其显示为滑块的形式，而不支持 range 类型的浏览器则会将其显示为普通的纯文本输入框，即以 type="text"来处理。所以不管怎样，我们都可以放心地使用 range 类型的 input 元素。

4.1.5　日期检出器类型的应用

日期检出器（Date Pickers）是网页中经常要用到的一种控件，在 HTML5 之前的版本中，并没有提供任何形式的日期检出器控件。在网页前端设计中，多是采用一些 JavaScript 框架来实现日期检出器控件的功能，如 jQuery UI、YUI 等，在具体使用时会比较麻烦。

HTML5 提供了多个可用于选取日期和时间的输入类型，即 6 种日期检出器控件，分别用于选择以下日期格式：日期、月、星期、时间、日期+时间和日期+时间+时区，如表 4.4 所示。

Note

表 4.4　日期检出器类型

输 入 类 型	HTML 代码	功能与说明
date	<input type="date">	选取日、月、年
month	<input type="month">	选取月、年
week	<input type="week">	选取周和年
time	<input type="time">	选取时间（小时和分钟）
datetime	<input type="datetime">	选取时间、日、月、年（UTC 时间）
datetime-local	<input type="datetime-local">	选取时间、日、月、年（本地时间）

UTC 是 Universal Time Coordinated 的英文缩写，即“协调世界时”，是由国际无线电咨询委员会规定和推荐，并由国际时间局（BIH）负责保持的以秒为基础的时间标度。简单地说，UTC 时间就是 0 时区的时间，而本地时间即地方时。例如，如果北京时间为早上 8 点，则 UTC 时间为 0 点，即 UTC 时间比北京时间晚 8 小时。

下面分别给出这些日期检出器类型的示例与运行结果。

1. date 类型

date 类型的日期检出器用于选取日、月、年，即选择一个具体的日期，如 2012 年 8 月 9 日，选择后会以 2012-08-09 的形式显示。

【示例 1】下面是 date 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请输入日期：  <input type="date" name=" date1" />
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.15 所示，在 Opera 浏览器中的运行结果如图 4.16 所示。Chrome 浏览器中显示为右侧带有微调按钮的数字输入框，可见该浏览器并不支持日期检出器控件。而在 Opera 浏览器中单击右侧小箭头时会显示出日期控件，用户可以使用控件来选择具体日期。

图 4.15　在 Chrome 浏览器中的运行结果

图 4.16　在 Opera 浏览器中的运行结果

2. month 类型

month 类型的日期检出器用于选取月、年，即选择一个具体的月份，如 2011 年 8 月，选择后会以 2011-08 的形式显示。

Note

【示例 2】下面是 month 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请输入月份：  <input type="month" name=" month1" />
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.17 所示，在 Opera 浏览器中的运行结果如图 4.18 所示。Chrome 浏览器中显示为右侧带有微调按钮的数字输入框，输入或微调时会只显示到月份，而不会显示日期。在 Opera 浏览器中单击右侧小箭头时会显示出日期控件，用户可以使用控件来选择具体月份，但不能选择具体日期。可以看到，整个月份中的日期都会以深灰色显示，单击该区域可以选择整个月份。

图 4.17　在 Chrome 浏览器中的运行结果

图 4.18　在 Opera 浏览器中的运行结果

3. week 类型

week 类型的日期检出器用于选取周和年，如 2012 年 8 月第 32 周，选择后会以 2012-W32 的形式显示。

【示例 3】下面是 week 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请选择年份和周数：  <input type="week" name="week1" />
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.19 所示，在 Opera 浏览器中的运行结果如图 4.20 所示。Chrome 浏览器中显示为右侧带有微调按钮的数字输入框，输入或微调时会显示年份和周数，

而不会显示日期。在 Opera 浏览器中单击右侧小箭头时会显示出日期控件，用户可以使用控件来选择具体的年份和周数，但不能选择具体日期。可以看到，整个月份中的日期都会以深灰色按周数显示，单击该区域可以选择某一周。

Note

图 4.19　在 Chrome 浏览器中的运行结果

图 4.20　在 Opera 浏览器中的运行结果

4. time 类型

time 类型的日期检出器用于选取时间，具体到小时和分钟，如 11 时 11 分，选择后会以 2011-11 的形式显示。

【示例 4】下面是 time 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请选择或输入时间：  <input type="time" name="time1" />
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.21 所示，在 Opera 浏览器中的运行结果如图 4.22 所示。

图 4.21　在 Chrome 浏览器中的运行结果

图 4.22　在 Opera 浏览器中的运行结果

除了可以使用微调按钮，还可以直接输入时间值。如果输入了错误的时间格式并单击“提交”按钮，则在 Chrome 浏览器中会显示“值无效。”的提示，如图 4.23 所示。而在 Opera 浏览器中则不存在这样的问题，因为该浏览器根本不允许输入错误的数值，而且会一直显示中间的冒号作为小时和分钟的间隔。

time 类型支持使用一些属性来限定时间的大小范围或合法的时间间隔，如表 4.4 所示。

表 4.4　time 类型的属性

属　性	值	描　述
max	time	规定允许的最大值
min	time	规定允许的最小值
step	number	规定合法的时间间隔
value	time	规定默认值

Note

【示例 5】可以使用下列代码来限定时间。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请选择或输入时间：<input type="time" name="time1" step="5" value="12:05:15">
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.24 所示，可以看到，在输入框中出现设置的默认值“12:05:15”，并且当单击微调按钮时，会以 5 分钟为单位递增或递减。当然，还可以使用 min 和 max 属性指定时间的范围。

图 4.23　显示“值无效”的提示

图 4.24　使用属性值限定时间类型

在 date 类型、month 类型、week 类型中也支持使用上述属性值。

5. datetime 类型

datetime 类型的日期检出器用于选取时间、日、月、年，其中时间为 UTC 时间。

【示例 6】下面是 datetime 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请选择或输入时间：<input type="datetime" name="datetime1" />
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.25 所示，在 Opera 浏览器中的运行结果如图 4.26

所示。

图 4.25　在 Chrome 浏览器中的运行结果

图 4.26　在 Opera 浏览器中的运行结果

6. datetime-local 类型

datetime-local 类型的日期检出器用于选取时间、日、月、年，其中时间为本地时间。

【示例 7】下面是 datetime-local 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请选择或输入时间：<input type="datetime-local" name="datetime-local1" />
<input type="submit" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.27 所示，在 Opera 浏览器中的运行结果如图 4.28 所示。

图 4.27　在 Chrome 浏览器中的运行结果

图 4.28　在 Opera 浏览器中的运行结果

4.1.6　search 类型的应用

search 类型的 input 元素提供用于输入搜索关键词的文本框。虽然在外观上看起来，search 类型的 input 元素与普通的 text 类型只是稍有区别，但实现起来却并不那么容易。search 类型提供的搜索框不只是 Google 或百度的搜索框，而是任意网页中的任意一个搜索框。目前大多数网站的搜索框都是用<input type="text">的方式来实现的，即采用纯文本的文本框，而 HTML5 中定义了专用于搜索框的 search 类型。

Note

【示例】下面是 search 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请输入搜索关键词：<input type="search" name="search1" />
<input type="submit" value="Go"/>
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.29 所示。如果在搜索框中输入要搜索的关键词，在搜索框右侧就会出现一个“×”按钮，如图 4.30 所示。单击该按钮可以清除已经输入的内容。在 Windows 系统中，新版的 Chrome 浏览器支持“×”按钮这一功能，其他浏览器则不支持。而 Mac OS X 中，新版的 Chrome 浏览器和 Safari 浏览器都支持这一功能。

图 4.29　search 类型的应用

图 4.30　输入关键词后出现“×”按钮

此外，Mac OS X 中的 Safari 浏览器会将搜索框渲染成圆角（如图 4.31 所示），而不是 Windows 系统中常见到的方角。

图 4.31　Mac OS X 中的圆角搜索框

在默认情况下，旧版的 Safari 浏览器不允许使用基本的 CSS 样式来控制<input type="search">搜索框。如果希望用自己的 CSS 样式来控制搜索框的样式，则可以强制 Safari 浏览器将<input type="search">搜索框当做普通文本框来处理，但需要将下面的规则加入样式表。

```
input[type="search"] {
    -webkit-appearance: textfield;
}
```

4.1.7　tel 类型的应用

tel 类型的 input 元素提供专门用于输入电话号码的文本框。它并不限定只输入数字，因为很多的电话号码还包括其他字符（如“+”、“-”、“(”、“)”等），如 86-0536-8888888。

【示例】下面是 tel 类型的一个应用示例。

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

```
<form action="demo_form.php" method="get">
请输入电话号码：<input type="tel" name="tel1" />
<input type="submit" value="提交"/>
</form>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行结果如图 4.32 所示。从某种程度上来说，所有的浏览器都支持 tel 类型的 input 元素，因为它们都会将其作为一个普通的文本框来显示。HTML5 规则并不需要浏览器执行任何特定的电话号码语法或以任何特别的方式来显示电话号码。iPhone 或 iPod touch 中的 Safari 浏览器遇到 tel 类型的 input 元素时，会自动变换触摸屏幕键盘以方便用户输入，如图 4.33 所示。

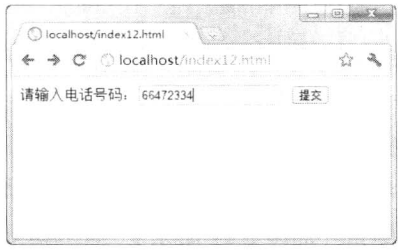


图 4.32　tel 类型的应用

图 4.33　iPod touch 中的屏幕键盘变化

4.1.8　color 类型的应用

color 类型的 input 元素提供专门用于设置颜色的文本框。通过单击文本框，可以快速打开拾色器面板，方便用户可视化选择一种颜色。

【示例】下面是 color 类型的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="demo_form.php" method="get">
请选择一种颜色：<input type="color" name="color1" />
<input type="submit" value="提交"/>
</form>
</body>
</html>
```

以上代码在 Opera 浏览器中的运行结果如图 4.34 所示，其他浏览器暂不支持。颜色选取面板中仅提供了 20 种常用的颜色，在选取之后会在下方显示十六进制的颜色值。如果希望选取其他颜色，则单击“其他”按钮，此时会打开 Windows 或 Mac OS 中传统的“颜色”拾取框。Windows 系统中的“颜色”拾取框如图 4.35 所示，Mac OS 系统中的“颜色”拾取框如图 4.36 所示。

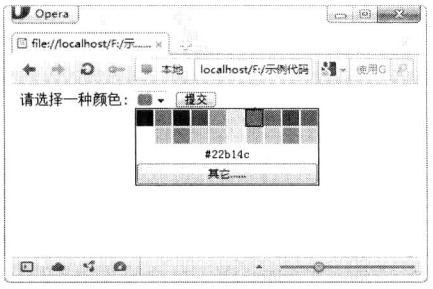


图 4.34　color 类型的应用

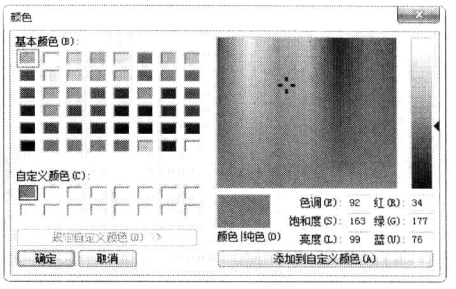


图 4.35　Windows 系统中的“颜色”拾取框

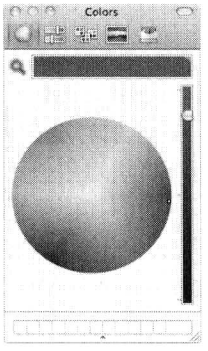


图 4.36　Mac OS 系统中的“颜色”拾取框

4.2　新增的 input 属性

input 元素不但增加了新的输入类型，而且还新增了几个 input 属性，用于指定输入类型的行为和限制，这些属性分别是 autocomplete、autofocus、form、form overrides、placeholder、height 和 width、min 和 max、step、list、pattern 以及 required。本节分别详细介绍这些新的 input 属性。

4.2.1　新增的 autocomplete 属性

多数浏览器都带有辅助用户完成输入的自动完成功能，只要开启了该功能，用户下次输入相同的内容时，浏览器就会自动完成内容的输入。

新增的 autocomplete 属性可以帮助用户在 input 类型的输入框中实现自动完成内容输入，这些 input 类型包括：text、search、url、telephone、email、password、datepickers、range 以及 color。不过，在某些浏览器中，可能需要首先启用浏览器本身的自动完成功能，才能使 autocomplete 属性起作用。目前只有 Opera 浏览器支持 autocomplete 属性。

autocomplete 属性同样适用于<form>标签，默认状态下表单的 autocomplete 属性是处于打开状态的，其中的输入类型继承所在表单的 autocomplete 状态。也可以单独将表单中某一输入类型的 autocomplete 状态设置为打开状态，这样可以更好地实现自动完成。

autocomplete 属性有 3 种值：on、off 和空值。例如，可以这样来指定 autocomplete 的属性值：

```
<input type="url" name="url1" autocomplete="on">
```

也可以像下面的代码中这样将表单的 autocomplete 属性值设置为“on”，而单独将其中某一输入

类型的 autocomplete 属性值设置为“off”。

```
<!DOCTYPE html>
<html>
<body>
<form action="/formexample.asp" method="get" autocomplete="on">
姓名：<input type="text" name="name1" /><br />
职业：<input type="text" name="career1" /><br />
电子邮件地址：<input type="email" name="email1" autocomplete="off" /><br />
<input type="submit" value="提交信息" />
</form>
</body>
</html>
```

autocomplete 属性设置为“on”时，可以使用 HTML5 中新增的 datalist 标签和 list 属性提供一个数据列表供用户进行选择。下面的示例代码说明如何综合应用脚本、autocomplete 属性、datalist 标签及 list 属性实现自动完成。在本例中，当用户将焦点定位到文本框中时，会自动出现一个城市列表供用户选择，如图 4.37 所示。而当用户单击页面的其他位置时，这个列表就会消失。此外，当用户输入时，该列表会随用户的输入进行更新，例如，当输入字母 Q 时，会自动更新列表，下拉列表显示以 Q 开头的城市名称，如图 4.38 所示。随着用户不断地输入新的字母，下面的列表还会随之变化。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
    <h2>HTML5 自动完成功能示例</h2>
    输入你最喜欢的城市名称<br><br>
<form autocompelete="on">
    <input type="text" id="city" list="cityList">
    <datalist id="cityList" style="display:none;">
        <option value="BeiJing">BeiJing</option>
        <option value="QingDao">QingDao</option>
        <option value="QingZhou">QingZhou</option>
        <option value="QingHai">QingHai</option>
    </datalist>
</form>
</body>
</html>
```

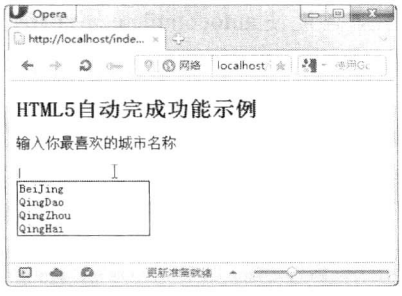


图 4.37　自动完成数据列表

图 4.38　数据列表随用户输入而更新

在 HTML5 之前，打开自动完成功能后，用户的浏览器会自动记录用户所输入的一些信息，从安全性和隐私的角度来考虑，存在较大的隐患。现在有了 autocomplete 属性，如果不希望用户的浏览器自动记录这些历史记录，则可以对 form 或 form 中的每个 input 元素都做单独的 autocomplete 属性设置。虽然 autocomplete 是 HTML5 中新增的属性，但其实该属性早在 IE5 中便已经存在了，在这之后才慢慢被其他浏览器所支持。

Note

4.2.2　新增的 autofocus 属性

在访问 Google 主页时，页面中的文字输入框会自动获得光标焦点，以方便输入搜索关键词，对大多数用户来说这是非常好的体验。目前的 Web 站点多采用 JavaScript 来实现让表单中某控件自动获取焦点，通常使用 Control.focus()编写 JavaScript 脚本来实现这一功能。

HTML5 中新增了 autofocus 属性，它是一个布尔值，可以使得在页面加载时，某表单控件自动获得焦点。这些控件可以是文本框、复选框、单选按钮和普通按钮等所有<input>标签的类型。autofocus 属性的使用示例如下所示。

```
<input type="text" name="user_name" autofocus="autofocus">
```

autofocus 属性的出现使得页面中的表单控件自动获取焦点变得非常容易，但要注意，在同一页面中只能指定一个 autofocus 属性值，所以必须谨慎使用。当页面中的表单控件比较多时，建议挑选最需要聚焦的那个控件来使用这一属性值，例如，一个搜索页面中的搜索文本框，或者一个同意某许可协议的“同意”按钮。

【示例 1】以下的示例说明如何合理地应用 autofocus 属性。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="GB2312">
<title>autofocus 属性应用示例</title>
</head>
<body>
<form>
    <p>请仔细阅读许可协议：</p>
    <p>
    <label for="textarea1"></label>
      <textarea name="textarea1" id="textarea1" cols="45" rows="5">许可协议许可协议许可协议许可协议许可协议许可协议许可协议许可协议许可协议</textarea>
        <br/>
    </p>
    <p>
      <input type="submit" value="同意" autofocus="on">
      <input type="submit" value="拒绝">
    </p>
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.39 所示。页面载入后，“同意”按钮自动获得焦点，因为通常希望用户直接单击该按钮。而如果将“拒绝”按钮的 autofocus 属性值设置为“on”，则

页面载入后焦点就在会“拒绝”按钮上，如图 4.40 所示，但从页面功用的角度来说却并不合适。正如以上所说，autofocus 属性应该谨慎使用，所以在指定 autofocus 时，应考虑页面最主要的目的是什么。

Note

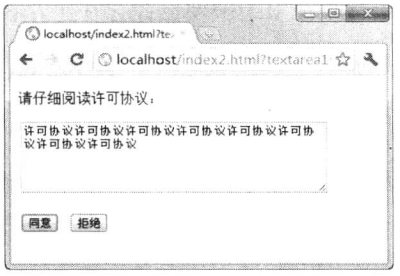


图 4.39 “同意”按钮自动获得焦点

图 4.40 “拒绝”按钮自动获得焦点

【示例 2】不支持 autofocus 属性的浏览器会将其忽略，如果要使得所有浏览器都能实现自动获得焦点，则可以在代码中加一小段脚本，以检测浏览器是否支持 autofocus 属性，代码如下。有关 autofocus 属性出现之前自动获取焦点的各种脚本实现方式，读者可以查阅相关参考资料，在此不再赘述。

```
<!DOCTYPE html>
<head>
<meta charset="GB2312">
<title>autofocus 属性应用示例</title>
</head>
<body>
<form name="form1">
  <input type="search" id="s" autofocus>
  <script>
    if (!("autofocus" in document.createElement("input"))) {
      document.getElementById("s").focus();
    }
  </script>
  <input type="submit" value="提交">
</form>
</body>
</html>
```

4.2.3 新增的 form 属性

在 HTML5 之前，如果用户要提交一个表单，必须把相关的控件元素都放在表单内部，即<form>和</form>标签之间。在提交表单时，会将页面中不是表单子元素的控件直接忽略掉。然而有些时候，可能需要一并提交表单之外的某些元素，而表单固有的缺陷使得这一要求不容易实现。

HTML5 中新增了一个 form 属性，使得这一问题得到了很好的解决。有了 form 属性，便可以把表单内的从属元素写在页面中的任一位置，然后只需要为这个元素指定 form 属性并设置属性值为该表单的 id。如此一来，便规定了该表单元素属于指定的这一表单。此外，form 属性也允许规定一个表单元素从属于多个表单。form 属性适用于所有的 input 输入类型，在使用时，必须引用所属表单的 id。

Note

【示例】下面是一个 form 属性应用的示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="" method="get" id="form1">
请输入姓名：<input type="text" name="name1" autofocus/>
<input type="submit"    value="提交"/>
</form>
<p>下面的输入框在 form 元素之外，但因为指定的 form 属性，并且值为表单的 id，所以该输入框仍然是表单的一部分。</p>
请输入住址：<input type="text" name="address1" form="form1" />
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.41 所示。如果填写姓名和住址并单击“提交”按钮，则 name1 和 address1 分别会被赋值为所填写的值。例如，如果在姓名处填写“zhangsan”，住址处填写“北京”，则单击“提交”按钮后，服务器端会接收到“name1= zhangsan”和“address1= 北京”。也可以在提交后观察浏览器的地址栏，可以看到有“name1=zhangsan&address=北京”字样，如图 4.42 所示。

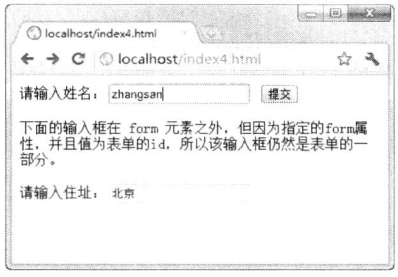


图 4.41　form 属性的应用

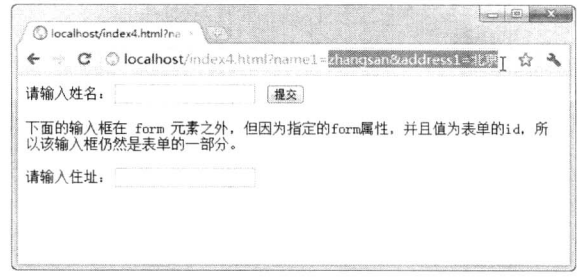


图 4.42　地址中要提交的数据

如果一个 form 属性要引用两个或两个以上的表单，则需要使用空隔将表单的 id 分隔开。例如：

```
<input type="text" name="address1" form="form1 form2 form3" />
```

4.2.4　新增的表单重写属性

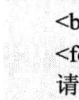
HTML5 中新增了几个表单重写（form override）属性，用于重写 form 元素的某些属性设定，这些表单属性包括以下几种。

- ☑ formaction：用于重写表单的 action 属性。
- ☑ formenctype：用于重写表单的 enctype 属性。
- ☑ formmethod：用于重写表单的 method 属性。
- ☑ formnovalidate：用于重写表单的 novalidate 属性。
- ☑ formtarget：用于重写表单的 target 属性。

应该注意的是，表单重写属性并不适用于所有的 input 输入类型，而是只适用于 submit 和 image 输入类型。

【示例】在 HTML5 之前，只能使用表单的 action 属性将表单内的所有元素统一提交到另一个页面。而使用 formaction 属性，则可以通过重写表单的 action 属性，实现将表单提交到不同的页面中去，

代码如下所示。



Note

```
<!DOCTYPE html>
<html>
<body>
<form action="1.asp" id="testform">
请输入电子邮件地址：  <input type="email" name="userid" /><br />
      <input type="submit" value="提交到页面 1" formaction="1.asp" />
      <input type="submit" value="提交到页面 2" formaction="2.asp" />
      <input type="submit" value="提交到页面 3" formaction="3.asp" />
</form>
</body>
</html>
```

4.2.5 新增的 height 与 width 属性

height 和 width 属性规定用于 image 类型的 input 标签的图像高度和宽度，这两个属性只适用于 image 类型的<input>标签。

【示例】下面是 height 与 width 属性应用的示例代码。

```
<!DOCTYPE html>
<html>
<body>
<form action="testform.asp" method="get">
请输入用户名: <input type="text" name="user_name" /><br />
<input type="image" src="submit.jpg" width="50" height="50" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.43 所示。该示例中，源图像的大小为宽 387 像素高 372 像素，使用以上代码将其显示大小限制为 50×50 像素。

图 4.43　height 与 width 属性的应用

4.2.6 新增的 list 属性

HTML5 中新增了一个 datalist 元素，可以实现数据列表的下拉效果，其外观类似 autocomplete，用户可从列表中选择，也可自行输入。而 list 属性用于指定输入框所绑定的 datalist 元素，其值是某个 datalist 的 id。

Note

【示例】下面是 list 属性应用的示例代码。

```
<!DOCTYPE html>
<html>
<body>
<form action="testform.asp" method="get">
请输入网址：  <input type="url" list="url_list" name="weblink" />
<datalist id="url_list">
    <option label="新浪" value="http://www.sina.com.cn" />
    <option label="搜狐" value="http://www.sohu.com" />
    <option label="网易" value="http://www.163.com" />
</datalist>
<input type="submit" value="提交" />
</form>
</body>
</html>
```

以上代码在 Opera 浏览器中的运行结果如图 4.44 所示。在本例中，单击输入框之后，就会弹出已定义的网址列表。目前支持这一属性的浏览器只有 Opera。

图 4.44　list 属性应用

list 属性适用于以下 input 输入类型：text、search、url、telephone、email、date pickers、number、range 和 color。

4.2.7　新增的 min、max 和 step 属性

HTML5 中新增的 min、max 和 step 属性用于为包含数字或日期的 input 输入类型规定限值，也就是给这些类型的输入框加一个数值的约束，适用于 date、pickers、number 和 range 标签。具体用途如下。

- ☑ max 属性：规定输入框所允许的最大值。
- ☑ min 属性：规定输入框所允许的最小值。
- ☑ step 属性：为输入框规定合法的数字间隔（或称为“步进”。如果 step="4"，则合法的数值是-4、0、4、8 等）。

【示例】在下面的示例中，显示一个数字输入框，并规定该输入框接受介于 0~12 之间的值，且数字间隔为 4（即合法的值为 0、4、8、12）。

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

```
<form action="testform.asp" method="get">
请输入数值：<input type="number" name=number1" min="0" max="12" step="4" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行结果如图 4.45 所示。在本例中，如果单击数字输入框右侧的微调按钮，则可以看到数字以 4 为步进值递增。而如果输入不合法的数值，如数字 3，则单击“提交”按钮时会显示“值无效。”的提示，如图 4.46 所示。

图 4.45　min、max 和 step 属性应用

图 4.46　显示“值无效”的提示

4.2.8　新增的 multiple 属性

在 HTML5 之前，input 输入类型中的 file 类型只支持选择单个文件来上传，而新增的 multiple 属性支持一次性选择多个文件，并且该属性同样支持新增的 email 类型。这一特性无疑为开发者提供了极大的方便，因为有了 HTML5 便不必再单独开发选择并提交多个文件的控件。

【示例】下面是 multiple 属性的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="testform.asp" method="get">
请选择要上传的多个文件：  <input type="file" name="img" multiple="multiple" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

以上代码在 Opera 浏览器中的运行结果如图 4.47 所示。如果单击“添加文件”按钮，则会允许在打开的对话框中选择多个文件。选择文件并单击“打开”按钮后会关闭对话框，同时输入框中会列出这些被选中文件的完整路径，如图 4.48 所示。

图 4.47　multiple 属性的应用

图 4.48　显示被选中文件的路径

4.2.9 新增的 pattern 属性

Note

pattern 属性用于验证 input 类型输入框中用户输入的内容是否与自定义的正则表达式相匹配，该属性适用于以下类型的<input>标签：text、search、url、telephone、email、password。其实许多 input 输入类型本身就是 HTML5“内建”的正则表达式，如 email、number、tel、url 等，使用这些输入类型，浏览器便能够检查用户的输入是否合乎既定的规则。

在一些用于处理字符串的程序或网页代码中，经常会用到一些用于查找或输入符合某些复杂规则的字符串的代码，而正则表达式正是用于描述一系列符合某个句法规则的代码。一个正则表达式通常被称为一个模式（pattern）。

pattern 属性允许用户自定义一个正则表达式，但用户的输入必须符合正则表达式所指定的规则。pattern 属性中的正则表达式语法与 JavaScript 中的正则表达式语法相匹配。

【示例】下面是 pattern 属性的一个应用示例。该示例的文本输入框规定必须输入 6 位数的邮政编码。

```
<!DOCTYPE html>
<html>
<body>
<form action="/testform.asp" method="get">
请输入邮政编码: <input type="text" name="zip_code" pattern="[0-9]{6}"
title="请输入 6 位数的邮政编码" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.49 所示。当指向输入框时，会出现“请输入 6 位数的邮政编码”的提示。如果输入的数字不是 6 位，则会出现错误提示，如图 4.50 所示。如果输入的并非规定的数字，而是字母，也会出现这样的错误提示。这是因为在 pattern="[0-9]{6}"中规定了必须输入 0~9 这样的阿拉伯数字，并且必须为 6 位数，有关正则表达式的知识可以参考相关图书或资料（如 Andrew Watt 的《正则表达式入门经典》，或上网搜索并阅读“正则表达式 30 分钟入门教程”以快速入门）。

图 4.49 pattern 属性的应用

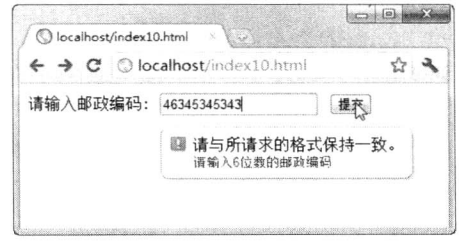


图 4.50 出现错误提示

4.2.10 新增的 placeholder 属性

placeholder 属性用于为 input 类型的输入框提供一种提示（hint），这些提示可以描述输入框期待用户输入何种内容，在输入框为空时显式出现，而当输入框获得焦点时则会消失。placeholder 属性适

用于以下类型的<input>标签：text、search、url、telephone、email 和 password。

【示例】下面是 placeholder 属性的一个应用示例。请注意比较本例与上例提示方法的不同。

```
<!DOCTYPE html>
<html>
<body>
<form action="/testform.asp" method="get">
请输入邮政编码：<input type="text" name="zip_code" pattern="[0-9]{6}"
placeholder="请输入 6 位数的邮政编码" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.51 所示。当输入框获得焦点时，提示文字消失，如图 4.52 所示。

图 4.51　placeholder 属性的应用

图 4.52　提示消失

4.2.11　新增的 required 属性

新增的 required 属性用于规定输入框填写的内容不能为空，否则不允许用户提交表单。该属性适用于以下 input 输入类型：text、search、url、telephone、email、password、date pickers、number、checkbox、radio 和 file。

【示例】下面是 required 属性的一个应用示例。该示例的文本输入框中必须输入内容，否则表单不能被提交。

```
<!DOCTYPE html>
<html>
<body>
<form action="/testform.asp" method="get">
请输入姓名：<input type="text" name="usr_name" required="required" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行结果如图 4.53 所示。当输入框内容为空并单击“提交”按钮时，会出现“请填写此字段。”的提示，只有在输入了内容之后才允许提交表单。

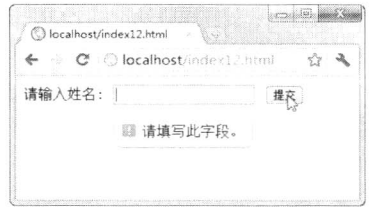


图 4.53　提示“请填写此字段”

Note

4.3　新增的 form 元素

HTML5 中新增了几个 form 元素，分别是 datalist、keygen 和 output，目前来看，只有 Opera 浏览器对这些元素支持得最好。本节将通过一些应用示例来说明这几个新元素的用法。

4.3.1　新增的 datalist 元素

datalist 元素用于为输入框提供一个可选的列表，用户可以直接选择列表中的某一预设的项，从而免去输入的麻烦。该列表由 datalist 中的 option 元素创建。如果用户不希望从列表中选择某项，也可以自行输入其他内容。

在实际应用中，如果要把 datalist 提供的列表绑定到某输入框，则需要使用输入框的 list 属性来引用 datalist 元素的 id，其应用示例在本章介绍 list 属性时已经提供，在此不再赘述。

应该注意的是，每一个 option 元素都必须设置 value 属性。

4.3.2　新增的 keygen 元素

keygen 元素是密钥对生成器（key-pair generator），能够使用户验证更为可靠。用户提交表单时会生成两个键，一个私钥，一个公钥，其中私钥（private key）会被存储在客户端，而公钥（public key）则会被发送到服务器。公钥可用于验证用户的客户端证书（client certificate）。如果各种新的浏览器对 keygen 元素的支持度再增强一些，则有望使其成为一种有用的安全标准。

【示例】下面是 keygen 属性的一个应用示例。

```
<!DOCTYPE html>
<html>
<body>
<form action="/testform.asp" method="get">
请输入用户名：<input type="text" name="usr_name" /><br>
请选择加密强度：<keygen name="security" /><br>
<input type="submit" value="提交" />
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 4.54 所示。在“请选择加密强度：”右侧的 keygen 元素中可以选择一种密钥强度，有 2048（高强度）和 1024（中等强度）两种。相比之下，Opera 浏览器对 keygen 元素的支持略胜一筹，该浏览器支持更多的密钥等级，如图 4.55 所示。

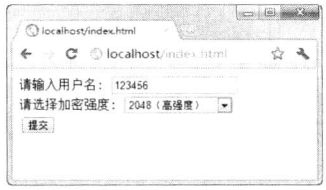


图 4.54　Chrome 浏览器中 keygen 属性的应用效果

图 4.55　Opera 浏览器提供的密钥等级

4.3.3　新增的 output 元素

output 元素用于在浏览器中显示计算结果或脚本输出，包含完整的开始和结束标签，其语法如下。

```
<output name="">Text</output>
```

【示例】下面是 output 属性的一个应用示例，可计算用户输入的两个数字的乘积。

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
    function multi(){
        a=parseInt(prompt("请输入第 1 个数字。",0));
        b=parseInt(prompt("请输入第 2 个数字。",0));
        document.forms["form"]["result"].value=a*b;
    }
</script>
</head>
<body onload="multi()">
<form action="/testform.asp" method="get" name="form">
两数的乘积为：<output name="result"></output>
</form>
</body>
</html>
```

在 Chrome 浏览器中运行以上代码，当页面载入时，会首先提示“请输入第 1 个数字”，如图 4.56 所示，输入并单击“确定”按扭后再根据提示输入第 2 个数字，如图 4.57 所示，再次单击“确定”按钮后，显示计算结果，如图 4.58 所示。

图 4.56　提示输入第 1 个数字

图 4.57　提示输入第 2 个数字

图 4.58　显示计算结果

Note

4.4　新增的 form 属性

HTML5 中新增了两个 form 属性，分别是 autocomplete 和 novalidate，本节通过示例介绍这两个 form 属性的用法。

4.4.1　新增的 autocomplete 属性

form 元素的 autocomplete 属性用于规定 form 中所有元素都拥有自动完成功能。该属性在介绍 input 属性时已经介绍过，其用法与之相同，只不过当 autocomplete 属性用于整个 form 时，所有从属于该 form 的元素便都具备自动完成功能。如果要使个别元素关闭自动完成功能，则单独为该元素指定 autocomplete="off"即可，具体方法可参见第 4.2 节有关 autocomplete 属性的介绍。

4.4.2　新增的 novalidate 属性

form 元素的 novalidate 属性用于在提交表单时取消整个表单的验证，即关闭对表单内所有元素的有效性检查。如果要只取消表单中较少部分内容的验证而不妨碍提交大部分内容，则可以将 formnovalidate 属性单独用于 form 中的这些元素。

【示例】下面是 novalidate 属性的一个应用示例。该示例中取消了整个表单的验证。

```
<!DOCTYPE html>
<html>
<body>
<form action="/testform.asp" method="get" novalidate="true">
请输入电子邮件地址：<input type="email" name="user_email" />
<input type="submit" value="提交" />
</form>
</body>
</html>
```

第5章

实战 HTML5 画布

（视频讲解：2 小时）

Canvas 是 HTML5 新增的开发跨平台动画和游戏的标准解决方案，能够实现对图像和视频进行像素级操作，这为 Web 图形和视频处理打开了广阔的空间。借助 HTML5 Canvas 技术，用户可以在 Web 中绘制各种图形。虽然在此之前也有基于 XML 的绘图技术，如 VML 和 SVG 等，但 Canvas 是基于像素的绘图，它相当于画板的 html 节点，开发者通过 JavaScript 脚本可以轻松实现任意绘图。

canvas 是 HTML5 新增的专门用来绘制图形的元素。在页面上放置一个 canvas 元素，就相当于在页面上放置了一块画布，可以在其中进行图形的描绘。在 canvas 元素里进行绘画，不需要用鼠标作画。事实上，canvas 元素只是一块无色透明的区域，需要利用 JavaScript 编写在其中进行绘画的脚本。从这个角度来说，读者可以把它理解为类似于其他开发语言中的 canvas 画布。本章将详细介绍 Canvas 的渲染上下文、坐标系统、画布状态、变形合成，以及高级的图像和视频处理技巧和应用案例。

5.1　认识 HTML5 canvas 元素

Note

HTML5 canvas 元素及随其而来的编程接口 Canvas API 应用前景极为广阔。简单地说，canvas 元素能够在网页中创建一块矩形区域，这块矩形区域可以称为画布，这其中可以绘制各种图形。可别小看了这个画布，它能实现无限的可能性。接下来让我们从最简单的部分开始，逐步认识 Canvas 的强大功能。

5.1.1　在页面中添加 canvas 元素

如果要向页面中添加 canvas 元素，可以使用下面的代码。默认情况下，Canvas 所创建的矩形区域大小为宽 300 像素、高 150 像素，也可以使用 width 与 height 属性来自定义其宽度和高度。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100"></canvas>
</body>
</html>
```

以上代码只是简单地创建了一个 Canvas 对象，在浏览器中打开的页面上什么都不会显示，但如果是在 Dreamweaver 的设计视图中，就可以看到一个矩形区域，如图 5.1 所示。

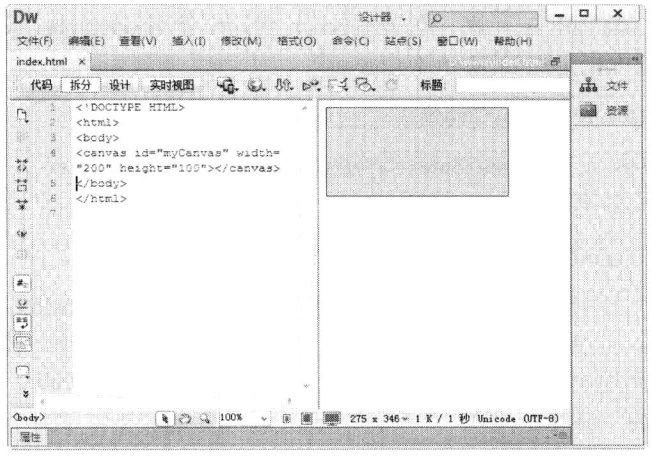


图 5.1　Dreamweaver 的设计视图中看到的矩形区域

可以在 canvas 元素中使用标准的 CSS 边框属性来设置边框的外观。例如，使用下面示例中的 style 属性，可以为 canvas 元素添加一个实心的边框。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.2 所示。

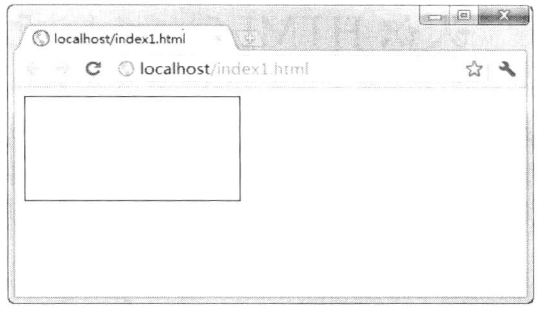


图 5.2　为 canvas 元素添加实心边框

观察以上代码，注意其中的 id="myCanvas"，在接下来的小节中将介绍 JavaScript 如何使用 id 来找到 canvas 元素。

5.1.2　Canvas 如何绘制图形

canvas 元素本身并不能实现图形绘制功能，绘制图形的工作需要由 JavaScript 来完成。使用 JavaScript 可以在 canvas 元素内部添加线条、图片和文字，也可以在其中绘画，还能够加入高级动画。

在 Canvas 中绘制图形的具体步骤如下：

第 1 步，在 HTML5 页面中添加 canvas 元素，必须定义 canvas 元素的 id 属性值以便接下来调用。

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

第 2 步，使用 id 寻找 canvas 元素。在 JavaScript 代码中使用 document.getElementById 等方法，利用先前在 canvas 元素中指定的 id 来寻找 Canvas。

```
var c=document.getElementById("myCanvas");
```

第 3 步，通过 canvas 元素的 getContext 方法来获取其上下文（Context），即创建 Context 对象，以获取允许进行绘制的 2D 环境。

```
var context=c.getContext("2d");
```

这里的 getContext("2d")是用于返回一个内建的 HTML5 对象（CanvasRenderingContext2D 对象），使用该对象可在 canvas 元素中绘制图形，目前强制支持的只有"2d"，即二维绘图。getContext("2d")对象能够实现一个画布所使用的大多数方法，例如，绘制路径、矩形、圆形、字符和添加图像（如下面将要介绍到的 fillStyle、fillRect 等）。如果将来 canvas 元素能够支持 3D，则 getContext()方法可能会允许使用"3d"这个字符串参数。

第 4 步，使用 JavaScript 进行绘制。例如，使用以下代码可以绘制一个位于画布中央的矩形。

```
context.fillStyle="#FF00FF";
context.fillRect(50,25,100,50);
```

这两行代码中，fillStyle 将要绘制的矩形的填充颜色定义为粉红色，而 fillRect 则指定了要绘制的矩形的位置和尺寸。图形的位置由前面的 Canvas 坐标值决定，尺寸由后面的宽度和高度值决定。在本例中，坐标值为（50,25），宽为 100 像素、高为 50 像素，根据这些数值，粉红色矩形将出现在画布的中央。

Note

【示例】下面给出完整的示例代码。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100"></canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.fillStyle="#FF00FF";
context.fillRect(50,25,100,50);
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.3 所示。在画布周围加上边框是为了更能清楚地看到中间矩形位于画布的什么位置。

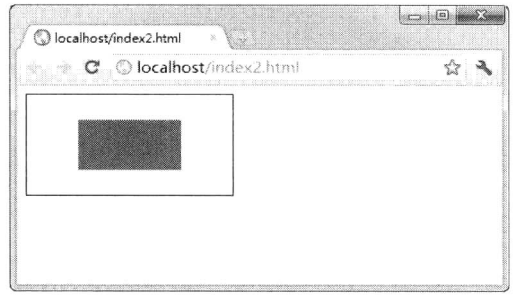


图 5.3　Canvas 绘制图形效果

5.1.3　认识 Canvas 坐标

在上面的示例中提到了坐标。在 Canvas 中绘制图形时，需要为图形指定具体位置，fillRect(50,25,100,50)中的前两个参数便是用于指定位置的 x 轴和 y 轴的坐标值。在 Canvas 中，坐标原点（0,0）位于 Canvas 的左上角，x 轴水平向右延伸，y 轴垂直向下延伸，如图 5.4 所示。

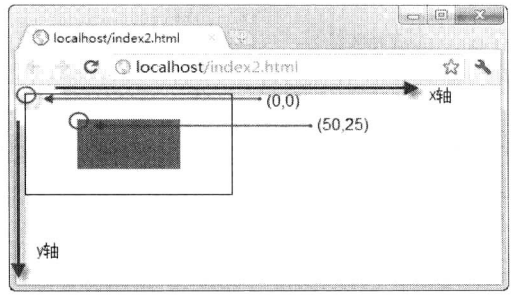


图 5.4　认识 Canvas 坐标

5.1.4　何时不用 Canvas

尽管 canvas 元素可以实现非常强大且复杂的功能，但如果 HTML 页面中有比 canvas 元素更合适

的元素存在，则不必再用 canvas。例如，用 canvas 元素来渲染 HTML 页面的标题样式标签（如 h1、h2 等）就不太合适。

5.1.5 如果浏览器不支持 Canvas

有些浏览器可能不支持 canvas 元素，因此就需要为这些浏览器提供替代显示的内容。方法比较简单，只需要直接在 canvas 元素内插入替代内容即可。不支持 canvas 的浏览器会忽略 canvas 元素而直接显示替代内容，支持 Canvas 的浏览器则会正常地渲染 Canvas。

【示例】把一行说明文字或者一幅替代图片插入 canvas 元素内，作为替代显示的内容。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100">
您的浏览器不支持 canvas 元素，请更新或更换您的浏览器。
</canvas>
</body>
</html>
```

以上代码在 IE6 浏览器中的运行效果如图 5.5 所示。因其不支持 canvas 元素，所以显示了 canvas 元素中插入的替代文本。

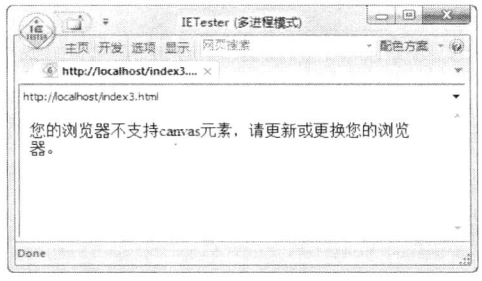


图 5.5 显示 canvas 元素中插入的替代文本

5.1.6 检测浏览器支持

除了用上述方法在不支持 canvas 的浏览器中显示替代文本之外，还可以使用 JavaScript 脚本来检测浏览器是否支持 canvas，方法是判断 getContext 函数是否存在。

【示例】以下代码在 IE6 浏览器中的运行效果如图 5.6 所示，因其不支持 canvas 元素，所以显示“您的浏览器不支持 Canvas!”。而在 Chrome 浏览器中的运行结果如图 5.7 所示，显示的是“您的浏览器支持 Canvas!”。当然也可以用 document.write 方法在网页中显示类似的信息。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="200" height="100">
<!--此处放置用于绘制图形的 JavaScript 代码。-->
</canvas>
<script type="text/javascript">
var canvas = document.getElementById("myCanvas");
```

```
if (canvas.getContext){
    alert("您的浏览器支持 Canvas!");
} else {
    alert("您的浏览器不支持 Canvas!");
}
</script>
</body>
</html>
```

图 5.6　显示“您的浏览器不支持 Canvas!”

图 5.7　显示“您的浏览器支持 Canvas!”

5.2　绘制简单图形

“万丈高楼平地起”，任何事物都是从最简单最基础的部分开始筑建，但最终却会形成或复杂或庞大的结构。HTML5 的 canvas 能够实现最简单直接的绘图，也能通过编写脚本实现极为复杂的应用，如各种美妙绝伦的图形、精彩纷呈的游戏等。本节首先介绍如何使用 Canvas 和 JavaScript 实现最简单的图形绘制，包括绘制直线、矩形和圆形，一起来初窥 Canvas 中绘图的基本原理和方法。

5.2.1　绘制直线

绘制直线可以使用下面示例中的 JavaScript 代码。其中调用了 3 个方法：moveTo、lineTo 和 stroke。

【示例】本示例绘制了一条贯穿画布的对角线，代码在 Chrome 浏览器中的运行效果如图 5.8 所示。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="200" height="100"></canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.moveTo(0,0);
context.lineTo(200,100);
context.stroke();
</script>
</body>
</html>
```

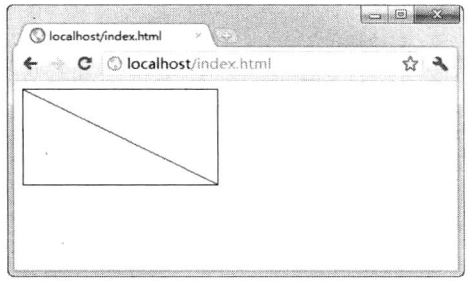


图 5.8　绘制直线

moveTo 方法用于建立新的子路径，并规定其起始点为（x,y），用法如下。

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

lineTo 方法用于从 moveTo 方法规定的起始点开始绘制一条到规定坐标的直线，如果前面没有用 moveTo 方法规定子路径的起始点，则 lineTo 方法等同于 moveTo 方法。例如，在上面的示例中，lineTo 方法绘制了一条从（0,0）到（200,100）的直线。lineTo 方法的用法如下。

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

上面的两个方法规定了直线路径的起点和终点，而 stroke 方法用于沿该路径绘制一条直线。

5.2.2　绘制矩形

绘制矩形可以使用下面示例中的 JavaScript 代码。

【示例 1】以下代码绘制了一个大小为 200×100 像素的粉红色矩形，且左上角坐标为（0,0），在 Chrome 浏览器中的运行效果如图 5.9 所示。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="150"></canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.fillStyle="#FF00FF";
context.fillRect(0,0,200,100);
</script>
</body>
</html>
```

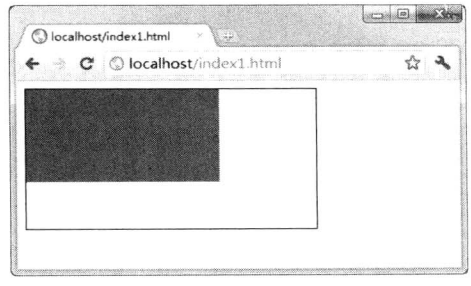


图 5.9　绘制矩形

在上面的示例代码中，fillStyle 属性用于指定要绘制的图形的填充颜色，而 fillRect 方法用于以指定的颜色填充一个矩形，括号中的参数用于指定矩形的坐标位置与大小。

除了可以使用 fillStyle 和 fillRect，还可以使用 strokeStyle 和 strokeRect。fill 是填充，而 stroke 是轮廓，所以用 strokeStyle 和 strokeRect 方法绘制的是矩形轮廓。例如，在以上代码中添加下面的代码可以在粉红色矩形上再绘制一个白色的矩形轮廓。

Note

```
context.strokeStyle="#FFFFFF";
context.strokeRect(0,0,100,50);
```

以上代码在 Chrome 浏览器中的运行效果如图 5.10 所示，在粉红色矩形上方绘制了一个大小为 100×50 的白色矩形轮廓，且左上角坐标为（0,0）。

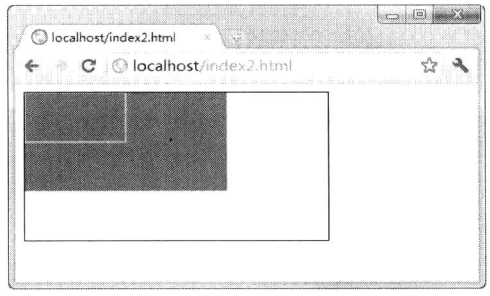


图 5.10　绘制矩形

在使用 fillStyle 或 strokeStyle 指定颜色值时，用于表示颜色值的是符合 CSS3 标准的字符串。

【示例 2】下面这些例子中 fillStyle 表示的是同一种颜色。

```
context.fillStyle="#FF0000";
context.fillRect(0,0,10,10);
context.fillStyle="red";
context.fillRect(20,20,10,10);
context.fillStyle="rgb(255,0,0)";
context.fillRect(40,40,10,10);
context.fillStyle="rgb(100%,0%,0%)";
context.fillRect(60,60,10,10);
context.fillStyle="rgba(255,0,0,1)";
context.fillRect(80,80,10,10);
```

综上所述，可以看出绘制矩形有两种方法，即 fillRect 和 strokeRect，前者用于绘制用颜色填充区域的矩形，后者用于绘制轮廓（或线条）。而为图形指定颜色用到了两个属性，即 fillStyle 和 strokeStyle，前者用于指定要绘制的填充区域的颜色，后者用于指定要绘制的轮廓（或线条）的颜色。

5.2.3　绘制圆形

接下来，介绍如何绘制圆形。在绘制圆形时，会用到 beginPath、arc、closePath 和 fill 这 4 个方法。

【示例 1】下面以示例进行说明，代码如下。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="150">
```

```
</canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.fillStyle="#FF00FF";
context.beginPath();
context.arc(100,75,50,0,Math.PI*2,true);
context.closePath();
context.fill();
</script>
</body>
</html>
```

以上代码在 300×150 像素大小的画布上绘制了一个半径为 50 像素的圆形，在 Chrome 浏览器中的运行效果如图 5.11 所示。

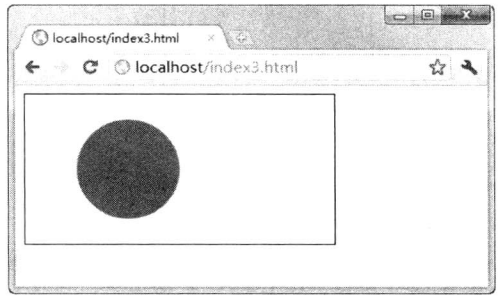


图 5.11　绘制圆形

圆形的绘制仍是采用绘制路径并填充颜色的方法。beginPath 方法用于开始绘制路径，closePath 方法用于结束绘制路径。调用 beginPath 方法后，在 Canvas 中进行一系列图形的绘制，绘制完成后，应该使用 closePath 方法将图形闭合起来。在这两个语句中间的 arc 方法用于绘制圆形，其用法如下。

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

arc 方法本意为用于绘制弧线，当采用适当的参数后，即可绘制圆形。参数中的 x，y 为起点坐标，radius 为圆形的半径，startAngle 为开始的角度，endAngle 为结束的角度，anticlockwise 为是否按顺时针方向进行绘制。

【示例 2】为了更好地理解这些方法，再看下面的示例。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="150">
</canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
for(var i=0;i<15;i++){
    context.strokeStyle="#FF00FF";
    context.beginPath();
    context.arc(0,150,i*10,0,Math.PI*3/2,true);
```

```
    context.stroke();
}
</script>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行效果如图 5.12 所示。在示例 2 中要用到 stroke 方法，该方法用于在显示设备中输出线条，而示例 1 中用 fill 方法输出的是填充颜色。这里没有用 closePath 方法，因为该方法会在路径不闭合时，补上一条线将路径闭合起来。例如，如果在以示例 2 中的 context.stroke();语句前添加 context.closePath();语句，则会得到如图 5.13 所示的输出结果。

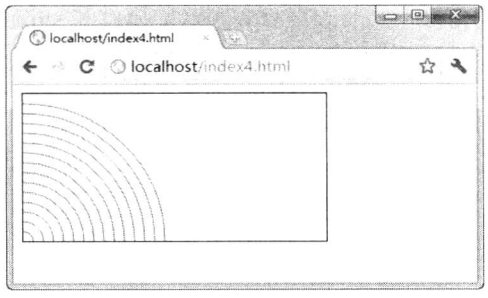


图 5.12　绘制弧线

图 5.13　封闭路径

5.2.4　绘制三角形

综合运用上述的一些方法和属性，使用绘制路径的方法可以自由绘制其他形状，如三角形。请看下面的示例。

【示例 1】以下代码可用于绘制实心三角形。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="200" height="200"></canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.fillStyle="red";
context.beginPath();
context.moveTo(25,25);
context.lineTo(150,25);
context.lineTo(25,150);
context.fill();
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.14 所示。

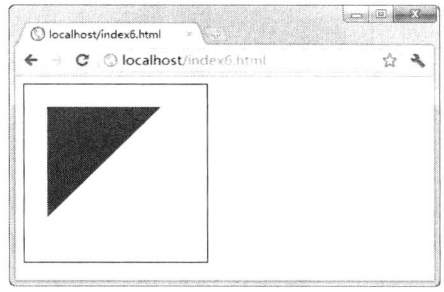


图 5.14　绘制实心三角形

【示例 2】如果要绘制空心三角形，即只有轮廓的三角形，则要改用 strokeStyle 属性和 stroke 方法，代码如下。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="200" height="200"></canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.strokeStyle="red";
context.beginPath();
context.moveTo(25,25);
context.lineTo(150,25);
context.lineTo(25,150);
context.closePath();
context.stroke();
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.15 所示。

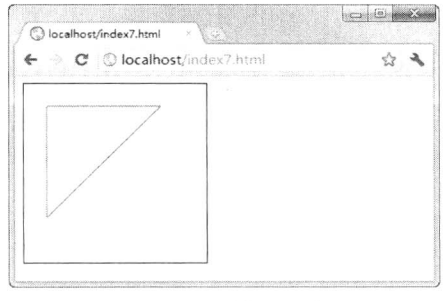


图 5.15　绘制空心三角形

5.2.5　清空画布

在 Canvas 中绘制了一些图形后，许多时候可能需要清除这些图形，就像一些绘图程序中的橡皮工具的功能。使用 clearRect 方法可清除指定的矩形区域内的所有图形，显示出画布的背景，该方法用法如下。

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

【示例】可以通过下面的示例来体验如何清空画布。

```
<!DOCTYPE html>
<html>
<script type="text/javascript">
function clearMap(){
    context.clearRect(0,0,300,200);
}
</script>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="200">
</canvas>
<script type="text/javascript">
    var c=document.getElementById("myCanvas");
    var context=c.getContext("2d");
    context.strokeStyle="#FF00FF";
    context.beginPath();
    context.arc(200,150,100,-Math.PI*1/6,-Math.PI*5/6,true);
    context.stroke();
</script></br>
<input name="" type="button"   value="清空画布" onClick="clearMap();">
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行效果如下。先是在画布上绘制一段弧线，如图 5.16 所示，如果单击“清空画布”按钮，则会清除这段弧线，如图 5.17 所示。

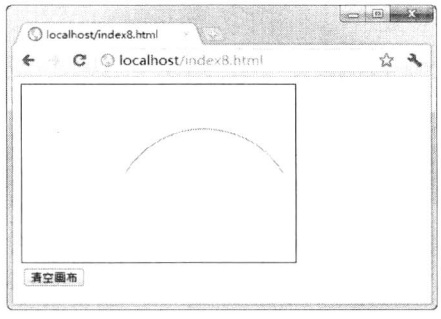


图 5.16　绘制弧线

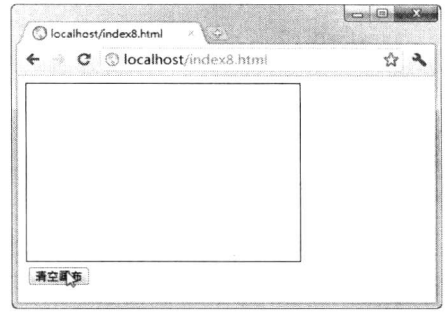


图 5.17　清空画布

5.3　绘制贝塞尔曲线

相对于绘制直线、矩形、圆形等简单图形而言，绘制曲线是比较有挑战性的，但是一旦洞悉其原理，就能够创建出许多复杂的图形。贝塞尔曲线在计算机图形学中的作用至关重要，其应用也非常广泛，如在一些数学软件、矢量绘图软件和三维动画软件中，经常会见到贝塞尔曲线，主要用于数值分析领域或产品设计和动画制作领域。本节介绍如何在 Canvas 中绘制贝塞尔曲线，包括二次方曲线和三次方曲线。有关贝塞尔曲线的知识可以参考相关书籍或维基百科中的条目“贝塞尔曲线”。

5.3.1 绘制二次方贝塞尔曲线

Note

quadraticCurveTo 方法用于绘制二次方贝塞尔曲线，其用法如下。

```
context.quadraticCurveTo(cp1x, cp1y, x, y);
```

其中参数 cp1x 和 cp1y 是控制点的坐标，x 和 y 是终点坐标。

【示例】本示例不但绘制了一条二次方贝塞尔曲线，还绘制出了其控制点和控制线。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="200"></canvas>
<script type="text/javascript">
    var c=document.getElementById("myCanvas");
    var context=c.getContext("2d");
    //下面开始绘制二次方贝塞尔曲线
    context.strokeStyle="dark";
    context.beginPath();
    context.moveTo(0,200);
    context.quadraticCurveTo(75,50,300,200);
    context.stroke();
    context.globalCompositeOperation="source-over";
    //下面绘制的直线用于表示上面曲线的控制点和控制线，控制点坐标即两直线的交点（75,50）
    context.strokeStyle="#ff00ff";
    context.beginPath();
    context.moveTo(75,50);
    context.lineTo(0,200);
    context.moveTo(75,50);
    context.lineTo(300,200);
    context.stroke();
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.18 所示，其中曲线即为二次方贝塞尔曲线，两条直线为控制线，两直线的交点即曲线的控制点。

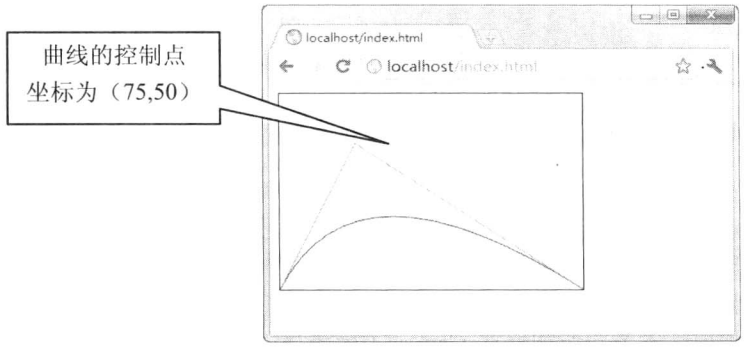


图 5.18　二次方贝塞尔曲线及其控制点

5.3.2　绘制三次方贝塞尔曲线

bezierCurveTo 方法用于绘制三次方贝塞尔曲线，其用法如下。

Note

```
context.bezierCurveTo(cp1x,cp1y,cp2x,cp2y, x, y);
```

其中参数 cp1x 和 cp1y 是第一个控制点的坐标，cp2x 和 cp2y 是第二个控制点的坐标，x 和 y 是终点坐标。

【示例】本示例不但绘制了一条三次方贝塞尔曲线，还绘制出了两个控制点和两条控制线。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="200"></canvas>
<script type="text/javascript">
    var c=document.getElementById("myCanvas");
    var context=c.getContext("2d");
    //下面开始绘制三次方贝塞尔曲线
    context.strokeStyle="dark";
    context.beginPath();
    context.moveTo(0,200);
    context.bezierCurveTo(25,50,75,50,300,200);
    context.stroke();
    context.globalCompositeOperation="source-over";
    //下面绘制的直线用于表示上面曲线的控制点和控制线，控制点坐标为（25,50）和（75,50）
    context.strokeStyle="#ff00ff";
    context.beginPath();
    context.moveTo(25,50);
    context.lineTo(0,200);
    context.moveTo(75,50);
    context.lineTo(300,200);
    context.stroke();
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.19 所示，其中曲线即为三次方贝塞尔曲线，两条直线为控制线，两直线上方的端点即为曲线的控制点。

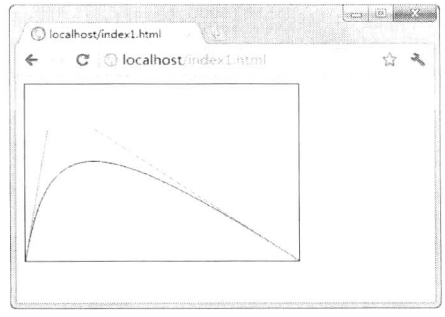


图 5.19　三次方贝塞尔曲线

5.4 图形的变换

Note

适当地运用图形的变换（Transformations，如旋转和缩放图形等），可以创建出大量复杂多变的图形。在了解图形的变换之前，应该首先了解一下 Canvas 状态的保存与恢复。

5.4.1 保存与恢复 Canvas 状态

Canvas 状态指的是当前画面所有样式、变形和裁切的一个快照，以堆（stack）的方式保存。save 和 restore 方法用于保存和恢复 Canvas 状态，这两个方法都不需要任何参数，用法如下。

```
context.save();
context.restore();
```

save 方法可以暂时将当前的状态保存到堆中，这些状态可以是各种属性（如 strokeStyle、fillStyle 和 globalCompositeOperation 等）的值、当前应用的变形、当前裁切的路径等。restore 方法用于将上一个保存的状态从堆中再次取出，恢复该状态的所有设置。

【示例】在本示例中，首先绘制一个矩形，填充颜色为#ff00ff，轮廓颜色为蓝色，然后保存这个状态，再绘制另外一个矩形，填充颜色为#ff0000，轮廓颜色为绿色，最后恢复第一个矩形的状态，并绘制两个小的矩形，则其中一个矩形填充颜色必为#ff00ff，另外矩形轮廓颜色必为蓝色，因为此时已经恢复了原来保存的状态，所以会沿用最先设定的属性值。

```
<!DOCTYPE html>
<html>
<body>
<head>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="200"></canvas>
<script type="text/javascript">
    var c=document.getElementById("myCanvas");
    var context=c.getContext("2d");
    //开始绘制矩形
    context.fillStyle="#ff00ff";
    context.strokeStyle="blue";
    context.fillRect(20,20,100,100);
    context.strokeRect(20,20,100,100);
    context.fill();
    context.stroke();
    //保存当前 Canvas 状态
    context.save();
    //绘制另外一个矩形
    context.fillStyle="#ff0000";
    context.strokeStyle="green";
    context.fillRect(140,20,100,100);
    context.strokeRect(140,20,100,100);
    context.fill();
    context.stroke();
    //恢复第一个矩形的状态
```

```
        context.restore();
        //绘制两个矩形
        context.fillRect(20,140,50,50);
        context.strokeRect(80,140,50,50);
    </script>
    </body>
    </html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.20 所示，可以尝试将 context.restore();一行删除，然后再查看代码的运行效果，比较一下有何不同。

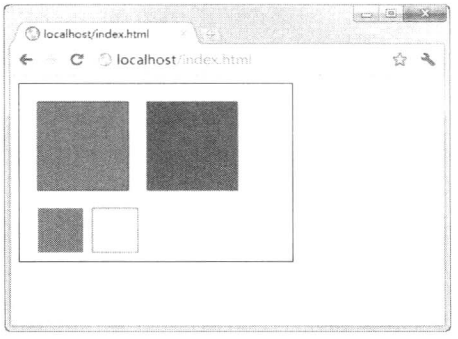


图 5.20　保存与恢复 Canvas 状态

5.4.2　移动坐标空间

前面讲过，画布的坐标空间默认以画布左上角（0,0）为原点，x 轴水平向右为正向，y 轴垂直向下为正向，该坐标空间的单位通常为像素。在绘制图形时，可以使用 translate 方法移动坐标空间，使画布的变换矩阵发生水平和垂直方向的偏移，其用法如下。

```
context.translate(dx, dy);
```

其中 dx 和 dy 分别为坐标原点沿水平和垂直两个方向的偏移量，如图 5.21 所示。

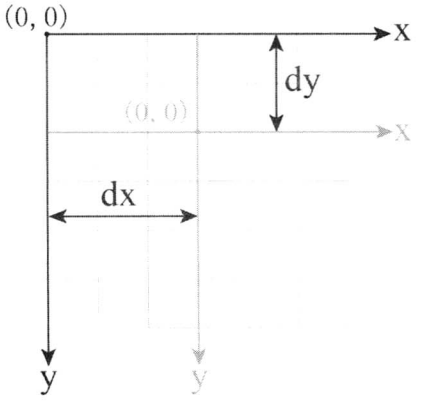


图 5.21　坐标空间的偏移示意图

在进行图形变换之前，最好养成先使用 save 方法保存当前状态的好习惯。在许多情况下，使用 restore 方法来自动恢复原来的状态要比手动恢复更加高效，特别是当重复某种运算时。

【示例】在本示例中，综合运用了 save、restore 和 translate 方法来绘制一个伞状图形。仔细阅读代码并体会一些关键语句的用途，有助于更好地理解。

Note

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function drawTop(ctx, fillStyle){
    ctx.fillStyle = fillStyle;
    ctx.beginPath();
    ctx.arc(0, 0, 30, 0,Math.PI,true);
    ctx.closePath();
    ctx.fill();
}
function drawGrip(ctx){
    ctx.save();
    ctx.fillStyle = "blue";
    ctx.fillRect(-1.5, 0, 1.5, 40);
    ctx.beginPath();
    ctx.strokeStyle="blue";
    ctx.arc(-5, 40, 4, Math.PI,Math.PI*2,true);
    ctx.stroke();
    ctx.closePath();
    ctx.restore();
}
function draw(){
    var ctx = document.getElementById('myCanvas').getContext("2d");
    //注意：所有的移动都是基于这一上下文
    ctx.translate(80,80);
    for (var i=1;i<10;i++){
        ctx.save();
        ctx.translate(60*i, 0);
        drawTop(ctx,"rgb("+(30*i)+","+(255-30*i)+",255)");
        drawGrip(ctx);
        ctx.restore();
    }
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="700" height="300"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.22 所示。可见，Canvas 中图形移动的实现，其实是通过改变画布的坐标原点来实现的，所谓的“移动图形”，只是看上去被移动了的样子，实际移动的是坐标空间。领会并掌握这种方法对于随心所欲地绘制图形非常有帮助。

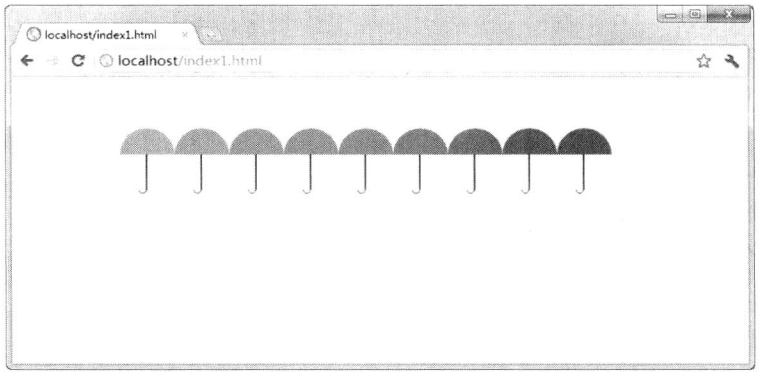


图 5.22　移动坐标空间

5.4.3　旋转坐标空间

rotate 方法用于以原点为中心旋转 Canvas，实质仍是旋转 Canvas 上下文对象的坐标空间，其用法如下。

```
context.rotate(angle);
```

rotate 方法只有一个参数，即旋转角度 angle，旋转角度以顺时针方向为正方向，以弧度为单位，旋转中心为 Canvas 的原点，如图 5.23 所示。

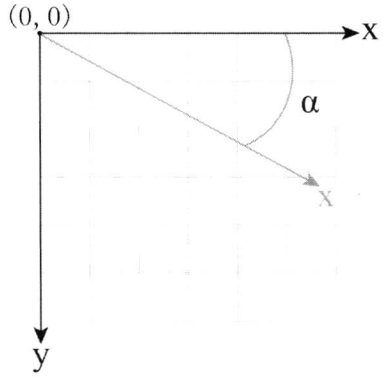


图 5.23　以原点为中心旋转 Canvas

【示例】以下代码在 Chrome 浏览器中的运行效果如图 5.24 所示。在本例中，每次开始绘制图形之前，先将坐标空间旋转 PI*（2/4+i/4），再将坐标空间沿 y 轴负方向移动 100，然后开始绘制图形，从而实现使图形沿一中心点平均旋转分布。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function drawTop(ctx, fillStyle){
    ctx.fillStyle = fillStyle;
    ctx.beginPath();
    ctx.arc(0,0,30,0,Math.PI,true);
    ctx.closePath();
```

```
        ctx.fill();
}
function drawGrip(ctx){
        ctx.save();
        ctx.fillStyle = "blue";
        ctx.fillRect(-1.5, 0, 1.5, 40);
        ctx.beginPath();
        ctx.strokeStyle="blue";
        ctx.arc(-5, 40, 4, Math.PI,Math.PI*2,true);
        ctx.stroke();
        ctx.closePath();
        ctx.restore();
}
function draw(){
        var ctx = document.getElementById('myCanvas').getContext("2d");
        ctx.translate(150,150);
        for (var i=1;i<9;i++)
        {
            ctx.save();
            ctx.rotate(Math.PI*(2/4+i/4));
            ctx.translate(0,-100);
            drawTop(ctx,"rgb("+(30*i)+","+(255-30*i)+",255)");
            drawGrip(ctx);
            ctx.restore();
        }
}
window.onload=function(){
        draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="700" height="300"></canvas>
</body>
</html>
```

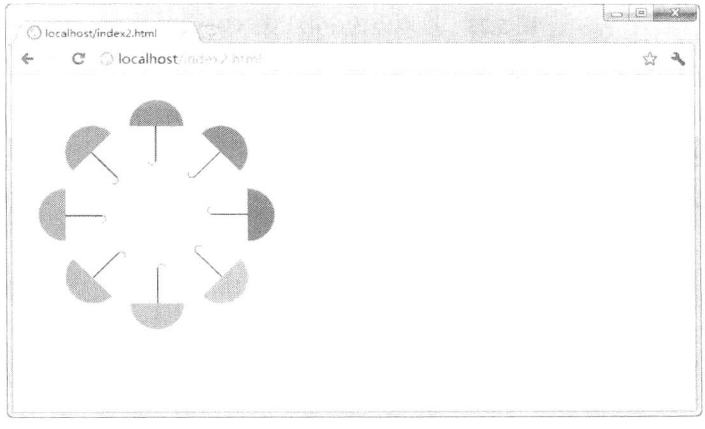


图 5.24　旋转坐标空间

5.4.4　缩放图形

Note

scale 方法用于增减 Canvas 上下文对象中的像素数目，从而实现图形或位图的放大或缩小，其用法如下。

```
context.scale(x,y);
```

其中 x，y 为必须接受的参数，x 为横轴的缩放因子，y 轴为纵轴的缩放因子，它们的值必须是正值。如果需要放大图形，则将参数值设置为大于 1 的数值，如果需要缩小图形，则将参数值设置为小于 1 的数值，当参数值等于 1 时则没有任何效果。

【示例】以下代码在 Chrome 浏览器中的运行效果如图 5.25 所示。在本例中，使用 scale(0.95,0.95) 来缩小图形到上次的 0.95，共循环 80 次，同时移动和旋转坐标空间，从而实现图形呈螺旋状由大到小的变化。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw(){
    var ctx = document.getElementById('myCanvas').getContext("2d");
    ctx.translate(200,20);
    for (var i=1;i<80;i++){
        ctx.save();
        ctx.translate(30,30);
        ctx.scale(0.95,0.95);
        ctx.rotate(Math.PI/12);
        ctx.beginPath();
        ctx.fillStyle="red";
        ctx.globalAlpha="0.4";
        ctx.arc(0,0,50,0,Math.PI*2,true);
        ctx.closePath();
        ctx.fill();
    }
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="700" height="300"></canvas>
</body>
</html>
```

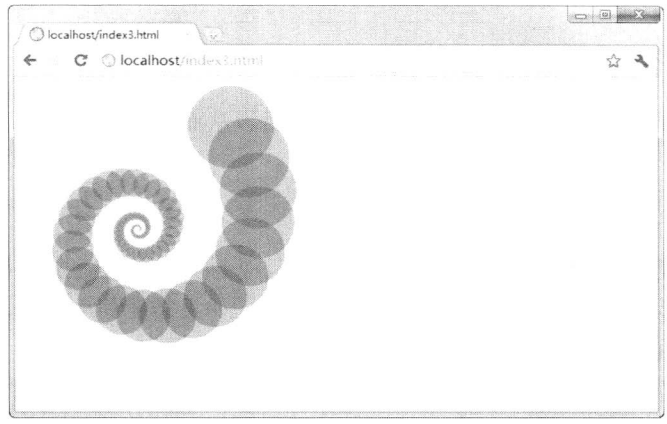


图 5.25　缩放图形

5.4.5　矩阵变换

transform 方法用于直接对变形矩阵作修改，即进行矩阵变换。矩阵变换常用于坐标变换不能达到预期效果的情况，能够实现比普通的坐标变换更为复杂的变形。

变换矩阵是数学线性代数中的一个概念，在计算机图形学中，这个矩阵能够实现二维图形的变形。Canvas 的上下文对象事实上便是创建了一个变换矩阵，在这个上下文对象中，一个元素经过渲染后可以得到一张位图，通过对这个矩阵进行变换，即对这个位图上每一点进行变换，从而使图形产生诸如平移、缩放、旋转、切变以及镜像反射等效果。目前，多数新版的浏览器已经支持 2D 的矩阵变换，但 3D 的矩阵变换还要过一段时间才能实现。transform 方法的用法如下。

```
context.transform(m11, m12, m21, m22, dx, dy);
```

该方法必须将当前的变换矩阵与下面的矩阵进行乘法运算。

```
m11     m21     dx
m12     m22     dy
0       0       1
```

要注意 m11、m12、m21、m22、dx 和 dy 这几个参数，它们是竖着排列的。通过下面的基本公式，我们可以了解到矩阵变换是如何计算的。具体的原理要参考有关矩阵乘法的知识，在此不作赘述。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m11 & m21 & dx \\ m12 & m22 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} (m11)x + (m21)y + dx \\ (m12)x + (m22)y + dy \\ 1 \end{bmatrix}$$

其中 x，y 为初始坐标，x'和 y'是矩阵变换之后得到的新坐标。也就是说，所谓的矩阵变换即使用变换矩阵对初始坐标进行变换，从而得到新的坐标，进而实现平移、缩放、旋转、切变以及镜像反射等效果。如果按照矩阵乘法的运算法则，可以将上述基本公式作以简化，即：

```
x'=(m11)x+(m21)y+dx
y'=(m12)x+(m22)y+dy
```

再来看前面讲过的移动（translate）、缩放（scale）和旋转（rotate）坐标空间，这 3 种方法其实都是矩阵变换的特例。下面使用矩阵变换来分别解释一下这 3 种方法。

Note

1. 移动（translate）

translate(x,y)可以用下面的 transform 方法来代替：

```
context.transform(0,1,1,0,dx,dy);
```

或

```
context.transform(1,0,0,1,dx,dy);
```

如果将这些参数值代入简化的基本公式，则以上的形式都可以这样来表示：

```
x'=x+dx
y'=y+dy
```

其中 dx 为原点沿 x 轴移动的数值，dy 为原点沿 y 轴移动的数值。

2. 缩放（scale）

scale(x,y)可以用下面的 transform 方法来代替：

```
context.transform(m11,0,0,m22,0,0);
```

或

```
context.transform(0,m12,m21,0,0,0);
```

如果将这些参数值代入简化的基本公式，则以上的形式都可以这样来表示：

```
x'=(m11)x
y'=(m22)y
```

或

```
x'=(m12)x
y'=(m21)y
```

此处 dx、dy 都为 0 表示坐标原点不变。m11、m22 或 m12、m21 为沿 x、y 轴放大的倍数。

3. 旋转（rotate）

rotate(angle)比较复杂一些，需要用到三角函数的知识，可以用下面的 transform 方法来代替：

```
context.transform(cosθ,sinθ,- sinθ, cosθ,0,0);
```

其中的 θ 为旋转角度的弧度值，dx、dy 都为 0 表示坐标原点不变。

如果将这些参数值代入简化的基本公式，则以上的形式都可以这样来表示：

```
x'=x* cosθ-y* sinθ
y'=x* sinθ+ y* cosθ
```

下面根据以上分析来替换前面介绍过的“缩放图形”示例的代码，那么应该可以用

```
ctx.transform(0.95,0,0,0.95,30,30);
```

来代替

```
ctx.translate(30,30);ctx.scale(0.95,0.95);
```

读者可以自行验证一下。但是先不要期望用一个 transform 方法来代替以下 3 条语句，因为 transform 方法非常复杂，有时间的话可以多参考一些矩阵变换的文章或书籍，以及计算机图形学的知识，然后再深入研究和实践如何实现更高级的应用。

```
ctx.translate(30,30);
ctx.scale(0.95,0.95);
ctx.rotate(Math.PI/12);
```

Note

setTransform 方法用于将当前的变换矩阵重置为最初的矩阵，然后以相同的参数调用 transform 方法，即先 set（重置）再 transform（变换），用法如下所示。

```
context.setTransform(m11, m12, m21, m22, dx, dy);
```

【示例】本示例使用 setTransform 方法将前面已经发生变换的矩阵首先重置为最初的矩阵，即恢复最初的原点，然后再将坐标原点改为（10,10），并以新的坐标为基准绘制一个蓝色的矩形。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw(){
    var ctx = document.getElementById('myCanvas').getContext("2d");
    ctx.translate(200,20);
    for (var i=1;i<90;i++){
        ctx.save();
        ctx.transform(0.95,0,0,0.95,30,30);
        ctx.rotate(Math.PI/12);
        ctx.beginPath();
        ctx.fillStyle="red";
        ctx.globalAlpha="0.4";
        ctx.arc(0,0,50,0,Math.PI*2,true);
        ctx.closePath();
        ctx.fill();
    }
    ctx.setTransform(1,0,0,1,10,10);
    ctx.fillStyle="blue";
    ctx.fillRect(0,0,50,50);
    ctx.fill();
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="700" height="300"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.26 所示。在本例中，使用 transform(0.95,0,0,0.95,30,30)来缩小图形到上次的 0.95，共循环 80 次，同时移动和旋转坐标空间，从而实现图形呈螺旋状由大到小的变化。

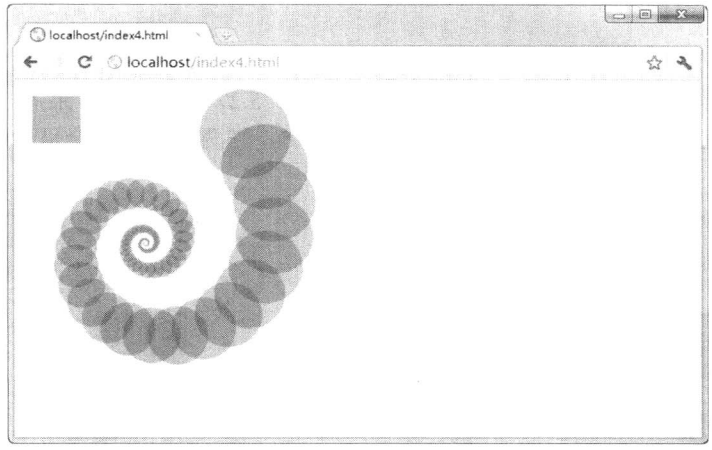


图 5.26　矩阵重置并变换

Note

5.5　图形的组合与裁切

当在 Canvas 中绘制多个图形时，可以指定图形的组合（Compositing）方式，使其呈现出不同的效果。本节通过示例介绍在 Canvas 中进行图形组合与裁切的方法与技巧。

5.5.1　图形的组合

当两个或两个以上的图形存在重叠区域时，默认情况下一个图形画在前一个图形之上。通过指定图形 globalCompositeOperation 属性的值可以改变图形的绘制顺序或绘制方式，从而实现更多种可能。此外，还可以使用 globalAlpha 属性来指定图形的透明度。

【示例】请看下面的示例，在本例中，设置所有图形的透明度为 1，即不透明，可以在 0~1 上取值从而改变图形的透明度。globalCompositeOperation 属性的值指定为 source-over，即默认设置，新的图形会覆盖在原有图形之上，也可以指定其他值，详见表 5.1。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="200"></canvas>
<script type="text/javascript">
    var c=document.getElementById("myCanvas");
    var context=c.getContext("2d");
    context.fillStyle="red";
    context.fillRect(50,25,100,100);
    context.fillStyle="green";
    context.globalCompositeOperation="source-over";
    context.beginPath();
    context.arc(150,125,50,0,Math.PI*2,true);
    context.closePath();
    context.fill();
</script>
```

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

以上代码在 Chrome 浏览器中的运行效果如图 5.27 所示。如果在上面代码中添加：context.globalAlpha=0.5;，即设置 globalAlpha 的值为 0.5，则两个图形都会呈现为半透明效果，如图 5.28 所示。

Note

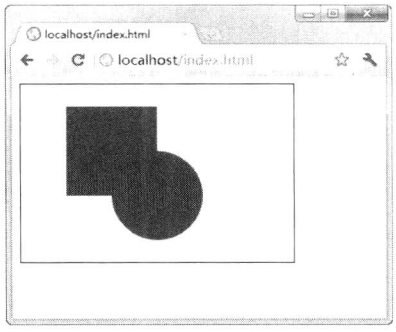


图 5.27　图形的组合

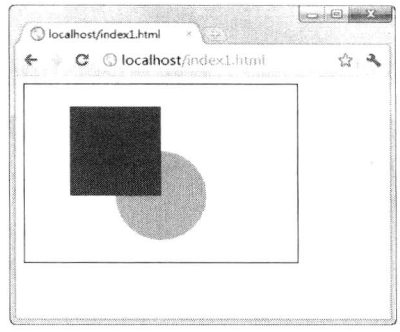
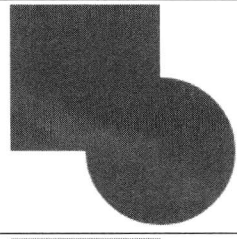
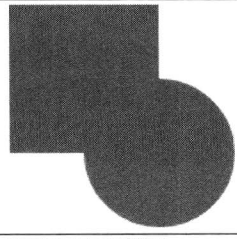
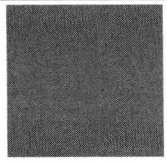


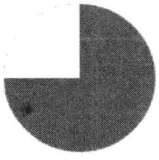
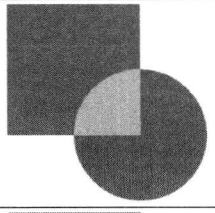
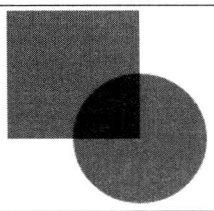
图 5.28　半透明效果

表 5.1 中列出了 globalCompositeOperation 属性所有可用的值。灵活使用这些属性值，必能创造出丰富多彩的图形。表中的图例矩形表示为 B，为先绘制的图形（原有内容为 destintation），圆形表示为 A，为后绘制的图形（新图形为 source）。在应用时注意 globalCompositeOperation 语句的位置应处在原有内容与新图形之间。Chrome 浏览器支持大多数属性值，无效的属性值在表中已经标出。Opera 浏览器对这些属性值的支持相对来说更好一些。

表 5.1　globalCompositeOperation 属性所有可用的值

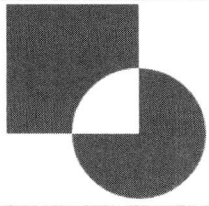
属　性　值	图形合成示例	说　　明
source-over（默认值）		A over B，这是默认设置，即新图形覆盖在原有内容之上
destination-over		B over A，即原有内容覆盖在新图形之上
source-atop		只绘制原有内容和新图形与原有内容重叠的部分，且新图形位于原有内容之上

续表

属　性　值	图形合成示例	说　　明
destination-atop		只绘制新图形和新图形与原有内容重叠的部分，且原有内容位于重叠部分之下
source-in		新图形只出现在与原有内容重叠的部分，其余区域变为透明
destination-in		原有内容只出现在与新图形重叠的部分，其余区域为透明
source-out		新图形中与原有内容不重叠的部分被保留
destination-out		原有内容中与新图形不重叠的部分被保留
lighter		两图形重叠的部分作加色处理
darker		两图形重叠的部分作减色处理

Note

续表

属　性　值	图形合成示例	说　　明
copy		只保留新图形。在 Chrome 浏览器中无效，Opera 11.5 中有效
xor		将重叠的部分变为透明

5.5.2　裁切路径

clip 方法用于裁切路径，其原理与绘制普通 Canvas 图形类似，只不过 clip 的作用是形成一个蒙版，没有被蒙版的区域会被隐藏。

【示例】如果绘制一个圆形，并进行裁切，则圆形之外的区域将不会绘制在 Canvas 上。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="300" height="300"></canvas>
<script type="text/javascript">
function draw(){
    var ctx = document.getElementById('myCanvas').getContext("2d");
    //绘制背景
    ctx.fillStyle="black";
    ctx.fillRect(0,0,300,300);
    ctx.fill();
    //绘制圆形
    ctx.beginPath();
    ctx.arc(150,150,130,0,Math.PI*2,true);
    //裁切路径
    ctx.clip();
    ctx.translate(200,20);
    for (var i=1;i<90;i++){
        ctx.save();
        ctx.transform(0.95,0,0,0.95,30,30);
        ctx.rotate(Math.PI/12);
        ctx.beginPath();
        ctx.fillStyle="red";
        ctx.globalAlpha="0.4";
        ctx.arc(0,0,50,0,Math.PI*2,true);
        ctx.closePath();
        ctx.fill();
```

```
        }
    }
    window.onload=function(){
        draw();
    }
    </script>
    </body>
    </html>
```


Note

以上代码在 Chrome 浏览器中的运行效果如图 5.29 所示。可以看到，只有圆形区域内的螺旋图形被显示出来，其余部分被“遮”住了。

图 5.29　图形的组合

5.6　更多的颜色和样式选项

在前面已经介绍过可以为图形设置填充颜色与轮廓颜色，使用的是 fillStyle 和 strokeStyle 方法。Canvas 支持更多的颜色和样式选项，具体还包括线型、渐变、图案、透明度和阴影。巧妙地运用这些可选项，对于绘制出引人注目的内容非常有帮助。

5.6.1　应用不同的线型

通过设置下面的 4 个属性值，可以为线条应用不同的线型，分别为线条的粗细、端点样式、两线段连接处样式和绘制交点的方式。

```
lineWidth = value
lineCap = type
lineJoin = type
miterLimit = value
```

下面分别通过实例详细介绍这些属性。

1. lineWidth（设置线条的粗细）

lineWidth 直译为“线宽”，即路径中心到两边的距离，也就是线条的粗细。lineWidth 属性的值必

须为正数，默认为 1.0。

【示例 1】在本示例中，使用 for 循环从画布上方到下方绘制了 12 条线宽依次递增的直线段。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
  var ctx = document.getElementById('myCanvas').getContext('2d');
  for (var i = 0; i < 12; i++){
    ctx.strokeStyle="red";
    ctx.lineWidth = 1+i;
    ctx.beginPath();
    ctx.moveTo(5,5+i*14);
    ctx.lineTo(140,5+i*14);
    ctx.stroke();
  }
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.30 所示。

图 5.30　lineWidth 示例

2. lineCap（设置端点样式）

lineCap 属性的值用于设置线段端点的样式，包括以下 3 种：butt，round 和 square，默认值为 butt。

【示例 2】在本示例中，从上到下绘制了 3 条蓝色的直线段，并依次设置上述 3 种属性值，两侧有两条红色的参考线，这样可以更加清楚地观察端点样式的区别。

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

Note

```
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    var lineCap = ['butt','round','square'];
    //绘制参考线
    ctx.strokeStyle = 'red';
    ctx.beginPath();
    ctx.moveTo(10,10);
    ctx.lineTo(10,150);
    ctx.moveTo(150,10);
    ctx.lineTo(150,150);
    ctx.stroke();
    //绘制直线段
    ctx.strokeStyle = 'blue';
    for (var i=0;i<lineCap.length;i++){
        ctx.lineWidth = 20;
        ctx.lineCap = lineCap[i];
        ctx.beginPath();
        ctx.moveTo(10,30+i*50);
        ctx.lineTo(150,30+i*50);
        ctx.stroke();
    }
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="300" height="300"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.31 所示，可以看到这 3 种端点样式从上到下依次为平头、圆头和方头。

图 5.31　lineCap 示例

3. lineJoin（设置连接处样式）

lineJoin 属性的值用于设置两条线段连接处的样式，包括以下 3 种：round，bevel 和 miter，默认值为 miter。

【示例 3】在本示例中，从左到右绘制了 3 条蓝色的折线，并依次设置上述 3 种属性值，观察拐角处（即直线段连接处）样式的区别。

Note

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    var lineJoin = ['round','bevel','miter'];
    ctx.strokeStyle = 'blue';
    for (var i=0;i<lineJoin.length;i++){
        ctx.lineWidth = 25;
        ctx.lineJoin = lineJoin[i];
        ctx.beginPath();
        ctx.moveTo(10+i*150,30);
        ctx.lineTo(100+i*150,30);
        ctx.lineTo(100+i*150,100);
        ctx.stroke();
    }
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="600" height="300"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.32 所示。

图 5.32　lineJoin 示例

4. miterLimit（设置绘制交点的方式）

miterLimit 属性的值用于设置两条线段连接处交点的绘制方式，当宽线条使用 lineJoin 属性并将其值设置为 miter 时，如果绘制两条线段并以锐角相交，所得到的斜面可能会非常长。当斜面过长时，图形就会显得不协调。miterLimit 属性的作用是为斜面的长度设置一个上限，默认为 10，即规定斜面

的长度不能超过线条宽度的 10 倍。当斜面的长度达到线条宽度的 10 倍时，就会变为斜角。如果 lineJoin 属性值为 round 或 bevel 时，miterLimit 属性无效。

【示例 4】通过本示例，可以观察当角度和 miterLimit 属性值发生变化时斜面长度的变化。在运行代码之前，也可以将 miterLimit 属性值改为固定值，以观察不同的值产生的效果。

Note

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw(){
    var ctx = document.getElementById('myCanvas').getContext('2d');
    for (var i=1;i<10;i++){
        ctx.strokeStyle = 'blue';
        ctx.lineWidth = 10;
        ctx.lineJoin = 'miter';
        ctx.miterLimit = i*10;
        ctx.beginPath();
        ctx.moveTo(10,i*30);
        ctx.lineTo(100,i*30);
        ctx.lineTo(10,33*i);
        ctx.stroke();
    }
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="1600" height="300"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.33 所示。

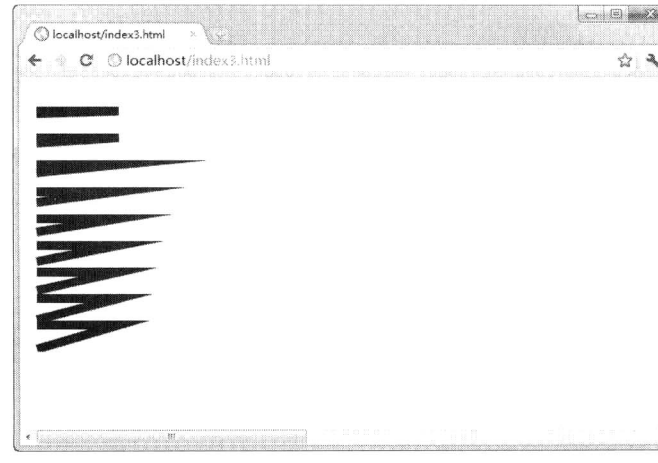


图 5.33　miterLimit 示例

5.6.2 绘制线性渐变

Note

渐变是在各种绘图软件中经常见到的一种为图形填充颜色或描边的方式，在 Canvas 中也可以绘制线性或径向的渐变。如果要绘制线性渐变，首先需要使用 createLinearGradient 方法创建 canvasGradient 对象，然后使用 addColorStop 方法进行上色。createLinearGradient 用法如下。

```
context.createLinearGradient(x1,y1,x2,y2);
```

其中参数 x1、y1 为渐变的起点，x2、y2 为渐变的终点。例如，可以像下面这样来创建一个 canvasGradient 对象。

```
var lineargradient = ctx.createLinearGradient(20,20,150,150);
```

然后即可使用 addColorStop 方法定义色标的位置并进行上色。

```
addColorStop(position, color)
```

其中参数 position 为渐变中色标的相对位置（或称“偏移值”），必须是一个从 0~1 的浮点值。渐变起点的偏移值为 0，终点的偏移值为 1。如果 position 值为 0.5，则表示色标会出现在渐变的正中间。

【示例】下面通过示例来演示如何绘制线性渐变。在本例中共添加了 8 个色标，分别为红、橙、黄、绿、青、蓝、紫、红。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    var lingrad = ctx.createLinearGradient(0,0,0,200);
    lingrad.addColorStop(0, '#ff0000');
    lingrad.addColorStop(1/7, '#ff9900');
    lingrad.addColorStop(2/7, '#ffff00');
    lingrad.addColorStop(3/7, '#00ff00');
    lingrad.addColorStop(4/7, '#00ffff');
    lingrad.addColorStop(5/7, '#0000ff');
    lingrad.addColorStop(6/7, '#ff00ff');
    lingrad.addColorStop(1, '#ff0000');
    ctx.fillStyle = lingrad;
    ctx.strokeStyle = lingrad;
    ctx.fillRect(10,10,200,200);
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="300" height="240"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.34 所示。

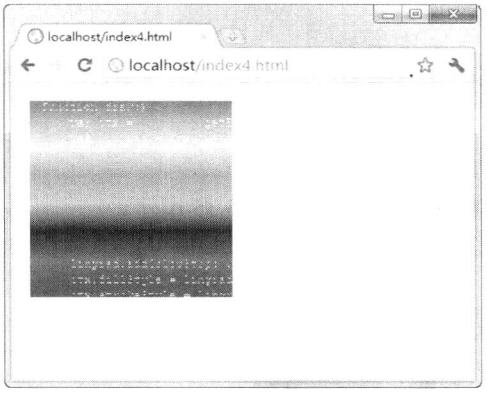


图 5.34　绘制线性渐变

使用 addColorStop 可以添加多个色标，色标的添加也并非一定要从 0 位置开始到 1 位置结束，而是可以在 0~1 之间任意添加，例如，从 0.3 处开始设置一个蓝色色标，再在 0.5 处设置一个红色色标，则从 0~0.3 都会填充为蓝色。从 0.3~0.5 为蓝色到红色的渐变，从 0.5~1 处则填充为红色。上面的示例中没有使用 strokeStyle 属性，但要说明的是，这个属性同样可以接受 Canvas 渐变对象。

5.6.3　绘制径向渐变

如果要绘制径向渐变，则首先需要使用 createRadialGradient 方法创建 canvasGradient 对象，然后使用 addColorStop 方法进行上色。createRadialGradient 用法如下。

```
context.createRadialGradient(x1,y1,r1,x2,y2,r2);
```

其中参数 x1、y1、r1 定义一个以（x1,y1）为原点、半径为 r1 的圆，x2、y2、r2 定义一个以（x2,y2）为原点、半径为 r2 的圆。例如，可以像下面这样来创建一个 canvasGradient 对象用于绘制径向渐变。然后即可使用 addColorStop 方法定义色标的位置并进行上色。

```
var radialgradient = ctx.createRadialGradient(100,100,30,100,100,200);
```

【示例】下面通过示例来演示如何绘制径向渐变。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    var radgrad = ctx.createRadialGradient(55,55,20,100,100,90);
    radgrad.addColorStop(0,'#ffffff');
    radgrad.addColorStop(0.75,'#333333');
    radgrad.addColorStop(1,'#000000');
    ctx.fillStyle = radgrad;
    ctx.fillRect(10,10,200,200);
}
window.onload=function(){
    draw();
```

```
}
</script>
</head>
<body>
<canvas id="myCanvas" width="300" height="240"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.35 所示。

图 5.35　绘制径向渐变

5.6.4　绘制图案

在多数绘图软件中都有填充图案这一功能，而在 Canvas 中，有 createPattern 方法用来实现图案效果，使用起来非常方便，其用法如下。

```
context.createPattern(image,type);
```

其中参数 image 为要引用的 image 对象或另一个 Canvas 对象，type 是所引用对象的平铺类型，必须为下面的字符串值之一。

- ☑ repeat：同时沿 x 轴与 y 轴方向平铺。
- ☑ repeat-x：沿 x 轴方向平铺。
- ☑ repeat-y：沿 y 轴方向平铺。
- ☑ no-repeat：不平铺。

创建图案的步骤与创建渐变有些类似，需要先创建出一个 pattern，然后再赋予其 fillStyle 属性或 strokeStyle 属性。

【示例】在本示例中，以一幅 PNG 格式的图像作为 image 对象创建图案，平铺方式为同时沿 x 轴与 y 轴方向平铺。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    //创建用于图案的新 image 对象
    var img = new Image();
```

Note

```
        img.src = 'images/icon.jpg';
        img.onload = function(){
        //创建图案
        var ptrn = ctx.createPattern(img,'repeat');
        ctx.fillStyle = ptrn;
        ctx.fillRect(0,0,600,600);
        }
}
window.onload=function(){
        draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="600" height="300"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.36 所示。

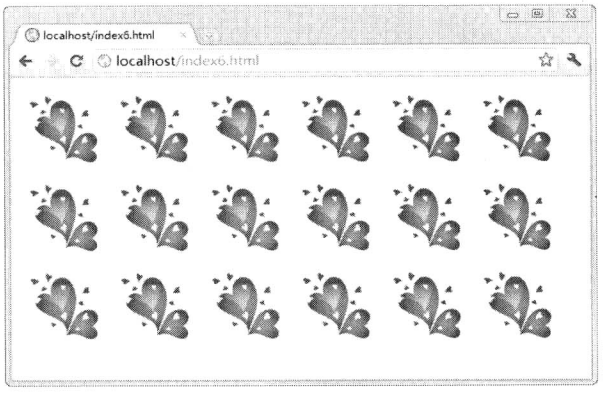


图 5.36　绘制图案

5.6.5　设置图形的透明度

在前面绘制图形的示例中，已经接触过如何使用 globalAlpha 属性设置图形的透明度，该方法适合为大量图形设置相同的透明度。除了使用 globalAlpha 属性之外，还可以通过设置色彩透明度的参数来为图形设置不同的透明度，这种方法相对于使用 globalAlpha 属性来说，更加灵活方便一些。

使用 rgba 方法可以设置具有透明度的颜色，其用法如下。

```
rgba(R,G,B,A)
```

其中 R、G、B 将颜色的红色、绿色和蓝色成分指定为 0~255 之间的十进制整数，A 把 alpha（不透明）成分指定为 0.0~1.0 的一个浮点数值，0.0 为完全透明，1.0 为完全不透明。例如，可以用 "rgba(255,0,0,0.5)"表示半透明的完全红色。

【示例】本示例是由前面的矩阵变换示例修改而来，从中可以看出用 rgba 设置透明度的灵活性和可操作性。

```
<!DOCTYPE html>
```

Note

```
<html>
<head>
<script language="javascript">
function draw(){
    var ctx = document.getElementById('myCanvas').getContext("2d");
    ctx.translate(200,20);
    for (var i=1;i<50;i++){
        ctx.save();
        ctx.transform(0.95,0,0,0.95,30,30);
        ctx.rotate(Math.PI/12);
        ctx.beginPath();
        ctx.fillStyle='rgba(255,0,0,'+(1-(i+10)/40)+')';
        ctx.arc(0,0,50,0,Math.PI*2,true);
        ctx.closePath();
        ctx.fill();
    }
}
window.onload=function(){
    draw();
}
</script>
</head>
<body>
<canvas id="myCanvas" width="700" height="300"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.37 所示。

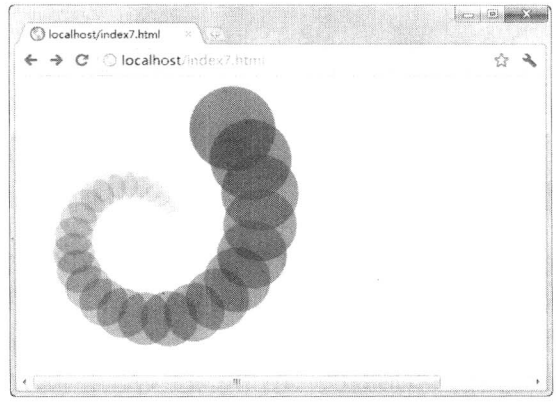


图 5.37　用 rgba 设置透明度

5.6.6　创建阴影

如果要在 Canvas 中创建阴影效果，需要用到下面这 4 个属性：shadowOffsetX、shadowOffsetY、shadowBlur 和 shadowColor，其用法如下。

```
context.shadowOffsetX = float;
context.shadowOffsetY = float;
```

```
context.shadowBlur = float;
context.shadowColor = color;
```

其中，shadowOffsetX 为阴影的水平偏移，shadowOffsetY 为阴影的垂直偏移，默认值都是 0，正值表示向下或向右延伸，负值表示向上或向左延伸。shadowBlur 为阴影羽化的程度，默认值为 0，其数值与像素数量无关，也不受矩阵变换的影响。shadowColor 为阴影的颜色，可以指定为一个 CSS 字符串或 Web 样式字符串，并且可以包含一个 alpha 部分来表示透明度，默认值是黑色。

Note

【示例】下面通过示例来演示如何创建文字阴影效果。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    //设置阴影
    ctx.shadowOffsetX = 3;
    ctx.shadowOffsetY = 3;
    ctx.shadowBlur = 2;
    ctx.shadowColor = "rgba(0, 0, 0, 0.5)";
    //绘制矩形
    ctx.fillStyle = "#33ccff";
    ctx.fillRect(20,20,300,60);
    ctx.fill();
    //绘制文本
    ctx.font = "45px  黑体";
    ctx.fillStyle = "white";
    ctx.fillText("HTML5+CSS3",30, 64);
}
window.onload=function(){
    draw();
}
</script>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body>
<canvas id="myCanvas" width="400" height="200"></canvas>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.38 所示。

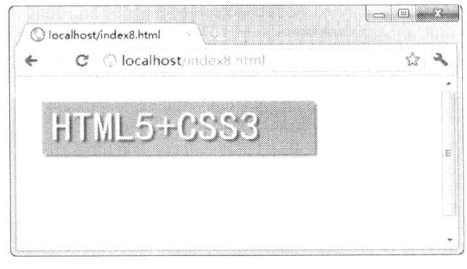


图 5.38　创建阴影

5.7 绘制文字

Note

在 5.6 节的示例中用到了 fillText 方法来绘制文字，本节详细介绍一下在 Canvas 中绘制文字的具体方法。绘制文字使用的是 fillText 方法和 strokeText 方法，分别用于以填充方式和轮廓方式绘制文字。

5.7.1 绘制填充文字

如果要以填充方式绘制文字，可以用 fillText 方法，其用法如下。

```
context.fillText(text,x,y,[maxWidth]);
```

其中 text 为要绘制的字符串，x、y 为绘制文字的横坐标与纵坐标，maxWidth 为可选参数，表示显示文字时的最大宽度。如果要绘制的字符串宽度大于指定的最大宽度，字体就会被调整为更“瘦”的字体（如果字体支持横向压缩）或更小的字号。

【示例】下面是绘制填充文字的一个示例。

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="350" height="200"></canvas>
<script language="javascript">
    var ctx = document.getElementById('myCanvas').getContext('2d');
    ctx.font = "italic 35px  黑体";
    ctx.fillStyle = "Red";
    ctx.fillText("绘制填充文字",30, 60,200);
    ctx.font = "bold 35px  宋体";
    ctx.fillStyle = "Blue";
    ctx.fillText("绘制填充文字",30, 100);
    ctx.font = "25px  隶书";
    ctx.fillStyle = "Green";
    ctx.fillText("绘制填充文字",30, 130);
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.39 所示。

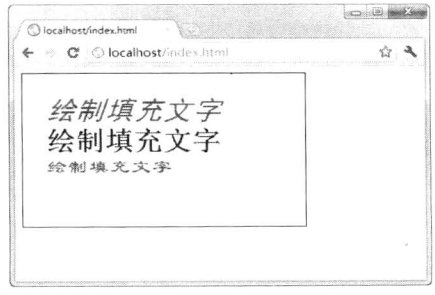


图 5.39　绘制填充文字

5.7.2　文字相关属性

在上例中用到了有关文字的一些属性，下面介绍一下这些属性以及未出现在上面的其他属性的用法。

Note

☑ font：用于指定正在绘制的文字的样式，其语法与 CSS 字体样式的指定方法相同。如果要在绘制文字时改变字体样式，只需要更改该属性的值即可。默认的字体样式为 10px sans-serif。例如，可以像下面这样来指定字体样式。

```
context.font="20pt Times new roman";
```

☑ textAlign：用于指定正在绘制的文字的对齐方式，有 left、right、center、start 和 end 5 种对齐方式，默认值为 start。

- ➢ left：左对齐。
- ➢ right：右对齐。
- ➢ center：居中对齐。
- ➢ start：如果文字从左到右排版则左对齐，从右到左排版则右对齐。
- ➢ end：如果文字从右到左排版则左对齐，从左到右排版则右对齐。

☑ textBaseline：用于指定正在绘制的文字的基线，有 top、hanging、middle、alphabetic、ideographic 和 bottom 6 种属性值，默认值为 alphabetic。

- ➢ top：文本基线与字元正方形空间顶部对齐。
- ➢ hanging：文本基线是悬挂的基线，当前不支持。
- ➢ middle：文本基线位于字元正方形空间的中间位置。
- ➢ alphabetic：指定文本基线为通常的字母基线。
- ➢ ideographic：指定文本基线为表意字基线，即如果表意字符的主体突出到字母基线的下方，则表意字基线与表意字符的底部对齐。
- ➢ bottom：文本基线与字元正方形空间底部的边界框对齐。因为表意基线不能识别下行字符，故可用此种基线来与表意字基线相区分。

当前许多支持 HTML5 的浏览器尚不支持某些属性值，如 hanging 和 ideographic。

5.7.3　绘制轮廓文字

如果要以轮廓方式绘制文字，可以用 strokeText 方法，其用法如下。

```
context.strokeText(text,x,y,[maxWidth]);
```

其中的参数与 fillText 相同，在此不再多作解释。

【示例】下面是绘制轮廓文字的一个示例。

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="350" height="200"></canvas>
<script language="javascript">
    var ctx = document.getElementById('myCanvas').getContext('2d');
```

Note

```
    ctx.font = "italic 35px 黑体";
    ctx.strokeStyle = "Red";
    ctx.strokeText("绘制轮廓文字",30, 60,200);
    ctx.font = "bold 35px 宋体";
    ctx.strokeStyle = "Blue";
    ctx.strokeText("绘制轮廓文字",30, 100);
    ctx.font = "25px 隶书";
    ctx.strokeStyle = "Green";
    ctx.strokeText("绘制轮廓文字",30, 130);
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.40 所示。

图 5.40　绘制轮廓文字

5.7.4　测量文字宽度

使用 measureText 方法可以测量当前所绘制文字中指定文字的宽度，该方法会返回一个 TextMetrics 对象，使用该对象的 width 属性可以得到指定文字参数后所绘制文字的总宽度，其用法如下。

```
metrics=context. measureText(text);
```

其中的参数 text 为要绘制的文字。

【示例】下面是测量文字宽度的一个示例。

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<canvas id="myCanvas" style="border:1px solid;" width="350" height="200"></canvas>
<script language="javascript">
    var ctx = document.getElementById('myCanvas').getContext('2d');
    ctx.font = "bold 20px 楷体";
    ctx.fillStyle="Blue";
    var txt1 = "滚滚长江东逝水，浪花淘尽英雄。";
    ctx.fillText(txt1,10,40);
    var txt2 = "以上字符串的宽度为：";
```

```
    var mtxt1 = ctx.measureText(txt1);
    var mtxt2 = ctx.measureText(txt2);
    ctx.font = "bold 15px 宋体";
    ctx.fillStyle="Red";
    ctx.fillText(txt2,10,80);
    ctx.fillText(mtxt1.width,mtxt2.width,80);
</script>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行效果如图 5.41 所示。

图 5.41　测量文字宽度

5.8　操作与使用图像

在 Canvas 中不仅可以绘制各种各样的图形，还可以直接引入图像。引入的图像可以改变大小，可用于创建图像切片、进行图片合成等。Canvas 支持多种图像格式，如 PNG、GIF、JPEG 等，其他的 canvas 元素也可以作为图像的来源。

5.8.1　向 Canvas 中引入图像

如果要向 Canvas 中引入图像，需要经过两个步骤：第一步是确定图像来源，第二步是使用 drawImage 方法将图像绘制到 Canvas 中。

确定图像来源可以有以下 4 种方式，从中选择一种合适的方式即可。

- ☑ 页面内的图片：如果已知图片元素的 ID，则可以通过 document.images 集合、document.getElementsByTagName 方法或 document.getElementById 方法获取页面内的该图片元素。
- ☑ 其他 canvas 元素：可以通过 document.getElementsByTagName 方法或 document.getElementById 方法获取已经准备好的 canvas 元素。例如，可以用这种方法为一个比较大的 Canvas 生成缩略图。

用脚本创建一个新的 Image 对象：使用脚本可以从零开始创建一个新的 Image 对象，不过这种方法存在一个缺点，即如果图像文件来源于网络且体积较大，则会花费较长的时间来装载，所以如果不希望因为图像文件装载而导致的漫长的等待，需要做好预装载的工作。

☑ 使用 data:url 方式引用图像：这种方法允许用 Base64 编码的字符串来定义一个图片，优点是图片可以即时使用，不必等待装载，而且迁移也非常容易。缺点是无法缓存图像，所以如果图片较大，则不太适宜用这种方法，因为这会导致嵌入的 url 数据相当庞大。

Note

在使用脚本创建新 Image 对象时，其方法如下所示。

```
var img = new Image();                    //创建新的 Image 对象
img.src = 'Image1.png';                   //设置图像路径
```

如果要解决图片预装载的问题，则可以使用下面的方法，即使用 onload 事件一边装载图像一边执行绘制图像的函数。

```
var img = new Image();                    //创建新的 Image 对象
img.onload = function(){
                                          //此处放置 drawImage 的语句
}
img.src = 'Image1.png';                   //设置图像路径
```

不管采用什么作为图像来源，之后的工作都是使用 drawImage 方法将图像绘制到 Canvas 中。drawImage 方法有以下 3 种用法。

```
context.drawImage(image,x,y);
context.drawImage(image, x, y, width, height);
context.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);
```

第一种是最基础和最简单的方法，其中 image 是引入的 image 对象或 Canvas 对象，x 和 y 目标是 Canvas 中要绘制到的起始位置的坐标。

【示例】本示例演示了如何使用上述两个步骤将图像引入到 Canvas 中。第二种和第三种 drawImage 方法将在后续的小节中单独介绍。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    var img = new Image();
    img.onload = function(){
        ctx.drawImage(img,0,0);
        ctx.font = "italic 60px Arial Black";
        ctx.shadowOffsetX = 3;
        ctx.shadowOffsetY = 3;
        ctx.shadowBlur = 2;
        ctx.shadowColor = "rgba(0, 0, 0, 0.7)";
        ctx.fillStyle = "White";
        ctx.fillText("COOL",50,165);
    }
    img.src = 'images/bg.jpg';
}
window.onload=function(){
    draw();
}
```

```
</script>
</head>
<body>
<canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行效果如图 5.42 所示。可以看到，通过引入外部图像作为文字的背景，可以缩减许多代码。

图 5.42　向 Canvas 中引入图像

5.8.2　改变图像大小

许多时候，需要在某个特定位置显示特定大小的图像，而且更新时希望不同图像文件显示在网页中的大小保持原来的统一尺寸，以保证网页布局与美观不受影响，这时需要用脚本动态改变图像的大小。当然，在其他场合，也可能需要改变图像大小以适应网页布局或其他功能的需要。

drawImage 的第二种用法可以使图片按指定的大小显示，其用法如下。

```
context.drawImage(image, x, y, width, height);
```

其中 width 和 height 分别是图像在 canvas 中显示的宽度和高度。

【示例】可以将上例中的代码稍作修改，以使得引入的图像调整为指定的大小，这样做的好处是可以让图像迁就页面布局。

```
<!DOCTYPE html>
<html>
<head>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    var img = new Image();
    img.onload = function(){
        ctx.drawImage(img,0,0,847,797);
        ctx.font = "italic 60px Arial Black";
        ctx.shadowOffsetX = 3;
        ctx.shadowOffsetY = 3;
        ctx.shadowBlur = 2;
        ctx.shadowColor = "rgba(0, 0, 0, 0.7)";
        ctx.fillStyle = "White";
```

```
            ctx.fillText("COOL",50,165);
        }
        img.src = 'images/bg.jpg';
    }
    window.onload=function(){
        draw();
    }
    </script>
    </head>
    <body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
    </body>
    </html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.43 所示。

图 5.43　改变图像大小

5.8.3　创建图像切片

drawImage 的第三种用法用于创建图像切片，其用法如下。

```
context.drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh);
```

其中 image 参数与前两用法相同，其余 8 个参数可以参考图 5.44。sx、sy 为源图像被切割区域的起始坐标，sw、sh 为源图像被切下来的宽度和高度，dx、dy 为被切割下来的源图像要放置到目标 Canvas 的起始坐标，dw、dh 为被切割下来的源图像放置到目标 Canvas 并显示的高度和宽度。

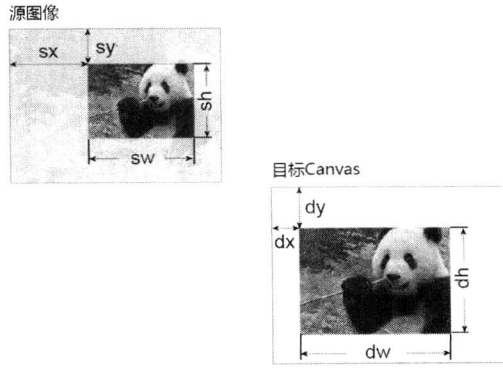


图 5.44　其余 8 个参数的图示

Note

【示例】下面通过一个示例来演示一下如何创建图像切片。

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<canvas id="myCanvas" width="600" height="380"></canvas>
<script language="javascript">
function draw() {
    var ctx = document.getElementById('myCanvas').getContext('2d');
    var img = new Image();
    img.onload = function(){
        ctx.drawImage(img,0,0);
        ctx.drawImage(img,30,40,140,180,0,240,140,180);
    }
    img.src = 'images/bg1.jpg';
}
window.onload=function(){
    draw();
}
</script>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 5.45 所示。其中上方显示的是源图像，下方是所创建的图像切片。

图 5.45　创建图像切片

第6章

HTML5 音频与视频

（视频讲解：50 分钟）

Web 上的多媒体应用经历了重大的改进，从最初简单的 MIDI 和 GIF 动画，发展到现在随处可见的 MP3 音乐、Flash 动画和各种在线视频，同时也产生了用于播放多媒体内容的各种工具或插件，如 Windows Media Player、Flash Player、Real Player 等。HTML5 中多媒体播放功能的出现，给原有的这些工具或插件带来了相当大的挑战，可以想象，用户如果不需要安装任何插件或工具就能在网页中直接播放音频与视频，该是多么美妙的体验。从这一点来说，HTML5 的竞争力可谓不同凡响，如果 HTML5 大行其道，必将给现有的插件或工具带来巨大的冲击。

本章将讨论如何在 HTML5 中实现音频与视频的播放和控制的革新，认识 HTML5 的两个重要元素 audio 和 video，介绍如何利用它们创建引人注目的应用。audio 和 video 元素的出现让 HTML5 的媒体应用多了新选择，开发人员不必使用插件就能播放音频和视频。对于这两个元素，HTML5 规范提供了通用、完整、可脚本化控制的 API。然后，会演示如何通过 API 编程的方式来控制页面中的音频和视频。最后将会探讨 HTML5 Audio 和 Video 在实际中的应用。

6.1　HTML5 多媒体技术概述

Note

依据 HTML5 的官方标准，并不需要为任何特定的音频或视频格式提供支持，所以浏览器厂商可以选择他们希望支持的格式。然而，如果未来能有一个固定的标准，最好能结束现在这种混乱的各自为战的局面。毕竟对于开发者来说，并不希望多做一些对媒体文件重编码的工作。未来，一定是越来越简化的，简单化才是真正的进步，按中国的古语来说，即大道至简，这也正是 HTML5 深受欢迎并大有前途的原因之一。不过，现在还有多种媒体格式，仍需要熟知各种能够被 HTML5 支持的媒体格式，并了解各浏览器目前对这些格式的兼容性问题。

不论是音频文件还是视频文件，实际上都只是一个容器文件，这点类似于压缩了一组文件的 ZIP 文件。视频文件（视频容器）包含了音频轨道、视频轨道和其他一些元数据。视频播放的时候，音频轨道和视频轨道是绑定在一起的。元数据部分包含了该视频的封面、标题、子标题、字幕等相关信息。

音频和视频的编码/解码器是一组算法，用来对一段特定音频或视频流进行解码和编码，以便音频和视频能够播放。原始的媒体文件体积非常大，假如不对其编码，那么构成一段视频和音频的数据可能会非常庞大，以至于在互联网上传播需耗费大量时间。若没有解码器，接收方就不能把编码过的数据重组为原始的媒体数据。编解码器可以读懂特定的容器格式，并且对其中的音频轨道和视频轨道解码。

6.1.1　关于编解码器

在介绍编解码器（codec）之前，先来熟悉一个关键概念——“容器”（Container）。音频文件与视频文件其实都只是一个“容器”（Container）文件，其中包含了音频或视频的音频轨道、视频轨道及其他元数据。元数据可以是音频或视频的标题、子标题、作者、艺术家、字幕等相关信息。

编解码器的作用是读取特定的容器格式，并对其中的音频与视频轨道进行解码，然后实现播放。音频与视频的原始数据一般都比较大，如果不对其进行编码就放到互联网上，那么其传播就会耗费大量时间，无法实现流畅传输或播放。大多数编码器对原始音频与视频文件进行了有损压缩，因为只有这样才能得到更小的文件大小和更高的压缩比。也有无损压缩的编码器，但在网页中没有什么优势。编解码器的数量非常多，这里只介绍常用的几种。

6.1.2　音频编解码器

目前经常用到的音频编解码器有以下几种：

☑　AAC

AAC（Advanced Audio Coding，高级音频编码）于 1997 年标准化，基于 MPEG-2 的音频编码技术，当时苹果将其作为 iTunes 的默认存储格式，因此而得以举世闻名并广为应用。AAC 是由 Fraunhofer IIS、杜比实验室、AT&T、Sony 等公司共同开发的，意在取代 MP3 格式。自 2000 年 MPEG-4 标准出现之后，AAC 重新集成了 MPEG-4 的特性，加入了 SBR 技术和 PS 技术，为了与之前的 MPEG-2 AAC 相区别，新一代的 AAC 又被称为 MPEG-4 AAC。

AAC 可以按任意比特率进行编码，通常情况下压缩比为 18:1，比 MP3 胜出很多。而 MP3 只能将比特率限制在一定范围内，即 320Kbps 之内。AAC 理论上能够编码高达 48 个声道，而且支持自定

义多个配置文件，以适应不同配置的计算机和不同速率的网络环境。如果使用高配，则能够在相同比特率下提高音质，但同时意味着编解码会比较费时。

目前，苹果产品（包括 iPod、Appler TV、QuickTime 等）、Sony Walkman、任天堂 NDSi 和魅族 M8 等 MP3 播放器都支持 AAC。此外计算机上如果安装过 AAC 解码器，则很多音乐播放软件也都能支持 AAC，如苹果的 iTunes。AAC 在移动电话领域的支持度已经很普遍，诺基亚、索尼爱立信、摩托罗拉等品牌均在其中高端产品中支持 AAC。

Note

☑ MPEG-1 音频层 3

注意： 这里要介绍的不是 MPEG-3，这是两个不同的概念。MPEG-1 音频层 3（MPEG-1 Audio Layer 3，动态图像专家压缩标准音频层 3）经常被称为 MP3，是现在经常听到和使用并且非常流行的一种音频格式，这是一种数字音频编码的有损压缩格式，可以大幅度减少音频数据量，并且对大多数用户来说，从听觉上效果并不会感到比原始音频有较为明显的下降，因此在互联网、手机、各种音乐播放器中应用非常广泛。

MP3 于 1991 年标准化，最多支持两个声道，并且可以被编码为不同的比特率，范围为 32Kbps~320Kbps，典型的比特率介于 128Kbps~320Kbps，而 CD 上未压缩的音频比特率为 1411.2Kbps。MP3 可以在文件编码时实现动态改变比特率，即使用“可变码率”（VBR）技术，在声音变化小的部分使用较小的比特率，声音变化较大的部分使用较大的比特率，方法是将音频切分为比特率不同的帧。

MP3 并未规定一个精确的编码规范，却详细地定义了解码算法与文件格式。在这种情况下，尽管编码方式非常多，却都可以使用同一播放器解码。在高比特率编码方面表现比较好并广泛被采用的编码器如 LAME，在低比特率编码上却未必表现良好。

MP3 格式的专利授权被很好地保护起来，主要是在美国和日本，不包括欧盟国家。专利保护限制了一些厂商在相关软件方面对 MP3 格式的支持，例如，微软公司倾向于发展自身的 Windows Media 格式，以避免与 MP3 相关的专利授权问题。

☑ Ogg Vorbis

Ogg Vorbis 也简称 Vorbis，Ogg 是指一种文件格式，或是一种“容器”，自由而且开放标准，在其中可以放入各式各样的编解码器。Ogg Vorbis 指的就是将 Vorbis 编码的音效包含在 Ogg 的容器中所形成的格式。后来 Ogg 通常指 Ogg Vorbis 这种音频文件格式，而在之前，.ogg 这一扩展名被用在任何 Ogg 支持格式下的内容，直到 2007 年才决定只留给 Vorbis 使用。

Vorbis 没有专利保护，因此得以在众多 Linux 发行版中直接播放。Vorbis 之所以开放源代码而且免专利使用费，就是为了与需要专利使用费的 MP3 和 AAC 一争高下。

6.1.3 视频编解码器

以上介绍的是音频编解码器，下面介绍经常用到的视频编解码器，主要有以下几种：

☑ H.264

H.264（或称“MPEG-4 第十部分”、“MPEG-4 AVC”、“MPEG-4 高级视频编码”）是由联合视频组（JVT，Joint Video Team）提出的高度压缩数字视频编解码器标准，于 2003 年实现标准化，其最初目标为以较低的比特率提供更好的视频质量，同时能够支持一切设备。

H.264 比以前的编解码器更能有效地进行编码，并能够适应不同的网络环境，它最大的优势是具有很高的数据压缩比率。如果图像质量相等，H.264 的压缩比为 MPEG-2 的两倍以上，MPEG-4 的 1.5~2 倍。正因为这一特点，H.264 在光盘存储、数字电视、网络电视、通信与多媒体方面都有广泛的应用。

例如，YouTube 现在采用 H.264 来对高清晰视频进行编码，然后使用 Flash Player 进行播放。

☑ VP8

VP8 由 On2 公司于 2008 年 9 月 13 日推出，目的是取代其前任 VP7 视频编解码器。2009 年，Google 收购了 On2 公司，并于 2010 年 5 月 19 日宣布将 VP8 以 BSD 许可证的形式开源，从此时开始，VP8 便成了一个免版权费用可自由使用的技术，任何使用都不受专利限制。

Note

也就是从 2010 年 5 月 19 日开始，Google 的 WebM 项目启动，目标是构建一个开放且免版权费用的视频文件格式，提供高质量的视频压缩以适应 HTML5 的需要。WebM 项目采用 VP8 视频编解码器和 Vorbis 音频编解码器，其封装格式以 Matroska 格式为基础。Adobe 已宣布 Adobe Flash Player 将在未来支持 VP8，但还没有宣布会支持 Vorbis 或 WebM。

☑ Ogg Theora

On2 公司先前开发的 VP3 后来被 Xiph.Org 基金会于 2002 年接手并将之冠名以 Theora，无专利限制，无许可证，开放源代码，旨在实现比 MPEG-4 Part 2 更好的编码效率。Theora 的发展比较缓慢，几至“冻结”，2008 年 11 月才发布最终正式版 1.0 版本，2009 年 11 月发布了改进后的 1.1 版本。

Theora 是一个可变比特率图像压缩格式，其图像流可以存储在任何容器文件格式中，最常用的是与 Vorbis 一起存储到 Ogg 容器。浏览器方面，Mozilla Firefox 3.5 提供了对于 Theora 格式的原生支持，即不需要任何插件就能播放 Theora 视频，其他浏览器如 Google Chrome 3.0.182.2、Opera 10.50 等之后的版本都提供了对该格式的支持。也可以使用相关浏览器插件来播放 Theora 视频，如 Annodex 的插件、Cortado、VLC 媒体播放器提供的针对 IE 及 Firefox 的插件等。

有一些编解码器受专利保护，有一些是免费的。例如，Vorbis 音频编解码和 Theora 视频编解码器可以免费使用，而要想使用 MPEG-4 和 H.264 的话就需要支付相关费用了。

最初，HTML5 规范中本来打算指定编解码器，但实施起来困难重重。如 Ogg Theora，有的厂商考虑到他们已有的硬件或软件不支持，所以不希望 HTML5 规范中包含 Ogg Theora。例如，Apple 的 iPhone 使用的硬件解码器就是 H.264 而不是 Theora。另一方面，免费系统若采用要支付专利费用的编解码器将不可避免地影响其下游分销业务。而专有编解码器的性能指标又是采用免费编解码器的浏览器厂商需要考虑的一个重要因素，这种情况导致了现在的僵局，没有任何一种编解码器可以被所有浏览器厂商接受并在其产品中提供支持。

目前，HTML5 规范放弃了对编解码器的要求，不过未来很有可能重新考虑。现在，开发人员能做的只能是熟悉各种浏览器的支持情况，针对不同的浏览器环境对媒体文件进行重编码。

我们非常希望随着时间的推移，不同编解码器的支持程度越来越高并最终汇集成统一版本，这样可以简化将来对通用媒体类型的选择。当然，也有可能某个编解码器逐渐强大起来并最终成为领域内的事实标准。此外，HTML5 的媒体标签内建了一套机制来挑选最适合在浏览器中播放的内容类型，进而实现不同的环境下媒体类型的简单切换。

HTML5 规范不支持如下特性：

☑ 流式音频和视频。因为目前 HTML5 视频规范中还没有比特率切换标准，所以对视频的支持只限于加载的全部媒体文件，但是将来一旦流媒体格式被 HTML5 支持，肯定有相关的设计规范。

☑ HTML5 的媒体受到 HTTP 跨源资源共享的限制。

☑ 全屏视频无法通过脚本控制。从安全性角度来看，让脚本元素控制全屏操作是不合适的。不过，如果要让用户在全屏方式下播放视频，浏览器可以提供其他控制手段。

☑ 对 audio 元素和 video 元素的访问尚未完全加入规范中。基于流行的字幕格式 SRT 的字幕支持规范（WebSRT）仍在编写中。

Note

6.2 浏览器支持概述

audio 和 video 元素的浏览器支持情况如表 6.1 所示，很多浏览器已经实现了对 HTML5 中 audio 和 video 元素的支持。

表 6.1 浏览器支持概述

浏 览 器	说 明
IE	9.0 及以上的版本支持
Firefox	3.5 及以上的版本支持。Theora 和 Vorbis 编解码器，Ogg 容器
Opera	10.5 及以上的版本支持。Theora 和 Vorbis 编解码器，Ogg 容器（10.5 及以上版本） VP8 和 Vorbis 编解码器，WebM 格式（10.6 及以上版本）
Chrome	3.0 及以上的版本支持。Theora 和 Vorbis 编解码器，Ogg 容器 H.264 和 AAC 编解码器，MPEG 4 容器
Safari	3.2 及以上的版本支持。H.264 和 AAC 编解码器，MPEG 4 容器

各浏览器支持的音频与视频格式可以从 http://www.findmebyip.com/litmus/网站公布的测试结果来作一概览。其中对于 HTML5 音频技术支持情况如图 6.1 所示。

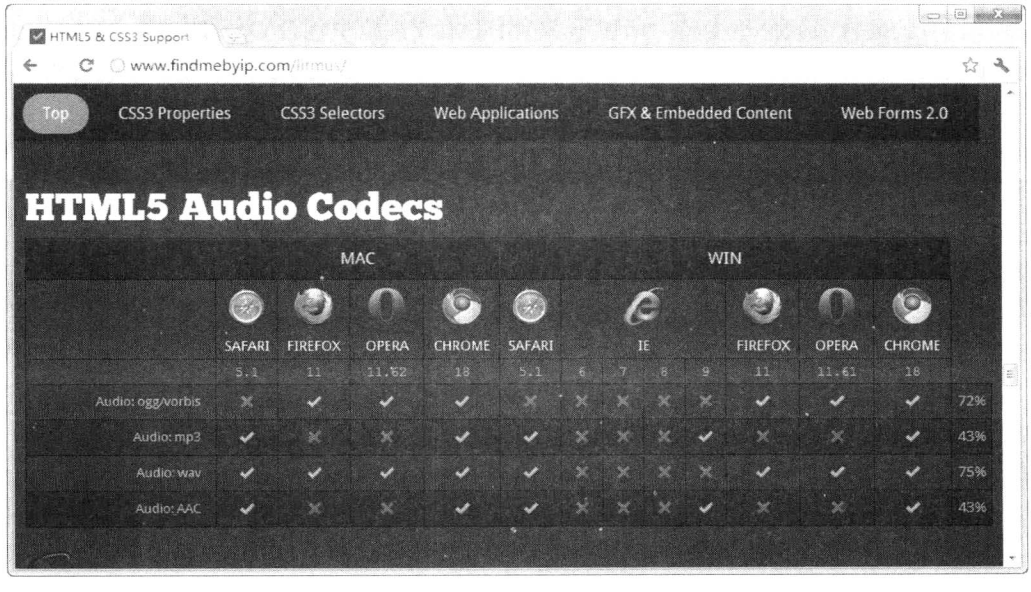


图 6.1 音频格式的浏览器支持情况

对于 HTML5 视频技术支持情况如图 6.2 所示。

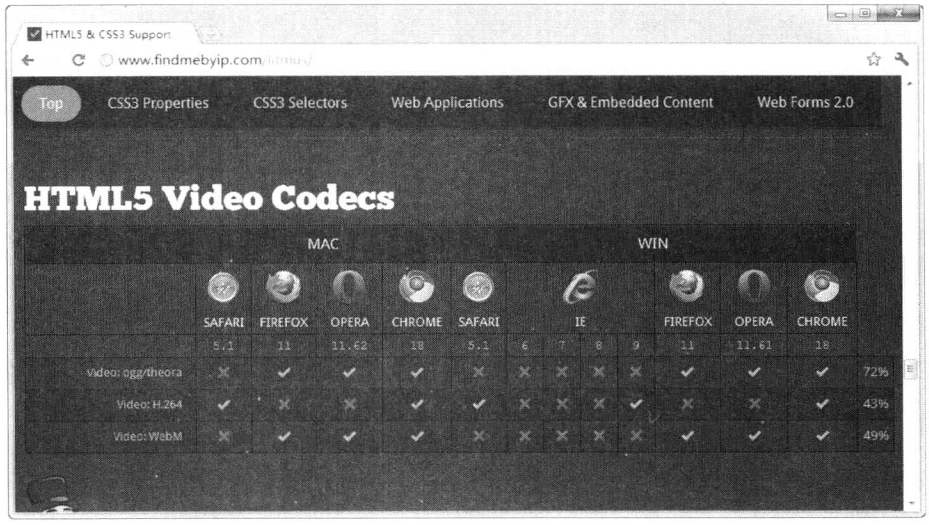


图 6.2　视频格式的浏览器支持情况

6.2.1　用 JavaScript 检测音频格式支持情况

如果要使用 JavaScript 检测浏览器支持哪些音频格式，则可以使用下面示例中的代码。使用 audio 元素的 canPlayType 方法可以检测浏览器支持的文件格式，该方法采用 MIME 类型与编码/解码器参数，并返回下列 3 个字符串值之一：probably、maybe 和空字符串。

【示例】以下代码在 Chrome 浏览器中的运行效果如图 6.3 所示，可以看到 Chrome 浏览器对 mp3、ogg 和 aac 这 3 种格式都支持。

```
<!DOCTYPE html>
<head>
<script type/javascript>
function checkAudio(){
    var myAudio = document.createElement('audio');
    if (myAudio.canPlayType) {
        if ( "" != myAudio.canPlayType('audio/mpeg')) {
            document.write("您的浏览器支持 mp3 编码。<br>");
        }
        if ( "" != myAudio.canPlayType('audio/ogg; codecs="vorbis"')) {
            document.write("您的浏览器支持 ogg 编码。<br>");
        }
        if ( "" != myAudio.canPlayType('audio/mp4; codecs="mp4a.40.5"')) {
            document.write("您的浏览器支持 aac 编码。");
        }
    }
    else {
        document.write("您的浏览器不支持要检测的音频格式。");
    }
}
window.onload=function()
{
    checkAudio();
```

```
}
    </script>
</head>
<body>
</body>
</html>
```

Note

如果 canPlayType 返回 probably 或 maybe，则下面的语句返回 true，而如果 canPlayType 返回空字符串，则该语句将返回 false，表示不支持此格式。

```
if ( "" != myAudio.canPlayType('audio/mpeg'))
```

因为 IE8 浏览器不支持 ogg 和 aac 编码格式，所以在 IE8 浏览器中会显示“您的浏览器支持 mp3 编码。”，如图 6.4 所示。

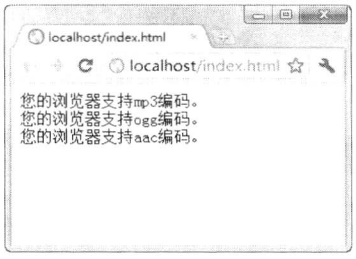


图 6.3　显示支持的格式

图 6.4　IE 浏览器支持 mp3 编码格式

6.2.2　用 JavaScript 检测视频格式支持情况

对于视频格式，同样可以使用上节介绍过的 canPlayType 方法，因为 video 元素同样支持该方法。

【示例】以下代码在 Chrome 浏览器中的运行结果如图 6.5 所示，可以看到以上 3 种格式都支持。因为 IE 浏览器不支持 vp8 和 theora 编码格式，仅支持 h264 编码格式，所以在 IE 浏览器中会显示“您的浏览器支持 h264 编码。”，如图 6.6 所示。

```
<!DOCTYPE html>
<head>
<script type/javascript>
function checkVideo(){
    var myVideo = document.createElement('video');
    if (myVideo.canPlayType) {
        if ( "" != myVideo.canPlayType('video/mp4;codecs="avc1.64001E"')) {
            document.write("您的浏览器支持 h264 编码。<br>");
        }
        if ( "" != myVideo.canPlayType('video/ogg; codecs="vp8"')) {
            document.write("您的浏览器支持 vp8 编码。<br>");
        }
        if ( "" != myVideo.canPlayType('video/ogg; codecs="theora"')) {
            document.write("您的浏览器支持 theora 编码。");
        }
    }
    else {
        document.write("您的浏览器不支持要检测的视频格式。");
```

```
        }
    }
    window.onload=function()
    {
        checkVideo();
    }
        </script>
    </head>
    <body>
    </body>
    </html>
```

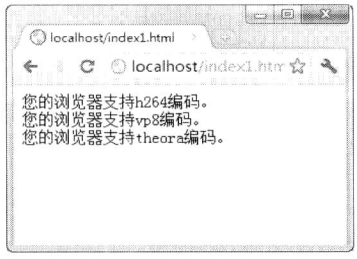


图 6.5　显示支持的格式

图 6.6　IE 浏览器支持 h264 编码格式

如果为不支持 HTML5 媒体的浏览器提供可选方式来显示视频，可以使用相同的方法，将以插件方式播放视频的代码作为备选内容，放在相同的位置即可。

```
<video src="video.ogg">
    <object data="videoplayer.swf" type="application/x-shockwave-flash">
        <param name="movie" value="video.swf"/>
    </object>
</video>
```

在 video 元素中嵌入显示 Flash 视频的 object 元素之后，如果浏览器支持 HTML5 视频，那么 HTML5 视频会优先显示，Flash 视频作为备选。不过在 HTML5 被广泛支持之前，可能需要提供多种视频格式。

6.3　在 HTML5 中播放音频

Web 上的音频播放从来都没有一个固定的标准，在访问相关网站时，会遇到各种各样的插件，比较常见的有 Windows Media Player、Real Player 等。HTML5 的问世，终于有望在音频播放领域实现统一的标准，从而告别这些纷繁复杂并令人生厌的插件。

6.3.1　认识 audio 元素

在 HTML5 中，使用新增的 audio 元素来播放声音文件或音频流，支持 Ogg Vorbis（简称 Ogg）、MP3 和 WAV 等音频格式（有关音频格式的知识稍后作详细介绍），其用法如下。

```
<audio src="samplesong.mp3" controls="controls">
```

```
</audio>
```

其中 src 属性用于指定要播放的声音文件，controls 属性用于提供播放、暂停和音量控件。

如果浏览器不支持 audio 元素，则可以在<audio>与</audio>之间插入一段替换内容，这样旧的浏览器就可以显示这些信息。例如：

Note

```
<audio src="samplesong.mp3" controls="controls">
您的浏览器不支持 audio 标签。
</audio>
```

替换内容不仅可以使用文本，还可以是一些其他音频插件，或者是声音文件的链接等。

6.3.2 播放音频

接下来通过一个完整的示例来演示如何在页面中播放音频。

【示例 1】在本示例中，使用了 source 元素来链接到不同的音频文件，浏览器会自动选择第一个可以识别的格式。

```
<!DOCTYPE html>
<html>
<body>
<audio controls="controls">
    <source src="samplesong.ogg" type="audio/ogg">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 6.7 所示，可以看到出现一个比较简单的音频播放器，包含了播放/暂停、位置、时间显示、音量控制这些常用控件。

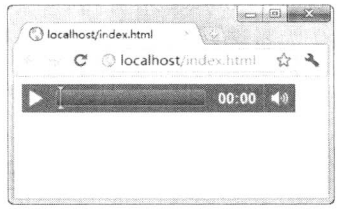


图 6.7 播放音频

HTML5 中的 audio 元素和 video 元素有很多相同之处。两者都支持的操作有播放/暂停、静音/清除静音、加载等，因此通用的动作被从 video 和 audio 元素中剥离出来并放到了媒体（media）元素部分。

audio 元素之间的内容是为不支持 audio 元素的浏览器准备的替换内容，如果用户用的是老版本浏览器，页面上就会显示这些文本信息。替换内容不局限于文本信息，还可以换成 Flash 播放器等视频播放插件，或者直接给出媒体文件的链接地址。

src 属性直接指向媒体文件就可以了。但是，万一浏览器不支持相关容器或者编解码器时，就需要用到备用声明了。备用声明中可以包含多种来源，浏览器可以从这些数据源中进行选择。

从上面的代码可以看到在 audio 元素中，使用两个新的 source 元素替换了先前的 src 属性。这样

可以让浏览器根据自身播放能力自动选择，挑选最佳的来源进行播放。对于来源，浏览器会按照声明顺序判断，如果支持的不止一种，那么浏览器会选择支持的第一个来源。数据源列表的排放顺序应按照用户体验由高到低或者服务器消耗由低到高的顺序列出。

Note

在支持 audio 元素的浏览器中运行上述代码，页面的显示结果与添加可选来源列表之前没有任何变化。但是如果浏览器不支持 Ogg Vorbis 格式，而是支持 MP3 格式，那上面的代码就起作用了。这种声明的好处在于当开发人员在与媒体文件进行交互操作的时候，不需要考虑实际上调用的是哪个容器和编解码器。浏览器给开发人员提供操作媒体文件的统一接口，这与所选择播放的文件来源无关。

此外，还有一种暗示浏览器应该使用哪个媒体来源的方式。一个媒体容器可能会支持多种类型的编解码器，因此就不难理解声明的源文件的扩展名可能会让浏览器误以为自己支持或不支持某种类型了。如果 type 属性中指定的类型与源文件不匹配，浏览器可能就会拒绝播放。指定类型的方式就是为 source 元素添加 type 属性。如果明确地知道浏览器支持某种类型，将它写明显然更明智。否则，最好省略 type 属性，让浏览器自己检测编码方式。此外还需要注意的是，WebM 格式只支持一种音频编解码器和一种视频编解码器，也就是说只要看到.webm 扩展名或者 video/WebM 的内容类型，就可以知道来源的全部情况了。如果浏览器支持 WebM，那么就可以播放所有.webm 文件了，容器和编解码器都可以在 type 特性中声明。

【示例 2】本示例演示了如何在页面中插入背景音乐。实际上使用 audio 元素实现循环播放一首背景音乐非常简单，只需在 audio 元素中设置 autoplay 和 loop 属性即可，详细代码如下所示。

```
<!DOCTYPE html>
<html>
<audio autoplay loop>
    <source src="Wah Game Loop.ogg">
    <source src="Wah Game Loop.mp3">
</audio>
</html>
```

6.4　在 HTML5 中播放视频

与音频类似，Web 上的视频播放也从来都没有一个固定的标准，大多数视频是通过像 Flash 这样的插件来显示的，不同的浏览器往往拥有不同的插件。HTML5 新增的 video 元素是现在及未来都非常可行的一种用于包含视频的标准方法。

6.4.1　认识 video 元素

在 HTML5 中，使用新增的 video 元素来播放视频文件或视频流，支持 Ogg、MPEG-4、WebM 等视频格式，其用法如下。

```
<video src="samplemovie.mp4" controls="controls">
</video>
```

其中 src 属性用于指定要播放的视频文件，controls 属性用于提供播放、暂停和音量控件，也可以包含宽度和高度属性。

如果浏览器不支持 video 元素，则可以在<video>与</video>之间插入一段替换内容，这样，旧的浏览器就可以显示这些信息。例如：

```
<video src=" samplemovie.mp4" controls="controls">
您的浏览器不支持 video 标签。
</video>
```

6.4.2 播放视频

Note

下面通过一个完整的示例来演示如何在页面内播放视频。

【示例 1】在本示例中，使用了 source 元素来链接到不同的视频文件，浏览器会自己选择第一个可以识别的格式。

```
<!DOCTYPE html>
<html>
<body>
<video controls="controls">
    <source src="medias/volcano.ogg" type="video/ogg">
    <source src="medias/volcano.mp4" type="video/mp4">
您的浏览器不支持 video 标签。
</video >
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 6.8 所示，可以看到出现一个比较简单的视频播放器，包含了播放/暂停、位置、时间显示和音量控制这些常用控件。

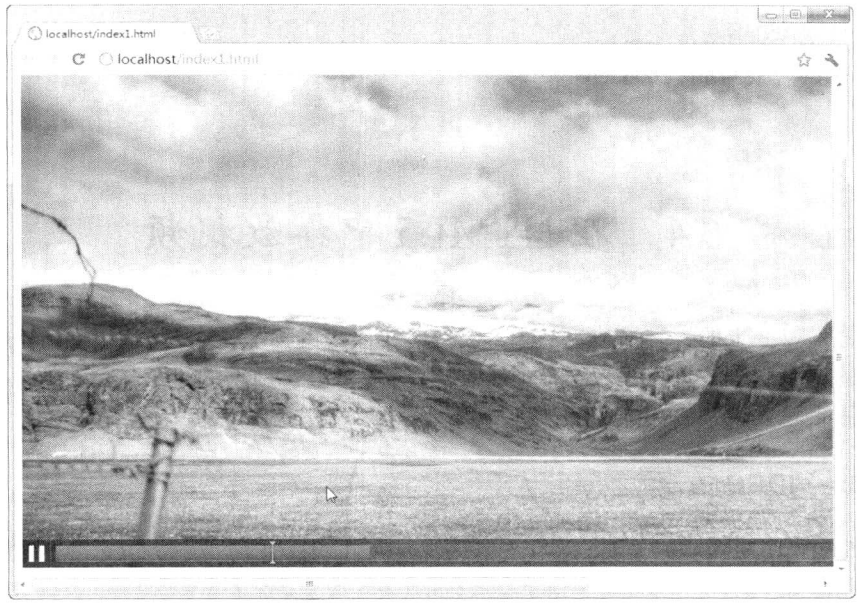


图 6.8 播放视频

在 audio 元素或 video 元素中指定 controls 属性可以在页面上以默认方式进行播放控制。如果不加这个特性，那么在播放的时候就不会显示控制界面。如果播放的是音频，那么页面上任何信息都不会出现，因为音频元素的唯一可视化信息就是对应的控制界面。如果播放的是视频，那么视频内容会显示。即使不添加 controls 属性也不会影响页面正常显示。有一种方法可以让没有 controls 属性的音频或视频正常播放，那就是在 audio 元素或 video 元素中设置另一个属性 autoplay。

```
<video autoplay>
    <source src="medias/volcano.ogg" type="video/ogg">
    <source src="medias/volcano.mp4" type="video/mp4">
您的浏览器不支持 video 标签。
</video >
```

Note

通过设置 autoplay 属性，不需要任何用户交互，音频或视频文件就会在加载完成后自动播放。不过大部分用户对这种方式会比较反感，所以应慎用 autoplay。在无任何提示的情况下，自动播放一段音频通常有两种用途，第一种是用来制造背景氛围，第二种是强制用户接收广告。这种方式的问题在于会干扰用户本机播放的其他音频，尤其会给依赖屏幕阅读功能进行 Web 内容导航的用户带来不便。

如果内置的控件不适应用户界面的布局，或者希望使用默认控件中没有的条件或动作来控制音频或视频文件，那么可以借助一些内置的 JavaScript 函数和属性来实现，简单说明如下。

- ☑ load()：该函数可以加载音频或者视频文件，为播放做准备，用来在播放前预加载。通常情况下不必调用，除非用于动态生成的元素。
- ☑ play()：该函数可以加载并播放音频或视频文件，除非音频或视频文件已经暂停在其他位置了，否则默认从开头播放。
- ☑ pause()：该函数用于暂停处于播放状态的音频或视频文件。
- ☑ canPlayType(type)：该函数可用于检测 video 元素是否支持给定 MIME 类型的文件。

canPlayType(type)函数有一个特殊的用途：向动态创建的 video 元素中传入某段视频的 MIME 类型后，仅仅通过一行脚本语句即可获得当前浏览器对相关视频类型的支持情况。

【示例 2】本示例演示如何通过在视频上移动鼠标来触发 play 和 pause 功能。当页面包含多个视频，且由用户来选择播放某个视频时，这个功能就非常适用了。如在用户将鼠标移到某个视频上时，播放简短的视频预览片段，用户单击后，播放完整的视频。具体演示代码如下所示。

```
<!DOCTYPE html>
<html>
<video id="movies" onmouseover="this.play()" onmouseout="this.pause()" autobuffer="true"
    width="400px" height="300px">
    <source src="medias/volcano.ogv" type='video/ogg; codecs="theora, vorbis"'>
    <source src="medias/volcano.mp4" type='video/mp4'>
</video>
</html>
```

在浏览器中预览，以上代码的显示效果如图 6.9 所示。

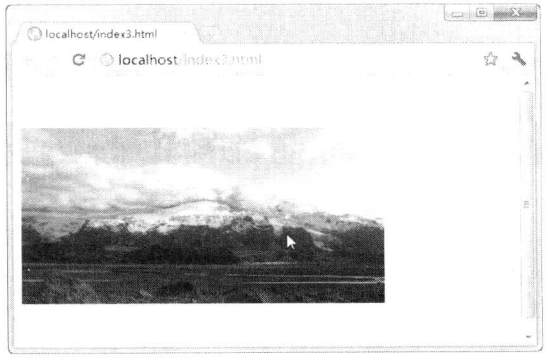


图 6.9　使用鼠标控制视频播放

6.5 音频与视频相关属性、方法与事件

Note

与所有其他的 HTML5 元素一样，audio 元素支持 HTML5 中全局属性和事件属性。此外，还支持以下几个属性。

6.5.1 音频与视频相关属性

audio 元素与 video 元素相关属性基本相同，下面对这些属性作详细介绍，并对某些属性给出示例。

☑ autobuffer 属性

可读写属性。使用该属性可以使 audio 或 video 元素实现自动缓冲，默认值为 false，即 audio 或 video 元素默认情况下并不自动缓冲。如果值为 true，则自动缓冲，但并不播放。如果使用了 autoplay 属性，则 autobuffer 属性会被忽略。其用法参见下面的代码。

```
<audio controls="controls" autobuffer="true">
    <source src="samplesong.ogg" type="audio/ogg">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ autoplay 属性

可读写属性。使用该属性可以实现页面加载音频后一旦就绪即开始自动播放，比使用脚本控制音频或视频播放更简便，其值也可以设置为 true 或 false。如果值为 true 或 autoplay，则当音频或视频缓冲到足够多时即开始播放。其用法参见下面的代码。

```
<audio controls="controls" autoplay="autoplay">
    <source src="samplesong.ogg" type="audio/ogg">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ buffered 属性

只读属性，用于返回一个 TimeRanges 对象，确认浏览器已经缓存媒体文件。

☑ controls 属性

可读写属性。该属性为布尔值，可以为媒体文件提供用于播放的控制条，包含播放、暂停、定位、时间显示、音量控制、全屏切换等常用控件。在将来的标准中，有望在播放控件中看到字幕和音轨。开发人员如果不希望使用浏览器默认的控制条，也可以使用脚本来自定义控制条。其用法参见下面的代码。

```
<audio controls="controls">
    <source src="samplesong.ogg" type="audio/ogg">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ currentSrc 属性

只读属性，无默认值。用于返回媒体数据的 URL 地址，如果媒体 URL 地址未指定，则返回一个空字符串。

☑ currentTime 属性

可读写属性，无默认值。用于获取或设置当前播放位置，返回值为时间，单位为秒。

☑ defaultPlaybackRate 属性

可读写属性，无默认值。用于获取或设置当前播放速率，前提是用户没有使用快进或快退控件。

☑ duration 属性

只读属性，无默认值。用于获取当前媒体的持续时间，返回值为时间，单位为秒。

☑ ended 属性

只读属性，无默认值。用于返回一个布尔值，以获悉媒体是否播放结束。

☑ error 属性

只读属性，无默认值。用于返回一个 MediaError 对象以表明当前的错误状态，如果没有出现错误，则返回 null。错误状态共有 4 个可能值，如下所示。

- MEDIA_ERR_ABORTED（数字值为 1）：媒体资源获取异常——媒体数据的下载过程因用户操作而终止。
- MEDIA_ERR_NETWORK（数字值为 2）：网络错误——在媒体数据已经就绪时用户停止了媒体下载资源的过程。
- MEDIA_ERR_DECODE（数字值为 3）：媒体解码错误——在媒体数据已经就绪时解码过程中出现了错误。
- MEDIA_ERR_SRC_NOT_SUPPORTED（数字值为 4）：媒体格式不被支持。

☑ initialTime 属性

只读属性，无默认值。用于获取最早的可用于回放的位置，返回值为时间，单位为秒。

☑ loop 属性

可读写属性。用于获取或设置当媒体文件播放结束时是否再重新开始播放。其使用方法如下所示。

```
<audio controls="controls" loop="loop">
    <source src="samplesong.mp3" type="audio/mpeg">
您的浏览器不支持 audio 标签。
</audio>
```

☑ muted 属性

可读写属性，无默认值。muted 属性为布尔值，用于获取或设置当前媒体播放是否开启静音，true 为开启静音，false 为未开启静音。如果对其赋值，则可以设置播放时是否静音。

☑ networkState 属性

只读属性。用于返回媒体的网络状态，共有 4 个可能值。

- NETWORK_EMPTY（数字值为 0）：元素尚未初始化。
- NETWORK_IDLE（数字值为 1）：加载完成，网络空闲。
- NETWORK_LOADING（数字值为 2）：媒体数据加载中。
- NETWORK_NO_SOURCE（数字值为 3）：因为不存在支持的编码格式，加载失败。

☑ paused 属性

只读属性，无默认值。用于返回一个布尔值，表示媒体是否暂停播放，true 表示暂停，flase 表示正在播放。

☑ playbackRate 属性

可读写属性，无默认值。用于读取或设置媒体资源播放的当前速率。

☑ played 属性

只读属性，无默认值。用于返回一个 TimeRanges 对象，标明媒体资源在浏览器中已播放的时间范围。TimeRanges 对象的 length 属性为已播放部分的时间段，该对象有两个方法，end 方法用于返回已播放时间段的结束时间，start 方法用于返回已播放时间段的开始时间，其用法参见下面的代码。

Note

```
var ranges = document.getElementById('myVideo').played;
for (var i=0; i<ranges.length; i++)
    var start = ranges.start(i);
    var end = ranges.end(i);
    alert("从" + start +"开始播放到" + end+"结束。");
}
```

☑ preload 属性

可读写属性，无默认值。用于定义视频是否预加载，该属性有 3 个可选值：none、metadata 和 auto。如果不用该属性，则默认为 auto，分别介绍如下。

- none：不进行预加载。当页面制作人员认为用户不希望播放此视频，或者减少 HTTP 请求时可用该值。
- metadata：部分预加载。使用此属性值，代表页面制作者认为用户不希望播放此视频，但为用户提供一些元数据（包括尺寸、第一帧、曲目列表和持续时间等）。
- auto：全部预加载。

preload 属性的用法如下所示。

```
<video src="samplemovie.mp4" preload="auto">
</video>
```

☑ readyState 属性

只读属性，无默认值。用于返回媒体当前播放位置的就绪状态，共有以下 5 个可能值。

- HAVE_NOTHING（数字值为 0）：当前播放位置没有有效的媒体资源。
- HAVE_METADATA（数字值为 1）：媒体资源确认存在且加载中，但当前位置没有能够加载到有效的媒体数据以进行播放。
- HAVE_CURRENT_DATA（数字值为 2）：已获取到当前播放数据，但没有足够的数据进行播放。
- HAVE_FUTURE_DATA（数字值为 3）：在当前位置已获取到后续播放媒体数据，可以进行播放。
- HAVE_ENOUGH_DATA（数字值为 4）：媒体数据可以进行播放，且浏览器确认媒体数据正以某一种速率进行加载并有足够的后续数据以继续进行播放，而不会使浏览器的播放进度赶上加载数据的末端。

☑ seekable 属性

只读属性，无默认值。用于返回一个 TimeRanges 对象，表明可以对当前媒体资源进行请求。

☑ seeking 属性

只读属性，无默认值。用于返回一个布尔值，表示浏览器是否正在请求某一播放位置的媒体数据，ture 表示浏览器正在请求数据，而 false 表示浏览器已经停止请求数据。

☑ src 属性

可读写属性，无默认值。用于指定媒体资源的 URL 地址，与<img>标签类似，可与 poster 属性连用。poster 属性用于指定一张替换图片，如果当前媒体数据无效，则显示该图片。其用法如下所示。

```
<video src="http://www.lidongbo.com/samplemovie.mp4" poster=" http://www.lidongbo.com/samplemovie.png">
</video>
```

Note

☑ volume 属性

可读写属性，无默认值。用于获取或设置媒体资源的播放音量，范围为 0.0~1.0，0.0 为静音，1.0 为最大音量。注意音量大小并非是线性变化的，如果同时使用了 muted 属性，则此属性会被忽略。

6.5.2　音频与视频相关方法

audio 元素与 video 元素相关，方法也基本相同，下面对这些方法作详细介绍，并对某些方法给出示例。

☑ canPlayType 方法

用于返回一个字符串以表明客户端是否能够播放指定的媒体类型，其用法如下。

```
var canPlay = media.canPlayType(type)
```

其中 media 指页面中的 audio 或 video 元素，参数 type 为客户端浏览器能够播放的媒体类型。该方法返回以下值之一。

- probably：表示浏览器确定支持此媒体类型。
- maybe：表示浏览器可能支持此媒体类型。
- 空字符串：表示浏览器不支持此媒体类型。

☑ load 方法

用于重置媒体元素并重新载入媒体，不返回任何值，该方法可中止任何正在进行的任务或事件。元素的 playbackRate 属性值会被强行设为 defaultPlaybackRate 属性的值，而且元素的 error 值会被强行设置为 null。

【示例】在本示例中，通过单击按钮可以重新载入另一个新的视频。

```
<video controls>
 <source src="video.ogv" type='video/ogg; codecs="theora, vorbis"'>
 <source src="video.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
您的浏览器不支持视频播放。
</video>
<script>
function loadNewVideo() {
  var video = document.getElementsByTagName('video')[0];
  var sources = video.getElementsByTagName('source');
  sources[0].src = 'video2.ogv';
  sources[1].src = 'video2.mp4';
  video.load(); // 用 load 方法载入新的视频
}
</script>
<input type="button" value="载入新的视频"
       onclick="loadNewVideo()">
```

☑ pause 方法

用于暂停媒体的播放，并将元素的 paused 属性的值强行设置为 true。

☑ play 方法

用于播放媒体，并将元素的 paused 属性的值强行设置为 false。

Note

6.5.3 音频与视频相关事件

audio 元素或 video 元素支持 HTML5 中的媒体事件（media events）属性，使用 JavaScript 脚本可以捕捉这些事件并对其进行处理。处理这些事件一般有下面两种方式。

一种是使用监听的方式，即使用 addEventListener 方法，其用法如下：

```
addEventListener("事件名",处理函数,处理方式)
```

另一种是直接赋值法，即常用的获取事件句柄的方法，例如，video.onplay=begin_playing，begin_playing 为处理函数。音频与视频相关事件及描述如表 6.2 所示。

表 6.2 音频与视频相关事件及描述

事　件	描　述
abort	浏览器在完全加载媒体数据之前中止获取媒体数据
canplay	浏览器能够开始播放媒体数据，但估计以当前速率播放不能直接将媒体播放完，即可能因播放期间需要缓冲而停止
canplaythrough	浏览器以当前速率可以直接播放完整个媒体资源，在此期间不需要缓冲
durationchange	媒体长度（duration 属性）改变
emptied	媒体资源元素突然为空时，可能是网络错误或加载错误等
ended	媒体播放已抵达结尾
error	在元素加载期间发生错误
loadeddata	已经加载当前播放位置的媒体数据
loadedmetadata	浏览器已经获取媒体元素的持续时间和尺寸
loadstart	浏览器开始加载媒体数据
pause	媒体数据暂停播放
play	媒体数据将要开始播放
playing	媒体数据已经开始播放
progress	浏览器正在获取媒体数据
ratechange	媒体数据的默认播放速率（defaultPlaybackRate 属性）改变或播放速率（playbackRate 属性）改变
readystatechange	就绪状态（ready-state）改变
seeked	浏览器停止请求数据，媒体元素的定位属性不再为真（seeking 属性值为 false）且定位已结束
seeking	浏览器正在请求数据，媒体元素的定位属性为真（seeking 属性值为 true）且定位已开始
stalled	浏览器获取媒体数据过程中出现异常
suspend	浏览器非主动获取媒体数据，但在取回整个媒体文件之前中止
timeupdate	媒体当前播放位置（currentTime 属性）发生改变
volumechange	媒体音量（volume 属性）改变或静音（muted 属性）
waiting	媒体已停止播放但打算继续播放

6.6　综 合 实 战

Note

本节将通过两个示例演示如何应用 audio 元素和 video 元素，并灵活使用 JavaScript 脚本控制 HTML5 多媒体播放。

6.6.1　用脚本控制音乐播放

如果需要在用户交互界面上播放一段音频，同时又不想被默认的控制界面影响页面显示效果，则可创建一个隐藏的 audio 元素，即不设置 controls 属性，或将其设置为 false，然后用自定义控制界面控制音频的播放。

【示例】本示例完整代码如下，演示效果如图 6.10 所示。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<style type="text/css">
body { background:url(images/bg.jpg) no-repeat;}
#toggle { position:absolute; left:311px; top:293px; }
</style>
</head>
<title></title>
<audio id="music">
    <source src="medias/Wah Game Loop.ogg">
    <source src="medias/Wah Game Loop.mp3">
</audio>
<button id="toggle" onclick="toggleSound()">播放</button>
<script type="text/javascript">
    function toggleSound() {
        var music = document.getElementById("music");
        var toggle = document.getElementById("toggle");
        if (music.paused) {
            music.play();
            toggle.innerHTML = "暂停";
        }
        else {
            music.pause();
            toggle.innerHTML ="播放";
        }
    }
  </script>
</html>
```

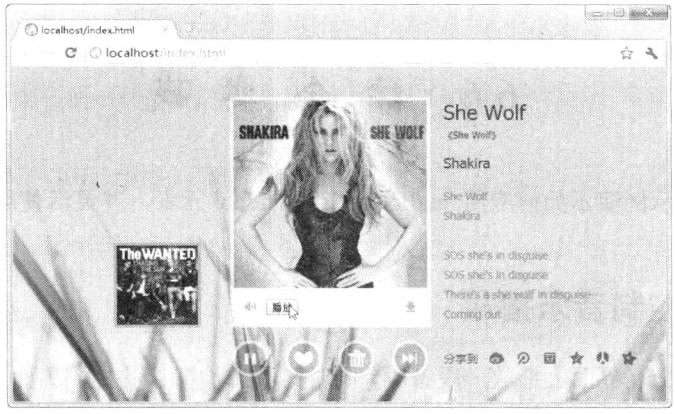


图 6.10　用脚本控制音乐播放

在上面示例中，先隐藏了用户控制界面，也没有将其设置为加载后自动播放，而是创建了一个具有切换功能的按钮，以脚本的方式控制音频播放，如下所示。

```
<button id="toggle" onclick="toggleSound()">播放</button>
```

按钮在初始化时会提示用户单击它以播放音频。每次单击时，都会触发 toggieSound()函数。在toggleSound()函数中，首先访问 DOM 中 audio 元素和 button 元素。

```
function toggleSound() {
    var music = document.getElementById("music");
    var toggle = document.getElementById("toggle");
    if (music.paused) {
       music.play();
       toggle.innerHTML = "暂停";
    }
}
```

通过访问 audio 元素的 paused 属性，可以检测到用户是否已经暂停播放。如果音频还没开始播放，那么 paused 属性默认值为 true，这种情况在用户第一次单击按钮的时候遇到。此时，需要调用 play()函数播放音频，同时修改按钮上的文字，提示再次单击就会暂停。

```
else {
   music.pause();
   toggle.innerHTML ="播放";
}
```

相反，如果音频没有暂停，则会使用 pause()函数将它暂停，然后更新按钮上的文字为“播放”，让用户知道下次单击的时候音频将继续播放。

6.6.2　查看视频帧画面

本示例将演示如何抓取 video 元素中的帧画面并显示在动态 canvas 上。当视频播放时，定期从视频中抓取图像帧并绘制到旁边的 canvas 上，当用户单击 canvas 上显示的任何一帧时，所播放的视频会跳转到相应的时间点。

【示例】本示例完整代码如下所示，演示效果如图 6.11 所示。

Note

```
<!DOCTYPE html>
<html>
<title></title>
<video id="movies" autoplay oncanplay="startVideo()" onended="stopTimeline()" autobuffer="true"
    width="400px" height="300px">
    <source src="medias/volcano.ogv" type='video/ogg; codecs="theora, vorbis"'>
    <source src="medias/volcano.mp4" type='video/mp4'>
</video>
<canvas id="timeline" width="400px" height="300px">
<script type="text/javascript">
    var updateInterval = 5000;
    var frameWidth = 100;
    var frameHeight = 75;
    var frameRows = 4;
    var frameColumns = 4;
    var frameGrid = frameRows * frameColumns;
    var frameCount = 0;
    var intervalId;
    var videoStarted = false;
    function startVideo() {
        if (videoStarted)
          return;
        videoStarted = true;
        updateFrame();
        intervalId = setInterval(updateFrame, updateInterval);
        var timeline = document.getElementById("timeline");
        timeline.onclick = function(evt) {
            var offX = evt.layerX - timeline.offsetLeft;
            var offY = evt.layerY - timeline.offsetTop;
            var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
            clickedFrame += Math.floor(offX / frameWidth);
            var seekedFrame = (((Math.floor(frameCount / frameGrid)) *
                                frameGrid) + clickedFrame);
            if (clickedFrame > (frameCount % 16))
                seekedFrame -= frameGrid;
            if (seekedFrame < 0)
              return;
            var video = document.getElementById("movies");
            video.currentTime = seekedFrame * updateInterval / 1000;
            frameCount = seekedFrame;
        }
    }
    function updateFrame() {
        var video = document.getElementById("movies");
        var timeline = document.getElementById("timeline");
        var ctx = timeline.getContext("2d");
        var framePosition = frameCount % frameGrid;
        var frameX = (framePosition % frameColumns) * frameWidth;
        var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
        ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY, frameWidth, frameHeight);
```

```
            frameCount++;
        }
        function stopTimeline() {
            clearInterval(intervalId);
        }
    </script>
</html>
```

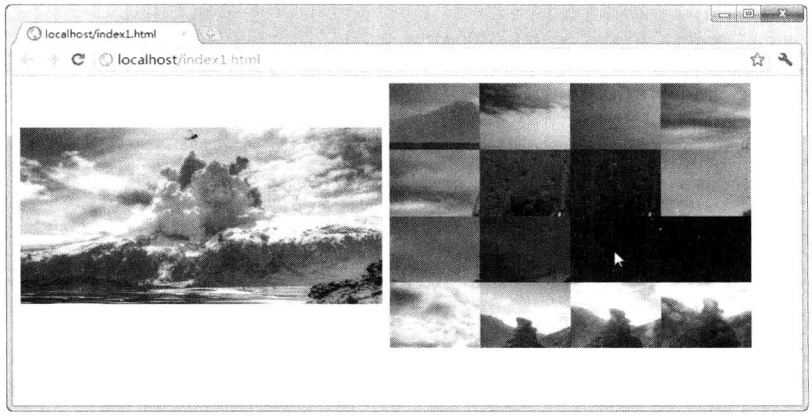


图 6.11　查看视频帧画面

具体操作步骤如下：

第 1 步，添加 video 和 canvas 元素。使用 video 元素播放视频。

```
<video id="movies" autoplay oncanplay="startVideo()" onended="stopTimeline()" autobuffer="true"
    width="400px" height="300px">
    <source src="medias/volcano.ogv" type='video/ogg; codecs="theora, vorbis"'>
    <source src="medias/volcano.mp4" type='video/mp4'>
</video>
```

video 元素声明了 autoplay 属性，这样页面加载完成后，视频会被自动播放。此外还增加了两个事件处理函数，当视频加载完毕，准备开始播放的时候，会触发 oncanplay 函数来执行预设的动作。当视频播放完后，会触发 onended 函数以停止帧的创建。

接着创建 id 为 timeline 的 canvas 元素，以固定的时间间隔在上面绘制视频帧画面。

```
<canvas id="timeline" width="400px" height="300px">
```

第 2 步，添加变量。创建必须的元素之后，为示例编写脚本代码，在脚本中声明一些变量，同时增强代码的可读性。

```
//定义时间间隔，以毫秒为单位
var updateInterval = 5000;
//定义抓取画面显示大小
var frameWidth = 100;
var frameHeight = 75;
//定义行列数
var frameRows = 4;
var frameColumns = 4;
var frameGrid = frameRows * frameColumns;
//定义当前帧
```

```
var frameCount = 0;
var intervalId;
//定义播放完毕取消定时器
var videoStarted = false;
```

Note

变量 updateInterval 控制抓取帧的频率，其单位是毫秒，5000 表示每 5 秒钟抓取一次。frameWidth 和 frameHeight 两个参数用来指定在 canvas 中展示的视频帧画面的大小。frameRcws、frameColumns 和 frameGrid 3 个参数决定了在画布中总共显示多少帧。为了跟踪当前播放的帧，定义了 frameCount 变量。frameCount 变量能够被所有函数调用。intervalId 用来停止控制抓取帧的计时器。videoStarted 标志变量用来确保每个示例只创建一个计时器。

第 3 步，添加 updateFrame 函数。整个示例的核心功能是抓取视频帧并绘制到 canvas 上，它是视频与 canvas 相结合的部分，具体代码如下：

```
//该函数负责把抓取的帧画面绘制到画布上
function updateFrame() {
    var video = document.getElementById("movies");
    var timeline = document.getElementById("timeline");
    var ctx = timeline.getContext("2d");
    //根据帧数计算当前播放位置，然后以视频为输入参数绘制图像
    var framePosition = frameCount % frameGrid;
    var frameX = (framePosition % frameColumns) * frameWidth;
    var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
    ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY, frameWidth, frameHeight);
    frameCount++;
}
```

在操作 canvas 前，首先需要获取 canvas 的二维上下文对象：

```
var ctx = timeline.getContext("2d");
```

这里设计按从左到右、从上到下的顺序填充 canvas 网格，所以需要精确计算从视频中截取的每帧应该对应到哪个 canvas 网格中。根据每帧的宽度和高度，可以计算出它们的起始绘制坐标。

```
var framePosition = frameCount % frameGrid;
var frameX = (framePosition % frameColumns) * frameWidth;
var frameY = (Math.floor(framePosition / frameRows)) * frameHeight;
```

最后是将图像绘制到 canvas 上的关键函数调用。这里向 drawImage ()函数中传入的不是图像，而是视频对象。

```
ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY, frameWidth, frameHeight);
```

canvas 的绘图顺序可以将视频源当做图像或者图案进行处理，这样开发人员就可以方便地修改视频并将其重新显示在其他位置。

当 canvas 使用视频作为绘制源时，画出的只是当前播放的帧。canvas 的显示图像不会随着视频的播放而动态更新，如果希望更新显示内容，需要在视频播放期间重新绘制图像。

第 4 步，定义 startVideo()函数。startVideo()函数负责定时更新画布上的帧画面图像。一旦视频加载成功并播放就会触发 startVideo()函数。因此每次页面加载都仅触发一次 startVideo()，除非视频重新播放。

在该函数中，当视频开始播放后，将抓取第一帧，接着会启用计时器来定期调用 updateFrame()

函数。

```
updateFrame();
intervalId = setInterval(updateFrame, updateInterval);
```

Note

第 5 步，处理用户单击。当用户单击某一帧图像时，将计算帧图像对应视频位置，然后定位到该位置进行播放。代码如下。

```
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
    var offX = evt.layerX - timeline.offsetLeft;
    var offY = evt.layerY - timeline.offsetTop;
    //计算哪个位置的帧被单击
    var clickedFrame = Math.floor(offY / frameHeight) * frameRows;
    clickedFrame += Math.floor(offX / frameWidth);
    //计算视频对应播放到哪一帧
    var seekedFrame = (((Math.floor(frameCount / frameGrid)) * frameGrid) + clickedFrame);
    //如果用户单击的帧位于当前帧之前，则设定是上一轮的帧
    if (clickedFrame > (frameCount % 16))
        seekedFrame -= frameGrid;
    //不允许跳出当前帧
    if (seekedFrame < 0)
      return;
    var video = document.getElementById("movies");
    video.currentTime = seekedFrame * updateInterval / 1000;
    frameCount = seekedFrame;
}
```

第 6 步，添加 stopTimeline()函数。最后要做的工作是在视频播放完毕时，停止视频抓取。

```
function stopTimeline() {
    clearInterval(intervalId);
}
```

视频播放完毕时会触发 onended()函数，stopTimeline()函数会在此时被调用。

第7章

Web 存储

（视频讲解：1 小时 20 分钟）

Web 应用的发展，使得客户端存储的用途也越来越多，而实现客户端存储的方式则是多种多样。最简单且兼容性最佳的方案是 Cookie，但是作为真正的客户端存储，Cookie 存在很多致命伤。此外，在 IE6 及以上版本中还可以使用 userData Behavior，在 Firefox 中可以使用 globalStorage，而在 Flash 插件环境中可以使用 Flash Local Storage，但是这几种方式都存在兼容性方面的局限性，因此都不是很理想的选择。

针对这种情况，HTML5 提出了更加理想的解决方案：如果存储复杂的数据，可以使用 Web Database，该方法可以像客户端程序一样使用 SQL；如果需要存储简单的 key/value（键值对）信息，可以使用 Web Storage。本章将主要从各个方面介绍 Web Storage 和 Web Database 的使用和应用。

7.1 认识 Web Storage

Note

Web Storage（Web 存储）提供了一种方式，可以让 Web 页面实现在客户端浏览器中以一键值对的形式在本地保存数据。在了解 HTML5 的 Web 存储之前，先来回顾一下 HTML4 中使用的 Cookies 存储机制。

7.1.1 Cookie 存储机制的优缺点

Cookie 是 HTML4 中在客户端存储简单用户信息的一种方式，它使用文本来存储信息，当有应用程序使用 Cookie 时，服务器端就会发送 Cookie 到客户端，客户端浏览器将保存该信息。下一次页面请求时，客户端浏览器就会把 Cookie 发送到服务器。Cookie 最典型的应用是用来保存用户信息、用户设置、密码记忆等。

使用 Cookie 有其优点，也有其缺陷，其优点主要表现在以下几个方面。

- ☑ 简单易用。
- ☑ 浏览器负责发送数据。
- ☑ 浏览器自动管理不同站点的 Cookie。

但经过长期地使用，Cookie 也越来越暴露出其先天的不足，主要有以下几点。

- ☑ 使用简单的文本存储数据，所以 Cookie 的安全性很差。Cookie 保存在客户端浏览器，很容易被黑客窃取。
- ☑ Cookie 中存储的数据容量有限，其上限为 4KB。
- ☑ 存储 Cookie 的数量有限，多数浏览器上限为 30 或 50 个，而 IE6 只支持每个域名存储 20 个 Cookie，也有部分浏览器存储 Cookie 的数量高达 300 个。
- ☑ 如果浏览器的安全配置为最高级别，则 Cookie 会失效。
- ☑ Cookie 不适合大量数据的存储，因为 Cookie 会由每个对服务器的请求来传递，从而造成 Cookie 速度缓慢效率低下。

就本地存储的大小而言，IE 的 userData 最少也能支持 640KB，IE8 后已经支持 DOM Storage，但是只支持 IE。Flash 到目前用得最多，其支持的容量为 Cookie 的 25 倍，即 100KB。之后 Google 推出的 Gears 不限制容量，但是需要安装额外的插件。而 HTML5 现在支持的容量至少为 5MB，对于本地存储而言已经足够使用，也有的浏览器会让用户自行设置。

7.1.2 为什么要用 Web Storage

HTML5 的 Web Storage 提供了两种在客户端存储数据的方法，即 localStorage 和 sessionStorage。

☑ localStorage

localStorage 是一种没有时间限制的数据存储方式，可以将数据保存在客户端的硬盘或其他存储器，存储时间可以是一天、两天、几周或几年，浏览器的关闭并不意味着数据也随之消失，当再次打开浏览器时，依然可以访问这些数据。localStorage 用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

☑ sessionStorage

sessionStorage 指的是针对一个 session 的数据存储，即将数据保存在 session 对象中。Web 中的

session 指的是用户在浏览某个网站时，从进入网站到关闭浏览器所经过的这段时间，可以称为用户与浏览器进行交互的“会话时间”。session 对象用来保存这个时间段内所有要保存的数据，当用户关闭浏览器后，这些数据会被删除。

sessionStorage 用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问，并且当会话结束后数据也随之销毁。因此 sessionStorage 不是一种持久化的本地存储，仅仅是会话级别的存储。

Note

从以上介绍可以看出，localStorage 可以永久保存数据，而 sessionStorage 只能暂时保存数据，这是两者之间的重要区别，在具体使用时应该注意。

Web Storage 存储机制比传统的 Cookie 更加强大，弥补了 Cookie 的诸多缺点，主要在以下两个方面做了加强。第一，Web Storage 提供了易于使用的 API 接口，只需设置键值即可使用，简单方便。第二，在存储容量方面可根据用户分配的磁盘配额进行存储，能够在每个用户域存储 5MB~10MB 的内容，用户不仅可以存储 session，还可以存储许多信息，如设置偏好、本地化的数据和离线数据等。

Web Storage 还提供了使用 JavaScript 编程的接口，开发者可以使用 JavaScript 客户端脚本实现许多以前只能在服务器端才能完成的工作。

7.1.3 Web Storage 的优缺点

数据的本地存储一直都是以本地客户端应用为主，因为本地客户端应用相对于 Web 存储（Web Storage）的优势显而易见。在 HTML5 之前，Web 应用的本地存储不能满足大量与复杂数据存储的需要。HTML5 在 Web 存储方面所做的重大改进，包括支持在客户端本地建立一个数据库，这是对 HTML 4 中的 Cookies 存储机制的改善和提高，其应用会越来越广泛。

- ☑ 存储空间：存储空间更大。IE8 下每个独立的存储空间为 10M，其他浏览器实现略有不同，但都比 Cookie 要大很多。
- ☑ 服务器：存储内容不会发送到服务器。当设置了 Cookie 后，Cookie 的内容会随着请求一并发送到服务器，这对于本地存储的数据是一种带宽浪费，而 Web Storage 中的数据则仅仅存在于本地，不会与服务器发生任何交互。
- ☑ 接口：更多丰富易用的接口。Web Storage 提供了一套更为丰富的接口，使得数据操作更为简便。
- ☑ 存储空间：独立的存储空间。每个域（包括子域）都有独立的存储空间，各个存储空间是完全独立的，因此不会造成数据混乱。

Web Storage 的缺点主要集中在其安全性方面，具体体现在以下两点：

- ☑ 浏览器会为每个域分配独立的存储空间，即脚本在域 A 中是无法访问到域 B 中的存储空间的，但是浏览器却不会检查脚本所在的域与当前域是否相同，即在域 B 中嵌入域 A 中的脚本依然可以访问域 B 中的数据。
- ☑ 存储在本地的数据未加密而且永远不会过期，极易造成隐私泄漏。

7.1.4 浏览器支持概述

在 HTML5 的各项特性中，Web Storage 的浏览器支持度是非常好的。其实，目前所有主流浏览器版本都在一定程度上支持 Web Storage，具体说明如表 7.1 所示。

表 7.1 浏览器支持概述

浏 览 器	说 明
IE	8.0 及以上的版本支持
Firefox	3.0 及以上的版本支持
Opera	10.5 及以上的版本支持
Chrome	3.0 及以上的版本支持
Safari	4.0 及以上的版本支持

Note

HTML5 的 Web Storage 因其广泛的支持度而成为 Web 应用中最安全的 API 之一。事实证明各浏览器在 API 方面的实现基本一致，存在一定的兼容性问题，但不影响正常的使用。

7.2 使用 Web Storage

Web Storage 的使用比较简单，下面分别介绍客户端数据的存储和获取，然后分析 localStorage 和 sessionStorage 之间的差异，最后了解其中一些高级特性，为 HTML5 数据开发奠定基础。

7.2.1 检查浏览器的支持性

在 Web Storage API 中，特定域名下的 Storage 数据库可以直接通过 window 对象访问。因此首先确定用户的浏览器是否支持 Web Storage 就非常重要。在编写代码时，只要检测 window. localStorage 和 window. sessionStorage 是否存在即可，详细代码如下。

```
function checkStorageSupport() {
    if(window.sessionStorage) {
        alert('This browser supports sessionStorage');
    } else {
        alert('This browser does NOT support sessionStorage');
    }
    if(window.localStorage) {
        alert('This browser supports localStorage');
    } else {
        alert('This browser does NOT support localStorage');
    }
}
```

许多浏览器不支持从文件系统直接访问文件式的 sessionStorage。所以，在运行本章示例之前，应当确保从 Web 服务器上获取页面。例如，可以通过本地虚拟服务器发出页面请求：

```
http://localhost/test.html
```

对于很多 API 来说，特定的浏览器可能只支持其部分功能，但是因为 Web Storage API 非常小，所以它已经得到了相当广泛的支持，不过出于安全考虑，即使浏览器本身支持 Web Storage，用户仍然可自行选择是否将其关闭。

☑ sessionStorage 测试

测试方法：打开页面 A，在页面 A 中写入当前的 session 数据，然后通过页面 A 中的链接或按钮

进入页面 B，如果页面 B 中能够访问到页面 A 中的数据，则说明浏览器将当前的页面 A、B 视为同一个 session，测试结果如表 7.2 所示。

表 7.2 sessionStorage 兼容性测试

浏 览 器	执行的运算	target="_blank"	window.open	ctrl + click	跨 域 访 问
IE	是	是	是	是	否
Firefox	是	是	是	否（null）	否
Chrome	是	是	是	否（undefined）	否
Safari	是	否	是	否（undefined）	否
Opera	是	否	否	否（undefined）	否

Note

上面主要针对 sessionStorage 的一些特性进行了测试，测试的重点在于各个浏览器对于 session 的定义以及跨域情况。从表 7.2 中可以看出，从安全性考虑，所有浏览器下的 session 数据都是不允许跨域访问的，跨子域也是不允许的。其他方面主流浏览器中的实现较为一致。

API 测试方法包括 setItem(key,value)、removeItem(key)、getItem(key)、clear()和 key(index)，属性包括 length 和 remainingSpace（非标准）。不过存储数据时可以简单地使用 localStorage.key=value 的方式。

标准中定义的接口在各个浏览器中都已实现，此外 IE 下新增了一个非标准的 remainingSpace 属性，用于获取存储空间中剩余的空间。测试结果如表 7.3 所示。

表 7.3 API 测试

浏 览 器	setItem	removeItem	getItem	clear	key	length	remainingSpace
IE	是	是	是	是	是	是	是
Firefox	是	是	是	是	是	是	否
Chrome	是	是	是	是	是	是	否
Safari	是	是	是	是	是	是	否
Opera	是	是	是	是	是	是	否

此外，关于 setItem(key,value)方法中的 value 类型，理论上可以是任意类型，不过实际上浏览器会调用 value 的 toString 方法来获取其字符串值并存储到本地，因此如果是自定义的类型则需要自己定义有意义的 toString 方法。

Web Storage 标准事件为 onStorage，当存储空间中的数据发生变化时触发。此外，IE 自定义了一个 onStoragecommit 事件，当数据写入时触发。onStorage 事件中的事件对象应该支持以下属性。

- ☑ key：被改变的键。
- ☑ oldValue：被改变键的旧值。
- ☑ newValue：被改变键的新值。
- ☑ url：被改变键的文档地址。
- ☑ storageArea：影响存储对象。

对于这一标准的实现，Webkit 内核的浏览器（Chrome、Safari）以及 Opera 是完全遵循标准的，IE 则只实现了 url，Firefox 则均未实现，具体测试结果如表 7.4 所示。

表 7.4　onStorage 事件对象属性测试

浏　览　器	key	oldValue	newValue	url	storageArea
IE	无	无	无	有	无
Firefox	无	无	无	无	无
Chrome	有	有	有	有	有
Safari	有	有	有	有	有
Opera	有	有	有	有	有

Note

此外，不同的浏览器事件注册的方式以及对象也不一致，其中 IE 和 Firefox 在 document 对象上注册，Chrome5 和 Opera 在 window 对象上注册，而 Safari 在 body 对象上注册。Firefox 必须使用 document.addEventListener 注册，否则无效。

7.2.2　设置和获取数据

下面介绍如何使用 sessionStorage 设置和获取网页中的简单数据。设置数据值很简单，具体用法如下：

```
window.sessionStorage.setItem('myFirstKey', 'myFirstValue');
```

使用上面的存储访问语句时，需要注意以下 3 点：

- ☑ 实现 Web Storage 的对象是 window 对象的子对象，因此 window.sessionStorage 包含了开发人员需要调用的函数。
- ☑ setItem 方法需要一个字符串类型的键和一个字符串类型的值来作为参数。虽然 Web Storage 支持传递非字符数据，但是目前浏览器可能还不支待其他数据类型。
- ☑ 调用的结果是将字符串 myFirstKey 设置到 sessionStorage 中，这些数据随后可以通过键 myFirstKey 获取。

获取数据需要调用 get Item 函数。例如，如果把下面的声明语句添加到前面的示例中，浏览器将弹出提示对话框，显示文本 myFirstValue。

```
alert(window.sessionStorage.get Item('myFirstKey'));
```

可以看出，使用 Web Storage 设置和获取数据非常简单。不过，访问 Storage 对象还有更简单的方法：可以使用点语法设置数据，使用这种方法，可完全避免调用 setItem 和 getItem，而只是根据键值的配对关系，直接在 sessionStorage 对象上设置和获取数据。使用这种方法设置数据调用代码可以改写为：

```
window.sessionStorage.myFirstKey = 'myFirstValue';
```

同样，获取数据的代码可以改写为：

```
alert(window.sessionStorage.myFirstKey);
```

JavaScript 允许开发人员设置和获取几乎任何对象的属性，那么为什么还要引入 sessionStorage 对象？其实，二者之间最大的区别在于作用。只要网页是同源的（包括规则、主机和端口），基于相同的键，就都能够在其他网页中获得设置在 sessionStorage 上的数据。在对同一页面后续多次加载的情况下也是如此。大部分开发者对页面重新加载时经常会丢失脚本数据，但通过 Web Storage 保存的数据不再如此了，重新加载页面后这些数据仍然存在。

有时候，一个应用程序会用到多个标签页或窗口中的数据，或多个视图共享的数据。在这种情况下，比较恰当的做法是使用 HTML5 Web Storage 的另一种实现方式 localStorage。localStorage 与 sessionStorage 用法相同，唯一的区别是访问它们的名称不同，分别通过 localStorage 和 sessionStorage 对象来访问。二者在行为上的差异主要是数据的保存时长及其共享方式。

Note

localStorage 数据的生命周期要比浏览器和窗口的生命周期长，同时被同源的多个窗口或者标签页共享；而 sessionStorage 数据的生命周期只在构建它们的窗口或者标签页中可见。

7.2.3 防止数据泄露

数据能够保存多久呢？对于设置在 sessionStorage 中的对象，只要浏览器窗口（或标签）不关闭，它们就会一直存在。当用户关闭窗口或浏览器时，sessionStorage 数据将被清除。所以开发人员不应该把真正有价值的东西放在里面，因为不能任何时候查询这些数据都存在。

sessionStorage 非常适合用于短时存在的流程中，如对话框和向导。如果数据需要存储在多个页面中，同时又不希望用户下一次访问应用程序时重新部署，则可将这些数据存储在 sessionStorage 中。以前，这类数据可能需要通过表单和 cookie 提交，并在页面加载时来回传递，而使用 Storage 可以避免这种开销。

sessionStorage 还有另外一种特殊用法：数据作用域。它解决了一个一直存在于诸多 Web 应用程序的问题，以购买机票的购物应用程序为例，在这个应用程序中，诸如理想的出发返回日期这样的用户偏好数据，可能会在浏览器和服务器间使用 cookie 来回传送，为的是在用户使用应用时，服务器应用程序能记住用户先前选择的偏好数据。

不过，用户打开多个窗口是很常见的，他们可能在查看旅游产品时同时打开多个窗口，比较不同代理商同一时间起飞的航班。这会导致 cookie 系统出现问题，因为如果一个用户在比较价格和是否有票等信息时会在浏览器窗口之间来回切换，很可能会在其中一个窗口设置 cookie 值，而在其他窗口中意外地将这些值应用到 URL 相同的另一个网页的后续操作中。这一现象也被称为数据泄露。其产生的根本原因在于 cookie 能够被同源网页共享。

而使用 sessionStorage 能够跨页面（使用该应用的页面）暂存如启程日期这样的临时数据，又不会将其泄露到用户仍在浏览的其他航班信息的窗口中。这样，不同的偏好信息就会被隔离在预订相应航班的窗口中。

7.2.4 Web Storage 的其他用法

在使用 sessionStorage 或 localStorage 对象的文档中，可以通过 window 对象来获取它们。除了名字和数据的生命周期外，它们的功能完全相同，其实现的接口代码如下。

```
interface Storage {
    readonly attribute unsigned long length;
    getter DOMString key(in unsigned long index);
    getter any getItem(in DOMString key);
    setter creator void setItem(in DOMString key, in any data);
    deleter void removeItem(in DOMString key);
    void clear();
};
```

具体说明如下。

Note

- ☑ 使用 length 属性获取目前 Storage 对象中存储的键值对的数量。注意，Storage 对象是同源的，这意味着 Storage 对象的长度只反映同源情况下的长度。
- ☑ key(index)方法允许获取一个指定位置的键。一般情况下，最有用的是遍历特定 Storage 对象的所有健。键的索引从 0 开始，即第一个键的索引是 0，最后一个键的索引是 index(length-1)。获取到键后，就可以用它来获取其相应的数据。除非键本身或者它前面的键被删除，否则其索引值会在给定 Storage 对象的生命周期内一直保留。
- ☑ getItem(key)函数是根据给定的键返回相应数据的一种方式，另一种方式是将 Storage 对象当做数组，而将键作为数组的索引。在这种情况下，如果 Storage 中不存在指定键，则返回 null。
- ☑ 与 getItem(key)函数类似，setItem(key, value)函数能够将数据存入指定键对应的位置。如果值已存在，则替换原值。需要注意的是设置数据时可能会出错。如果用户已关闭了网站的存储，或者存储已达到其最大容量，那么此时设置数据将会抛出错误。因此，在需要设置数据的场合，务必保证应用程序能够处理此类异常。
- ☑ removeItem(key)函数的作用是删除数据项，如果数据存储在键参数下，则调用此函数会将相应的数据项删除。如果键参数没有对应数据，则不执行任何操作。

提示：与某些数据集或数据框架不同，删除数据项时不会将原有数据作为结果返回。在删除操作前请确保已经存储相应数据的副本。

- ☑ clear()函数能删除存储列表中的所有数据。空的 Storage 对象调用 clear()方法也是安全的，此时调用不执行任何操作。

7.2.5 Web Storage 事件监测

某些复杂情况下，多个网页、标签页或者 Worker 都需要访问存储的数据。此时，应用程序可能会在存储数据被修改后触发一系列操作。对于这种情况，Web Storage 内建立了一套事件通知机制，它可以将数据更新通知发送给监听者。无论监听窗口本身是否存储过数据，与执行存储操作的窗口同源的每个窗口的 window 对象上都会触发 Web Storage 事件。

添加如下事件监听器，即可接收同源窗口的 Storage 事件：

```
window.addEventListener("storage", displayStorageEvent, true);
```

其中事件类型参数是 storage，这样只要有同源的 Storage 事件发生（包括 SessionStorage 和 LocalStorage 触发的事件），已注册的所有事件侦听器作为事件处理程序都会接收到相应的 Storage 事件。Storage 事件的接口形式如下所示。

```
interface StorageEvent : Event {
    readonly attribute DOMString key;
    readonly attribute any oldValue;
    readonly attribute any newValue;
    readonly attribute DOMString url;
    readonly attribute Storage storageArea;
};
```

StorageEvent 对象是传入事件处理程序的第一个对象，包含了与存储变化有关的所有必要信息。key 属性包含了存储中被更新或删除的键。oldValue 属性包含了更新前键对应的数据，newValue

属性包含更新后的数据。如果是新添加的数据，则 oldValue 属性值为 null，如果是被删除的数据，newValue 属性值为 null。url 属性指向 Storage 事件发生的源。storageArea 属性是一个引用，它指向值发生改变的 localStorage 或 sessionStorage 对象，如此一来，处理程序就可以方便地查询到 Storage 中的当前值，或基于其他 Storage 的改变而执行其他操作。

Note

【示例】下面的代码是一个简单的事件处理程序，它以提示框的形式显示在当前页面上触发的 Storage 事件的详细信息。

```
function displayStorageEvent(e) {
    var logged = "key:" + e.key + ", newValue:" + e.newValue + ", oldValue:" + e.oldValue + ", url:" + e.url + ",
storageArea:" + e.storageArea;
    alert(logged);
}
window.addEventListener("storage", displayStorageEvent, true);
```

7.2.6 实例 1：设计网页皮肤

在网页设计中，会用 JavaScript 动态设计网页皮肤，当用户选择某种皮肤样式之后，再次访问该网站或者页面，都将显示相同的皮肤样式。

对于皮肤配置数据，最适合使用 localStorage 进行存储，这样每次访问页面时，都会自动调用 localStorage 数据设置页面样式，避免用户每次访问页面时都重新设置选项，效果如图 7.1 所示。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body onload="colorload();">
<script type="text/javascript">
    // 检测浏览器是否支持 localStorage
    if (typeof localStorage === 'undefined') {
        window.alert("您的浏览器不支持 localStorage。");
    }else{
        window.alert("您的浏览器支持 localStorage。");
    var storage = localStorage;
    // 设置 DIV 背景颜色为红色，并保存 localStorage
    function redSet() {
        var value = "red";
        document.getElementById("divblock").style.backgroundColor = value;
        window.localStorage.setItem("DivBackGroundColor", value);
    }
    // 设置 DIV 背景颜色为绿色，并保存 localStorage
    function greenSet() {
        var value = "green";
        document.getElementById("divblock").style.backgroundColor = value;
        window.localStorage.setItem("DivBackGroundColor", value);
    }
    // 设置 DIV 背景颜色为蓝色，并保存 localStorage
    function blueSet() {
        var value = "blue";
```

```
            document.getElementById("divblock").style.backgroundColor = "blue";
            window.localStorage.setItem("DivBackGroundColor", value);
        }
        function colorload() {
            document.getElementById("divblock").style.backgroundColor = window.localStorage.getItem
("DivBackGroundColor");
        }
    }
    </script>
    <section id="main">
    <button id="redbutton" onclick="redSet()">红色</button>
    <button id="greenbutton" onclick="greenSet()">绿色</button>
    <button id="bluebutton" onclick="blueSet()">蓝色</button>
    <div id="divblock" style="width:500px; height:500px;"></div>
    </body>
    </html>
```

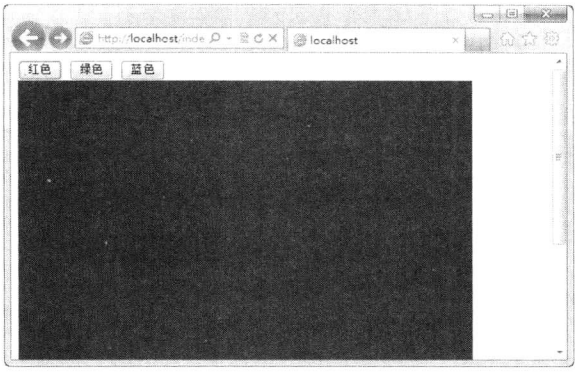


图 7.1　网页皮肤选项

7.2.7　实例 2：跟踪 localStorage 数据

本示例中将调用 localStorage 对象的相关属性和方法，演示如何动态跟踪本地化数据。演示效果如图 7.2 所示。

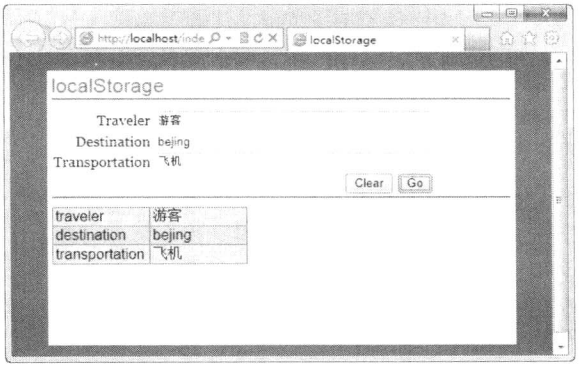


图 7.2　跟踪 localStorage 数据

首先构建一个交互表单的结构。

Note

```
<div id="content">
    <h1> localStorage </h1>
    <div id="form">
        <form id="travelForm">
            <table class="form">
                <tr>
                    <td class="label"> Traveler </td>
                    <td><input type="text" name="traveler" /></td>
                </tr>
                <tr>
                    <td class="label"> Destination </td>
                    <td><input type="text" name="destination" /></td>
                </tr>
                <tr>
                    <td class="label"> Transportation </td>
                    <td><input type="text" name="transportation" /></td>
                </tr>
                <tr>
                    <td colspan="2" class="button"><input id="formSubmit" type="button" value="Clear"
onClick="javascript:dbClear()" />
                                <input id="formSubmit" type="button" value="Go"
onClick="javascript:dbGo()" /> </td>
                </tr>
            </table>
            <input id="inputAction" type="hidden" name="action" value="add" />
            <input id="inputKey" type="hidden" name="key" value="0" />
        </form>
    </div>
    <div id="results"> </div>
</div>
```

然后使用 JavaScript 采集、存储和读写 localStorage 数据。

```
var t = new bwTable();
var db = getLocalStorage() || dispError('Local Storage not supported.');
function getLocalStorage() {
    try {
        if( !! window.localStorage ) return window.localStorage;
    } catch(e) {
        return undefined;
    }
}
function dispResults() {
    if(errorMessage) {
        element('results').innerHTML = errorMessage;
        return;
    }
    var t = new bwTable();
    t.addRow( ['traveler', db.getItem('traveler')] );
    t.addRow( ['destination', db.getItem('destination')] );
    t.addRow( ['transportation', db.getItem('transportation')] );
    element('results').innerHTML = t.getTableHTML();
```

Note

```
}
function dbGo() {
    if(errorMessage) return;
    var f = element('travelForm');
    db.setItem('traveler', f.elements['traveler'].value);
    db.setItem('destination', f.elements['destination'].value);
    db.setItem('transportation', f.elements['transportation'].value);
    dispResults();
}
function dbClear() {
    if(errorMessage) return;
    db.clear();
    dispResults();
}
function initDisp() {
    dispResults();
}
window.onload = function() {
    initDisp();
}
```

7.2.8 实例 3：设计计数器

sessionStorage 可以作为会话计数器，localStorage 则可以作为 Web 应用访问计数器。声明一个 localStorage 计数变量，当刷新页面时，会看到计数器数值在增长，即使关闭浏览器窗口，然后重新访问页面，计数器会继续计数。而 sessionStorage 计数变量只能够在当前会话期间显示页面访问量，即刷新页面会看到计数器在增长，而当关闭浏览器窗口，然后再试一次，计数器已经重置了，演示效果如图 7.3 所示。

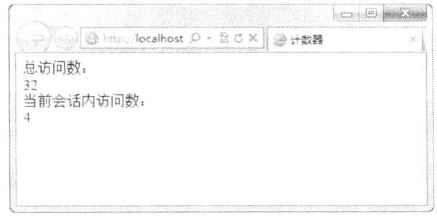


图 7.3　Web 应用计数器

计数器代码如下。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>计数器</title>
</head>
<body>
<script type="text/javascript">
if(localStorage.pagecount) {
    localStorage.pagecount = Number(localStorage.pagecount) + 1;
} else {
```

```
    localStorage.pagecount = 1;
}
document.write("总访问数：<br />" + localStorage.pagecount );
if(sessionStorage.pagecount) {
    sessionStorage.pagecount = Number(sessionStorage.pagecount) + 1;
} else {
    sessionStorage.pagecount = 1;
}
document.write("<br />当前会话内访问数：<br />" + sessionStorage.pagecount );
</script>
</body>
</html>
```

Note

7.2.9 综合应用：Web 应用项目实时跟踪

现在，来将前面所学知识应用到 Web 应用程序中。随着应用变得越来越复杂，无需服务器交互而管理尽可能多的数据变得越来越重要。将数据存储在本地客户端，进而从本地而不是远程获取数据，既可降低网络流量，又可提升浏览器响应能力。

一个困扰开发人员的常见问题是，当用户从应用程序的一个页面切换到另一个页面时如何管理数据。传统实现方式是由服务器存储数据，当用户在网页间切换时来回传递数据。还有一种方法是应用程序尽可能地让用户停留在一个动态更新的网页上。不过用户更习惯于在页面间切换，当用户返回到应用程序的某个页面时，如果能够快速获取数据并加以显示，无疑是增强用户体验的非常好的方式。

在本示例应用程序中，将演示用户在一个网站的页面间切换时，如何将应用程序的临时数据存储在本地，以及如何从每个页面的 Storage 中快速加载。

为了达到这一目的，需要引进一个示例网站，它应该可以保存和恢复 Storage 数据。本示例已经做好了一个包含一个页面的 Storage 数据管理和跟踪网站。当然，用户可以选择任意网站来做演示。关键是网站要包含多个页面，且能让用户在它们之间轻松切换。

这是一个简单的 Web 应用实时跟踪项目，使用 HTML5 的 localStorage 实现，可以在主流浏览器以及 iPhone 或 iPad 上运行。演示效果如图 7.4 所示。

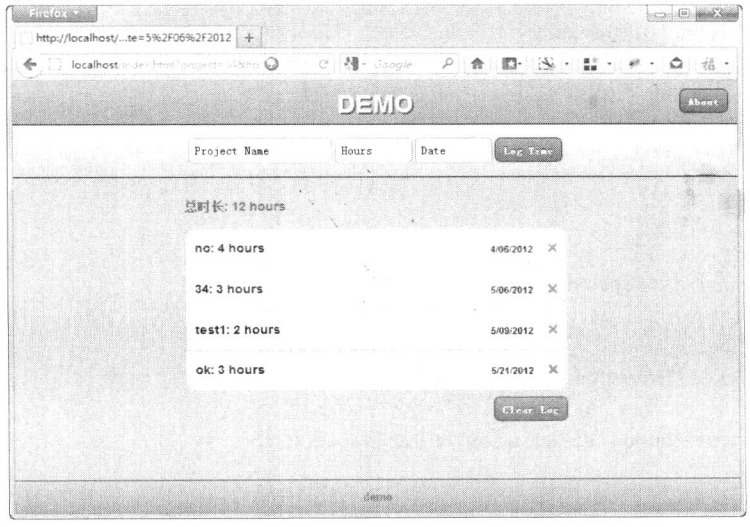


图 7.4 Web 应用项目实时跟踪

操作步骤：

第 1 步，首先构建 Web Storage 本地化管理器页面结构，代码如下。

Note

```
<!DOCTYPE html>
<html lang="en" manifest="monkeeTime.manifest">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title></title>
    <link rel="stylesheet" href="css/reset.css" type="text/css">
    <link rel="stylesheet" href="css/screen.css" type="text/css">
    <link rel="stylesheet" href="scripts/ui-themes/smoothness/jquery-ui-1.8.custom.css" type="text/css">
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script>
    <script type="text/javascript" src="scripts/jquery-ui-1.8.custom.min.js"></script>
    <script src="scripts/monkeeTime.js"></script>
</head>
<body>
    <div id="about" class="hidden">
        <p><strong></p>
        <p><button type="button" id="closeButton">Close</button></p>
    </div>
    <div id="clearLog-message" title="Clear Log" class="hidden">
        确定要清除所有的数据？
    </div>
    <div id="emptyFields-message" title="Missing Required Fields" class="hidden">
        必须提供相应数据。
    </div>
    <header>
        <button type="button" id="aboutButton">About</button>
        <h1>DEMO</h1>
        <form id="logForm">
            <input type="text" name="project" value="Project Name" onfocus="if(this.value == 'Project Name')
this.value = '';" onblur="if(this.value == '') this.value = 'Project Name';">
            <input type="number" name="hours" value="Hours" class="shortField" onfocus="if(this.value ==
'Hours') this.value = '';" onblur="if(this.value == '') this.value = 'Hours';">
            <input type="text" name="date" value="Date" id="date" class="shortField" onfocus="if(this.value
== 'Date') this.value = '';" onblur="if(this.value == '') this.value = 'Date';">
            <input type="submit" value="Log Time">
        </form>
    </header>
    <section>
        <header>
            <p>总时长: <span>0</span> hours</p>
        </header>
        <ul id="theLog">
            <li class="empty">Loading&hellip;</li>
        </ul>
        <button type="button" id="clearLog">Clear Log</button>
    </section>
    <footer>
        <p>demo</p>
```

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

整个页面包含 4 部分结构。

- ☑ 提示框部分，这部分包含 3 个提示框结构： 、 和 ，这 3 个提示框在默认状态下隐藏显示，当触发特定事件之后才能够正常显示。
- ☑ <header>部分是一个表单域结构，允许用户在其中执行 Storage 数据添加操作。
- ☑ <section>部分是一个信息列表模块，在该部分包含一个<header>提示框和一个列表结构，列表显示所有 Storage 数据。
- ☑ <footer>部分是一个待扩展的部分。

Note

第 2 步，引入 jQuery 库，本示例借用了 jQuery 基础库以及 UI 部件，故在头部区域导入 jQuery 基础库和 UI 库。

```
<link rel="stylesheet" href="scripts/ui-themes/smoothness/jquery-ui-1.8.custom.css" type="text/css">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"></script>
<script type="text/javascript" src="scripts/jquery-ui-1.8.custom.min.js"></script>
```

第 3 步，在表单结构中添加初始化脚本，实现文本框的半自动化提示信息。

```
<input type="text" name="project" value="Project Name" onfocus="if(this.value == 'Project Name') this.value = '';" onblur="if(this.value == '') this.value = 'Project Name';">
<input type="number" name="hours" value="Hours" class="shortField" onfocus="if(this.value == 'Hours') this.value = '';" onblur="if(this.value == '') this.value = 'Hours';">
<input type="text" name="date" value="Date" id="date" class="shortField" onfocus="if(this.value == 'Date') this.value = '';" onblur="if(this.value == '') this.value = 'Date';">
```

这里主要通过 onfocus 和 onblur 事件进行跟踪，实现自动填写文本框的提示信息。

第 4 步，从 localStorage 检索所有列表项，并把它们以列表形式显示出来。通过 localStorage 技术从本地读取用户存储的所有添加的项目管理数据，并把它显示出来。

```
function getAllItems() {
    var timeLog = "";
    var i = 0;
    var logLength = localStorage.length-1;
    var totalHours = 0.0;
     //处理每一个项目，并创建一个列表项
     for (i = 0; i <= logLength; i++) {
          var itemKey = localStorage.key(i);
          var values = localStorage.getItem(itemKey);
          values = values.split(";");
          var project = values[0];
          var hours = values[1];
          var date = values[2];
          timeLog += '<li><strong>' + project + '</strong>: ' + hours + ' hours <span class="delete">&times;<span class="hidden" title="Delete Time" id="' + itemKey + '">Are you sure you want to delete ' + project + ' from the log?</span></span> <span class="date">' + date + '</span></li>';
          totalHours = totalHours + parseInt(hours);
```

```
        }
        if (timeLog == "")
            timeLog = '<li class="empty">Log Currently Empty</li>';
    //显示总小时数
    $("section header p span").html(totalHours);
    //更新了所有项目的日志
    $("#theLog").html(timeLog);
    //最后移除底部边框
    $("ul li:last-child").css("border", 0);
    deleteItem(); //在最新的名单中，绑定每个项目对话框
}
```

Note

第 5 步，编写删除项目函数。显示对话框提示删除单个项目，如果允许则执行删除操作。

```
function deleteItem() {
    var deleteItemDialog = new Array();
    $(".delete").each(function() {
        var id = $(this).find("span").attr("id");
        deleteItemDialog[id] = $("#"+id).dialog({
            autoOpen: false,
            resizable: false,
            height: 140,
            modal: true,
            show: 'fold',
            buttons: {
                'Delete Time': function() {
                    localStorage.removeItem(id);
                    getAllItems();
                    $(this).dialog("close");
                },
                Cancel: function() {
                    $(this).dialog("close");
                }
            }
        });
        $(this).click(function(){
            deleteItemDialog[id].dialog("open");
            return false;
        });
    });
}
```

第 6 步，完成两个功能函数之后，在页面初始化事件中调用 getAllItems()，当提交表单时，把对应的项目数据保存到 localStorage 中，当单击清除所有项目时，则弹出提示对话框，并调用 localStorage.clear()方法和 getAllItems()函数，代码如下。

```
$(document).ready(function() {
    if (typeof(localStorage) == 'undefined' ) {
        alert('Your browser does not support HTML5 localStorage. Try upgrading.');
    }
    else {
```

Note

```
getAllItems();
$("#logForm").submit(function(){
    var newDate = new Date();
    var itemId = newDate.getTime();
    var values = new Array();
    var project = $("input[name='project']").val();
    var hours = $("input[name='hours']").val();
    var date = $("input[name='date']").val();
    //去除 HTML 标签
    project = project.replace(/(<([^>]+)>)/ig, "");
    //编码特殊字符
    // name = name.replace(/&/,"&amp;");
    // name = name.replace(/</,"&lt;");
    // name = name.replace(/>/,"&gt;");
    values.push(project);
    values.push(hours);
    values.push(date);
    if (project == "" || project == "Project Name" || hours == "" || hours == "Hours" || date == "" || date
== "Date") {
        $("#emptyFields-message").dialog({
            resizable: false,
            height:140,
            modal: true,
            show: 'fold',
            buttons: {
                Ok: function() {
                    $(this).dialog('close');
                }
            }
        });
    } else {
        try {
            localStorage.setItem(itemId, values.join(';'));
        } catch (e) {
            if (e == QUOTA_EXCEEDED_ERR) {
                alert('Quota exceeded!');
            }
        }
        getAllItems();
    }
});
$("#clearLog").click(function() {
    $("#clearLog-message").dialog({
        resizable: false,
        height:140,
        modal: true,
        show: 'fold',
        buttons: {
            'Delete all items': function() {
                localStorage.clear();
```

```
                        getAllItems();
                        $(this).dialog('close');
                    },
                    Cancel: function() {
                        $(this).dialog('close');
                    }
                }
            });
        });
        $("#aboutButton").click(function() {
            $("#about").slideDown("slow");
        });
        $("#closeButton").click(function() {
            $("#about").slideUp("slow");
        });
    }
});
```

Note

7.3 Web SQL 数据库

HTML5 大大丰富了本地可存储的内容，添加了很多功能将原本必须要保存在服务器上的数据转为保存在本地，从而提高了 Web 应用程序的性能，减轻了服务器端的负担，使 Web 时代重新回到了客户端为重、服务器端为轻的时代。其中，一项非常重要的功能就是数据库的本地存储功能。在 HTML5 中内置了一个可以通过 SQL 语言来访问的数据库。在 HTML 4 中，数据库只能放在服务器端，只能通过服务器来访问数据库，但是在 HTML5 中，可以像访问本地文件那样轻松地对内置数据库进行直接访问了。

7.3.1 Web SQL 数据库概述

以键值对的形式存储数据的 Storage API 在数据持久化方面已经很强大了，但是 HTML5 应用同样可以访问索引数据库。数据库 API 的具体细节仍在完善中，并有多个方案。Web SQL Database 是其中之一，并已经在 Safari、Chrome 和 Opera 中实现了，如表 7.5 所示显示了浏览器对于 Web SQL Database 的支持情况。

表 7.5 浏览器支持概述

浏 览 器	说 明
IE	不支持
Firefox	不支持
Opera	10.5 及以上的版本支持
Chrome	3.0 及以上的版本支持
Safari	3.2 及以上的版本支持

Web SQL Database 允许应用程序通过一个异步 JavaScript 接口访向 SQLLite 数据库。虽然它既不是常见 Web 平台的一部分，也不是 HTML5 规范最终推荐的数据库 API，但当针对如 Safari 移动版这

样的特定平台时，SQL API 很有用。在任何情况下，SQL API 在浏览器中的数据库处理能力都是无可比拟的。与其他 Storage API 一样，浏览器能够限制同源页面可用 Storage 的大小，并且当用户数据被清除时，Storage 中的数据也会被清除。

Note

Web SQL 数据库 API 实际上不是 HTML5 规范的组成部分，而是单独的规范。它通过一套 API 来操纵客户端的数据库。虽然 Web SQL Database 已经在 Safari、Chrome 和 Opera 中实现，但是 IE、Firefox 中并没有实现它，而且 WHATWG 也停止对 Web SQL Database 的开发。由于标准认定直接执行 SQL 语句不可取，Web SQL Database 已被极新的规范——索引数据库（Indexed Database，原为 WebSimpleDB）所取代。索引数据库更简便，而且不依赖于特定的 SQL 数据库版本。目前浏览器正在逐步实现对索引数据库的支持。

7.3.2 使用 Web SQL 数据库

SQL 数据库已经得到了广泛的应用，所以 HTML5 也采用了这种数据库作为本地数据库。如果先掌握了 SQLLite 数据库的基本知识，再学习如何使用 HTML5 的数据库就不困难了。

HTML5 数据库 API 是以一个独立规范形式出现，它包含以下 3 个核心方法。

☑ openDatabase：使用现有数据库或创建新数据库的方式创建数据库对象。

☑ transaction：允许用户根据情况控制事务提交或回滚。

☑ executeSql：用于执行真实的 SQL 查询。

使用 JavaScript 脚本编写 SQLLite 数据库有两个必要的步骤：

☑ 创建访问数据库的对象。

☑ 使用事务处理。

1. 创建或打开数据库

首先，必须使用 openDatabase 方法来创建一个访问数据库的对象。具体用法如下所示。

```
Database openDatabase(in DOMString name, in DOMString version, in DOMString displayName, in unsigned
long estimatedSize, in optional DatabaseCallback creationCallback)
```

openDatabase 方法可以打开已经存在的数据库，如果不存在则创建。openDatabase 中 5 个参数分别表示：数据库名、版本号、描述、数据库大小和创建回调。没有创建回调时也可以创建数据库。

【示例 1】创建了一个数据库对象 db，名称是 Todo，版本编号为 0.1。db 还带有描述信息和大概的大小值。浏览器可使用这个描述与用户进行交流，说明数据库是用来做什么的。利用代码中提供的大小值，浏览器可以为内容留出足够的存储空间。如果需要，这个大小是可以改变的，所以没有必要预先假设允许用户使用多少空间。

```
db = openDatabase("ToDo", "0.1", "A list of to do items.", 200000);
```

为了检测之前创建的连接是否成功，可以检查数据库对象是否为 null：

```
if(!db)
    alert("Failed to connect to database.");
```

注意：使用中绝不可以假设该连接已经成功建立，即使过去对于某个用户它是成功的。一个连接会失败有多个原因：也许浏览器出于安全原因拒绝访问，也许设备存储有限。面对活跃而快速进化的潜在浏览器，对用户机器、软件及其能力作出假设是非常不明智的行为。如当用户使用手持设备时，他们可自由处置的数据可能只有几兆字节。

2. 访问和操作数据库

实际访问数据库的时候，还需要调用 transaction 方法，用来执行事务处理。使用事务处理，可以防止在对数据库进行访问及执行有关操作的时候受到外界的干扰。因为在 Web 上，同时会有许多人都在对页面进行访问。如果在访问数据库的过程中，正在操作的数据被别的用户修改，会引起很多意想不到的后果。因此，可以使用事务来达到在完成操作之前，阻止别的用户访问数据库的目的。

Note

transaction 方法的使用方法如下所示。

```
db.transaction( function(tx) {})
```

transaction 方法使用一个回调函数作为参数。在这个函数中，执行访问数据库的语句。

在 transaction 的回调函数内，使用了作为参数传递给回调函数的 transaction 对象的 executeSql 方法。executeSql 方法的完整定义如下所示。

```
transaction.executeSql(sqlquery,[],dataHandler, errorHandler):
```

该方法使用 4 个参数，第一个参数为需要执行的 SQL 语句。

第二个参数为 SQL 语句中所有使用到的参数的数组。在 executeSql 方法中，将 SQL 语句中所要使用到的参数先用“?”代替，然后依次将这些参数组成数组放在第二个参数中，如下所示。

```
transaction.executeSql("UPDATE people set age-? where name=?;",[age, name]);
```

第三个参数为执行 sql 语句成功时调用的回调函数。该回调函数的传递方法如下所示。

```
function dataRandler(transaction, results){//执行 SQL 语句成功时的处理
}
```

该回调函数使用两个参数，第一个参数为 transaction 对象，第二个参数为执行查询操作时返回的查询到的结果数据集对象。

第四个参数为执行 SQL 语句出错时调用的回调函数。该回调函数的传递方法如下所示。

```
function errorHandler(transaction,errmeg) {//执行 sql 语句出错时的处理
}
```

该回调函数使用两个参数，第一个参数为 transaction 对象，第二个参数为执行发生错误时的错误信息文字。

【示例 2】下面将在 mydatabase 数据库中创建表 t1，并执行数据插入操作，完成插入两条记录。

```
var db = openDatabase('mydatabase', '2.0', my db', 2 * 1024);
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
    tx.executeSql('INSERT INTO t1 (id, log) VALUES (1, "foobar")');
    tx.executeSql('INSERT INTO t1 (id, log) VALUES (2, "logmsg")');
});
```

在插入新记录时，还可以传递动态值：

```
var db = openDatabase(' mydatabase ', '2.0', 'my db', 2 * 1024);
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
    tx.executeSql('INSERT INTO t1 (id,log) VALUES (?, ?'), [e_id, e_log];   //e_id 和 e_log 是外部变量
});
```

当执行查询操作时，从查询到的结果数据集中依次把数据取出到页面中，最简单的方法是使用 for 语句循环。结果数据集对象有一个 rows 属性，其中保存了查询到的每条记录，记录的条数可以用 rows.length 来获取，可以用 for 循环，用 rows[index]或 rows.Item ([index])的形式来依次取出每条数据。在 JavaScript 脚本中，一般采用 rows[index]的形式。另外在 Chrome 浏览器中，不支持 rows.Item ([index]) 的形式。

Note

【示例 3】如果要读取已经存在的记录，可以使用一个回调函数来捕获结果，并通过 for 语句循环显示每条记录。

```
var db = openDatabase(mydatabase, '2.0', 'my db', 2*1024);        db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS t1 (id unique, log)');
    tx.executeSql('INSERT INTO t1 (id, log) VALUES (1, "foobar")');
    tx.executeSql('INSERT INTO t1 (id, log) VALUES (2, "logmsg")');
});
db.transaction(function (tx) {
    tx.executeSql('SELECT * FROM t1, [], function (tx, results) {
    var len = results.rows.length, i;
    msg = "<p>Found rows: " + len + "</p>";
    document.querySelector('#status').innerHTML +=    msg;
    for (i = 0; i < len; i++){
        alert(results.rows.item(i).log );
    }
 }, null);
});
```

7.3.3 实例 1：创建简单的本地数据库

本实例将完整地演示 Web SQL Database API 的使用，包括建立数据库、建立表格、插入数据、查询数据、将查询结果显示。在最新版本的 Chrome、Safari 或 Opera 浏览器中输出结果如图 7.5 所示。

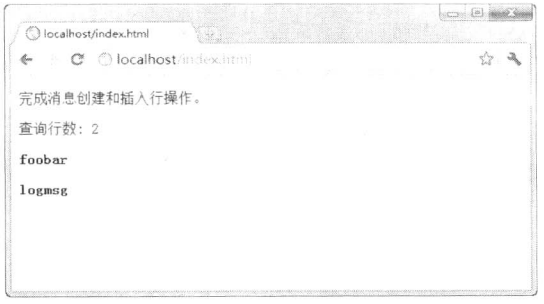


图 7.5 创建简单的本地数据库

实例完整代码如下：

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
    var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
    var msg;
    db.transaction(function(tx) {
```

Note

```
            tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
            tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
            tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
            msg = '<p>完成消息创建和插入行操作。</p>';
            document.querySelector('#status').innerHTML = msg;
        });
        db.transaction(function(tx) {
            tx.executeSql('SELECT * FROM LOGS', [], function(tx, results) {
                var len = results.rows.length, i;
                msg = "<p>查询行数：" + len + "</p>";
                document.querySelector('#status').innerHTML += msg;
                for( i = 0; i < len; i++) {
            msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
            document.querySelector('#status').innerHTML += msg;
                }
            }, null);
        });
    </script>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        </head>
    <body>
    <div id="status" name="status">
    </div>
    </body>
    </html>
```

其中第 5 行的 var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);建立一个名称为 mydb 的数据库，它的版本为 1.0，描述信息为 Test DB，大小为 2M 字节。可以看到此时有数据库建立，但并无表格建立，如图 7.6 所示。

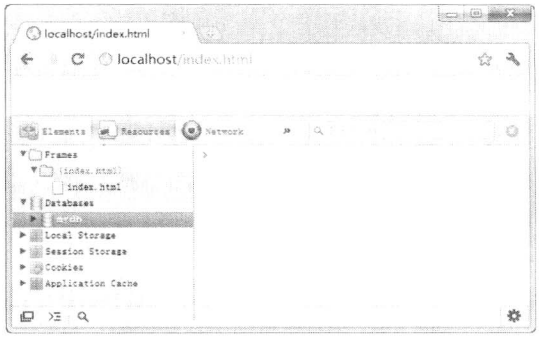


图 7.6　创建数据库 mydb

openDatabase 方法打开一个已经存在的数据库，如果数据库不存在则创建数据库，创建数据库包括数据库名、版本号、描述、数据库大小和创建回调函数。最后一个参数用于创建回调函数，在创建数据库时调用，但即使没有这个参数，一样可以在运行时创建数据库。

第 7 行~第 13 行代码：

```
db.transaction(function(tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
    tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
```

```
    tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
    msg = '<p>完成消息创建和插入行操作。</p>';
    document.querySelector('#status').innerHTML = msg;
});
```

Note

通过第 8 行语句可以在 mydb 数据库中建立一个 LOGS 表格。在这里只执行创建表格语句，而不执行后面两个插入操作时，将在 Chrome 中看到在数据库 mydb 中建立的 LOGS 表格，但表格 LOGS 为空。

第 9 行和第 10 行执行插入操作，在插入新记录时，还可以传递动态值。

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
    tx.executeSql('INSERT INTO LOGS (id,log) VALUES (?, ?'), [e_id, e_log];
});
```

这里的 e_id 和 e_log 为外部变量，executeSql 在数组参数中将每个变量映射到“？”。在插入操作执行后，可以在 Chrome 中看到数据库的状态，可以看到插入的数据，此时并未执行查询语句，页面中并未出现查询结果，如图 7.7 所示。

图 7.7 创建数据表并插入数据

如果要读取已经存在的记录，需使用一个回调函数捕获结果，如上面的第 15 行~第 25 行代码如下。

```
db.transaction(function(tx) {
    tx.executeSql('SELECT * FROM LOGS', [], function(tx, results) {
        var len = results.rows.length, i;
        msg = "<p>查询行数：" + len + "</p>";
        document.querySelector('#status').innerHTML += msg;
        for( i = 0; i < len; i++) {
    msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
    document.querySelector('#status').innerHTML += msg;
        }
    }, null);
});
```

执行查询之后，将信息输出到页面中，可以看到页面中的查询数据。

注意：如果不必要，不要使用 Web SQL Database，因为它会让代码更加复杂（如匿名内部类的内部函数、回调函数等）。大多数情况下，本地存储或会话存储就能够完成相应的任务，

尤其是能够保持对象状态持久化的情况。通过这些 HTML5 Web SQL Database API 接口，可以获得更多功能，相信以后会出现一些非常优秀的、建立在这些 API 之上的应用程序。

7.3.4 实例 2：批量存储本地数据

Note

Web SQL Database 操作数据比较烦琐，为了提高代码执行效率，下面通过一个示例演示如何通过数组实现快速存储数据。

```
<!DOCTYPE html>
<title>Web SQL Database</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<script>
    var db = openDatabase('db', '1.0', 'my first database', 2 * 1024 * 1024);
    function log(id, name) {
        var row = document.createElement("tr");
        var idCell = document.createElement("td");
        var nameCell = document.createElement("td");
        idCell.textContent = id;
        nameCell.textContent = name;
        row.appendChild(idCell);
        row.appendChild(nameCell);
        document.getElementById("racers").appendChild(row);
    }
    function doQuery() {
        db.transaction(function (tx) {
            tx.executeSql('SELECT * from mytable', [], function(tx, result) {
                for (var i=0; i<result.rows.length; i++) {
                    var item = result.rows.item(i);
                    log(item.id, item.name);
                }
            });
        });
    }
    function initDatabase() {
        var names = ["张三", "李四", "王五", "赵六", "侯七", "abc", "def"];
        db.transaction(function (tx) {
            tx.executeSql('CREATE TABLE IF NOT EXISTS mytable(id integer primary key autoincrement,
name)');
            for (var i=0; i<names.length; i++) {
                tx.executeSql('INSERT INTO mytable (name) VALUES (?)', [names[i]]);
            }
            doQuery();
        });
    }
    initDatabase();
</script>
<table id="racers" border="1" cellspacing="0" style="width:100%">
    <th>Id</th>
    <th>Name</th>
</table>
```

首先，创建一个本地数据库 db，把预存储的数据放在数组 names 中，使用 for 语句执行批量操作，这样就省略了编写大量的 executeSql 语句，当数据量比较大时，这种批量操作方式就更加有效。

执行 initDatabase()函数，完成数据初始化存储操作，然后调用 doQuery()函数，再次使用 for 语句把本地数据库中的数据读取出来，并通过 log()显示函数把数据显示在页面表格中。最后显示效果如图 7.8 所示。

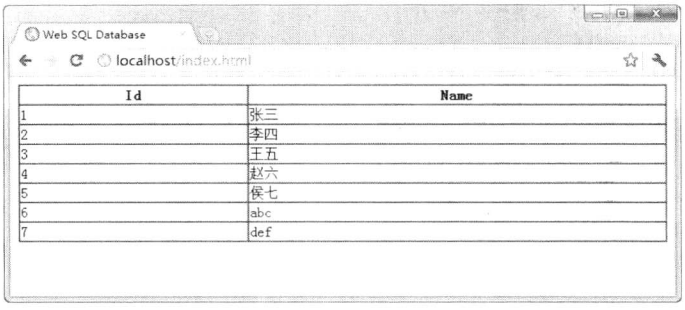


图 7.8 批量存储本地数据

数据库操作可能需要花点时间才能完成。不过，在获得查询结果集之前，查询操作会在后台运行，以避免阻塞脚本的执行。executeSQL()的第三个参数是回调函数，查询得到的事务和结果集将作为参数供此回调函数使用。

7.3.5 综合应用：Web Storage 和 Web SQL 混合开发

Web Storage 使用 getItem 方法读取数据，虽然这种一对一的数据读写方法使用起来比较方便，但是在实际使用过程中作用并不是很大，因为如果要保存的数据量比较大的话，使用这种方法会非常麻烦。

例如，以一个简单 Web 留言本示例看一下如何利用 Web Storage 来保存和读取大量数据。使用一个多行文本框来输入数据，当单击按钮时将文本框中的数据保存到 localStorage 中，在表单下部放置一个 p 元素来显示保存后的数据。

如果只保存文本框中的内容，并不能知道该内容是什么时候写好的，所以保存该内容的同时，也保存了当前日期和时间，并将该日期和时间一并显示在 p 元素中。

利用 Web Storage 保存数据时，数据必须是“键名/键值”这样的格式，所以将文本框的内容作为键值，保存时的日期和时间作为键名来进行保存，计算机中对于日期和时间的值是以时间戳的形式进行管理的，所以保存时不可能存在重复的键名。

整个页面的具体代码如下，页面显示效果如图 7.9 所示。

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>Web 留言本</title>
<script type="text/javascript">
function saveStorage(id){
    var data = document.getElementById(id).value;
    var time = new Date().getTime();
    localStorage.setItem(time,data);
    alert("数据已保存。");
```

```
        loadStorage('msg');
    }
    function loadStorage(id){
        var result = '<table border="1">';
        for(var i = 0;i < localStorage.length;i++) {
            var key = localStorage.key(i);
            var value = localStorage.getItem(key);
            var date = new Date();
            date.setTime(key);
            var datestr = date.toGMTString();
            result += '<tr><td>' + value + '</td><td>' + datestr + '</td></tr>';
        }
        result += '</table>';
        var target = document.getElementById(id);
        target.innerHTML = result;
    }
    function clearStorage(){
        localStorage.clear();
        alert("全部数据被清除。");
        loadStorage('msg');
    }
    </script>
    </head>
    <body>
    <h1>Web 留言本</h1>
    <textarea id="memo" cols="60" rows="10"></textarea><br>
    <input type="button" value="追加" onclick="saveStorage('memo');">
    <input type="button" value="初始化" onclick="clearStorage('msg');">
    <hr>
    <p id="msg"></p>
    </body>
    </html>
```

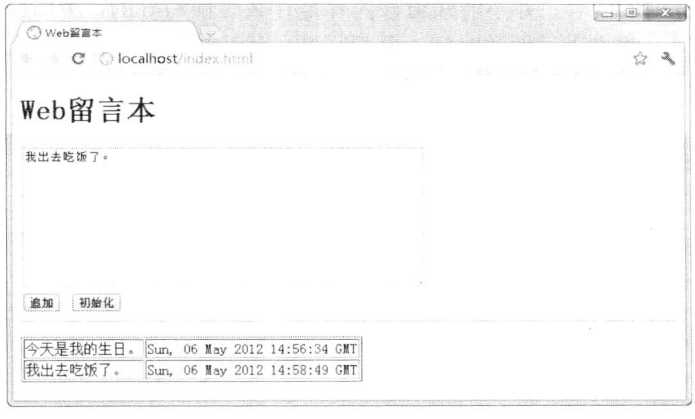


图 7.9　使用 localStorage 存储数据的 Web 留言本

在该页面中，除了输入数据用的文本框与显示数据用的 p 元素之外，还放置了追加按钮与初始化按钮，单击追加按钮来保存数据，单击初始化按钮来消除全部数据，如图 7.9 所示。

在 JavaScript 脚本里含有 3 个供按钮调用的函数，分别是 saveStorage、loadStorage 和 clearStorage，简单说明如下。

Note

- ☑ saveStorage 函数：这个函数比较简单，使用 new Date().getTime()语句得到了当前的日期和时间，然后调用 localStorage.setItem 方法，将得到的时间作为键值，并将文本框中的数据作为键名进行保存。保存完毕后，重新调用脚本中的 loadStorage 函数在页面上重新显示保存后的数据。
- ☑ loadStorage 函数：取得保存后的所有数据，然后以表格的形式进行显示。取得全部数据的时候，需要用到 localStorage 两个比较重要的属性。
  - ➢ loadStorage.length：返回所有保存在 localStorage 中的数据的条数。
  - ➢ localStorage.key(index)：将想要得到数据的索引号作为 index 参数传入，可以得到 localStorage 中与这个索引号对应的数据。如想得到第 6 条数据，传入的 index 为 5（index 是从 0 开始计算的）。

先用 localStorage.length 属性获取保存数据的条数，然后做一个循环，在循环内用一个变量，从 0 开始将该变量作为 index 参数传入 localStorage.key(index)属性，每次循环时该变量加 1，通过这种方法取得保存在 localStorage 中的所有数据。

- ☑ clearStorage 函数：将 localStorage 中保存的数据全部清除，在这个函数中只有一条语句 localStorage.clear();，当调用 localStorage 的 clear 方法时，所有保存在 localStorage 中的数据会被全部清除。

那么能不能将 Web Storage 作为简易数据库来利用呢?

如果想要将 Web Storage 作为数据库来利用，必须要考虑几个问题。

首先，在数据库中，大多数表都分为几列，怎样对列来进行管理呢？其次，怎样对数据进行检索呢？如果能够解决这些问题，就可以将 Web Storage 作为数据库来利用了。

例如，设计一个客户联系信息管理网页。客户的联系信息包括姓名、E-mail 地址、电话号码、备注列，把它们保存在 localStorage 中，如果输入客户的姓名并且进行检索，可以获取这个客户的所有联系信息。

首先，用客户的姓名作为键名来保存数据，这样在获取客户其他信息的时候会比较方便，那么怎样将客户的联系信息分几列来进行保存呢？

要做到这一点，需要使用 JSON 数据格式。用这种 JSON 格式作为文本来保存对象，获取该对象时再通过 JSON 格式来获取，就可以在 Web Storage 中保存和读取具有复杂结构的数据了。

整个页面的具体代码如下：

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title></title>
<script type="text/javascript">
function saveStorage(){
    var data = new Object;
    data.name = document.getElementById('name').value;
    data.email = document.getElementById('email').value;
    data.tel = document.getElementById('tel').value;
    data.memo = document.getElementById('memo').value;
    var str = JSON.stringify(data);
    localStorage.setItem(data.name,str);
```

```
        alert("数据已保存。");
    }
    function findStorage(id){
        var find = document.getElementById('find').value;
        var str = localStorage.getItem(find);
        var data =   JSON.parse(str);
        var result = "姓名: " + data.name + '<br>';
        result += "EMAIL: " + data.email + '<br>';
        result += "电话号码: " + data.tel   + '<br>';
        result += "备注: " + data.memo + '<br>';
        var target = document.getElementById(id);
        target.innerHTML = result;
    }
    </script>
    </head>
    <body>
    <h1>使用 Web Storage 模拟数据库</h1>
    <table>
        <tr><td>姓名:</td><td><input type="text" id="name"></td></tr>
        <tr><td>EMAIL:</td><td><input type="text" id="email"></td></tr>
        <tr><td>电话号码:</td><td><input type="text" id="tel"></td></tr>
        <tr><td>备注:</td><td><input type="text" id="memo"></td></tr>
        <tr>
            <td></td>
            <td><input type="button" value="保存" onclick="saveStorage();"></td>
        </tr>
    </table>
    <hr>
    <p>检索:<input type="text" id="find">
            <input type="button" value="检索" onclick="findStorage('msg');">
    </p>
    <p id="msg"></p>
    </body>
```

在 JavaScript 脚本中存放了两个函数，分别是保存数据用的 saveStorage 函数与检索数据用的 findStorage 函数。

saveStorage 函数中的流程如下：

第 1 步，从各输入文本框中获取数据。

第 2 步，创建对象，将获取的数据作为对象的属性进行保存。

第 3 步，将对象转换成 JSON 格式的文本数据。

第 4 步，将文本数据保存在 localStorage 中。

为了将数据保存在一个对象中，使用 new Object 语句创建了一个对象，将各种数据保存在该对象的各个属性中，然后，为了将对象转换成 JSON 格式的文本数据，使用了 JSON 对象的 stringify 方法，该方法的使用方法如下所示。

```
var str = JSON.tringify(data);
```

该方法接受一个参数 data，它表示要转换成 JSON 格式文本数据的对象，这个方法的作用是将对象转换成 JSON 格式的文本数据，并将其返回。

Note

findStorage 函数中的流程如下：

第 1 步，在 localStorage 中将检索用的姓名作为键值，获取对应的数据。

第 2 步，将获取的数据转换成 JSON 对象。

第 3 步，取得 JSON 对象的各个属性值，创建要输出的内容。

第 4 步，将要输出的内容在页面上输出。

该函数的关键是使用 JSON 对象的 parse 方法，将从 localStorage 中获取的数据转换成 JSON 对象。该函数的使用方法如下所示。

```
var data = JSON.parse(str);
```

该方法接受一个参数 str，它表示从 localStorage 中取得的数据，该方法的作用是将传入的数据转换成 JSON 对象，并且将该对象返回。

上面示例的关键是利用了 JSON 对象的 stringify 方法与 parse 方法。注意，这个 JSON 对象只被大部分最新版本的浏览器所支持，而不是所有浏览器、所有版本都支持。现在支持 JSON 对象的浏览器有 IE 8+、Firefox 3.6+、Chrome +、Safari 5+和 Opera 10+。

示例的演示效果如图 7.10 所示。

图 7.10　使用 Web Storage 模拟数据库

上面示例演示了如何使用 Web Storage 设计 Web 留言本，下面来看一下怎样利用 Web SQL 数据库实现同样的功能。

首先，页面中包含一个可输入姓名的文本框，一个输入留言的文本框，以及一个保存数据时用的按钮。在按钮下面放置了一个表格，保存数据后从数据库中重新取得所有数据，然后把数据显示在这个表格中。单击按钮时，调用 saveData 函数，保存数据时的处理都被写在了这个函数里。

另外，打开页面时将调用 init 函数，将数据库中全部已保存的留言信息显示在表格中。

整个页面的具体代码如下：

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>使用 Web SQL 设计 Web 留言本</title>
<script type="text/javascript">
var datatable = null;
var db = openDatabase('MyData', '', 'My Database', 102400);
```

```
    function init(){
        datatable = document.getElementById("datatable");
        showAllData();
    }
    function removeAllData(){
        for (var i =datatable.childNodes.length-1; i>=0; i--){
            datatable.removeChild(datatable.childNodes[i]);
        }
        var tr = document.createElement('tr');
        var th1 = document.createElement('th');
        var th2 = document.createElement('th');
        var th3 = document.createElement('th');
        th1.innerHTML = '姓名';
        th2.innerHTML = '留言';
        th3.innerHTML = '时间';
        tr.appendChild(th1);
        tr.appendChild(th2);
        tr.appendChild(th3);
        datatable.appendChild(tr);
    }
    function showData(row) {
        var tr = document.createElement('tr');
        var td1 = document.createElement('td');
        td1.innerHTML = row.name;
        var td2 = document.createElement('td');
        td2.innerHTML = row.message;
        var td3 = document.createElement('td');
        var t = new Date();
        t.setTime(row.time);
        td3.innerHTML=t.toLocaleDateString()+" "+t.toLocaleTimeString();
        tr.appendChild(td1);
        tr.appendChild(td2);
        tr.appendChild(td3);
        datatable.appendChild(tr);
    }
    function showAllData(){
        db.transaction(function(tx) {
            tx.executeSql('CREATE TABLE IF NOT EXISTS MsgData(name TEXT, message TEXT, time
INTEGER)', []);
            tx.executeSql('SELECT * FROM MsgData', [], function(tx, rs) {
                removeAllData();
                for(var i = 0; i < rs.rows.length; i++){
                    showData(rs.rows.item(i));
                }
            });
        });
    }
    function addData(name, message, time) {
        db.transaction(function(tx) {
            tx.executeSql('INSERT INTO MsgData VALUES(?, ?, ?)',[name, message, time],function(tx, rs)
```

Note

```
            {
                alert("成功保存数据!");
            },
            function(tx, error) {
                alert(error.source + "::" + error.message);
            });
        });
}
function saveData(){
    var name = document.getElementById('name').value;
    var memo = document.getElementById('memo').value;
    var time = new Date().getTime();
    addData(name,memo,time);
    showAllData();
}
</script>
</head>
<body onload="init();">
<h1>使用 Web SQL 设计 Web 留言本</h1>
<table>
    <tr><td>姓名：</td><td><input type="text" id="name"></td></tr>
    <tr><td>留言：</td><td><input type="text" id="memo"></td></tr>
    <tr>
<td></td>
<td><input type="button" value="保存" onclick="saveData();"></td></tr>
</table><hr>
<table id="datatable" border="1"></table>
<p id="msg"></p>
</body>
</html>
```

下面重点分析 JavaScript 脚本代码。

☑ 打开数据库

打开数据库的代码如下所示。

```
var datatable = null;
var db = openDatabase('MyData', '', 'My Database', 102400);
```

在 JavaScript 脚本一开始，使用了一个变量 datatable。用这个变量来代表页面中的 table 元素。db 变量代表使用 openDatabase 方法创建的数据库访问对象。在示例中创建了 MyData 数据库并对其进行访问。

☑ 初始化

编写 init 函数，该函数在页面打开时调用。为了在打开页面时就往页面表格中装入数据，所以在该函数中首先设定变量 datatable 为页面中的表格，然后调用脚本中另一个函数 showAllData 来显示数据。

☑ 清除表格中当前显示的数据

removeAllData 函数是在 showAllData 函数中被调用的一个必不可少的函数，它的作用是将页面中 table 元素下的子元素全部清除，只留下一个空表格框架，然后填入表头。这样在页面表格中当前显示的数据就全部被清除了，以便重新读取数据并装入表格。

☑ 显示数据

showData 函数使用一个 row 参数，该参数表示从数据库中读取到的一行数据。该函数在页面表格中使用 tr 元素添加一行，并使用 td 元素添加各列，然后将传入的这行数据分别填入表格中添加的这一行所对应的各列中。

Note

☑ 显示全部数据

showAllData 函数使用 transaction 方法，在该方法的回调函数中执行 executeSql 方法获取全部数据。获取数据之后，首先调用 removeAllData 函数初始化页面表格，将该表格中当前显示的数据全部清除，然后在循环中调用 showData 函数，将获取到的每一条数据作为参数传入，在页面上的表格中逐条显示获取到的每条数据。

☑ 追加数据

addData 函数在 saveData 函数中被调用。在 addData 函数中，使用 transaction 方法，在该方法的回调函数中执行 executeSql 方法，将作为参数传入的数据保存在数据库中。

☑ 保存数据

saveData 函数首先调用 addData 函数追加数据，然后调用 showAllData 函数重新显示表格中的全部数据。

示例演示效果如图 7.11 所示。

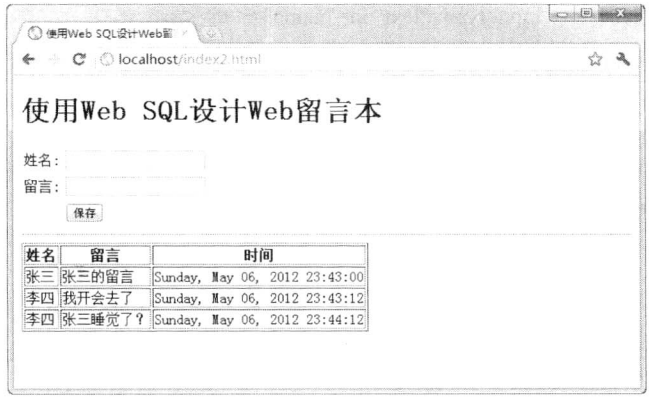


图 7.11　使用 Web SQL 设计 Web 留言本

第8章

离线应用

（视频讲解：23分钟）

所有的浏览器都有自己的缓存机制，但那些机制并不可靠，而且难以控制，在Web开发时经常会因为浏览器缓存问题而让人烦恼不已。HTML5通过ApplicationCache接口解决了这个问题，并且使离线存储成为可能，离线存储使得Web应用可以在用户离线的状况下进行访问。

HTML5离线存储（Offline Storage）功能非常强大，它的核心应用是：在用户没有与因特网连接时，依然能够访问站点或应用，在用户与因特网连接时，自动更新缓存数据。所以，如下所示，可以利用HTML5的离线存储功能开发出一些丰富的基于Web的应用。离线缓存技术包含了两部分内容。

- manifest缓存清单：manifest缓存文件包含了一些需要缓存的资源清单。
- JavaScript接口：提供了用于更新缓存文件的方法以及对缓存文件的操作。

HTML5应用不需要始终保持与网络连接，目前最新的主流的浏览器中都已添加了对HTML5的离线存储功能的支持。本章将详细讲解如何创建HTML5离线Web应用，介绍HTML5离线功能中的离线资源缓存、在线状态检测、本地数据存储等内容，并举例说明如何使用HTML5的新特性开发离线应用，帮助开发人员更加灵活地控制缓存资源的加载。

8.1 HTML5 离线应用概述

Note

HTML5 作为新一代 HTML 标准，包含了对离线应用的支持。在页面中加载数据时，允许用户自定义一些要缓存的图片、Flash、CSS、JavaScript 和 HTML 等文件，在不能联网的情况下，还可以用那些缓存的文件，这就是 HTML5 的离线应用。不过当网络连接正常时，Web 应用程序可以保证及时更新，因为用户每次使用，应用程序都会从远程位置更新加载相关数据。

8.1.1 为什么要学习 HTML5 离线应用

想一想，打开一个页面，加载完成后突然断网了，刷新页面后内容就没了，这是什么感觉。如果刷新页面后还是刚才页面，在新窗口中重新访问该页面，输入相同的网址，在断网的状态下打开的还是原来那个页面，这又是什么感觉。

Web 2.0 鼓励个人的参与，每个人都是 Web 内容的撰写者。如果 Web 应用能够提供离线功能，让用户在不能联网时也能进行内容撰写，有网络时再同步到 Web 上，就大大方便了用户的使用。

越来越多的应用被移植到 Web 上，但事实上，网络连接中断时有发生，如外出旅行、身处无网环境等。间断性的网络连接一直是网络计算系统的致命弱点，如果应用程序完全依赖于与网络的通信，而网络又无法连接时，用户就无法正常使用应用程序了。

但是移动因特网快速发展，在全球互联的时代，离线应用具有巨大的实用价值。可以说如今网络无处不在，而且非常稳定，不存在没有网络的情况。如果经常需要外出，或者移动设备信号不好，或者应用程序只需要偶尔进行网络通信，那么只要在本地存储了应用资源，无论是否连接网络都可用。随着完全依赖于浏览器的设备的出现，Web 应用程序在不稳定的网络状态下还能持续工作就变得更加重要。在这方面，不需要持续连接网络的桌面应用程序历来被认为比 Web 应用程序更有优势。

HTML5 的缓存控制机制综合了 Web 应用和桌面应用两者的优势，基于 Web 技术构建的 Web 应用程序，可在浏览器中运行并在线更新，也可在脱机情况下使用。然而，因为目前的 Web 服务器不为脱机应用程序提供任何默认的缓存行为，所以要想使用离线应用功能，必须在应用中明确声明。

HTML5 的离线应用缓存使得在无网络连接状态下运行应用程序成为可能，这类应用程序用处很多，如在起草电子邮件草稿时就无需连接因特网。HTML5 中引入了离线应用缓存，有了它 Web 应用程序就可以在没有网络连接的情况下运行。

应用程序开发人员可以指定 HTML5 应用程序中的具体资源（如 HTML、CSS、JavaScript 和图像等）在脱机时可用。离线应用的用处很多，简单举例说明如下：

- ☑ 阅读和撰写电子邮件。
- ☑ 编辑文档。
- ☑ 编辑和显示演示文档。
- ☑ 创建待办事宜列表。
- ☑ 用户可以离线访问 Web 应用，不用时刻保持与因特网的连接。
- ☑ 因为文件被缓存在本地，提升了页面加载速度。
- ☑ 离线应用只加载被修改过的资源，因此大大降低了用户请求对服务器造成的负载压力。

在 Web 应用中使用缓存的原因之一是为了支持离线应用。使用离线存储，避免了加载应用程序时所需的常规网络请求。如果缓存清单（Cache Manifest）文件是最新的，浏览器就无需检查其他资

源是否最新。大部分应用程序可以非常迅速地从本地应用缓存中加载完成。此外，从缓存中加载资源可节省带宽，而不必用多个 HTTP 请求确定资源是否已经更新，这对于移动 Web 应用是至关重要的。目前，加载速度慢是 Web 应用比不上桌面应用的一个方面，缓存则可以解决这一问题。

开发人员可以直接控制应用程序缓存。利用缓存清单文件可将相关资源组织到同一个逻辑应用中。这样一来，Web 应用就拥有了本来只属于桌面应用的特性。用户可以充分发挥想象力，尝试用一些更巧妙的方式利用这些特性。

Note

缓存清单文件中标识的资源构成了应用缓存（Application Cache），它是浏览器持久性存储资源的地方，通常在硬盘上。有些浏览器向用户提供了查看应用程序缓存中数据的方法。例如，在最新版本的 Firefox 中，about:cache 页面会显示应用程序缓存的详细信息，提供了查看缓存中的每个文件的方法，如图 8.1 所示。

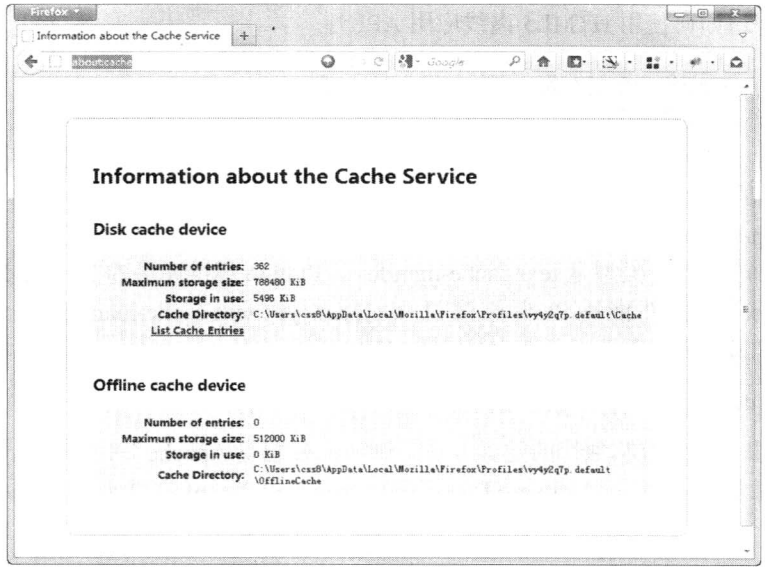


图 8.1 Firefox 的 about:cache 页面

8.1.2 浏览器支持概述

各浏览器对 HTML5 离线应用的支持情况如表 8.1 所示，从中可以看到目前大部分浏览器已经支持 HTML5 离线应用。

表 8.1 浏览器支持概述

浏 览 器	说 明
IE	不支持
Firefox	3.5 及以上的版本支持
Opera	10.6 及以上的版本支持
Chrome	4.0 及以上的版本支持
Safari	4.0 及以上的版本支持
iPhone	2.0 及以上的版本支持
Android	2.0 及以上的版本支持

HTML5 离线应用的支持程度不同，在使用之前建议先测试浏览器的支持情况。

8.2 HTML5 离线应用详解

Note

当准备开发基于 HTML5 离线存储的应用时，首先要确定服务器是否支持对 manifest 文件的解析，目前服务器都默认不支持，相信 HTML5 最终发布后，服务器应该都支持 manifest 文件。如果不支持，则需要用户手动设置，在服务器 mime.types 文件中添加。

为什么要求服务器一定要支持 manifest 文件？因为做离线存储必须要有一个 manifest 文件来存储需要缓存在用户机器上的资源路径，且这个 manifest 文件的路径将在 HTML 的文件中使用。下面我们将更加细致地探讨如何使用 HTML5 离线应用 API。

8.2.1 解析 manifest 文件

离线应用包含一个 manifest 清单文件，此文件中列出了浏览器为离线应用缓存的所有资源。实际上，manifest 文件是一个文本文件，它罗列了离线访问应用时所需缓存的文件清单。

注意，引用 manifest 文件的页面，不管有没有罗列清单，都会被缓存。

manifest 文件的 MIME 类型是 text/cache-manifest，Python 标准库中的 SimpleHTTPServer 模块对扩展名为.manifest 的文件能配以头部信息 Content-type:text/cache-manifest，配置方法是打开 PYTHON_HOME/Lib/mimetypes.py 文件并添加一行代码：

```
'. manifest': 'text/cache-manifest manifest';
```

不同的 Web 服务器都有其独特的配置方法。例如，要配置 Apache HTTP 服务器，开发人员需要将下面一行代码添加到 Apache Software Foundation\Apache2.2\conf 文件夹的 mime.types 文件中，如图 8.2 所示。

```
text/cache-manifest manifest
```

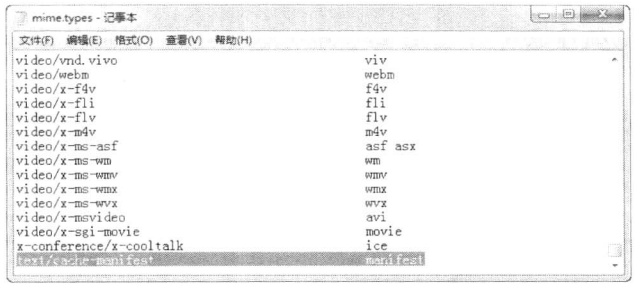


图 8.2 配置 Apache HTTP 服务器

Manifest 文件内容的基本格式要求如下：

☑ 第一行必须以 CACHE MANIFEST 开头，紧接着是文件的路径或注释。

☑ 注释必须以“#”开头。

☑ 必须声明一个白名单，这个白名单指定的文件将在用户连接因特网后访问，它必须在 NETWORK:的下一行。NETWORK 部分罗列的资源，无论缓存中存在与否，均从网络获取。

先写 CACHE MANIFEST，然后换行，每行单列资源文件，每行的换行符可以是 CR、LF 或者

CRLF，格式很灵活，但文本编码格式必须是 UTF-8。UTF-8 是多数文本编辑器经常输出的编码格式。

【示例 1】创建一个以 manifest 为扩展名的文件，命名为 cacheData.manifest，在这个文件中将指定一些文件的路径，如 HTML、CSS、JavaScript、Images。下面是一个完整的 manifest 文件的内容：

Note

```
CACHE MANIFEST
#version 1.0
login.html
static/css/i.css
static/img/png/alipay-i-logo-big.png
static/img/png/alipay-i-icons.png
static/js/mui-min.js
NETWORK:
static/img/png/button-ok.png
CACHE:
static/img/png/login-slider-bg.png
FALLBACK:
static/img/png/alipay-bank-icbc.png static/img/png/alipay-bank-cmb.png
```

第一行中的 CACHE MANIFEST 是必需的，每个站点都有 5MB 的空间来存储这些数据，如果 manifest 文件或文件里所列的文件无法加载，整个缓存更新过程将无法进行，浏览器会使用最后一次成功的缓存。

如果没有指定标题，默认就是 CACHE MANIFEST 部分。下面的 manifest 文件示例中指定了两个要缓存的文件：

```
CACHE MANIFEST
application.js
style.css
```

添加到 CACHE MANIFEST 区块中的文件，无论应用程序是否在线，浏览器都会从应用程序缓存中获取该文件。没有必要在这里列出应用程序的主 HTML 资源，因为最初指向 manifest 文件的 HTML 文档会被隐含包含进来。但是，如果希望缓存多个 HTML 文件，或者希望将多个 HTML 文件作为支持缓存的应用程序的可选入口，则需将这些文件都列在 CACHE MANIFEST 中。

如果需要，用户还可以添加以下内容：

☑　在进入因特网后，增加一个缓存内容，这些文件的路径必须在 CACHE:的下一行。

☑　增加备份，这些文件的路径必须在 FALLBACK: 的下一行，格式如下。

```
FALLBACK:
static/img/png/alipay-bank-icbc.png static/img/png/alipay-bank-cmb.png
```

FALLBACK 部分提供了获取不到缓存资源时的备选资源路径。第一个文件的路径和第二个文件的路径中间有一个空格，这个 FALLBACK:的作用是：当第一个文件缓存不成功时，或无法找到时，它会缓存第二个文件。

【示例 2】当无法获取 app/ajax 时，所有对 app/ajax/及其子路径的请求都会被转发给 default.html 文件来处理。

```
# 缓存文件列表
about.html
html5.css
index.html
```

```
happy-trails-rc.gif
lake-tahoe.JPG
#不缓存注册页面
NETWORK·
signup.html
FALLBACK
signup.html offline.html
/app/ajax/ default.html
```

Note

【示例 3】 #表示注释行标识符，但它还有一个小作用。Web 应用的缓存只有在 manifest 文件被修改的情况下才会被更新，所以如果只是修改了被缓存的文件，那么用户本地的缓存还是不会被更新的，但是可以通过修改 manifest 文件来告诉浏览器需要更新缓存。利用这点，可以更新注释。

```
CACHE MANIFEST
# wanz app v1

# 指明缓存入口
CACHE:
index.html
style.css
images/logo.png
scripts/main.js

# 以下资源必须在线访问
NETWORK:
login.php

# 如果 index.php 无法访问则用 404.html 代替
FALLBACK:
/index.php /404.html
```

上面 manifest 文件包括 3 个节点，简单说明如下。

☑ CACHE:

这个是 manifest 文件的默认入口，在此入口之后罗列的文件，或直接写在 CACHE MANIFEST 后的文件，在它们下载到本地后会被缓存起来。

☑ NETWORK:

可选的，在此节后面所罗列的文件是需要访问网络的，即使用户离线访问，也会直接跳过缓存而访问服务器。

☑ FALLBACK:

可选的，用来指定资源无法访问时的回调页面。每一行包括两个 URL，第一个是资源文件 URL，第二个是回调页面 URL。

提示：以上描述的这些节点是没有先后顺序的，而且在同一个 manifest 中可以多次出现。

修改注释行的文件版本：

```
# wanz app v2
```

这样做有 3 个好处：

☑ 可以很明确地了解离线 Web 应用的版本。

☑ 通过简单地修改版本号就可以轻易地通知浏览器进行更新。
☑ 可以配合 JavaScript 程序来完成缓存更新。

8.2.2 搭建离线应用程序

Note

创建好了 cacheData.manifest 文件，下面就需要在 HTML 文件中指定文档的 manifest 属性为 cache.mnifest 文件的路径。

```
<html manifest="cacheData.manifest">
…
</html>
```

当然，这个 manifest 的文件路径用绝对路径和相对路径都可以，甚至可以引用其他服务器上的 manifest 文件。该文件所对应的 mime-type 应该是 text/cache-manifest，所以需要配置服务器来发送对应的 MIME 类型信息。

由于有某些浏览器中仅仅添加这一属性，可能并不能很好地工作，所以一定要用 HTML5 文档声明方式创建 HTML 页面。

```
<!DOCTYPE html>
<html manifest="cacheData.manifest">
…
</html>
```

8.2.3 检查浏览器是否支持

使用离线应用 API 前，建议使用脚本先检查浏览器是否支持它。检测方法如下。

```
if(window.applicationCache) {
    //浏览器支持的离线应用
}
```

8.2.4 离线缓存更新实现

到目前为止，你应该了解如何来实现离线文件的存储了，简单地说只需要 3 步：

第 1 步，配置服务器 manifest 文件的 MIME 类型。

第 2 步，编写 manifest 文件。

第 3 步，在页面的 html 元素的 manifest 属性中引用 manifest 文件。

完成上述 3 步之后，即使拔掉网线，也可以访问页面。但是，当继续丰富页面，修改 JavaScript 代码和 CSS 来实现更炫的页面效果，然后将更新内容上传到服务器时，结果会发现自己无法看到最新的页面效果。那是因为本地浏览器还没有更新 HTML5 的离线存储文件。那么如何更新 HTML5 离线缓存？

更新 HTML5 离线缓存有 3 种方法：

☑ 清除离线存储的数据。这个不一定就是清理浏览器历史记录就可以做到的，因为不同浏览器管理离线存储的方式不同。例如，在 Firefox 中需要选择【选项】|【高级】|【网络】|【脱机存储】命令，然后在其中清除离线存储数据。

☑ 修改 manifest 文件。修改了 manifest 文件里所罗列的文件也不会更新缓存，而是要更新 manifest 文件。

☑ 使用 JavaScript 编写更新程序。

8.2.5 JavaScript 接口实现

Note

指定的缓存文件如果有更新怎么办？因为用户所访问的仍然是缓存在本地的文件，无法看到服务器更新后的内容或数据，因此还需要在文件更新的同时，更新用户机器上的缓存文件，这就需要用到 JavaScript 为 HTML5 这一特性提供的接口。

下面重点讲解 JavaScript 脚本控制缓存的动态方法。先简单介绍下 ApplicationCache（http://www.whatwg.org/specs/web-apps/current-work/#applicationcache）和它的几个重点属性及方法。

ApplicationCache API 是一个操作应用缓存的接口，新的 window.applicationCache 对象可触发一系列与缓存状态相关的事件。该对象有一个数值型属性 window.applicationCache.status，它代表了缓存的状态。缓存状态共有 6 种，说明如表 8.2 所示。

表 8.2 缓存状态说明

Status 值	说　明
0	UNCACHED（未缓存）
1	IDLE（空闲）
2	CHECKING（检查中）
3	DOWNLOADING（下载中）
4	UPDATEREADY（更新就绪）
5	OBSOLETE（过期）

目前，互联网上大部分的页面都没有指定缓存清单，所以这些页面的状态就是 UNCACHED（未缓存）。IDLE（空闲）是带有缓存清单的应用程序的典型状态。处于空闲状态说明应用程序的所有资源都已被浏览器缓存，当前不需要更新。如果缓存曾经有效，但现在 manifest 文件丢失，则缓存进入 OBSOLETE（过期）状态。对于上述各种状态，API 包含了与之对应的事件和回调特性。

例如，当缓存更新完成进入空闲状态时，会触发 cached 事件。此时，可能会通知用户，应用程序已处于离线模式可用的状态，可以断开网络连接了。如表 8.3 所示是一些与缓存状态有关的常见事件。

表 8.3 缓存事件说明

事　件	说　明
oncached	IDLE（空闲）
onchecking	CHECKING（检查中）
ondownloading	DOWNLOADING（下载中）
onupdateready	UPDATEREADY（更新就绪）
onobsolete	OBSOLETE（过期）

此外，没有可用更新或者发生错误时，还有一些表示更新状态的事件：onerror、onnoupdate 和 onprogress。

window.applicationCache 有一个 update()方法，调用 update()方法会请求浏览器更新缓存。包括检查新版本的 manifest 文件并下载必要的新资源。如果没有缓存或者缓存已过期，则会抛出错误。

其中常用代码说明如下：

Note

```
//返回应用于当前 window 对象文档的 ApplicationCache 对象
cache = window.applicationCache
//返回应用于当前 shared worker 的 ApplicationCache 对象 [shared worker]
cache = self.applicationCache
//返回当前应用的缓存状态，status 有 6 种无符号短整型值的状态，说明如表 8.2 所示
cache.status
//调用当前应用资源下载过程
cache.update()
//更新到最新的缓存，该方法不会使之前加载的资源突然被重新加载
cache.swapCache()
```

调用 swapCache()方法，图片不会重新加载，样式和脚本也不会重新渲染或解析，唯一的变化是在此之后发出请求页面的资源是最新的，applicationCache 对象和缓存宿主的关系是一一对应的，window 对象的 applicationCache 属性会返回关联 window 对象的活动文档的 applicationCache 对象。在获取 status 属性时，它返回当前 applicationCache 的状态，它的值有以下几种状态。

- ☑ UNCACHED(0)：ApplicationCache 对象的缓存宿主与应用缓存无关联。
- ☑ IDLE(1)：应用缓存已经是最新的，并且没有标记为 obsolete。
- ☑ CHECKING(2)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 checking。
- ☑ DOWNLOADING(3)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 downloading。
- ☑ UPDATEREADY(4)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 idle，并且没有标记为 obsolete，但是缓存不是最新的。
- ☑ OBSOLETE(5)：ApplicationCache 对象的缓存宿主已经和一个应用缓存关联，并且该缓存的更新状态是 obsolete。

如果 update 方法被调用了，浏览器就必须在后台调用应用缓存下载过程；如果 swapCache 方法被调用了，浏览器会执行以下步骤：

第 1 步，检查 ApplicationCache 的缓存宿主是否与应用缓存关联。

第 2 步，让 cache 成为 ApplicationCache 对象的缓存宿主关联的应用缓存。

第 3 步，如果 cache 的应用缓存组被标记为 obsolete，那么就取消 cache 与 ApplicationCache 对象的缓存宿主的关联并取消这些步骤，此时所有资源都会从网络中下载而不是从缓存中读取。

第 4 步，检查在同一个缓存组中是否存在完成标志为“完成”的应用缓存，并且版本比 cache 更新。

第 5 步，让完成标志为“完成”的新 cache 成为最新的应用缓存。

第 6 步，取消 cache 与 ApplicationCache 对象的缓存宿主的关联并用新 cache 代替关联。

通过下面的代码可以来检查当前页面缓存的状态：

```
var appCache = window.applicationCache;
switch (appCache.status) {
    case appCache.UNCACHED:
        //UNCACHED == 0
        alert('UNCACHED');
        break;
    case appCache.IDLE:
        //IDLE == 1
```

Note

```
        alert('IDLE');
        break;
    case appCache.CHECKING:
        //CHECKING == 2
        alert('CHECKING');
        break;
    case appCache.DOWNLOADING:
        //DOWNLOADING == 3
        alert('DOWNLOADING');
        break;
    case appCache.UPDATEREADY:
        //UPDATEREADY == 4
        alert('UPDATEREADY');
        break;
    case appCache.OBSOLETE:
        //OBSOLETE == 5
        alert('OBSOLETE');
        break;
    default:
        alert('UKNOWN CACHE STATUS');
        break;
};
```

一个更新的实现过程大概是这样的：首先调用 applicationCache.update()使得浏览器开始尝试更新，前提是 manifest 文件是更新过的（如修改 manifest 版本号）；在 applicationCache.status 为 UPDATEREADY 状态时就可以调用 applicationCache.swapCache()来将旧的缓存更新为新的。

```
var appCache = window.applicationCache;
appCache.update();                          //开始更新
if (appCache.status == window.applicationCache.UPDATEREADY) {
    appCache.swapCache();                   //得到最新版本缓存列表，并且成功下载资源，更新缓存到最新
}
```

更新过程也可以很简单，但是一个好的应用少不了容错处理，如 Ajax 技术一样，需要对更新过程进行监控，处理各种异常或提示等待状态来使 Web 应用更强壮、用户体验更好，因此需要了解 applicationCache 的更新过程所触发的事件，主要包括 onchecking、onerror、onnoupdate、ondownloading、onprogress、onupdateready、oncached 和 onobsolete。

要对更新错误进行处理，可以这样写：

```
var appCache = window.applicationCache;
//请求 manifest 文件时返回 404 或 410，下载失败
//或 manifest 文件在下载过程中源文件被修改会触发 error 事件
appCache.addEventListener('error', handleCacheError, false);
function handleCacheError(e) {
    alert('Error: Cache failed to update!');
};
```

不管是 manifest 文件，还是它所罗列的资源文件下载失败，整个更新过程就终止了，浏览器会使用上一个最新的缓存。更多关于事件的内容请访问 http://www.whatwg.org/specs/web-apps/current-work/#event-handlers。

8.2.6　离线存储事件监听

Note

HTML5 引入了一些新的事件，用来让应用程序检测网络是否正常连接。应用程序处于在线状态和离线状态会有不同的行为模式。是否处于在线状态可以通过检测 window.navigator 对象的属性来判断。

首先，navigator.onLine 是一个标明浏览器是否处于在线状态的布尔属性。当然，onLine 值为 true 并不能保证 Web 应用程序在用户的机器上一定能访问到相应的服务器。而当其值为 false 时，不管浏览器是否真正联网，应用程序都不会尝试进行网络连接。查看页面状态是在线还是离线的代码如下：

```
//当页面加载时，设置状态为 online 或者 offline
function loadDemo() {
    if(navigator.onLine) {
        log("Online");
    } else {
        log("Offline");
    }
}
//增加事件监听，当在线状态发生变化时，将触发响应
window.addEventListener("online", function(e) {
    log("Online");
}, true);
window.addEventListener("offline", function(e) {
    log("Offline");
}, true);
```

在支持 HTML5 离线存储的浏览器中，window 对象有一个 applicationcache 属性，通过 window .applicationcache 可以获得一个 DOMApplicationCache 对象，这个对象来自 DOMApplicationCache 类，这个类有一系列的属性和方法，具体请参阅 Safari HTML5 Applicationcache DOMApplicationCache 开发文档。

首先获取 DOMApplicationCache 对象。

```
var cache = window.applicationcache;
```

接着触发 cache 对象的一些事件来检测缓存是否成功。

```
/*oncached 事件表示：当更新已经处理完成，并且存储
 * 如果一切正常，这里 cache 的状态应该是 4
*/
cache.addEventListener('cached', function() {
    console.log('Cached,Status:' + cache.status);
}, false);
/*onchecking 事件表示：当更新已经开始进行，但资源还没有开始下载，即刚刚获取到最新的资源
 * 如果一切正常，这里 cache 的状态应该是 2
*/
cache.addEventListener('checking', function() {
    console.log('Checking,Status:' + cache.status);
}, false);
/*ondownloading 事件表示：开始下载最新的资源。
 * 如果一切正常，这里 cache 的状态应该是 3
*/
```

```
cache.addEventListener('downloading', function() {
    console.log('Downloading,Status:' + cache.status);
}, false);
/*onerror 事件表示：有错误发生，manifest 文件找不到或服务端有错误或资源找不到都会触发 onerror 事件。
 * 如果一切正常，这里 cache 的状态应该是 0
*/
cache.addEventListener('error', function() {
    console.log('Error,Status:' + cache.status);
}, false);
/*onnoupdate 事件表示：更新已经处理完成，但是 manifest 文件还未改变，处理闲置状态。
 * 如果一切正常，这里 cache 的状态应该是 1
*/
cache.addEventListener('noupdate', function() {
    console.log('Noupdate,Status:' + cache.status);
}, false);
/*onupdateready 事件表示：更新已经处理完成，新的缓存可以使用。
 * 如果一切正常，这里 cache 的状态应该是 4
*/
cache.addEventListener('updateready', function() {
    console.log('Updateready,Status:' + cache.status);
    cache.swapCache();
}, false);
```

通过以上代码可以发现，只有当 DOMApplicationCache 对象触发了 updateready 事件时，才真正更新了缓存文件。

如果在开发过程当中就开始对离线存储功能做单元测试，那么每一次修改文件都必须更新 manifest 文件中的内容，即使更新了一个注释，整个 manifest 文件也会更新，DOMApplicationCache 对象也会触发上述的一系列事件，直到新的缓存文件可用为止。通常情况下都是通过更新 manifest 文件中的版本号来触发 onupdateready 事件。

8.3 实战 1：缓存首页

离线应用并不复杂，为了方便读者快速、简便地理解离线缓存的应用，下面将通过一个简单的首页缓存示例演示 HTML 离线缓存的应用。整个过程只需要简单的 5 步即可完成，当然，要设计更加复杂的离线应用，还需要读者结合 HTML5 中其他新技术，并进行更加复杂的设置才行。

操作步骤：

第 1 步，添加 HTML5 DOCTYPE。

创建符合规范的 HTML5 文档。HTML5 DOCTYPE 相比于 XHTML 版本的 DOCTYPE 而言，要简单明了很多。

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>缓存首页</title>
<link href="style.css" type="text/css" rel="stylesheet" media="screen">
<meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0;">
```

```
</head>
<body>
<div id="container">
    <header class="ma-class-en-css">
        <h1 id ="logo"><a href="#">HTML5</a></h1>
    </header>
    <div id="content">
        <h2>HTML5</h2>
        <p>HTML 标准自 1999 年 12 月发布 HTML 4.01 后，后继的 HTML5 和其他标准被束之高阁，为了推动 web 标准化运动的发展，一些公司联合起来，成立了一个叫做 Web Hypertext Application Technology Working Group（WHATWG，Web 超文本应用技术工作组）的组织，HTML5 草案的前身名为 Web Applications 1.0，于 2004 年被 WHATWG 提出，2007 年被 W3C 接纳，并成立了新的 HTML 工作团队。</p>
        <p>HTML5 的第一份正式草案已于 2008 年 1 月 22 日公布。HTML5 有两大特点：首先，强化了 Web 网页的表现性能。其次，追加了本地数据库等 Web 应用的功能。</p>
    </div>
    <footer>html5 by <a href="#">WHATWG</a></footer>
</div>
</body>
</html>
```

然后另存为 index.html，放在站点根目录下。

第 2 步，添加.htaccess 支持。

在创建用于缓存页面的 manifest 清单文件之前，先要在.htaccess 文件中添加以下代码，具体说明请参照 8.2.1 节内容。

```
AddType text/cache-manifest .manifest
```

该指令可以确保每个 manifest 文件为 text/cache-manifest MIME 类型。如果 MIME 类型不对，那么整个清单将没有任何效果，页面将无法离线应用。

注意： 本章示例都是在 Apache HTTP Server 2.2 服务器环境下运行，读者在测试之前，应该在本地计算机中构建虚拟的 Apache 服务器环境。

提示： htaccess 文件被称为分布式配置文件，是 Apache 服务器中的一个配置文件，它提供了针对目录改变配置的方法，负责相关目录下的网页配置。通过 htaccess 文件，可以帮用户实现网页重定向、自定义错误页面、改变文件扩展名、允许或阻止特定的用户或者目录的访问、禁止目录列表、配置默认文档等功能。

启用.htaccess，需要修改 httpd.conf，启用 AllowOverride，并可以用 AllowOverride 限制特定命令的使用。如果需要使用.htaccess 以外的其他文件名，可以用 AccessFileName 指令来改变。例如，需要使用.config，则可以在服务器配置文件中按以下方法配置：AccessFileName .config。

第 3 步，创建 manifest 文件。

配置服务器之后，就可以创建 manifest 清单文件。新建一个文本文档，另存名为 offline.manifest，然后输入以下代码。

```
CACHE MANIFEST
#This is a comment

CACHE:
index.html
```

```
style.css
image.jpg
image-med.jpg
image-small.jpg
notre-dame.jpg
```

Note

在 CACHE 声明之后，罗列出所有需要缓存的文件。这对于缓存简单页面来说已经足够。但是 HTML5 缓存还有更多可能。例如，考虑以下 manifest 文件：

```
CACHE MANIFEST
#This is a comment

CACHE:
index.html
style.css

NETWORK:
search.php
login.php

FALLBACK:
/api offline.html
```

其中 CACHE 声明用于缓存 index.html 和 style.css 文件。同时，NETWORK 声明用于指定无需缓存的文件，如登录页面。最后一个是 FALLBACK 声明，这个声明允许在资源不可用的情况下，将用户重定向到特定文件，如 offline.html。

第 4 步，关联 manifest 文件到 HTML 文档。

设计完 manifest 文件和 HTML 文档，还需要将 manifest 文件关联到 HTML 文档中。使用 html 元素的 manifest 属性：

```
<html manifest="/offline.manifest">
```

第 5 步，测试文档。

完成后，使用 Firefox 3.5+本地访问 index.html 文件，会看到浏览器提示信息，如图 8.3 所示。其他高级浏览器不会提醒是否允许缓存，而是默认自动缓存。

图 8.3　测试首页离线缓存

此后即使服务器停止工作或者无法上网，用户依然可以访问本地缓存中的该首页。如果没有离线存储的支持，则当服务器停止工作或者无法上网时访问首页时，将会显示如图 8.4 所示的效果。

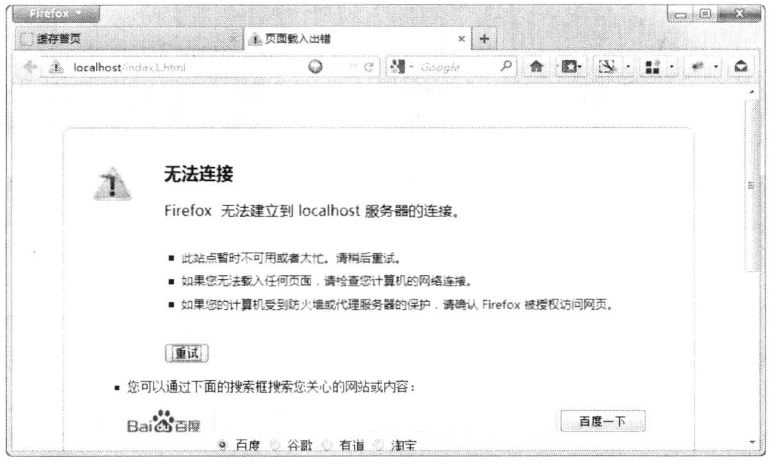


图 8.4　不支持离线缓存的效果

Note

8.4　实战 2：离线编辑内容

本案例将在上一实战基础上拓展使用 HTML5 开发离线应用。示例中会用到 HTML5 离线缓存、在线状态检测和 DOM Storage 等功能。

案例设计：开发一个便签管理的 Web 应用程序，用户可以在其中添加和删除便签。它支持离线功能，允许用户在离线状态下添加、删除便签，并且当在线以后能够同步到服务器上。

操作步骤：

第 1 步，设计应用程序 UI。

这个程序的界面很简单，如图 8.5 所示。当用户单击 New Note 按钮，则可以在弹出框中创建新的便签，双击某便签就可以删除该便签。

新建文档，输入下面代码，然后保存为 index.html。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>离线编辑内容</title>
<script type="text/javascript" src="server.js"></script>
<script type="text/javascript" src="data.js"></script>
<script type="text/javascript" src="UI.js"></script>
</head>
<body onload = "SyncWithServer()">
<input type="button" value="New Note" onclick="newNote()">
<ul id="list">
</ul>
</body>
</html>
```

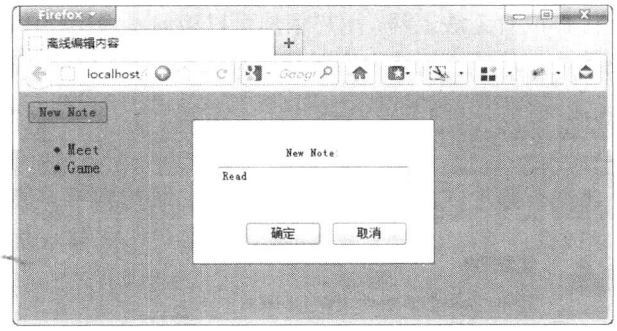


图 8.5　便签管理界面 UI

在 body 中声明了一个按钮和一个无序列表。当单击 New Note 按钮时，newNote 函数将被调用，它用来添加一条新的便签。而无序列表初始为空，它是用来显示便签的列表。

第 2 步，设计 cache manifest 文件。

定义 cache manifest 文件，声明需要缓存的资源。本例需要缓存 index.html、server.js、data.js 和 UI.js 等 4 个文件。除了前面列出的 index.html 外，server.js、data.js 和 UI.js 分别包含服务器相关、数据存储和用户界面代码。

cache manifest 文件的源代码如下：

```
CACHE MANIFEST
index.html
server.js
data.js
UI.js
```

保存为 notes.manifest，然后关联到 HTML 文档中：

```
<html manifest="notes.manifest">
```

第 3 步，设计用户界面代码。

用户界面代码定义在 UI.js 文件中，详细代码如下：

```
function newNote() {
    var title = window.prompt("New Note:");
    if(title) {
        add(title);
    }
}
function add(title) {
    //在界面中添加
    addUIItem(title);
    //在数据中添加
    addDataItem(title);
}
function remove(title) {
    //从界面中删除
    removeUIItem(title);
    //从数据中删除
    removeDataItem(title);
}
```

```
function addUIItem(title) {
    var item = document.createElement("li");
    item.setAttribute("ondblclick", "remove('" + title + "')");
    item.innerHTML = title;
    var list = document.getElementById("list");
    list.appendChild(item);
}
function removeUIItem(title) {
    var list = document.getElementById("list");
    for(var i = 0; i < list.children.length; i++) {
        if(list.children[i].innerHTML == title) {
            list.removeChild(list.children[i]);
        }
    }
}
```

Note

UI.js 中的代码包含添加便签和删除便签的界面操作。

☑ 添加便签：用户单击 New Note 按钮，newNote 函数被调用，newNote 函数会弹出对话框，用户可输入新便签内容。newNote 调用 add 函数。add 函数分别调用 addUIItem 和 addDataItem 添加页面元素和数据，addDataItem 代码将在后面列出。addUIItem 函数在页面列表中添加一项并指明 ondblclick 事件的处理函数是 remove，使得双击时可以删除便签。

☑ 删除便签：用户双击某便签时，调用 remove 函数。remove 函数分别调用 removeUIItem 和 removeDataItem 删除页面元素和数据，removeDataItem 将在后面列出。removeUIItem 函数删除页面列表中的相应项。

第 4 步，设计数据存储代码。

数据存储代码定义在 data.js 中，详细代码如下：

```
var storage = window['localStorage'];
function addDataItem(title) {
    if(navigator.onLine)//在线状态
    {
        addServerItem(title);
    } else//离线状态
    {
        var str = storage.getItem("toAdd");
        if(str == null) {
            str = title;
        } else {
            str = str + "," + title;
        }
        storage.setItem("toAdd", str);
    }
}
function removeDataItem(title) {
    if(navigator.onLine)//在线状态
    {
        removeServerItem(title);
    } else//离线状态
    {
```

Note

```
            var str = storage.getItem("toRemove");
            if(str == null) {
                str = title;
            } else {
                str = str + "," + title;
            }
            storage.setItem("toRemove", str);
        }
    }
    function SyncWithServer() {
        //如果当前是离线状态，不需要做任何处理
        if(navigator.onLine == false)
            return;
        var i = 0;
        //和服务器同步添加操作
        var str = storage.getItem("toAdd");
        if(str != null) {
            var addItems = str.split(",");
            for( i = 0; i < addItems.length; i++) {
                addDataItem(addItems[i]);
            }
            storage.removeItem("toAdd");
        }
        //和服务器同步删除操作
        str = storage.getItem("toRemove");
        if(str != null) {
            var removeItems = str.split(",");
            for( i = 0; i < removeItems.length; i++) {
                removeDataItem(removeItems[i]);
            }
            storage.removeItem("toRemove");
        }
        //删除界面中的所有便签
        var list = document.getElementById("list");
        while(list.lastChild != list.firstElementChild)
        list.removeChild(list.lastChild);
        if(list.firstElementChild)
            list.removeChild(list.firstElementChild);
        //从服务器获取全部便签，并显示在界面中
        var allItems = getServerItems();
        if(allItems != "") {
            var items = allItems.split(",");
            for( i = 0; i < items.length; i++) {
                addUIItem(items[i]);
            }
        }
    }
    window.addEventListener("online", SyncWithServer,false);
```

data.js中的代码包含添加便签、删除便签和与服务器同步等数据操作。其中用到了navigator.onLine

属性、online 事件、DOM Storage 等 HTML5 新功能。

☑　添加便签：addDataItem

通过 navigator.onLine 判断是否在线。如果在线，那么调用 addServerItem 直接把数据存储到服务器上。addServerItem 将在后面列出。如果离线，那么把数据添加到 localStorage 的“toAdd”项中。

Note

☑　删除便签：removeDataItem

通过 navigator.onLine 判断是否在线。如果在线，那么调用 removeServerItem 直接在服务器上删除数据。removeServerItem 将在后面列出。如果离线，那么把数据添加到 localStorage 的“toRemove”项中。

☑　数据同步：SyncWithServer

在 data.js 的最后一行，注册了 window 的 online 事件处理函数 SyncWithServer。当 online 事件发生时，SyncWithServer 将被调用。其功能如下。

- 如果 navigator.onLine 表示当前离线，则不做任何操作。
- 把 localStorage 中“toAdd”项的所有数据添加到服务器上，并删除“toAdd”项。
- 把 localStorage 中“toRemove”项的所有数据从服务器中删除，并删除“toRemove”项。
- 删除当前页面列表中的所有便签。
- 调用 getServerItems 从服务器获取所有便签，并添加在页面列表中。getServerItems 将在后面列出。

第 5 步，设计服务器相关代码。

服务器相关代码定义在 server.js 中，详细代码如下：

```
function addServerItem(title) {
    //在服务器中添加一项
}
function removeServerItem(title) {
    //在服务器中删除一项
}
function getServerItems() {
    //返回服务器中存储的便签列表
}
```

由于这部分代码与服务器有关，这里只说明各个函数的功能，具体实现可以根据不同服务器编写代码。在服务器中添加一项，需调用 addServerItem 函数；在服务器中删除一项，则调用 removeServerItem 函数；返回服务器中存储的便签列表，则调用 getServerItems 函数。

8.5　实战 3：离线跟踪

本示例将跟踪用户的活动位置，即使在间断性网络连接或无连接的情况下，用户只需要将一款具有 HTML5 Geolocation 功能和支持 HTML5 Web 浏览器的手机带在身上，即身在信号不好的区域无法联网，也能定位并记录自身位置。

案例设计：

在离线状态下，Geolocation API 可以在使用硬件地理定位的设备（如 GPS ）上继续工作，但在使用 IP 定位的设备上不可以，因为 IP 地理定位设备需要连接网络，以便将客户端的 IP 地址映射为坐标。离线应用程序还可以通过本地存储或者 Web SQL Database 这样的接口访向本地存储。界面显

示效果如图 8.6 所示。

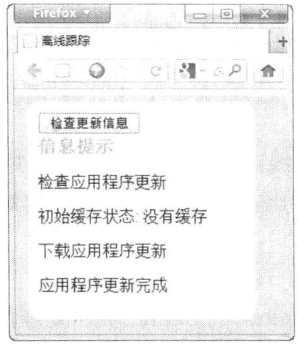


图 8.6　离线跟踪界面 UI

运行此应用程序需要 Web 服务器提供所需静态信息。注意，manifest 文件的内容类型必须配置为 text/cache-manifest 发送到浏览器。如果文件类型不正确，即使浏览器支持应用缓存也会返回缓存错误。

提示：要运行本示例的全部功能，服务器还需要具备接收地理位置数据的功能。服务器端的主要任务是存储、分析和提供这些数据。静态应用程序中，数据的获取不局限于同源。

操作步骤：

第 1 步，创建 manifest 文件。

新建 tracker.manifest 文件。在文件中列出应用程序需要缓存的资源。详细代码如下：

```
CACHE MANIFEST
# JavaScript
offline.js
#tracker.js
log.js

# stylesheets
html5.css

# images
```

第 2 步，创建构成界面的 HTML 和 CSS。

考虑到 index.html 和 html5.css 会被存储到缓存中，所以应用程序将从缓存中提取这两个文件。index.html 源代码如下：

```
<!DOCTYPE html>
<html lang="en" manifest="tracker.manifest">
<head>
<title>离线跟踪</title>
<script src="log.js"></script>
<script src="offline.js"></script>
<script src="tracker.js"></script>
<link rel="stylesheet" href="html5.css">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
```

```
<section>
    <article>
        <button id="installButton">检查更新信息</button>
        <h3>信息提示</h3>
        <div id="info"> </div>
    </article>
</section>
</body>
</html>
```

Note

从实现应用程序的离线功能分析，需要注意两个问题：

☑ HTML 元素的 manifest 属性。因为 html 元素在 HTML5 中是可选的，HTML 可以省略它。但是，如果希望应用程序支持缓存，就不能省略 html 元素，并需要在该元素中设置 manifest 属性，因为应用程序是否缓存离线文件取决于是否指定了 manifest 文件。

☑ 在页面中插入一个按钮，它的作用是让用户能够手动安装 Web 应用程序，以支持离线情况。

第 3 步，创建离线 JavaScript。

在本示例中，JavaScript 文件由多个<script>标签包含的.js 文件组成，这些.js 脚本会同 HTML 和 CSS 文件一起存储到缓存中。

☑ offline.js

```
//跟踪离线缓存事件状态
window.applicationCache.onchecking = function(e) {
    log("检查应用程序更新");
}
window.applicationCache.onnoupdate = function(e) {
    log("发现没有应用更新");
}
window.applicationCache.onupdateready = function(e) {
    log("应用程序更新完成");
}
window.applicationCache.onobsolete = function(e) {
    log("应用过时");
}
window.applicationCache.ondownloading = function(e) {
    log("下载应用程序更新");
}
window.applicationCache.oncached = function(e) {
    log("应用程序缓存");
}
window.applicationCache.onerror = function(e) {
    log("应用程序缓存错误");
}
window.addEventListener("online", function(e) {
    log("在线");
}, true);
window.addEventListener("offline", function(e) {
    log("离线");
}, true);
//把离线状态码转换信息提示
```

```
showCacheStatus = function(n) {
    statusMessages = ["没有缓存","空闲","正在检查...","正在下载...","更新完成","过时"];
    return statusMessages[n];
}
install = function() {
    log("正在检查更新");
    try {
        window.applicationCache.update();
    } catch (e) {
        applicationCache.onerror();
    }
}
onload = function(e) {
    if (!window.applicationCache) {
        log("你的浏览器不支持离线缓存。");
        return;
    }
    if (!navigator.geolocation) {
        log("你的浏览器不支持 HTML5 定位。");
        return;
    }
    if (!window.localStorage) {
        log("你的浏览器不支持本地存储。");
        return;
    }
    log("初始缓存状态: " + showCacheStatus(window.applicationCache.status));
    document.getElementById("installButton").onclick = install;
    if(navigator.onLine) {
        uploadLocations();
    }
}
```

Note

☑ log.js

```
log = function() {
    var p = document.createElement("p");
    var message = Array.prototype.join.call(arguments, " ");
    p.innerHTML = message;
    document.getElementById("info").appendChild(p);
}
```

第 4 步，检查 applicationCache 的支持情况。

除了离线应用缓存，示例中还使用了地理定位和本地存储。在页面加载前应确保浏览器支持这两种功能。

```
if (!window.applicationCache) {
    log("你的浏览器不支持离线缓存。");
    return;
}
if (!navigator.geolocation) {
    log("你的浏览器不支持 HTML5 定位。");
```

Note

```
        return;
    }
    if (!window.localStorage) {
        log("你的浏览器不支持本地存储。");
        return;
    }
    log("初始缓存状态: " + showCacheStatus(window.applicationCache.status));
    document.getElementById("installButton").onclick = install;
```

第 5 步，为更新按钮添加处理函数。

下面的代码用来处理更新行为，其作用是更新应用缓存。

```
    install = function() {
        log("正在检查更新");
        try {
            window.applicationCache.update();
        } catch (e) {
            applicationCache.onerror();
        }
    }
```

单击按钮后将检查缓存区，并更新需要更新的缓存资源。当所有可用更新都下载完毕之后，将在用户界面显示一条消息，告诉用户应用程序已安装成功，可以在离线模式下运行了。

第 6 步，添加 Geolocation 跟踪代码。

下面的代码是地理定位示例代码。它是 trackerja 文件的一部分。

```
    var handlePositionUpdate = function(e) {
        var latitude = e.coords.latitude;
        var longitude = e.coords.longitude;
        log("Position update:", latitude, longitude);
        if(navigator.onLine) {
            uploadLocations(latitude, longitude);
        }
        storeLocation(latitude, longitude);
    }
    var handlePositionError = function(e) {
        log("Position error");
    }
    var uploadLocations = function(latitude, longitude) {
        var request = new XMLHttpRequest();
        request.open("POST", "http://geodata.example.net:8000/geoupload", true);
        request.send(localStorage.locations);
    }
    var geolocationConfig = {
        "maximumAge" : 20000
    };
    navigator.geolocation.watchPosition(handlePositionUpdate, handlePositionError, geolocationConfig);
```

第 7 步，添加存储功能代码。

当应用程序处于离线状态时，需要将数据更新写入本地存储，接下来就添加这方面的代码。

```
var storeLocation = function(latitude, longitude) {
    var locations = JSON.parse(localStorage.locations || "[]");
    locations.push({
        "latitude" : latitude,
        "longitude" : longitude
    });
    localStorage.locations = JSON.stringify(locations);
}
```

Note

通过 HTML5 的 localStorage 保存坐标。因为 localStorage 可以将数据存储在本地浏览器中；所以它特别适用于具有离线功能的应用程序。本地存储中的数据在将来的会话中可用。当网络连接恢复正常后，应用程序就可以与远程服务器进行数据同步。

使用 Storage 还有一个好处，那就是当上传请求失败后可以通过 Storage 得到恢复。如果应用程序遇到某种原因导致的网络错误，或着应用程序被关闭，数据会被存储以便下次再进行传输。

第 8 步，添加离线事件处理程序。

位置更新处理程序运行时，会检查网络连接状态。如果应用程序在线，事件处理函数会存储并上传当前坐标，如果应用程序离线，事件处理函数只存储不上传。当应用程序重新连接到网络后，事件处理函数会在 UI 上显示在线状态，并在后台上传之前存储的所有数据。

```
window.addEventListener("online", function(e) {
    log("Online");
}, true);
window.addEventListener("offline", function(e) {
    log("Offline");
}, true);
```

网络连接状态在应用程序没有真正运行时可能会发生改变，如用户关闭了浏览器、刷新页面或跳转到其他网站。为了应对这些情况，离线应用程序在每次页面加载时都会检查与服务器的连接情况。如果连接正常，会尝试与远程服务器同步数据。

```
if(navigator.onLine) {
    uploadLocations();
}
```

为确保应用中所需的文件能够成功缓存，需要将这些文件指定在 manifest 文件中，随后在应用程序的主页面中进行引用，然后，添加监听器监听在线和离线状态的变化，进而基于因特网连接与否让网站执行不同的操作。

第9章

Workers 多线程处理

（视频讲解：1 小时 01 分钟）

在运行大型、复杂的 JavaScript 脚本的时候经常会出现浏览器假死的现象，那么能不能让这些代码在后台运行，或者让 JavaScript 函数在多个进程中同时运行呢？HTML5 Web Workers 正是为了解决这些问题而出现。

HTML5 Web Workers 可以让 Web 应用程序具备后台处理能力。它支持多线程处理功能，因此使用了 HTML5 的 JavaScript 应用程序可以充分利用多核 CPU 带来的优势。将耗时长的任务分配给HTML5 Web Workers执行，这样就避免了页面有时反应迟缓，甚至假死现象。Web Workers 可以创建一个不影响前台处理的后台线程，并且在这个后台线程中创建多个子线程。这样可以将耗时较长的处理交给后台线程去运行，从而解决了 HTML5 之前因为某个处理耗时过长而提示用户脚本运行时间过长，导致用户不得不结束这个处理的尴尬状况。

本章将介绍 Web Workers 基本知识，帮助读者使用 Web Workers 在 Web 网站或应用程序中创建一个后台线程。掌握在前台页面与后台线程进行数据交互时所使用到的方法与事件，能够在 JavaScript 脚本中实现前台页面与后台线程之间的数据交互，能够在主线程之间嵌套子线程的方法，利用 JavaScript 脚本在主线程之中创建一个或多个子线程，能够实现主线程与子线程、子线程与子线程之间的数据传递。了解在后台线程中可以使用的 JavaScript 脚本中的对象、方法和事件。

9.1 认识 Web Workers

Note

Web Workers 为网页脚本提供了一种能在后台进程中运行的方法。当创建 Worker 对象后，Web Workers 就可以通过 postMessage 方法向任务池发送任务请求，执行完之后再通过 postMessage 返回消息给创建者指定的事件处理程序，然后通过 onmessage 捕获返回消息，实现前后台数据的交互。

9.1.1 Web Workers 概述

在 Web 应用程序中，Web Workers 是一项后台处理技术。在此之前使用 JavaScript 创建的 Web 程序中，因为所有的处理都是在单线程内执行，所以如果脚本所需运行时间太长，程序界面会长时间处于停止响应状态，甚至当等待时间超出一定的限度，浏览器会提示脚本运行时间过长需要中断正在执行的处理。

为了解决这个问题，HTML5 新增了一个 Web Workers API。使用这个 API，用户可以很容易地创建在后台运行的线程，这个线程被称为 Worker，如果将可能耗费较长时间的处理交给后台执行，则对用户在前台页面中执行的操作没有影响。

尽管 Web Workers 功能强大，但也不是万能的，例如，在 Web Workers 中执行的脚本不能访问该页面的 window 对象，因此 Web Workers 不能直接访问 Web 页面和 DOM API，虽然 Web Workers 不会导致浏览器 UI 停止响应，但是仍然会消耗 CPU 周期，导致系统反应速度变慢。

如果开发人员创建的 Web 应用程序需要执行一些后台数据处理，但又不希望这些数据处理任务影响 Web 页面本身的交互性，那么可以通过 Web Workers 生成一个 Worker 去执行数据处理任务。同时添加一个事件监听器进行监听，并与之进行数据交互。

Web Workers 的另一个用途是可以监听由后台服务器广播的消息，收到后台服务器的消息后，将其显示在 Web 页面上。这种与后台服务器对话的场景，Web Workers 可能会使用到 Web Sockets 或 Server-Sent 事件。

Web Workers 接口可以创建真正的系统级别的进程，还可以使用 XMLHttpRequest 来处理 I/O，无论 responseXML 和 channel 属性是否为 null。使用它可以很容易设计并发操作效果，这将会很有趣。例如，在做网站下载的时候使用 Worker，或者使用 Worker 实现处理扩展功能。

注意：后台进程（包括 Web Workers 进程）不能对 DOM 进行操作。如果希望后台程序处理的结果能够改变 DOM，只能通过返回消息给创建者的回调函数进行处理。

Web Workers 能够为用户做些什么？

- ☑ 加载一个 JavaScript 文件，进行大量的复杂计算，而不挂起主进程，并通过 postMessage 和 onMessage 进行通信。
- ☑ 可以在 Worker 中通过 importScripts(url)方法加载 JavaScript 脚本文件。
- ☑ 可以使用 setTimeout()、clearTimeout()、setInterval()和 clearInterval()。
- ☑ 可以使用 XMLHttpRequest 进行异步请求。
- ☑ 可以访问 navigator 的部分属性。
- ☑ 可以使用 JavaScript 核心对象。

Web Workers 的局限性：

- ☑ 不能跨域加载 JavaScript。

- ☑ Worker 内代码不能访问 DOM。
- ☑ 各个浏览器对 Worker 的实现还没有完全完善。不是每个浏览器都支持所有新特性。
- ☑ 使用 Web Workers 加载数据没有 JSONP 和 Ajax 加载数据高效。

Note

9.1.2 浏览器支持概述

各浏览器对 HTML5 Web Workers 的支持情况如表 9.1 所示，从中可以看到目前浏览器对 Web Workers 的支持情况各不相同，但 Web Workers 已经得到了大部分主流浏览器的支持，并且仍在持续更新发展中。

表 9.1 浏览器支持情况

浏 览 器	说 明
IE	不支持
Firefox	3.5 及以上的版本支持
Opera	10.6 及以上的版本支持
Chrome	3.0 及以上的版本支持
Safari	4.0 及以上的版本支持

9.1.3 熟悉 Web Workers 成员

Web Workers 的使用方法非常简单，只需创建一个 Web Workers 对象，并传入希望执行的 JavaScript 文件即可。然后在页面中设置一个事件监听器，用来监听由 Web Workers 对象发来的消息和错误信息。如果想要在页面与 Web Workers 之间建立通信，数据需通过 postMessage()函数传递。对于 Web Workers JavaScript 文件中的代码也是如此，必须通过设置事件处理程序来处理发来的消息和错误信息，通过调用 postMessage()函数实现与页面的数据交互。

在使用 Web Workers 之前，读者应该熟悉线程中可用的变量、函数与类。在线程调用的 JavaScript 脚本文件中所有可用的变量、函数与类如下所示。

- ☑ self：self 关键值用来表示本线程范围内的作用域。
- ☑ postMessage(meseage)：向创建线程的源窗口发送消息。
- ☑ onmessage：获取接收消息的事件句柄。
- ☑ importScripts(urls)：导入其他 JavaScript 脚本文件。参数为该脚本文件的 URL 地址，可以导入多个脚本文件。导入的脚本文件必须与使用该线程文件的页面在同一个域中，并在同一个端口中。

```
importScripts("worker.js","worker1.js","worker2.js");
```

- ☑ navigator 对象：与 window.navigator 对象类似，具有 appName、platform、userAgent 和 appVersion 属性。它们可以用来标识浏览器的字符。
- ☑ sessionStorage/localStorage：在线程中可以使用 Web Storage。
- ☑ XMLHttpRequest：在线程中可以处理 Ajax 请求。
- ☑ Web Workers：在线程中可以嵌套线程。
- ☑ setTimeout()/setInterval()：在线程中可以实现定时处理。
- ☑ close：结束本线程。
- ☑ eval()、isNaN()、escape()等：可以使用所有 JavaScript 核心函数。

☑ object：可以创建和使用本地对象。

☑ WebSockets：可以使用 Web Sockets API 向服务器发送和接收信息。

Note

9.2 使用 Web Workers

使用 Web Workers 的步骤十分简单。只要在 Worker 类的构造器中，将需要在后台线程中执行的脚本文件的 URL 地址作为参数，然后创建 Worker 对象。

```
var worker = new Worker("worker.js");
```

注意：在后台线程中是不能访问页面或窗口对象的，此时如果在后台线程的脚本文件中使用 window 对象或 document 对象，则会引发错误。

另外，可以通过发送和接收消息与后台线程互相传递数据。通过对 Worker 对象的 onmessage 事件获取可以在后台线程之中接收消息。

```
worker.onmessage=function( event){
    //处理收到的消息
}
```

使用 Worker 对象的 postMessage 方法可以给后台线程发送消息。发送的消息是文本数据，但也可以是任何 JavaScript 对象，需要通过 JSON 对象的 stringify 方法将其转换成文本数据。

```
worker.postMessage(meseage);
```

通过获取 Worker 对象的 onmessage 事件句柄及 Worker 对象的 postMessage 方法可以实现线程内部消息的接收和发送。

9.2.1 检查浏览器支持性

在调用 Web Workers API 函数之前。应该确认当前浏览器是否支持。如果不支持，可以提供一些备用信息，提醒用户使用最新的浏览器。例如，下面代码可以用来测试浏览器的支持情况。

```
function testWorker() {
    if (typeof(Worker) !== "undefined") {
        document.getElementById("support").innerHTML = "浏览器不支持 HTML5 Web Workers";
    }
}
```

在上面代码中，使用 testWorker 函数来检测浏览器的支持情况，可在页面加载时调用该函数。调用 typeof(Worker)会返回全局 window 对象的 Worker 属性，如果浏览器不支持 Web Workers API，则返回结果将是 undefined。上面这段代码在检测了浏览器支持性之后，会将检测结果反馈到页面上。

9.2.2 创建 Web Workers

调用 Worker 构造函数就可以简单创建一个 Worker。Web Workers 在初始化时会接受一个 JavaScript 文件的 URL 地址，参数 URI 表示要执行的脚本文件地址。其中包含了供 Worker 执行的代码。这段代码会设置事件监听器，并与生成 Worker 的容器进行通信。JavaScript 文件的 URL 可以是相对或者

绝对路径，只要是相同的协议、主机和端口即可。

```
worker = new Worker("echoWorker.js");
```

如果想获取 Worker 进程的返回值，可以通过 onmessage 属性来绑定一个事件处理程序。

Note

```
var myWorker = new Worker('easyui.js');
myWorker.onmessage = function(event){
    alert('Called back by the worker!');
};
```

第一行用来创建和运行 Worker 进程，第 2 行设置 Worker 的 onmessage 属性用来绑定指定的事件处理程序，当 Worker 的 postMessage()方法被调用时，这个被绑定的程序就会被触发。

对于由多个 JavaScript 文件组成的应用程序来说，可以通过包含 script 元素的方式，在页面加载的时候同步加载 JavaScript 文件。由于 Web Workers 没有访问 document 对象的权限，所以 Worker 只能使用 importScripts 方法导入其他 JavaScript 文件。importScripts()是全局函数，该函数可以将脚本或库导入到它们的作用域中，导入的 JavaScript 文件只会在某一个已有的 Worker 中加载和执行。多个脚本的导入同样也可以使用 importScripts()函数，它们会按顺序执行。

importScripts()函数可以接受空的参数或多个脚本 URL 参数，下面这些形式都是合法的：

```
importScripts();
importScripts('foo.js');
importScripts('foo.js','bar.js');
```

JavaScript 会加载列出的每一个脚本文件，然后运行并初始化。这些脚本中的任何全局对象都可以被 worker 使用。

注意：importScripts 下载脚本顺序可能不一样，但执行的顺序一定是按 importScripts 中列出的顺序进行，而且是同步的，在所有脚本加载完并运行结束后 importScripts 才会返回。

Web Workers 能够嵌套使用，以创建子 Worker。

```
var subWorker = new Worker("subWorker.js");
```

用户可以创建多个 Workers。子 Worker 必须存在于同一个父页面下，且它的 URL 必须与父 Worker 的地址同源，这样可以很好地维持它们的依赖关系。

Web Workers 可以使用 setTimeout 和 setInterval。如果希望 Web Workers 进程周期性地运行，而不是不停地循环下去的话，这两个方法非常有用。

9.2.3 与 Web Workers 通信

使用后台线程时不能访问页面或窗口对象，但是并不代表后台线程不能与页面之间进行数据交互。为了实现页面与 Web Workers 通信，可以调用 postMessage 函数传入所需数据，同时将建立一个监听器，用来监听由 Web Workers 发送到页面的消息。

为建立页面和 Web Workers 之间的通信，首先在页面中添加对 postMessage 函数的调用，如下所示。

```
document.getElementById("helloButton").onclick = function() {
    worker.postMessage("Hi,How are you");
}
```

当用户单击按钮后，相应信息会被发送给 Web Workers，然后将事件监听器添加到页面中，用来监听从 Web Workers 发来的信息。

Note

```
worker.addEventListener("message", messageHandler, true);
function messageHandler(e) {
    //process message from worker
}
```

编写 HTML5 Web Workers JavaScript 文件。在该文件中，需要添加事件监听器以监听发来的消息，并且通过调用 postMessage 函数实现与页面之间的通信。

为了完成页面与 Web Workers 之间的通信功能。首先，添加代码调用 postMessage 函数。例如，在 messageHandler 函数中可以添加如下代码。

```
function messageHandler(e) {
    postMessage("worker says: " + e.data + " too");
}
```

接下来，在 Web Workers JavaScript 文件中添加事件监听器，以处理从页面发来的信息：

```
addEventListener("message", messageHandler, true);
```

接收到信息后会马上调用 messageHandler 函数以保证信息能及时返回。

通过 postMessage 函数将对象传递到 Worker 或者从中返回对象，这些对象将被自动转换为 JSON 格式。

```
var onmessage = function(e){
    postMessage(e.data);
};
```

注意：在 Worker 中进出的对象不能包含函数和循环引用，因为 JSON 不支持它们。

在 Web Workers 脚本中如果发生未处理的错误，会引发 Web Workers 对象的错误事件。特别是在调试用到 Web Workers 脚本时，对错误事件的监听就显得尤为重要。下面显示的是 Web Workers JavaScript 文件中的错误处理函数，它将错误记录在控制台上。

```
function errorHandler(e) {
    console.log(e.message, e);
}
```

为了处理错误，还必须在主页上添加一个事件监听器：

```
worker.addEventListener("error", errorHandler, true);
```

当 Worker 发生运行时错误时，它的 onerror 事件就会被触发。该事件接收一个 error 的事件，该事件不会冒泡，并且可以取消。要取消该事件可以使用 preventDefault()方法。该错误事件有以下 3 个属性。

- ☑ message：可读的错误信息。
- ☑ filename：发生错误的脚本文件名称。
- ☑ lineno：发生错误的脚本所在文件的行数。

Web Workers 不能自行终止，但能够被启用它们的页面所终止。调用 terminate 函数可以终止后台进程。被终止的 Web Workers 将不再响应任何信息或者执行任何其他计算。终止之后，Worker 不能被重新启动，但可以使用同样的 URL 创建一个新的 Worker。

```
cworker.terminate();
```

如果需要马上终止一个正在运行中的 Worker，则可以调用它的 terminate()方法：

```
myWorker.terminate();
```

这样一个 Worker 进程就被结束了。

Note

9.2.4　使用 Web Workers 上机练习

【示例 1】本示例演示了如何使用 Web Workers 在控制台显示一个提示信息。首先，设计主页面代码。

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <script type="text/javascript">
            //Web 页主线程
            var worker = new Worker("worker.js");          //创建一个 Worker 对象并向它传递将在新线程中
                                                            执行的脚本的 URL
            worker.postMessage("hello world");             //向 Worker 发送数据
            worker.onmessage = function(evt) {             //接收 Worker 传过来的数据函数
                console.log(evt.data);                     //输出 Worker 发送来的数据
            }
        </script>
    </head>
    <body></body>
</html>
```

下面是线程脚本文件 worker.js 代码。

```
onmessage = function(evt) {
    var d = evt.data;                                      //通过 evt.data 获得发送来的数据
    postMessage(d);                                        //将获取到的数据发送会主线程
}
```

在 Chrome 浏览器中访问主页文件，则可以在控制台中看到输出的信息，表示程序执行成功，如图 9.1 所示。

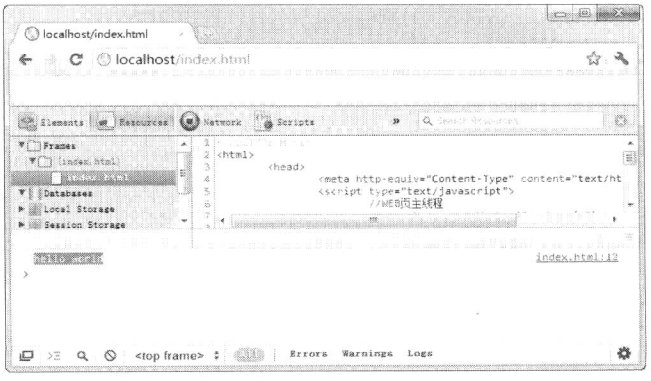


图 9.1　在控制台中查看信息

通过上面示例可以看到使用 Web Workers 应该包括下面两部分：

☑ 定义主页线程

- 通过 worker = new Worker(url)加载一个 JavaScript 文件，创建一个 Worker，同时返回一个 Worker 实例。
- 通过 worker.postMessage(data)方法向 Worker 发送数据。
- 绑定 worker.onmessage 事件接收 Worker 响应的数据。
- 使用 worker.terminate()可以终止一个 Worker 执行。

☑ 定义 Worker 线程

- 通过 postMessage(data)方法向主线程发送数据。
- 绑定 onmessage 事件接收主线程发送过来的数据。

Note

【示例 2】本示例演示如何创建 Web Workers，手动控制 Web Workers 与页面进行通信的一般方法，同时设置如何处理异常，以及如何停止 Worker 任务处理。

首先，设计主页文件（index.html），并在该文件脚本中定义一个主线程。

```
<!DOCTYPE html><head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<p id="support">你的浏览器不支持 HTML5 Web Workers</p>
<button id="stopButton" >停止任务</button>
<button id="helloButton" >发送消息</button>
<script>
function stopWorker() {                          //终止线程
    worker.terminate();
}
function messageHandler(e) {                     //显示线程响应信息
    console.log(e.data);
}
function errorHandler(e) {                       //线程错误处理
    console.warn(e.message, e);
}
function loadDemo() {
    if( typeof (Worker) !== "undefined") {
        document.getElementById("support").innerHTML = "你的浏览器支持 HTML5 Web Workers";
        worker = new Worker("worker.js");
        worker.addEventListener("message", messageHandler, true);
        worker.addEventListener("error", errorHandler, true);
        document.getElementById("helloButton").onclick = function() {
            worker.postMessage("ok");
        }
        document.getElementById("stopButton").onclick = stopWorker;
    }
}
window.addEventListener("load", loadDemo, true);
</script>
```

然后，设计线程脚本文件（worker.js）的代码。

```
function messageHandler(e) {
    postMessage("worker says: " + e.data + " too");
```

```
}
addEventListener("message", messageHandler, true);
```

在主页和线程脚本文件中，分别使用 addEventListener 方法把回调函数绑定到线程监听事件中。

最后，在 Chrome 浏览器中访问主页文件，单击“发送消息”按钮，则可以在控制台中看到输出的信息，表示程序手动控制线程交互执行成功，如图 9.2 所示。

图 9.2　在控制台中查看信息

使用 addEventListener 方法注册后台线程的响应事件比较麻烦，可以把它修改为下面这种传统写法。

☑　主线程脚本（index.html）

```
window.onload = function() {
    if( typeof (Worker) !== "undefined") {
        document.getElementById("support").innerHTML = "你的浏览器支持 HTML5 Web Workers";
        worker = new Worker("worker.js");
        worker.onmessage = function(e) {
            console.log(e.data);
        }
        worker.onerror = function(e) {
            console.warn(e.message, e);
        }
        document.getElementById("helloButton").onclick = function() {
            worker.postMessage("ok");
        }
        document.getElementById("stopButton").onclick = function() {
            worker.terminate();
        };
    }
}
```

☑　Worker 线程文件（worker.js）

```
onmessage = function(e) {
    postMessage("worker says: " + e.data );
}
```

9.3 案 例 实 战

Note

本节将通过多个示例演示如何灵活应用 Web Workers，实现并发式应用程序开发。

9.3.1 使用多线程实现后台运算

本示例设计一个文本框，允许用户在该文本框中输入数字，然后单击按钮，在后台计算从 1 到给定数值的和。虽然对于从 1 到给定数值的求和计算只需要用一个求和公式就可以了，但是本示例中为了展示后台线程的使用方法，采取循环计算的方法。

为了方便比较单线程与多线程的运算差异，首先采用传统方式设计一个单线程计算页面，页面代码如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta charset="utf-8">
<script type="text/javascript">
function calculate() {
    var num = parseInt(document.getElementById("num").value, 10);
    var result = 0;
    for (var i = 0; i <= num; i++) {
      result += i;
    }
    alert("合计值为" + result + "。");
}
</script>
</head>
<body>
输入数值:<input type="text" id="num">
<button onclick="calculate()">计算</button>
</body>
</html>
```

保存页面，然后在浏览器中预览，执行上面这段代码，在文本框中输入数值，然后单击“计算”按钮。可以看到，在弹出提示对话框之前，用户是不能在该页面上执行操作的。虽然在文本框中输入比较小的值时，不会出现延迟问题，但是当用户在该文本框中输入特别巨大的数字，如 1 亿以上的值时，浏览器会弹出提示对话框，提示脚本运行时间过长，如图 9.3 所示。

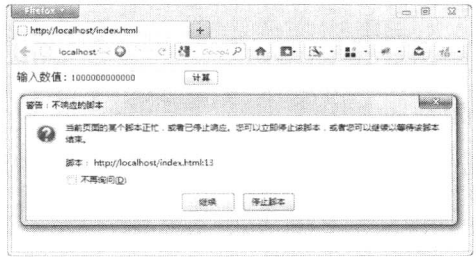


图 9.3 Firefox 浏览器提示脚本运行时间过长

重写该页面脚本，使用 Web Workers 把页面中比较耗时的运算放在后台运行，这样在上例的文本框中无论输入多么大的数值都可以正常运算了。操作步骤如下：

第 1 步，设计主页面，在该页面中创建一个 Worker，然后导入汇总计算的外部 JavaScript 文件。通过 postMessage 方法将用户输入的数字传递给 Worker，并通过 onmessage 事件回调函数接收运算的结果。

Note

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
var worker = new Worker("SumCalculate.js");        //创建执行运算的线程
worker.onmessage = function(event) {               //接收从线程中传出的计算结果
    alert("合计值为" + event.data + "。");
};
function calculate() {
    var num = parseInt(document.getElementById("num").value, 10);
    worker.postMessage(num);                       //将数值传给线程
}
</script>
</head>
<body>
输入数值:<input type="text" id="num">
<button onclick="calculate()">计算</button>
</body>
```

第 2 步，把对于给定值的求和运算放到线程中单独执行，且把线程代码单独存储在 SumCalculate.js 脚本文件中。

```
onmessage = function(event) {
    var num = event.data;
    var result = 0;
    for (var i = 0; i <= num; i++)
        result += i;
    postMessage(result);                           //向线程创建源送回消息
}
```

第 3 步，在支持 Web Workers 的浏览器中预览，如 Firefox、Safari、Chrome 和 Opera 等浏览器，在 Firefox 中的运行效果如图 9.4 所示。

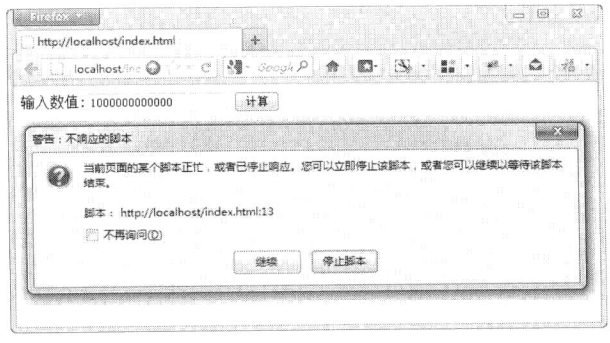


图 9.4　Firefox 浏览器中多线程运行效果

9.3.2 在后台过滤值

Note

在 Web 应用中，建议用户把非即时性的任务处理放在后台实现，以减轻前台处理的压力。本示例设计在页面上随机生成一个整数的数组，然后将该整数数组传入线程，让后台帮助挑选出该数组中可以被 3 整除的数字，然后显示在页面表格中。读者可以借助这种设计思路，把字符串、数组、列表中的数据都采取该方法显示在页面表格、表单控件甚至统计图中。

操作步骤：

第 1 步，设计前台页面代码，该页面的 HTML 代码部分包含一个空白表格，在前台脚本中随机生成整数数组，然后发送到后台线程挑选出能够被 3 整除的数字，再传回前台脚本，在前台脚本中根据挑选结果动态创建表格中的行、列，并将挑选出来的数字显示在表格中。

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<style type="text/css">
body { font: normal 11px auto "Trebuchet MS", Verdana, Arial, Helvetica, sans-serif; color: #4f6b72; background: #E6EAE9; }
table { width: 700px; padding: 0; margin: 0; }
td { border-right: 1px solid #C1DAD7; border-bottom: 1px solid #C1DAD7; background: #fff; font-size:11px; padding: 6px 6px 6px 12px; color: #4f6b72; text-align:center; }
</style>
<script type="text/javascript">
var intArray=new Array(200);                    //随机数组
var intStr="";
//生成 200 个随机数
for(var i=0;i<200;i++){
    intArray[i]=parseInt(Math.random()*200);
    if(i!=0)
        intStr+=";";                            //用分号作随机数组的分隔符
    intStr+=intArray[i];
}
//向后台线程提交随机数组
var worker = new Worker("script.js");
worker.postMessage(intStr);
//从线程中取得计算结果
worker.onmessage = function(event) {
    if(event.data!="") {
        var j,k,tr,td;
        var intArray=event.data.split(";");
        var table=document.getElementById("table");
        for(var i=0;i<intArray.length;i++){
            j=parseInt(i/10,0);
            k=i%10;
            if(k==0) {                          //如果该行不存在，则添加行
                tr=document.createElement("tr");
                tr.id="tr"+j;
                table.appendChild(tr);
            }
```

Note

```
            else {                                    //如果该行存在，则获取该行
                tr=document.getElementById("tr"+j);
            }
            td=document.createElement("td");
            tr.appendChild(td);
            td.innerHTML=intArray[j*10+k];
        }
    }
};
</script>
</head>
<body>
<table id="table">
</table>
</body>
```

第 2 步，将后台线程中需要处理的任务代码存放在脚本文件 script.js 中，详细代码如下。

```
onmessage = function(event) {
    var data = event.data;
    var returnStr;
    var intArray=data.split(";");
    returnStr="";
    for(var i=0;i<intArray.length;i++){
        if(parseInt(intArray[i])%3==0) {
            if(returnStr!="")
                returnStr+=";";
            returnStr+=intArray[i];
        }
    }
    postMessage(returnStr);                          //返回 3 的倍数拼接成的字符串
}
```

第 3 步，在浏览器中预览，则运行效果如图 9.5 所示。

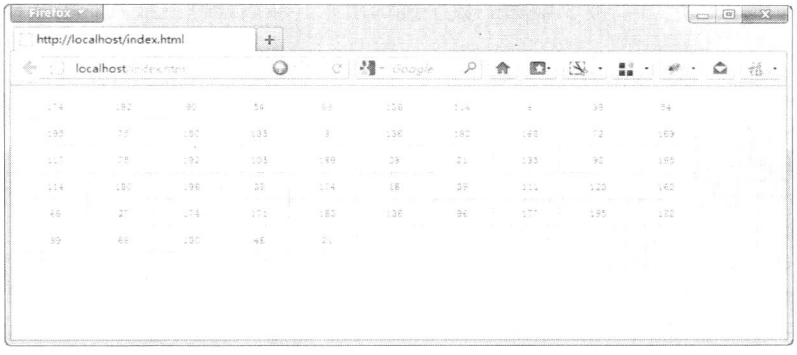


图 9.5 在后台过滤值

9.3.3 多任务并发处理

利用线程可以嵌套的特性，可以在 Web 应用中实现多个任务并发处理，这样能够提高 Web 应用

程序的执行效率和反应速度。同时通过线程嵌套把一个较大的后台任务切分成几个子线程，在每个子线程中各自完成相对独立的一部分工作。

本示例将在 9.3.2 节示例的基础上把主页脚本中随机生成数组的工作放到后台线程中进行，然后使用另一个子线程在随机数组中挑选可以被 3 整除的数字。对于数组的传递以及挑选结果的传递均采用 JSON 对象来进行转换，以验证是否能在线程之间进行 JavaScript 对象的传递工作。

Note

操作步骤：

第 1 步，在主页面中定义一个线程。设计不向该线程发送数据，在 onmessage 事件回调函数中进行后期数据处理，并把返回的数据显示在页面中。

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<style type="text/css">
body { font: normal 11px auto "Trebuchet MS", Verdana, Arial, Helvetica, sans-serif; color: #4f6b72; background: #E6EAE9; }
table { width: 700px; padding: 0; margin: 0; }
td { border-right: 1px solid #C1DAD7; border-bottom: 1px solid #C1DAD7; background: #fff; font-size:11px; padding: 6px 6px 6px 12px; color: #4f6b72; text-align:center; }
</style>
<script type="text/javascript">
var worker = new Worker("script.js");
worker.postMessage("");
worker.onmessage = function(event) {};
</script>
</head>
<body>
<table id="table">
</table>
</body>
```

第 2 步，在后台主线程文件 script.js 中，随机生成由 200 个整数构成的数组，然后把这个数组提交到子线程，在子线程中把可以被 3 整除的数字挑选出来，然后送回主线程，主线程再把挑选结果送回页面进行显示。

```
onmessage=function(event){
    var intArray=new Array(200);
    for(var i=0;i<200;i++)
        intArray[i]=parseInt(Math.random()*200);
    var worker;
    worker=new Worker("worker2.js");                    //创建子线程
    worker.postMessage(JSON.stringify(intArray));       //把随机数组提交给子线程进行挑选工作
    worker.onmessage = function(event) {
        postMessage(event.data);                        //把挑选结果返回主页面
    }
}
```

在上面代码中，向子线程中提交消息时使用的是 worker.postMessage()方法，而向主页面提交消息时使用 postMessage()方法。在线程中，向子线程提交消息时使用子线程对象的 postMessage()方法，

而向本线程的创建源发送消息时直接使用 postMessage()方法即可。

第 3 步，设计子线程的任务处理代码。下面是子线程代码，子线程在接收到的随机数组中挑选能被 3 整除的数字，然后拼接成字符串并返回。

Note

```
onmessage = function(event) {
    var intArray= JSON.parse(event.data);                    //还原整数数组
    var returnStr;
    returnStr="";
    for(var i=0;i<intArray.length;i++){
        if(parseInt(intArray[i])%3==0){
            if(returnStr!="")
                returnStr+=";";
            returnStr+=intArray[i];
        }
    }
    postMessage(returnStr);                                  //返回拼接字符串
    close();                                                 //关闭子线程
}
```

在子线程中向发送源发送回消息后，如果该子线程不再使用，则应该用 close 语句关闭子线程。

第 4 步，在主页面的主线程回调函数中处理后台线程返回的数据，并将这些数据显示在页面中。

```
//从线程中取得计算结果
worker.onmessage = function(event) {
    if(event.data!=""){
        var j,k,tr,td;
        var intArray=event.data.split(";");
        var table=document.getElementById("table");
        for(var i=0;i<intArray.length;i++){
            j=parseInt(i/10,0);
            k=i%10;
            if(k==0){
                tr=document.createElement("tr");
                tr.id="tr"+j;
                table.appendChild(tr);
            }
            else {
                tr=document.getElementById("tr"+j);
            }
            td=document.createElement("td");
            tr.appendChild(td);
            td.innerHTML=intArray[j*10+k];
        }
    }
};
```

第 5 步，此时在浏览器中预览，则会看到类似图 9.6 中所示的运行效果。

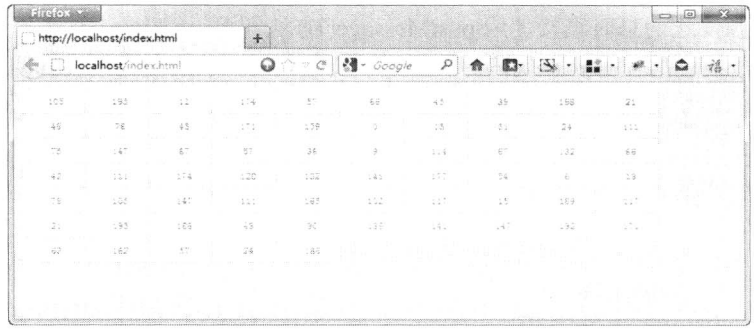


图 9.6　多任务并发处理

9.3.4　在多线程之间通信

当主线程嵌套多个子线程时，子线程之间可以通过下面几个步骤进行通信：

第 1 步，先创建发送数据的子线程。

第 2 步，执行子线程中的任务，然后把要传递的数据发送给主线程。

第 3 步，在主线程接收到子线程传回来的消息时，创建接收数据的子线程，然后把发送数据的子线程中返回的消息传递给接收数据的子线程。

第 4 步，执行接收数据子线程中的代码。

本示例继续在前面示例的基础上，将创建随机数组的工作也放到了一个单独的子线程中，在该线程中创建随机数组，然后将随机数组传递到另一个子线程中进行能够被 3 整除的数字挑选工作，最后把挑选结果传递回主页面进行显示。

操作步骤：

第 1 步，完成主页面的设计。包括 HTML 结构和 CSS 样式。在主页脚本中创建一个主线程，定义请求数据为空，在主线程响应事件 onmessage 回调函数中处理后台返回的处理数据，并把它们显示在页面中。

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<style type="text/css">
body { font: normal 11px auto "Trebuchet MS", Verdana, Arial, Helvetica, sans-serif; color: #4f6b72; background: #E6EAE9; }
table { width: 700px; padding: 0; margin: 0; }
td { border-right: 1px solid #C1DAD7; border-bottom: 1px solid #C1DAD7; background: #fff; font-size:11px; padding: 6px 6px 6px 12px; color: #4f6b72; text-align:center; }
</style>
<script type="text/javascript">
var worker = new Worker("script.js");
worker.postMessage("");
worker.onmessage = function(event) {
    if(event.data!=""){
        var j,k,tr,td;
        var intArray=event.data.split(";");
        var table=document.getElementById("table");
        for(var i=0;i<intArray.length;i++){
```

Note

```
                j=parseInt(i/10,0);
                k=i%10;
                if(k==0){
                    tr=document.createElement("tr");
                    tr.id="tr"+j;
                    table.appendChild(tr);
                }
                else {
                    tr=document.getElementById("tr"+j);
                }
                td=document.createElement("td");
                tr.appendChild(td);
                td.innerHTML=intArray[j*10+k];
            }
        }
};
</script>
</head>
<body>
<table id="table">
</table>
</body>
```

第 2 步，修改主线程中的代码。在主线程中定义一个子线程（发送数据），让其随机生成 200 个数字，并返回这个随机数组。在该子线程的回调函数中再定义一个子线程（接收数据），把接收到的随机数组传递给它，并接收该线程过滤后的数组。

```
onmessage=function(event){
    var worker;
    worker=new Worker("worker1.js");          //创建发送数据的子线程
    worker.postMessage("");
    worker.onmessage = function(event) {
        var data=event.data;                  //接收子线程中的数据：创建好的随机数组
        worker=new Worker("worker2.js");      //创建接收数据的子线程
        worker.postMessage(data);             //把从发送数据子线程中发回的消息传递给接收数据的子线程
        worker.onmessage = function(event) {
            var data=event.data;              //获取接收数据子线程中传回数据
            postMessage(data);                //把挑选结果发送回主页面
        }
    }
}
```

第 3 步，在发送数据的子线程中创建了一个 200 个整数构成的随机数组。然后把它转换为字符串并返回，最后关闭该子线程。

```
onmessage = function(event) {
    var intArray=new Array(200);
    for(var i=0;i<200;i++)
        intArray[i]=parseInt(Math.random()*200);
    postMessage(JSON.stringify(intArray));
    close();
}
```

第 4 步，在接收数据子线程中对接收到的随机数组中挑选能被 3 整除的数字，然后拼接成字符串并返回。

```
onmessage = function(event) {
    var intArray= JSON.parse(event.data);          //还原整数数组
    var returnStr;
    returnStr="";
    for(var i=0;i<intArray.length;i++){
        if(parseInt(intArray[i])%3==0){
            if(returnStr!="")
                returnStr+=";";
            returnStr+=intArray[i];
        }
    }
    postMessage(returnStr);                        //返回拼接字符串
    close();                                       //关闭子线程
}
```

9.3.5 使用线程技术计算 Fibonacci 数列

Fibonacci 数列是比较经典的数学规律，它以递归的方法定义：

F0=0

F1=1

Fn=F(n−1)+F(n−2)（n≥2，n∈N*）

使用 JavaScript 实现 Fibonacci 数列运算的一般方法如下：

```
var fibonacci =function(n) {
    return n <2? n : arguments.callee(n -1) + arguments.callee(n -2);
};
```

在 Chrome 浏览器中如果调用 fibonacci(39);，则执行时间需要大约为 19097 毫秒 ，而要计算 40 的 Fibonacci 数列时，浏览器就会罢工，直接提示脚本忙。

由于 JavaScript 是单线程执行的，在求数列的过程中浏览器不能执行其他脚本，UI 渲染线程也会被挂起，从而导致浏览器进入假死状态。

下面尝试使用 Web Workers 将数列计算过程放入一个新线程里，避免单线程计算所带来的问题。首先，定义主页文件。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>web worker fibonacci</title>
<script type="text/javascript">
  onload =function(){
      var worker =new Worker('fibonacci.js');
      worker.addEventListener('message', function(event) {
        var timer2 = (new Date()).valueOf();
          console.log( '结果：'+event.data, '时间:'+ timer2, '用时：'+ ( timer2   - timer ) );
      }, false);
      var timer = (new Date()).valueOf();
```

Note

```
        console.log('开始计算：40','时间:'+ timer );
        setTimeout(function(){
            console.log('定时器函数在计算数列时执行了', '时间:'+ (new Date()).valueOf() );
        },1000);
        worker.postMessage(40);
        console.log('我在计算数列的时候执行了', '时间:'+ (new Date()).valueOf() );
    }
    </script>
</head>
<body>
</body>
</html>
```

在主页脚本中定义创建一个线程，把 Fibonacci 数列计算任务交给新线程来完成。

然后，在新线程文件中（fibonacci.js）输入下面的代码。

```
var fibonacci =function(n) {
    return n <2? n : arguments.callee(n-1) + arguments.callee(n-2);
};
onmessage =function(event) {
    var n = parseInt(event.data, 10);
    postMessage(fibonacci(n));
};
```

在 Chrome 浏览器中访问主页文件，则可以在控制台中看到输出的信息，如图 9.7 所示。

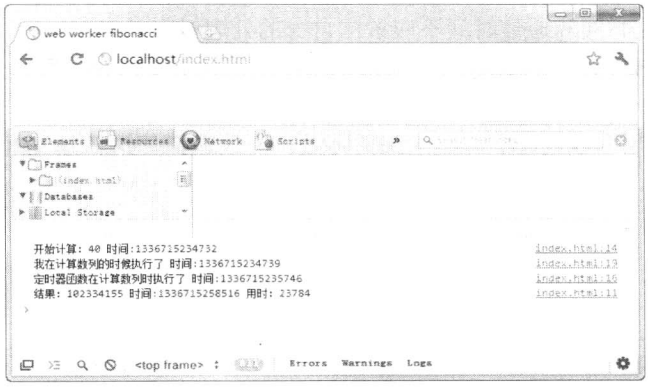


图 9.7 在控制台中查看信息

9.3.6 使用多线程绘图

JavaScript 一个最原始的缺陷就是它的单线程任务处理，这个局限性意味着一个长时间运行的进程会冻结主窗口。通常说的浏览器 UI 线程阻塞，这是由于主线程在处理所有的可视化元素及其相关任务，如绘制、刷新、动画、用户输入事件等。

这种线程过载的严重后果是页面被冻结，且用户不能再与 Web 应用进行交互。因此，Web 应用的用户体验就非常差，用户可能会关掉这个 Tab 或者整个浏览器。为了避免发生这种情况，浏览器已经引入一种保护机制，当脚本运行时间过长时，就会弹出警告。

但是，这种机制并不能正确分辨究竟是一段脚本编写有问题，还是这段脚本确实需要更多的时间来完成它的工作。尽管如此，由于它阻塞了 UI 线程，所以提示一下比让用户无限等待要好。

为了避免出现此类情况，传统做法是通过 setTimeout 和 setInterval 方法试图模拟并行任务，或者通过 XMLHttpRequest 对象，也可以异步地处理 HTTP 请求，避免从远程服务器载入资源时冻结 UI。当然，有了 Web Workers 技术之后，情况会有很大的好转。

Note

上面几节示例主要演示了使用 Web Workers 强化数学科学计算，如光线跟踪、分形、素数等。这些科学计算是理解 Web Workers 工作方式的很好示例，但是很少能给用户一些关于如何在真实世界的应用中使用它们的具体观点。当然，Web Workers 还有很多新的有趣应用：

- ☑ 图像处理。通过从 canvas 或 video 元素中获取数据，可以把图像分割成几个不同的区域，并把它们推送给并行的不同 Worker。这样就会在多线程中受益，此时运行速度明显加快。
- ☑ 大量数据检索。在调用 XMLHTTPRequest 处理大量数据时，如果处理这些数据所需的时间长短非常重要，最好在 Web Workers 中来做这些，避免冻结 UI 线程。这样可以保持一个可交互的应用。
- ☑ 背景数据分析。由于在使用 Web Workers 的时候，可以考虑 JavaScript 的新应用场景。例如，可以想像在不影响 UI 体验的情况下实时处理用户输入。利用这样一种可能，可以设计像 Word（Office Web Apps 套装）一样的应用，当用户打字时，后台在词典中进行查找，帮助用户自动纠错等。
- ☑ 针对本地数据的并发请求，Web Workers APIS 定义了一个在后台运行脚本的办法。因此用户可以执行一些存在于主页面之外的线程而不影响页面的渲染性能。

此外，如果在视频游戏世界，可以考虑将人工智能或者物理引擎的数据发送到 Web Workers。通常情况下，只要不需要 DOM，任何可能影响用户体验的、耗时的 JavaScript 代码都可以使用 Web Workers 并发处理。然而，使用 Web Workers 时也需要注意以下 3 点：

- ☑ Worker 初始化时间和通信时间不应该比自身的处理时间长。
- ☑ 使用多个 Worker 时，将会占用大量系统资源，消耗更多内存。
- ☑ 代码块之间的依赖关系需要一些同步的逻辑。并行就没有那么简单。

本示例演示如何使用多线程绘制光线追踪的特效。光线追踪使用一些 CPU 密集型的数学计算，据此来模拟光线的路径。通过大量并发数学计算来模拟一些诸如反射、折射、材质等效果，演示效果如图 9.8 所示。

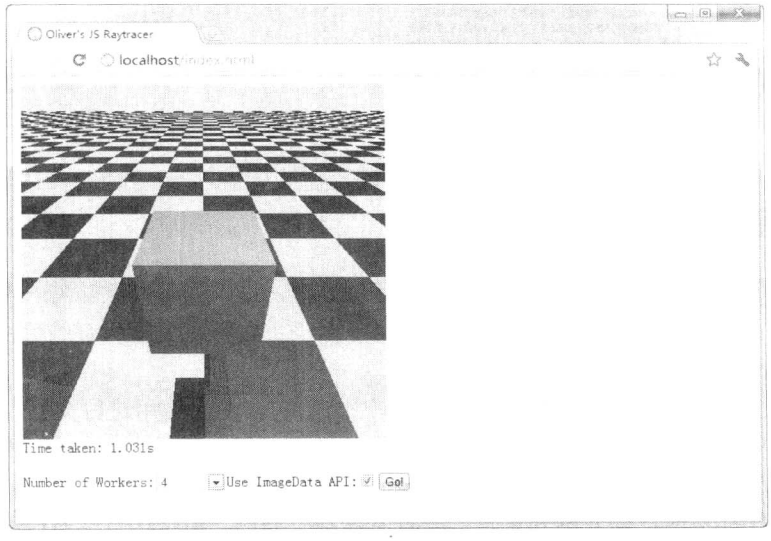


图 9.8　光线追踪特效

如果不用多线程并发计算和渲染，则可以看到浏览器要完成整个图像绘制需要 2 秒多，如图 9.9 所示。而选用 16 个子线程并发计算，则浏览器会快速完成图像绘制，仅需要 1 秒多，如图 9.10 所示。

本示例完整源代码请参考光盘示例，下面简单介绍两处最关键的代码。

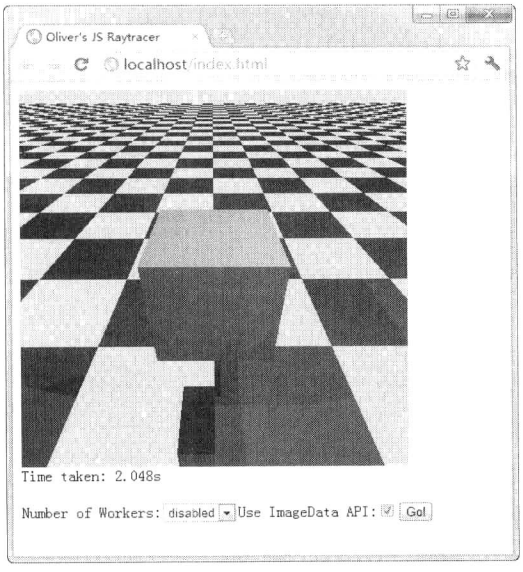


图 9.9　单线程处理

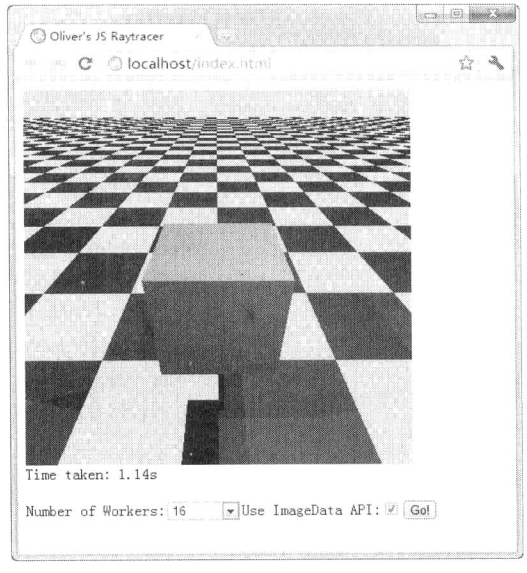


图 9.10　16 个线程处理

在主页面包含的 jobqueue.js 文件中，通过一个分支结构检测浏览器是否支持 Web Workers，如果支持，则允许选择设置多线程项，并多线程进行处理，否则只能够使用单线程处理任务。

```
function JobQueue(numWorkers) {
    this.workersAvailable = true;
    this.intialised = false;
    var cancelled = false;
    if (numWorkers <= 0 || !window.Worker) {
        …
    } else {
        …
    }
    this.stop = function (f) {
        cancelled = true;
        cancelCallback = f;
    }
}
```

为 GO 按钮绑定一个 click 事件处理函数，在该函数中使用 for 循环体创建多个子线程，实现并发数据计算，以提供图像绘制和渲染速度。

```
for (jobCount = 0; jobCount < numWorkers; ++jobCount) {
    worker = new Worker("render-task.js");
    worker.runNextTask = function() {
        this.postMessage(this.queue.shift());
    }
    worker.onmessage = function() {
```

Note

```
            if (--jobCount === 0) {
                self.initialised = true;
                for (var i = 0; i < workers.length; ++i)
                    workers[i].onmessage = function(evt) {
                        if (cancelled) {
                            if (++cancelledWorkers === runningWorkers) {
                                cancelCallback();
                            }
                            return;
                        }
                        var res = parse(evt.data);
                        if (jobs[res.jobid].done)
                            jobs[res.jobid].done(res.result);
                        if (--jobCount === 0)
                                queueDone();
                        if (this.queue.length)
                            this.runNextTask();
                        else
                            --runningWorkers;
                    };
                done();
            }
        };
        worker.postMessage(message);
        workers[jobCount] = worker;
    }
```

在子线程文件 render-task.js 中，尝试调用绘图、渲染函数完成图形绘制功能。

```
onmessage = function(event) {
    try {
        var message = parse(event.data);
        var type = message.type;
        if (message.type === "init") {
            _scene = message.scene();
            processRenderCommand = message.processRenderCommand;
            postMessage("initDone");
        } else {
            result = processRenderCommand(message.command);
            postMessage("({jobid:" + message.jobid + ", result:" + result + "})");
        }
    } catch(e) {
        postMessage(e);
    }
};
```

在这个应用脚本中，有两个方法明显地占用了大多数时间：renderScene()和 getPixelColor()。

getPixelColor()方法的目的是计算当前的像素。光线追踪是一个像素一个像素地渲染场景。使用 getPixelColor()方法之后再调用 rayTrace()方法执行渲染阴影、环境光等操作。这是本示例应用的核心

部分。如果分析 rayTrace()方法的代码，会发现这些代码没有任何 DOM 依赖。因此非常适合把它们设置为并行处理。此外，我们很容易将图像渲染拆分到几个线程中，因为每个像素的计算之间没有同步进行的必要。每个像素的操作与它们的邻居是独立的，因为在本例中没有使用抗锯齿。

使用浏览器预览，可以看出不使用 Worker 和使用 4 个 Worker 的显著区别。

Note

- ☑ 使用单线程的 processRenderCommand()方法几乎占据了全部可用 CPU，并且场景绘制耗时 2.048 秒。
- ☑ 使用 16 个 Worker 之后，processRenderCommand()方法在 16 个并行的线程中执行。我们甚至可以在右侧一栏看到它们的 Worker Id。这次场景绘制耗时 1.031 秒，受益是明显的：绘制速度快了一倍。

9.4　综合应用：模拟退火算法

模拟退火算法来源于固体退火原理，将固体加温至充分高，再让其缓缓冷却，加温时，固体内部粒子随温度升高逐渐变为无序状，内能增大，而缓缓冷却时粒子渐趋有序，使每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

如果直接使用 JavaScript 完成模拟退火算法运算，则会造成卡机，甚至假死现象，Mozbox 的 Paul Rouget 用模拟退火算法演示了 Web Workers 的优越性，下面就来分析如何使用 Web Workers 实现这个应用，示例演示效果如图 9.11 所示。

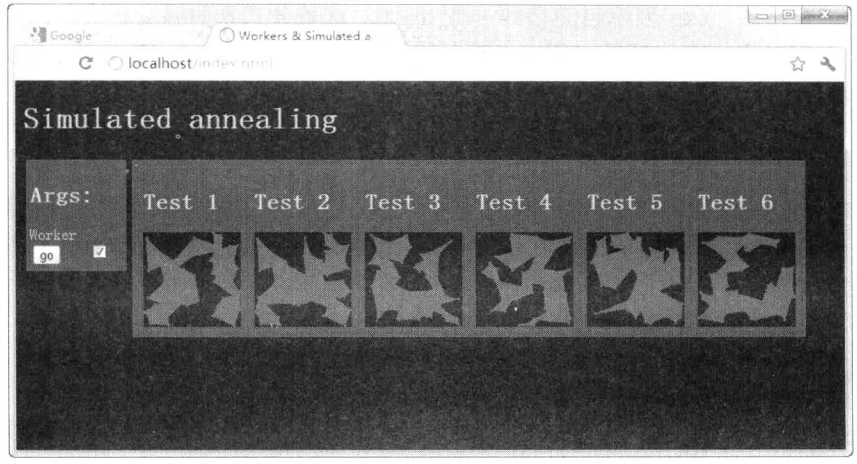


图 9.11　模拟退火算法

9.4.1　认识模拟退火算法

认识模拟退火算法前，先了解爬山算法。爬山算法是一种简单的贪心搜索算法，该算法每次从当前解的临近解空间中选择一个最优解作为当前解，直到得到一个局部最优解。

爬山算法实现很简单，其主要缺点是会陷入局部最优解，而不一定能搜索到全局最优解。如图 9.12 所示，假设 C 点为当前解，爬山算法搜索到 A 点这个局部最优解就会停止搜索，因为在 A 点无论向哪个方向小幅度移动都不能得到更优的解。

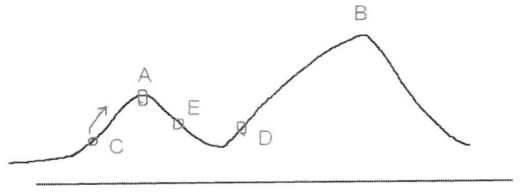


图 9.12　爬山算法

爬山算法是地道的贪心法，每次都只选择一个当前最优解，因此只能搜索到局部的最优值。模拟退火算法其实也是一种贪心算法，但是它的搜索过程引入了随机因素。模拟退火算法以一定的概率来接受一个比当前解要差的解，因此有可能会跳出这个局部的最优解，达到全局的最优解。以图 9.12 为例，模拟退火算法在搜索到局部最优解 A 后，会以一定的概率接受到 E 的移动。也许经过几次这样的不是局部最优的移动后会到达 D 点，于是就跳出了局部最大值 A。

模拟退火算法描述如下：

若 J（Y（i+1））>=J（Y（i））（即移动后得到更优解），则总是接受该移动；

若 J（Y（i+1））<J（Y（i））（即移动后的解比当前解要差），则以一定的概率接受移动，而且这个概率随着时间推移逐渐降低（逐渐降低才能趋向稳定）。

这里的一定概率的计算参考了金属冶炼的退火过程，这也是模拟退火算法名称的由来。

根据热力学的原理，在温度为 T 时，出现能量差为 dE 的降温的概率为 P（dE），表示为：

P（dE）=exp（dE/（kT））

其中 k 是一个常数，exp 表示自然指数，且 dE<0。这条公式表明温度越高，出现一次能量差为 dE 的降温的概率就越大；温度越低，则出现降温的概率就越小。又由于 dE 总是小于 0（否则就不叫退火了），因此 dE/kT<0，所以 P（dE）的函数取值范围是（0,1）。

随着温度 T 的降低，P（dE）会逐渐降低。将一次向较差解的移动看做一次温度跳变过程，我们以概率 P（dE）来接受这样的移动。

关于爬山算法与模拟退火算法，有一个有趣的比喻：爬山算法是兔子朝着比现在高的地方跳去。它找到了不远处的最高山峰。但是这座山不一定是珠穆朗玛峰。这就是爬山算法，它不能保证局部最优值就是全局最优值。模拟退火算法是兔子喝醉了，它随机地跳了很长时间。这期间，它可能走向高处，也可能踏入平地，但是它渐渐清醒了并朝最高处跳去。

9.4.2　编写应用主页面

下面代码是调用 Web Workers 的 HTML 主页面代码。为了方便说明，该 HTML 示例也化繁为简。本例目的不在于建立漂亮的界面，而是通过搭建一个简洁的框架，演示如何控制 Web Workers 并实际运行。应用程序的页面嵌入了 canvas 元素来显示输入的图像。页面上还有一个用于开始执行运算的按钮，以及一个复选框，决定是否采用 Web Workers 多线程模式。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-us" lang="en-us">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Workers &amp;amp; Simulated annealing/</title>
    <script type="application/javascript" src="main.js"></script>
```

Note

```
<script type="application/javascript" src="canvas.js"></script>
<script type="application/javascript" src="ground.js"></script>
<script type="application/javascript" src="annealing.js"></script>
<script type="application/javascript" src="points.js"></script>
<style type="text/css">
body { background: black; color:#CCCCCC; }
canvas { background: black; }
canvas:hover { background: #444444; }
div { float: left; border :1px solid #444444; padding:3px; margin: 3px; background:#3B3B3B; }
label { display: block; clear: both; }
label > input { width: 3em; float: right; }
.args { width: 100px; }
#gobutton { display: none; }
</style>
</head>
<body onload="controls.doLoad()">
<h1>Simulated annealing</h1>
<div id="args">
    <h2>Args:</h2>
    <label for="worker">Worker
    <input type="checkbox" id="worker"/>
</label>
    <input type="button" onclick="controls.go()" value="go"/>
</div>
<div id="groundContainer"/>
</body>
</html>
```

接下来在 main.js 文件中编写 GO 按钮的 click 事件响应函数，该函数以 controls.go()方法呈现，调用该方法将初始化页面，并创建 6 个 Test 实例，完成 6 幅图像的绘制。

```
var controls = {
  doLoad: function() {
  },
  go: function() {
    var useThreads = document.getElementById("worker").checked;
    function Test(name, p) {
      this.g = ground.newGround(name, p);
      this.name = name;
      var self = this;
      setTimeout(function() {
        self.go();
      }, 10);
    };
    Test.prototype = {
      go: function() {
        ground.drawPoints(this.g);
        var opts = {
          points: this.g.allPoints,
          t0: 1,
          g: 0.99,
```

Note

```
            stepsPerT: 10
          }
          var listener = {
            ctx: this.g,
            name: this.name,
            onNewMin: function(p) {
            },
            onDone: function(p) {
              ground.clear(this.ctx);
              ground.drawPath(this.ctx, p);
              ground.drawPoints(this.ctx);
            }
          };
          var a;
          if (useThreads)
            a = new ThreadedAnnealing();
          else
            a = new Annealing();
          a.init(opts, 200, 200, listener);
          a.go();
        }
      };
      var t1 = new Test("Test 1", p1);
      var t2 = new Test("Test 2", p2);
      var t3 = new Test("Test 3", p3);
      var t4 = new Test("Test 4", p4);
      var t5 = new Test("Test 5", p5);
      var t6 = new Test("Test 6", p6);
    }
};
```

9.4.3 编写 worker.js

现在，将 Worker 用来与页面通信的代码添加到 worker.js 文件中。Web Workers 完成运算即可使用 postMessage 通知页面。我们将利用这个信息更新主页上的图像。Worker 创建完成后，会先使用 importScripts('annealing.js');命令导入退火算法文件 annealing.js，然后实例化一个退火算法对象，然后在 Worker 线程的 onmessage 回调函数中执行退火运算，并返回运算结果，详细代码如下。

```
importScripts('annealing.js');
var annealing = new Annealing();
onmessage = function(event) {
  var method = event.data[0];
  if (method == "init") {
    var args = event.data[1];
    var listener = {
      onNewMin: function(p) {
        postMessage(["onNewMin", p]);
      },
      onDone: function(p) {
        postMessage(["onDone", p]);
```

```
        }
      };
      annealing.init(args.opts, args.width, args.height, listener);
    }
    if (method == "go") {
      annealing.go();
    }
}
```

Note

9.4.4 与 Web Workers 通信

在 index.html 文件中，可以通过给 Worker 发送一些代表运算任务的数据和参数来使用 Worker。方法是使用 postMessage 函数发送一个 JavaScript 对象。这个 JavaScript 对象包含了 Worker 负责处理的 RGBA 图像数据阵列、图像的尺寸和像素范围。每个 Worker 根据接收到的信息分别对图像的不同部分进行处理，然后在 canvas.js 中通过 canvas 元素来显示输入的图像数据。

```
function Annealing() {};
Annealing.prototype = {
  card: 0,
  width: 0,
  height: 0,
  k: 0,
  points: [],
  weight: null,
  currentPath: [],
  minWeight: null,
  minPath: [],
  timeout: null,
  t: null,
  t0: null,
  g: null,
  stepsPerT: null,
  step: 0,
  init: function(opts, width, height, listener) {
    this.points = opts.points;
    this.card = this.points.length;
    this.t0 = opts.t0;
    this.g = opts.g;
    this.stepsPerT    = opts.stepsPerT;
    this.stopValue = 0.000000001;
    this.listener = listener;
    this.width = width;
    this.height = height;
    for (var i = 0; i < this.card; i++)
      this.currentPath.push(i);
    this.t = this.t0;
  },
  go: function() {
    while (true) {
      this.cycle();
```

```
        if (this.t < this.stopValue)
            break;
    }
    if (this.listener)
        this.listener.onDone(this.minPath);
},
adoptPath: function(p, w) {
    this.currentPath = p;
    this.currentWeight = w;
    if (w < this.minWeight) {
        this.minWeight = w;
        this.minPath = p.slice(0);
        return true;
    }
    return false;
},
cycle: function() {
    this.step += 1;
    var newMin = false;
    var tmpPath = this.oneStep();
    var w = this.computeWeight(tmpPath);
    if (!this.currentWeight) {
        this.minWeight = w;
        this.minPath = tmpPath.slice(0);
        newMin = this.adoptPath(tmpPath, w);
    } else {
        var df = w - this.currentWeight;
        if (df > 0) {
            var p = Math.random();
            if (p <= Math.exp(-1 * df / this.t)) {
                newMin = this.adoptPath(tmpPath, w);
            }
        } else {
            newMin = this.adoptPath(tmpPath, w);
        }
    }
    if (this.step == this.stepsPerT) {
        this.step = 0;
        this.t *= this.g;
    }
    //if (newMin && this.listener)
    // this.listener.onNewMin(this.minPath);
},
computeWeight: function(path) {
    var weight = 0;
    for (var i = 0; i < this.card; i++) {
        var idx = path[i];
        var prevIdx;
        if (i == 0) {
            prevIdx = path[this.card - 1];
```

Note

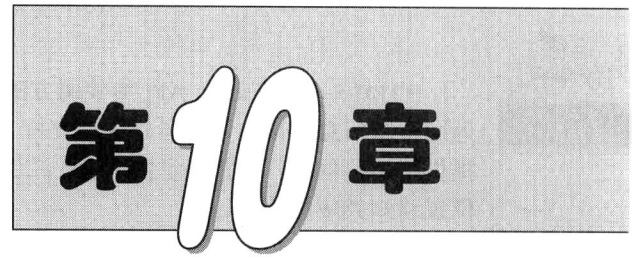
```
        } else {
            prevIdx = path[i - 1];
        }
        var x0 = this.points[prevIdx].x;
        var y0 = this.points[prevIdx].y;
        var x1 = this.points[idx].x;
        var y1 = this.points[idx].y;

        weight += Math.sqrt(
            (x1 - x0) * (x1 - x0) +
            (y1 - y0) * (y1 - y0)
        );
    }
    return weight / (200 * this.card);
},
oneStep: function() {
    var i = Math.round(Math.random() * this.card);
    var j = Math.round(Math.random() * this.card);
    while (j == i)
        j = Math.round(Math.random() * this.card);
    var t = i;
    if (i > j) {
        i = j;
        j = t;
    }
    var v1 = this.currentPath.slice(0, i);
    var v2 = this.currentPath.slice(i, j);
    var v3 = this.currentPath.slice(j, this.card);
    return v1.concat(v2.reverse().concat(v3));
}
}
function ThreadedAnnealing() {
    this.worker = new Worker("Worker.js");
};
ThreadedAnnealing.prototype = {
    init: function(opts, w, h, l) {
        this.worker.onmessage = function(event) {
            var msg = event.data[0];
            var p = event.data[1];
            l[msg](p);
        };
        this.worker.onerror = function(event) {
            dump("WORKER ERROR");
        };
        this.worker.postMessage(["init", {
                opts: opts,
                width: w,
                height: h
            }]);
    },
```

```
    go: function() {
        this.worker.postMessage(["go"]);
    }
}
```

Note

postMessage 可以对 imageData 对象进行高效序列化，以便通过 canvas API 使用。不过一些支持 Worker 和 postMessage API 的浏览器也许还不能支持 postMessage 的这种扩展的序列化能力。例如，Firefox 3.5 不能通过 postMessage 传送 imageData 对象，不过未来版本可能会提供此支持。因此，在本章中介绍的图像处理示例中，以传送 imageData.data（数据序列化方式间 JavaScript 数组一样）的方式代替传送 imageData 对象本身在 Web Workers 执行运算任务时，它们同时将其状态和结果返回到页面上。同样地，消息包含了一个 JavaScript 对象，这个对象中含有图像数据和标记处理范围的坐标信息。

HTML 页面上的消息处理程序接收上述数据，并用新的像素值更新 canvas。处理后的图像数据到达时，HTML 页面即时显示结果。现在，我们创建了一个可以处理图像的示例应用，并且具有利用多核 CPU 的潜在优势。此外，Web Workers 执行时，程序也不会锁定用户界面，不会使用户界面停止响应。



Geolocation 地理位置

HTML5 Geolocation API 是 HTML5 新增的地理位置应用程序接口，它提供了一个可以准确感知浏览器用户当前位置的方法。如果浏览器支持，且设备具有定位功能，就能够直接使用这组 API 来获取当前位置信息。该 Geolocation API 可以应用于移动设备中的地理定位。

Geolocation API 允许用户在 Web 应用程序中共享位置信息，使其能够享受位置感知服务。本章将介绍 HTML5 Geolocation 位置信息的来源：纬度、经度和其他特性，以及获取这些数据的途径（GPS、Wi-Fi 和蜂窝站点等）。然后将讨论 HTML5 地理定位数据的隐私问题，以及浏览器如何使用这些数据。最后，将深入探讨 HTML5 Geolocation API 在实际中的应用。目前有两种类型的定位请求 API：单次定位请求和重复性的位置更新请求。本章会分别介绍这两种请求的使用方式和适用场景，并演示如何使用它们构建实用的 HTML5 Geolocation 应用程序。

10.1 位置信息概述

Note

HTML5 Geolocation API 的使用方法相当简单。请求一个位置信息，如果用户同意，浏览器就会返回位置信息，该位置信息是通过支持 HTML5 地理定位功能的底层设备。例如，笔记本电脑或手机提供给浏览器的。位置信息由纬度、经度坐标和一些其他元数据组成。有了这些位置信息就可以构建位置感知类应用程序。

10.1.1 为什么要学习 Geolocation

设想这样一个场景：有一个 Web 应用程序，它可以向用户提供附近不远处某商店运动鞋的打折优惠信息。使用 HTML5 Geolocation API，可以请求其他用户共享他们的位置，如果他们同意，应用程序就可以提供相关信息，告诉用户去附近哪家商店可以挑选到打折的鞋子。

HTML5 Geolocation 技术的另一个应用场景是构建计算行走（跑步）路程的应用程序。想象一下，在开始跑步时通过手机浏览器启动应用程序的记录功能。在用户移动过程中，应用程序会记录已跑过的距离，还可以把跑步过程对应的坐标显示在地图上，甚至可以显示出海拔信息。如果用户正在和其他选手一起参加跑步比赛，应用程序甚至可以显示其对手的位置。

还有一种 HTMLS Geolocation 应用场景是基于 GPS 导航的社交网络应用，可以用它看到好友们当前所处的位置，如知道了好友的方位，就可以挑选合适的咖啡馆。此外，还有很多特殊的应用。

10.1.2 位置信息表示方式

位置信息主要由一对纬度和经度坐标组成，例如：

```
Latitude: 39.17222, Longitude: -120.13778
```

在这里，纬度（距离赤道以北或以南的数值表示）是 39.172 22，经度（距离英国格林威治以东或以西的数值表示）是 120.137 78，经纬度坐标可以用以下两种方式表示：

- ☑ 十进制格式，如 39.172 22。
- ☑ DMS 角度格式，如 39° 20'。

HTML5 Geolocation API 返回坐标的格式为十进制格式。

除了纬度和经度坐标，HTML5 Geolocation 还提供位置坐标的准确度，并提供其他一些元数据，具体情况取决于浏览器所在的硬件设备，这些元数据包括海拔、海拔准确度、行驶方向和速度等。如果这些元数据不存在则返回 null。

10.1.3 位置信息来源

HTML5 Geolocation API 不指定设备使用哪种底层技术来定位应用程序的用户。相反，它只是用于检索信息的 API，而且通过该 API 检索到的数据只具有某种程度的准确性。它并不能保证设备返回的实际位置是精确的。设备可以使用下列数据。

- ☑ IP 地址
- ☑ 三维坐标
  - ➢ GPS 全球定位系统。

- ➢ 从 RFID、Wi-Fi 和蓝牙到 Wi-Fi 的 MAC 地址。
- ➢ GSM 或 CDMA 手机的 ID。

☑ 用户自定义数据

为了保证更高的准确度，许多设备使用一个或多个数据源的组合。

Note

10.1.4 IP 定位

在 HTML5 Geolocation API 之前，基于 IP 地址的地理定位方法是获得位置信息的唯一方式，但其返回的位置信息通常并不靠谱，基于 IP 地址的地理定位的实现方式是，自动查找用户的 IP 地址，然后检索其注册的物理地址。因此，如果用户的 IP 地址是 ISP 提供的，其位置往往就由服务供应商的物理地址决定，该地址可能距离用户数千米。

优点：

☑ 任何地方都可用。

☑ 在服务器端处理。

缺点：

☑ 不精确，一般精确到城市级。

☑ 运算代价大，并经常出错。

许多网站会根据由 IP 地址得到的位置信息来做广告，所以在实际中可能会遇到这样的情况：你到其他国家旅行，访问非本地网站时却突然看到了本地广告，基于访问网站所在国家或地区的 IP 地址。

10.1.5 GPS 定位

GPS 定位是通过收集运行在地球周围的多个 GPS 卫星的信号实现的，但 GPS 定位时间可能较长，因此它不适合需要快速响应的应用程序。因为获取 GPS 定位数据需要时间可能较长，所以开发人员可能需要异步查询用户位置。可以添加一个状态栏以显示正在重新获取应用程序用户的位置。

优点：

☑ 比较精确。

缺点：

☑ 定位时间长，耗电量大。

☑ 室内效果不好。

☑ 需要硬件设备支持。

10.1.6 Wi-Fi 定位

基于 Wi-Fi 的地理定位信息是通过三角距离计算得出的，这个三角距离指的是用户当前位置到已知的多个 Wi-Fi 接入点的距离。不同于 GPS，Wi-Fi 定位在室内也非常准确。

优点：

☑ 精确。

☑ 可在室内便用。

☑ 可以简单、快捷地定位。

缺点：

☑ 适合在大城市，在乡村等无接入点或者接入点较少的地区效果不好。

10.1.7 手机定位

Note

基于手机的地理定位信息是通过用户到一些基站的三角距离确定的。这种方法可提供相当准确的位置结果。这种方法通常同基于 Wi-Fi 和基于 GPS 的地理定位信息结合使用。

优点：

☑ 相当准确。

☑ 可在室内使用。

☑ 可以简单、快捷地定位。

缺点：

☑ 在基站较少的偏远地区效果不好。

10.1.8 自定义定位

除了通过编程计算出用户的位置外，也可以允许用户自定义其位置。应用程序可能允许用户输入他们的地址、邮政编码和其他一些详细信息。应用程序可以利用这些信息来提供位置感知服务。

优点：

☑ 可以获得比程序定位服务更准确的位置数据。

☑ 允许地理定位服务的结果作为备用位置信息。

☑ 用户自行输入可能比自动检测更快。

缺点：

☑ 可能不准确，特别是当用户位置变更后。

10.2 使用 Geolocation API

本节将更详细地探讨 HTML5 Geulocation API 的使用方法。在 HTML5 中，为 window.navigator 对象新增了一个 geolocation 属性，可以使用 Geolocation API 对该属性进行访问，window.navigator 对象的 geolocation 属性存在 3 个方法，利用这些方法可以实现位置信息的读取。

10.2.1 检查浏览器支持性

各浏览器对 HTML5 Geolocation 的支持程度不同，并且还在不断更新。在 HTML5 的所有功能中，HTML5 Geolocation 是第一批被全部接受和实现的功能之一，这对于开发人员来说是个好消息。相关规范已达到一个非常成熟的阶段，不会做太大的改变。各浏览器对 HTML5 Geolocation 的支持情况如表 10.1 所示。

表 10.1 浏览器支持概述

浏 览 器	说 明
IE	通过 Gears 插件支持
Firefox	3.5 及以上的版本支持
Opera	10 及以上的版本支持
Chrome	2.0 及以上的版本支持
Safari	4.0 及以上的版本支持

由于浏览器的支持情况不同，在使用之前最好先检查浏览器是否支持 HTML5 Geolocation API，确保浏览器支持其所要完成的所有工作。这样当浏览器不支持时，就可以提供一些替代文本，以提示用户升级浏览器或安装插件来增强现有浏览器功能。

Note

```
function loadDemo() {
    if(navigator.geolocation) {
        document.getElementById("support").innerHTML = "支持 HTML5 Geolocation";
    } else {
        document.getElementById("support").innerHTML = "当前浏览器不支持 HTML5 Geolocation";
    }
}
```

在上面代码中，loadDemo 函数测试了浏览器的支持情况，这个函数是在页面加载的时候被调用的。如果存在地理定位对象，navigator.geolocation 调用将返回该对象，否则将触发错误。页面上预先定义的 support 元素会根据检测结果显示支持情况的提示信息。

10.2.2 获取当前地理位置

使用 getCurrentPosition 方法可以取得用户当前的地理位置信息，该方法的用法如下所示。

```
void getCurrentPosition(onSuccess, onError, options) ;
```

第 1 个参数为获取当前地理位置信息成功时所执行的回调函数，第 2 个参数为获取当前地理位置信息失败时所执行的回调函数，第 3 个参数为一些可选属性的列表。其中，第 2、3 个参数为可选属性。

getCurrentPosition 方法中的第 1 个参数为获取当前地理位置信息成功时所执行的回调函数。该参数的使用方法如下所示。

```
navigator.geolocation.getCurrentPosition(function(position){
    //获取成功时的处理
}
```

在获取地理位置信息成功时执行的回调函数中，用到了一个参数 position，它代表一个 position 对象，我们将在后面小节中对这个对象进行具体介绍。

getCurrentPosition 方法中的第 2 个参数为获取当前地理位置信息失败时所执行的回调函数。如果获取地理位置信息失败，可以通过该回调函数把错误信息提示给用户。当在浏览器中打开使用了 Geolocation API 来获得用户当前位置信息的页面时，浏览器会询问用户是否共享位置信息。如果在该画面中拒绝共享的话，也会发生错误。

该回调函数使用一个 error 对象作为参数，该对象具有以下两个属性。

☑ code 属性

code 属性包含 3 个值，简单说明如下：

- ➢ 当属性值为 1 时，表示用户拒绝了位置服务。
- ➢ 当属性值为 2 时，表示获取不到位置信息。
- ➢ 当属性值为 3 时，表示获取信息超时错误。

☑ message 属性

message 属性为一个字符串，在该字符串中包含了错误信息，这个错误信息在开发和调试时将很有用。因为有些浏览器中不支持 message 属性，如 Firefox。

在 getCurrentPosition 方法中使用第二个参数捕获错误信息的具体使用方法如下所示。

Note

```
navigator.geolocation.getCurrentPosition(
        function(position){
            var cords = position.coords;
            showMap(coords.latitude, coords.longitude,coords.accuracy);
        },
        //捕获错误信息
        function (error){
            var errorTypes = {
                1:位置服务被拒绝
                2:获取不到位置信息
                3:获取信息超时
             }
            alert( errorTypes[error.code]+ ":,不能确定当前地理位置");
        }
);
```

getCurrentPosition 方法中的第三个参数可以省略，它是一些可选属性的列表，这些可选属性说明如下。

☑ enableHighAccuracy

是否要求高精度的地理位置信息，这个参数在很多设备上设置了都不起作用，因为在设备上使用时需要结合设备电量、具体地理情况来综合考虑。因此，多数情况下把该属性设为默认，由设备自身来调整。

☑ timeout

对地理位置信息的获取操作做一个超时限制（单位为毫秒）。如果在该时间内未获取到地理位置信息，则返回错误。

☑ maximumAge

对地理位置信息进行缓存的有效时间的单位为毫秒。例如，maximumAge : 120000（1 分钟是 60000 毫秒），如果 10 点整的时候获取过一次地理位置信息，10:01 的时候，再次调用 navigator.geolocation.getCurrentPosition 重新获取地理位置信息，则返回的依然为 10:00 时的数据（因为设置的缓存有效时间为 2 分钟）。超过这个时间后缓存的地理位置信息被废弃，尝试重新获取地理位置信息。如果该值被指定为 0，则无条件重新获取新的地理位置信息。

对于这些可选属性的具体设置方法如下所示。

```
navigator.geolocation.getCurrentPosition(
        function(position){
          //获取地理位置信息成功时所做的处理
        },
        function(error){
          //获取地理位置信息失败时所做的处理
        },
          //以下为可选属性
        {
          //设置缓存有效时间为 2 分钟
          maximumAge: 60*1000*2,
          //5 秒钟内未获取到地理位置信息则返回错误
          timeout: 5000
```

```
        }
    }
```

10.2.3 监视位置信息

Note

使用 watchPosition 方法可以持续获取用户的当前地理位置信息，它会定期地自动获取。watchPosition 方法基本语法如下所示。

```
int watchCurrentPosition(onSuccess, onError, options) ;
```

该方法参数的说明和使用与 getCurrentPosition 方法相同。调用该方法后会返回一个数字，这个数字的用法与 JavaScript 脚本中 setInterval 方法的返回值用法类似，可以被 clearWatch 方法使用，以停止对当前地理位置信息的监视。

10.2.4 停止获取位置信息

使用 clearWatch 方法可以停止对当前用户的地理位置信息的监视。具体用法如下所示。

```
void clearWatch(watchId);
```

参数 watchId 为调用 watchCurrentPosition 方法监视地理位置信息时的返回参数。

10.2.5 隐私保护

HTML5 Geolocation 规范提供了一套保护用户隐私的机制。除非得到用户明确许可，否则不可获取位置信息。具体设置步骤如下：

第 1 步，用户从浏览器中打开位置感知应用程序。

第 2 步，应用程序 Web 页面加载，然后通过 Geolocation 函数调用请求位置坐标。浏览器拦截这一请求，然后请求用户授权。

第 3 步，如果用户同意，浏览器从其宿主设备中检索坐标信息，如 IP 地址、Wi-Fi 或 GPS 坐标，这是浏览器的内部功能。

第 4 步，浏览器将坐标发送给受信任的外部定位服务，它返回一个详细位置信息，并将该位置信息发回给 HTML5 Geolocation 应用程序。

提示，应用程序不能直接访问设备，它只能请求浏览器来代表它访问设备。

访问使用 HTML5 Geolocation API 的页面时，会触发隐私保护机制。如果仅仅是添加 HTML5 Geolocation 代码，而不被任何方法调用，则不会触发隐私保护机制。只要所添加的 HTML5 Geolocation 代码被执行，浏览器就会提示用户应用程序要共享位置。执行 HTML5 Geolocation 的方式很多，例如，可以调用 navigator.geolocation.getCurrentPosition 方法等。

除了询问用户是否允许共享其位置之外，Firefox 等一些浏览器还可以让用户选择记住该网站的位置服务权限，以便下次访问的时候不再弹出提示框，类似于在浏览器中记住某些网站的密码。

10.2.6 处理位置信息

因为位置数据属于敏感信息，所以接收到之后，必须小心地处理、存储和重传。如果用户没有授权存储这些数据，那么应用程序应该在相应任务完成后立即删除它。如果要重传位置数据，建议先对其进行加密。在收集地理定位数据时，应用程序应该着重提示用户以下内容：

Note

- ☑ 会收集位置数据。
- ☑ 为什么收集位置数据。
- ☑ 位置数据将保存多久。
- ☑ 怎样保证数据的安全。
- ☑ 位置数据怎样共享，如果同意则共享。
- ☑ 用户怎样检查和更新他们的位置数据。

10.2.7 使用 position 对象

如果获取地理位置信息成功，则可以在获取成功后的回调函数中通过访问 position 对象的属性来得到这些地理位置信息。position 对象具有以下这些属性。

- ☑ 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（不能获取时为 null）。
- ☑ accuracy：获取到的纬度或经度的精度（以米为单位）。
- ☑ altitudeAccurancy：获取到的海拔高度的精度（以米为单位）。
- ☑ heading：设备的前进方向。用面朝正北方向的顺时针旋转角度来表示（不能获取时为 null）。
- ☑ speed：设备的前进速度（以米/秒为单位，不能获取时为 null）。
- ☑ timestamp：获取地理位置信息时的时间。

【示例】本示例使用 getCurrentPosition 方法获取当前位置的地理信息，并且在页面中显示 position 对象中的所有属性。

```
<!DOCTYPE html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<title></title>
<script type="text/javascript" src=http://maps.google.com/maps/api/js?sensor=false></script>
<script type="text/javascript">
function showObject(obj,k){
    //递归显示 object
    if(!obj){return;}
    for(var i in obj){
        if(typeof(obj[i])!="object" || obj[i]==null){
            for(var j=0;j<k;j++){
                document.write("    ");
            }
            document.write(i + " : " + obj[i] + "<br/>");
        }
        else
        {
            document.write(i + " : " + "<br/>");
            showObject(obj[i],k+1);
        }
    }
}
function get_location(){
    if(navigator.geolocation)
```

```
navigator.geolocation.getCurrentPosition(show_map,handle_error,{enableHighAccuracy:true,maximumAge:1000});
        else
            alert("你的浏览器不支持使用 HTML5 来获取地理位置信息。");
    }
    function handle_error(err){
        //错误处理
         switch(err.code){
              case 1 :
                   alert("位置服务被拒绝。");
                   break;
              case 2:
                   alert("暂时获取不到位置信息。");
                   break;
              case 3:
                   alert("获取信息超时。");
                   break;
              default:
                   alert("未知错误。");
                   break;
        }
    }
    function show_map(position){
        //显示地理信息
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;
        showObject(position,0);
    }
    get_location();
    </script>
    </head>
    <body>
    <div id="map" style="width:400px; height:400px"></div>
    </body>
```

这段代码的运行效果在不同设备的浏览器上也各不相同，具体运行效果取决于运行浏览器的设备。

10.3 案 例 实 战

上面几节中主要讨论了 Geolocation 的基本应用。下面将使用 Geolocation 技术构建简单有用的 Web 应用程序，通过这些应用程序可以了解到 HTML5 Geolocation API 的强大之处。

10.3.1 使用 Google 地图

本示例介绍如何在页面上显示一幅 Google 地图，并且把用户的当前地理位置标注在地图上面，如果用户的位置发生改变，将把之前在地图上的标记自动更新到新的位置上。当然，要在页面中使用 Google 地图，需要用到 Google Map API。

具体操作步骤如下所示。

第 1 步，在页面中导入 Google Map API 的脚本文件，导入方法如下所示。

```
<script type="text/javascript" src=http://maps.google.com/maps/api/js?sensor=false />
```

第 2 步，设定地图参数，设定方法如下所示。

```
//指定一个 Google 地图上的坐标点，同时指定该坐标点的横坐标和纵坐标
var latlng = new google.maps.LatLng(coords.latitude, coords.longitude);
var myOptions = {
    zoom: 14,                                    //设定放大倍数
    center: latlng,                              //将地圈中心点设定为指定的坐标点
    mapTypeId: google.maps.MapTypeId.ROADMAP     //指定地圈类型
};
```

本例将用户当前位置的纬度、经度设定为页面打开时 Google 地图的中心点。

第 3 步，创建地图，并在页面中显示。

```
var map1= new google.maps.Map(document.getElementById("map"), myOptions);
```

上面代码将地图显示在 id 为“map”的 div 元素中。

第 4 步，在地图上创建标记。

```
var marker = new google.maps.Marker({
    position: latlng,                            //将前面指定的坐标点标注出来
    map: map1                                    //设置在 map1 变量代表的地图中标注
});
```

第 5 步，设置标注窗口并指定标注窗口中的注释文字。

```
var infowindow = new google.maps.InfoWindow({
    content: "当前位置!"                          //指定标注窗口中的注释文字
});
```

第 6 步，打开标注窗口。

```
infowindow.open(map1, marker);
```

整个示例的完整代码如下所示。

```
<!DOCTYPE html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<title></title>
<script type="text/javascript" src=http://maps.google.com/maps/api/js?sensor=false></script>
<script type="text/javascript">
  function init() {
    //取得当前地理位置
    navigator.geolocation.getCurrentPosition(function(position) {
        var coords = position.coords;
        //设定地图参数，将用户的当前位置的纬度、经度设定为地图的中心点。
        var latlng = new google.maps.LatLng(coords.latitude, coords.longitude);
        var myOptions = {
          zoom: 14,
          center: latlng,
          mapTypeId: google.maps.MapTypeId.ROADMAP
```

Note

```
        };
        //创建地图并在“map”的 div 中显示
        var map1;
        map1= new google.maps.Map(document.getElementById("map"), myOptions);
        //在地图上创建标记
        var marker = new google.maps.Marker({
            position: latlng,
            map: map1
        });
        //设定标注窗口，并指定该窗口中的注释文字
        var infowindow = new google.maps.InfoWindow({
          content: "当前位置!"
        });
        //打开标注窗口
        infowindow.open(map1, marker);
    });
  }
</script>
</head>
<body onload="init()">
  <div id="map" style="width:400px; height:400px"></div>
</body>
```

10.3.2 跟踪行走速度

要想快速确定在一定时间内的行走距离，通常可以使用 GPS 导航系统或计步器这样的专用设备。基于 HTML5 Geolocation 提供的强大服务，开发人员可以创建一个网页来跟踪从网页被加载的地方到目前所在位置所经过的距离。只要在手机浏览器中打开本示例页面并授予其位置访问的权限，每隔几秒钟，应用程序就会根据刚才走过的距离更新，并将其增加到总距离中。

该示例使用 watchPosition()函数，每当有新的位置返回，就将其与最后保存的位置进行比较以计算距离。所有缓存位置数据的生命周期都不能大于 205。

具体操作步骤如下所示。

第 1 步，创建示例程序的 HTML 显示代码。这里重点介绍处理数据的脚本，并把所有最新的数据以简单的表格形式展示，分行显示纬度、经度、准确度和以毫秒为单位的时间。此外，显示走过距离的概要情况，还会显示一些文本类统计信息，以便用户能够看到走过距离的概况。

```
<h2>当前位置:</h2>
<table border="1">
    <tr>
        <th width="40" scope="col">纬度</th>
        <td width="114" id="latitude">?</td>
    </tr>
    <tr>
        <td> 经度</td>
        <td id="longitude">?</td>
    </tr>
    <tr>
        <td>准确性</td>
```

```
            <td id="accuracy">?</td>
        </tr>
        <tr>
            <td>最后时间</td>
            <td id="timestamp">?</td>
        </tr>
</table>
<h4 id="currDist">当前行驶距离: 0.0 km</h4>
<h4 id="totalDist">总距离: 0.0 km</h4>
```

表中的数值都是默认值。开始接收到数据后，应用程序会进行相应的更新。

第 2 步，处理 Geolocation 数据。在 Geolocation 数据处理部分，第一段 JavaScript 代码会检测浏览器是否支持 HTML5 Geolocation，然后将检测结果显示在页面上。最后，代码会请求监测用户位置。

```
var totalDistance = 0.0;
var lastLat;
var lastLong;
function updateStatus(message) {
    document.getElementById("status").innerHTML = message;
}
function loadDemo() {
    if(navigator.geolocation) {
            updateStatus("HTML5 Geolocation is supported in your browser.");
            navigator.geolocation.watchPosition(updateLocation,
            handleLocationError,
            {maximumAge:20000}
        );
    }
}
window.addEventListener("load", loadDemo, true);
```

第 3 步，对于距离跟踪器而言，这些出错处理已经足够了，它将检查收到的所有错误编号，并更新页面上的状态信息。

```
function handleLocationError(error) {
    switch (error.code) {
        case 0:
            updateStatus("There was an error while retrieving your location: " + error.message);
            break;
        case 1:
            updateStatus("The user prevented this page from retrieving a location.");
            break;
        case 2:
            updateStatus("The browser was unable to determine your location: " +
            error.message);
            break;
        case 3:
            updateStatus("The browser timed out before retrieving the location.");
            break;
    }
}
```

需要注意的是，将位置监测中的 maximumAge 选项设置为{maximumAge:20000}，这将告诉定位服务，所有缓存的位置数据的生命期都不能大于 20 秒（或 20 000 毫秒），设置此选项可以使页面处于定期更新的状态。可以随意调整参数的大小，并试着同时调整缓存的大小。

第 4 步，大部分工作都将在 updateLocation()函数中实现，此函数中将使用最新数据来更新页面并计算路程。

Note

```
function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var accuracy = position.coords.accuracy;
    var timestamp = position.timestamp;
    document.getElementById("latitude").innerHTML = latitude;
    document.getElementById("longitude").innerHTML = longitude;
    document.getElementById("accuracy").innerHTML = accuracy;
    document.getElementById("timestamp").innerHTML = timestamp;
}
```

收到一组最新坐标数据后，首先要做的是记录所有信息，收集纬度、经度、准确度和时间，然后将这些数据更新到表格中。

应用程序可能选择不显示时间。时间主要供程序使用，它对最终用户没有多大意义，所以可以替换成更便于用户识别的时间指示器，或将其完全删除。

准确度是以 m 为单位的，任何数据都依赖于其准确度。即使不显示给用户，也应该在代码中考虑准确度。显示不准确的值会向用户提供错误的位置信息。因此，将过滤掉所有低精度的位置更新数据。

```
if (accuracy >= 500) {
    updateStatus("Need more accurate values to calculate distance.");
    return;
}
```

第 5 步，计算移动的距离。假设前面已经收到了一个准确的位置，将更新移动的总距离并将其显示给用户，同时还将储存当前值以备后面做比较。为了使界面不太乱，在计算数值时可采用四舍五入或截断的方式。

```
if ((lastLat !== null) && (lastLong !== null)) {
    var currentDistance = distance(latitude, longitude, lastLat, lastLong);
    document.getElementById("currDist").innerHTML = "Current distance traveled: " +
currentDistance.toFixed(4) + " km";
    totalDistance += currentDistance;
    document.getElementById("totalDist").innerHTML = "Total distance traveled: " + totalDistance.toFixed(4) +
" km";
    lastLat = latitude;
    lastLong = longitude;
    updateStatus("Location successfully updated.");
}
```

示例代码的内容，包括错误处理在内，总共不超过 200 行，在这么简短的 HTML 脚本中，已构建了一个能够持续监测用户位置变化的示例应用程序，几乎完整地演示了 Geolocation API 的使用。尽管示例不适用于台式机，但是不妨在支持地理定位功能的手机或移动设备上实践。

整个示例的完整代码如下所示。

Note

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title></title>
</head>
<body onload="loadDemo()">
<h2>当前位置:</h2>
<table border="1">
    <tr>
        <th width="40" scope="col">纬度</th>
        <td width="114" id="latitude">?</td>
    </tr>
    <tr>
        <td> 经度</td>
        <td id="longitude">?</td>
    </tr>
    <tr>
        <td>准确性</td>
        <td id="accuracy">?</td>
    </tr>
    <tr>
        <td>最后时间</td>
        <td id="timestamp">?</td>
    </tr>
</table>
<h4 id="currDist">当前行驶距离: 0.0 km</h4>
<h4 id="totalDist">总距离: 0.0 km</h4>
<script type="text/javascript">
    var totalDistance = 0.0;
    var lastLat;
    var lastLong;
    Number.prototype.toRadians = function() {
      return this * Math.PI / 180;
    }
    function distance(latitude1, longitude1, latitude2, longitude2) {
      var R = 6371;
      var deltaLatitude = (latitude2-latitude1).toRadians();
      var deltaLongitude = (longitude2-longitude1).toRadians();
      latitude1 = latitude1.toRadians(), latitude2 = latitude2.toRadians();
      var a = Math.sin(deltaLatitude/2) *
              Math.sin(deltaLatitude/2) +
              Math.cos(latitude1) *
              Math.cos(latitude2) *
              Math.sin(deltaLongitude/2) *
              Math.sin(deltaLongitude/2);
      var c = 2 * Math.atan2(Math.sqrt(a),
                             Math.sqrt(1-a));
      var d = R * c;
      return d;
```

Note

```
}
function updateStatus(message) {
    document.getElementById("status").innerHTML = message;
}
function loadDemo() {
    if(navigator.geolocation) {
        updateStatus("HTML5 Geolocation is supported in your browser.");
        navigator.geolocation.watchPosition(updateLocation,
                                            handleLocationError,
                                            {maximumAge:20000});
    }
}
function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var accuracy = position.coords.accuracy;
    var timestamp = position.timestamp;
    document.getElementById("latitude").innerHTML = latitude;
    document.getElementById("longitude").innerHTML = longitude;
    document.getElementById("accuracy").innerHTML = accuracy;
    document.getElementById("timestamp").innerHTML = timestamp;
    if (accuracy >= 500) {
        updateStatus("Need more accurate values to calculate distance.");
        return;
    }
    if ((lastLat != null) && (lastLong != null)) {
        var currentDistance = distance(latitude, longitude, lastLat, lastLong);
        document.getElementById("currDist").innerHTML =
          "Current distance traveled: " + currentDistance.toFixed(4) + " km";
        totalDistance += currentDistance;
        document.getElementById("totalDist").innerHTML =
          "Total distance traveled: " + currentDistance.toFixed(4) + " km";
    }
    lastLat = latitude;
    lastLong = longitude;
    updateStatus("Location successfully updated.");
}
function handleLocationError(error) {
    switch(error.code){
    case 0:
      updateStatus("There was an error while retrieving your location: " + error.message);
      break;
    case 1:
      updateStatus("The user prevented this page from retrieving a location.");
      break;
    case 2:
      updateStatus("The browser was unable to determine your location: " + error.message);
      break;
    case 3:
      updateStatus("The browser timed out before retrieving the location.");
```

```
            break;
        }
    }
</script>
</body>
</html>
```

Note

第11章

CSS3 概述

（视频讲解：1 小时 01 分钟）

CSS3 是 CSS 技术的最新升级版本，它是由 Adobe Systems、Apple、Google、HP、IBM、Microsoft、Mozilla、Opera、Sun 等许多 Web 界的巨头联合组成的一个名为 CSS Working Group 的组织共同协商策划的。2010 年在 HTML5 成为 IT 界人士关注的焦点的同时，CSS3 也开始慢慢地普及开来。

目前，对 B/S（Browser/Server，浏览器/服务器模式）结构的 Web 服务，CSS 是不可缺少的技术。在 DIV+CSS 技术时代，CSS 显得格外重要。CSS 可以将 Web 前端的 HTML 代码与页面布局、美化，以及添加的一些特殊效果的代码分隔开来。这样不仅提高了 Web 前端开发的效率，更好地优化了前端设计，并且在修改网页样式时也非常轻松。

对使用过 DIV+CSS 技术的用户来说，CSS 并不陌生。它可以对网页页面进行布局设计，如文本链接样式、鼠标样式和文本显示样式等，还可以统一地控制 HMTL 中各标记的显示属性，并且扩充精确指定网页元素位置、外观以及创建特殊效果的功能。

11.1 回顾 CSS

Note

CSS3 带来的一些改变，为用户创造新的具有冲击力的设计提供了更多方法。另外，CSS3 能完全向后兼容，这样用户就不必更改先前的设计，浏览器也会继续支持 CSS2。

11.1.1 CSS 发展简史

CSS（Cascading Style Sheets，层叠样式表）是一种将表示样式应用到标记的系统。CSS 以设计、改变其 HTML 页面的样式而知名，并且适用于 Web 和其他媒介，如 XML 文档中。有效使用 CSS 对于分离内容及其表示非常关键。

早在 1982 年，蒂姆·伯纳斯·李为使世界各地的物理学家能够方便地进行合作研究以及信息共享，创造了 HTML（Hypertext Markup Language）超文本置标语言。到 1990 年他发明了世界上第一个 WWW（World Wide Web）浏览器。

之后，他把这项发明介绍给了他在 CERN 公司工作的朋友，当时网页浏览器被其世界各地的成员用来浏览 CERN 庞大的电话薄。

1993 年，NCSA 推出了 Mosaic 浏览器并迅速爆红，成为世界上第一个被广泛应用的浏览器，推动了互联网迅猛发展。而在随后的 5 年里，Netscape 和 Microsoft 两大软件巨头掀起了一场互联网浏览器之争，最后 Microsoft 以 Internet Explorer 完胜告终，极大地推动了互联网的发展，并把网络带到了成千上万的个人计算机的用户面前。

而就在 HTML 迅猛发展的 20 世纪 90 年代，CSS 也以各种形式应运而生。不同的浏览器结合各自 HTML 语法结构实现了很多不同样式的语言支持，用户可以使用这些样式语言来调节网页的显示方式。

最初的 HTML 版本只含有很少的显示属性，用户可以自己来决定网页应该怎样被显示，但随着 HTML 的成长，HTML 增加了很多显示功能。

1994 年，哈坤·利（Hakon Wium Lie）为 HTML 语言提出了 CSS 样式表的构想。伯特·波斯（Bert Bos）当时正在设计一个叫做 Argo 的浏览器，决定与 Hakon Wium Lie 一起合作设计 CSS，于是形成了 CSS 的最初版本。在芝加哥的一次会议上，Hakon Wium Lie 第一次正式提出了 CSS 的建议。

1995 年，Hakon Wium Lie 与 Bert Bos 一起再次展示这个建议。当时 W3C 刚刚建立，W3C 对 CSS 的发展很感兴趣，为此专门组织了一次讨论会，并于 1996 年 12 月推出了 CSS 规范的第一个版本。这一规范立即引起了各方的积极响应，随即 MicroSoft 公司和 Netscape 公司纷纷表示自己的浏览器能够支持 CSS 1.0。从此，CSS 技术在浏览器中的发展便一马平川。

1998 年，W3C 发布了 CSS 的第二个版本（即 CSS 2.0），这也是至今流行最广、目前主流浏览器都采用的标准。2001 年 5 月 W3C 开始进行 CSS3 标准的制定，到目前为止该标准还没有最终定稿，只是提出一个规范。

11.1.2 CSS 1.0 和 CSS 2.0 概述

如果没有出现 CSS，网页中文本的字体、颜色、大小将变得非常单调、暗淡无光，并且通过 HTML 标签来实现，代码非常烦琐。最终生成的 HTML 代码的长度一定臃肿不堪，下面分别对 CSS 1.0 和 CSS 2.0 进行一下了解。

Note

1. CSS 1.0 版本

1997 年 W3C 颁布 HTML4 标准的同时也发布了有关样式单的第一个标准 CSS 1.0。CSS 1.0 提供有关字体、颜色、位置和文本属性的基本信息，该版本已经得到了目前解析 HTML 和 XML 的浏览器的广泛支持。

CSS 1.0 较为全面地规定了文档的显示样式，大致可分为选择器、样式属性、伪类和对象等几个大的部分。

☑　选择器

要使用 CSS 对 HTML 页面中的元素实现一对一，一对多或者多对一的控制，就需要用到 CSS 选择器。CSS 选择器大致分为派生选择器、ID 选择器和类选择器几种，用来定义希望应用样式的 HTML 元素或者标签。

☑　样式属性

该属性主要包括 Font 字体、Text 文本、Background 背景、Position 定位、Dimensions 尺寸、Layout 布局、Margins 外边框、Border 边框、Padding 内边框、List 列表、Table 表格和 Scrollbar 滚动条等，用于定义网页的一些样式变化。每个属性都有一个值，属性和值被冒号分开并由花括号包围，这样就组成了一个完整的样式声明。

☑　伪类属性

一般情况下，样式的应用都需要用户指定应用样式的 HTML 节点，然后动态地指定节点（也就是说节点的样式需要动态变化），这就需要用伪类来完成。在 CSS 1.0 中主要定义了针对锚对象 a 的 link、hover、active、visited 和针对节点的 first-letter、first-child、first-line 等几个伪类属性。

☑　保存方式

通过 CSS 所编写的样式代码，用户可以直接存储在 HTML 网页中，也可以将 CSS 样式代码存储为独立的样式表文件。

不管以哪种方式保存，样式表都包含将样式应用于特定元素类型的样式规则。在外部使用时，样式表规则放在外部样式表文档中，文件扩展名为.css。

【示例 1】CSS 1.0 是对 HTML 语法的一次重大革新。以前的 HTML 版本中，各种样式功能的实现是通过 HTML 标记元素实现的，例如，在本示例中将一段文本中一部分文字变成红色，代码如下。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body>
<p><font color="red">这一段文字是红色！ </font></p>
</body>
</html>
```

以上代码在浏览器中的运行效果如图 11.1 所示，在标记之内的文字内容显示为红色。

图 11.1　HTML 方式设置文字颜色

【示例 2】使用 CSS 把这些 HTML 标记属性化，以上面示例为例，在 CSS 样式代码中可以重新定义上述代码。

Note

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body>
<p style="color:red">这一段文字是红色！ </p>
</body>
</html>
```

在浏览器中的运行效果如图 11.2 所示，所显示的 标记之内的文字为红色，但是本示例代码以 style 属性来设置文字的颜色。

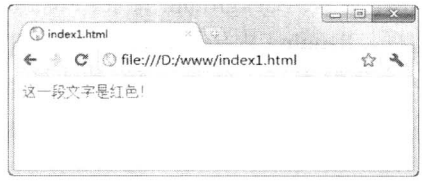


图 11.2　CSS 方式设置文字颜色

CSS 的引入使得文档的可读性被大大加强、文档的结构设计也更加灵活。用户在编写 HTML 代码时，可以自己决定文档的显示结构和样式。

由于各个浏览器厂商相互竞争，使浏览器之间出现相互不兼容的样式标记。CSS 的出现从一定程度上解决了 HTML 跨浏览器通用性的问题。

另外，随着后来 DHTML 的出现，开发人员可以使用一些 Script 标记语言来动态修改 HTML 节点的样式，这极大地增强了 CSS 的表现力。

2. CSS 2.0/CSS 2.1 版本

1998 年 W3C 发布了 CSS 的第二个版本，即 CSS 2.0 版本。CSS2 规范是基于 CSS1 设计的，其中包含了 CSS1 所有的功能，并扩充和改进了很多更加强大的属性。

☑ 选择器

CSS2 提供了更多强大的选择器，用来定位 HTML 节点或者标记，如表 11.1 所示。

表 11.1　不同字符在选择器中的含义

符 号 名 称	含　义	示　例
*（星号）	用于匹配任何标记	* {color:red}
>（大于号）	用于指定父子节点关系	AAA > CCC > DDD {color:red} 匹配父节点分别为 CCC 和 AAA 的节点 DDD 节点
□（空格）	用于匹配在某一层有某个父节点的节点	AAA CCC DDD {color:blue} 匹配包含父节点 CCC 的父节点 AAA 的节点 DDD
+（加号）	用于表示在同一个级别节点之间的关系	AAA + BBB {color:red}表示，有一个兄弟节点 AAA 的节点 BBB
名称[表达式]	选择包含特殊属性值的节点	BBB[text="xyz"] {color:blue} 表示包含 text 属性值为 xyz 的 BBB 标签

【示例 3】通过表 11.1，读者已经了解了 CSS2 中的一些选择器。本示例将演示如何使用选择器控制文档样式，同时了解相邻（即兄弟节点）选择器使用方法。

Note

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
/*添加兄弟节点选择器*/
li + li {
    font-weight: bold;
    font-family: "黑体";
    font-size: 14px;
    color: #0033CC;
}
</style>
</head>
<body>
<div>
    <ul>
        <li>第 1 章　学习 HTML5</li>
        <li>第 2 章　HTML5 标记</li>
        <li>第 3 章　HTML5 插入图片</li>
    </ul>
    <ol>
        <li>HTML 的发展</li>
        <li>HTML5 的新增功能</li>
        <li>如何学习 HTML5</li>
    </ol>
</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.3 所示。在显示的效果中，可以看到 标记中文本样式与 标记中文本样式的设置相同，因此，可以确定两个标记中的 标记所使用相同的样式设置。

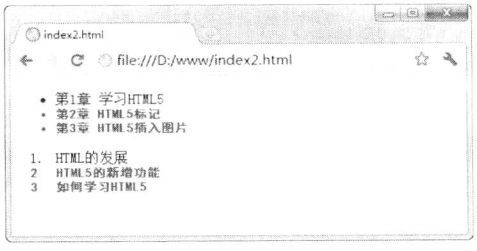


图 11.3　相邻标记应用相同样式

☑　位置模型

虽然在 CSS1 中已经定义了一些关于位置（Positioning）的属性，但在 CSS2 中进一步增强了这部分功能，如增加了 relative、absolute 和 fixed 等几个值，如表 11.2 所示。

表 11.2　CSS2 关于位置属性增加值

值　名　称	含　　义
relative	元素框相对父节点偏移某个距离，元素仍保持其未定位前的形状，它原本所占的空间仍保留
absolute	元素框从文档流中完全删除，并相对于其包含块定位。包含块可能是文档中的另一个元素或者是初始包含块。元素原先在正常文档流中所占的空间会关闭，就好像元素原来不存在一样
fixed	元素框的表现类似于 absolute 值，不过其包含块是视窗本身
z-index	用来指定相互重叠的元素的叠放顺序，数值较大的元素会叠放在数值较小的元素前面

Note

【示例 4】本示例演示了如何使用 CSS 来对文本内容进行定位。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
h2.pos_abs {
    position: absolute;
    left: 150px;
    top: 100px
}
</style>
</head>
<body>
<h2 class="pos_abs">该内容是通过定义后的位置</h2>
<p>通过绝对定位，元素可以放置到页面上的任何位置。下面的标题距离页面左侧 150px，距离页面顶部 100px。</p>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.4 所示。在显示的结果中，用户可以看到“该内容是通过定义后的位置”文本内容，已经显示到 标记中文本内容的下方，这说明已经通过样式对该段文本进行了定位操作。

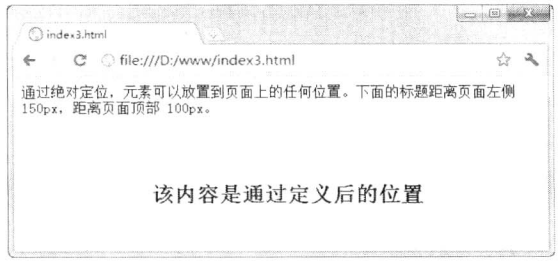


图 11.4　文本内容定位

在 CSS2 中，新的定位模型提供的这些属性，可以更加容易地建立列式及复杂布局，将布局的一部分与另一部分重叠，还可以完成之前需要使用多个表格才能完成的任务。

☑　布局、表格样式

display 属性用来规定元素应该生成的框架类型，但在 CSS1 中只有少数几个属性。CSS2 对该属性进行了许多扩充，用户可以用该属性指定元素是否会显示以及如何显示，也可以使用该属性配合位

置和浮动进行页面的布局。另外，用户还可以将一个非表格的结构化文档显示为一个表格样式。

☑　媒体类型

CSS2 引入了媒体类型，用于对不同的媒体类型定义不同的样式。可用的媒体类型如表 11.3 所示。

Note

表 11.3　CSS2 中媒体类型

属性名称	含　义	属性名称	含　义
aural	用于语音和音频合成器	braille	用于盲人用点字法触觉回馈设备
embossed	用于分页的盲人用点字法打印	handheld	用于小型手持的设备
print	用于打印机	projection	用于方案展示如幻灯片
screen	用于计算机显示器		

☑　伪类

在 CSS2 中，不但增加了:focus（将样式添加到被选中的元素）、:first-child（将特殊的样式添加到元素的第一个子元素）、:lang（允许创作者来定义指定的元素中使用的语言）等几个新的伪类，同时还扩充了伪类的使用范围，使得伪类不但可以和原来一样应用于 a 锚标记，还可以应用到一个类和标签上，如:link:hover、myClass:hover 等。

☑　光标样式

CSS2 的另一大亮点就是增加了 cursor 属性，用于指定设备应该显示怎样的光标类型。光标的可选类型如表 11.4 所示。

表 11.4　cursor 属性值

值　名　称	含　义
default	默认光标（通常是一个箭头）
auto	默认。浏览器设置的光标
crosshair	光标呈现为十字线
pointer	光标呈现为指示链接的指针（一只手）
move	此光标指示某对象可被移动
e-resize	此光标指示矩形框的边缘可被向右（东）移动
ne-resize	此光标指示矩形框的边缘可被向上及向右移动（北/东）
nw-resize	此光标指示矩形框的边缘可被向上及向左移动（北/西）
n-resize	此光标指示矩形框的边缘可被向上（北）移动
se-resize	此光标指示矩形框的边缘可被向下及向右移动（南/东）
sw-resize	此光标指示矩形框的边缘可被向下及向左移动（南/西）
s-resize	此光标指示矩形框的边缘可被向下移动（北/西）
w-resize	此光标指示矩形框的边缘可被向左移动（西）
text	此光标指示文本
wait	此光标指示程序正忙（通常是一只表或沙漏形状）
help	此光标指示可用的帮助（通常是一个问号或一个气球）

【示例 5】本示例演示了如何使用 CSS 的 cursor 属性设置不同的光标样式效果。

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

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<p>请把鼠标移动到单词上，可以看到鼠标指针发生变化：</p>
<div style="cursor:auto"> Auto 值，浏览器设置的光标。</div>
<div style="cursor:crosshair"> Crosshair 值，光标呈现为十字线。</div>
<div style="cursor:default"> Default 值，默认光标（通常是一个箭头）。</div>
<div style="cursor:pointer"> Pointer 值，光标呈现为指示链接的指针（一只手）</div>
<div style="cursor:move"> Move 值，光标指示某对象可被移动。</div>
<div style="cursor:e-resize"> e-resize 值，光标指示矩形框的边缘可被向右（东）移动。</div>
<div style="cursor:ne-resize"> ne-resize 值，光标指示矩形框的边缘可被向上及向右移动（北/东）。</div>
<div style="cursor:nw-resize"> nw-resize 值，光标指示矩形框的边缘可被向上及向左移动（北/西）。</div>
<div style="cursor:n-resize"> n-resize 值，光标指示矩形框的边缘可被向上（北）移动。</div>
<div style="cursor:se-resize"> se-resize 值，光标指示矩形框的边缘可被向下及向右移动（南/东）。</div>
<div style="cursor:sw-resize"> sw-resize 值，光标指示矩形框的边缘可被向下及向左移动（南/西）。</div>
<div style="cursor:s-resize"> s-resize 值，光标指示矩形框的边缘可被向下移动（北/西）。</div>
<div style="cursor:w-resize"> w-resize 值，光标指示矩形框的边缘可被向左移动（西）。</div>
<div style="cursor:text"> text 值，光标指示文本。</div>
<div style="cursor:wait"> wait 值，光标指示程序正忙（通常是一只表或沙漏）。</div>
<div style="cursor:help"> help 值，光标指示可用的帮助（通常是一个问号或一个气球）。</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.5 所示。将光标移至不同的文本之上，即可改变光标的样式，如图 11.6 所示。

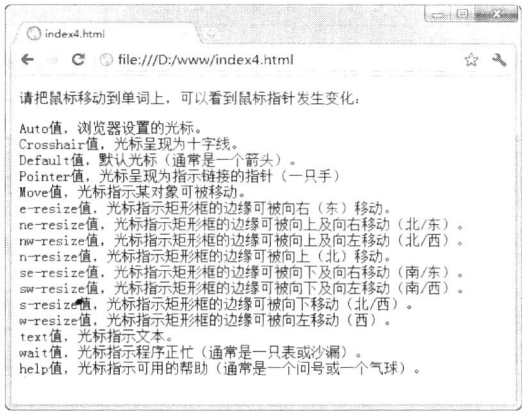


图 11.5　代码运行效果

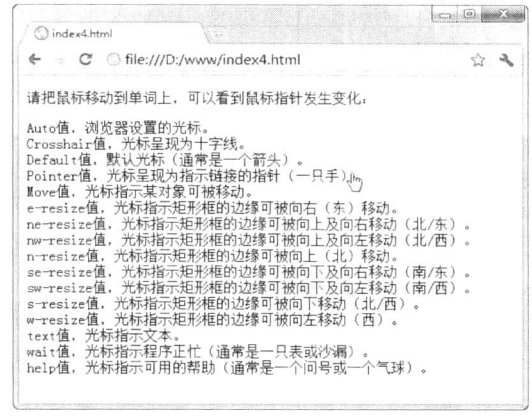


图 11.6　光标样式效果

在 CSS2 规范中，除了上述一些属性的增强外，还包含了一些其他属性，此处就不再一一列出了。这些新功能极大地增强了 CSS 和 HTML 的表现能力。总的来说，CSS 的优势及作用是显而易见的，主要体现在以下几方面：

- ☑ 提高页面浏览速度。
- ☑ 易于维护和改版。
- ☑ 使用 CSS 布局更符合现在的 W3C 标准。
- ☑ 表现和内容相分离。

一个普通的网页，通过给这个网页添加 CSS 规则，可以得到十分美观的页面。不用改动 HTML，

只是通过添加不同的 CSS 规则，用户就可以得到各种不同样式的网页。

11.1.3　CSS 与 DIV 标记之缘

Note

DIV+CSS 是 Web 设计标准，它是一种网页的布局方法。这一种网页布局方式与传统中通过表格（Table）布局定位方式，可实现网页页面内容与表现相分离。

提及 DIV+CSS 组合，还要从 XHTML 说起。XHTML 是一种在 HTML 基础上优化和改进的新语言，目的是基于 XML 应用与强大的数据转换能力，适应未来网络应用更多的需求。

因此，在 HTML 4.0 的基础上，用 XML 的规则对其进行扩展，得到了 XHTML。所以，建立 XHTML 的目的就是实现 HTML 向 XML 的过渡。目前，国际上在网站设计中推崇的 Web 标准就是基于 XHTML 的应用（即通常所说的“DIV+CSS”）。

“DIV+CSS”这种叫法其实是一种错误的叫法，而标准的叫法应该是 XHTML+CSS。因为 DIV 与 Table 都是 XHTML 或 HTML 语言中的一个标记，而 CSS 只是一种样式表现。DIV+CSS 布局与传统 Table 相比，具有以下特点：

☑　使页面载入得更快。

由于将大部分美化页面的代码都写在 CSS 文件中，使得页面体积容量变得更小。相对于表格嵌套的方式，将页面分隔成独立的更多的区域，在打开页面的时候逐层加载，而不是像表格嵌套那样将整个页面圈在一个大表格里，使得加载速度很慢。

☑　降低流量费用。

页面体积变小，浏览速度变快，这对于某些控制主机流量的网站来说是最大的优势了。

☑　修改设计时更有效率。

DIV+CSS 已经将内容与表现分隔开来，所以修改网页时只需对指定内容修改即可。例如，用户想修改页面的表现力，可以直接修改 CSS 代码，这样不会影响数据内容。而修改数据内容，则不会影响显示效果。

☑　保持视觉的一致性。

以往表格嵌套的制作方法，会使得页面与页面或者区域与区域之间的显示效果有所偏差，或者表格之间出现间隙。而使用 DIV+CSS 的制作方法，将所有页面或所有区域统一用 CSS 文件控制，就避免了不同区域或不同页面出现效果偏差。

☑　更好地被搜索引擎收录。

因为很多代码被写入到 CSS 中，网页页面中的大部分内容将被突出，而这恰好可以被搜索引擎采集收录，方便浏览用户对网端的搜索。

☑　对浏览者和浏览器更具亲和力。

由于 CSS 富含丰富的样式，使页面更具灵活性。它可以根据不同的浏览器或者版本，体现出不同的效果，也可以根据地区、时间、人物等不同条件，显示出不同的效果。

【示例 1】本示例演示了如何使用表格控制网页内容，使读者了解一下 Table+CSS 的布局方式。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>Table 布局</title>
<style type="text/css">
/*定义表格中的边框和字体样式*/
table {
    border: #000 solid 2px;
```

```
    font-size: 14px;
    font-family: "迷你简黑体";
}
/*定义单元格的边框样式*/
table tr td { border: #039 solid 1px; }
</style>
</head>
<body>
<table width="400" border="0" cellpadding="0" cellspacing="0">
    <tr>
        <th align="left">消费项目</th>
        <th align="right">一月</th>
        <th align="right">二月</th>
    </tr>
    <tr>
        <td align="left">衣服</td>
        <td align="right">241 元</td>
        <td align="right">150 元</td>
    </tr>
    <tr>
        <td align="left">化妆品</td>
        <td align="right">30 元</td>
        <td align="right">14 元</td>
    </tr>
    <tr>
        <td align="left">食物</td>
        <td align="right">730 元</td>
        <td align="right">650 元</td>
    </tr>
    <tr>
        <th align="left">总计</th>
        <th align="right">1001 元</th>
        <th align="right">814 元</th>
    </tr>
</table>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.7 所示。

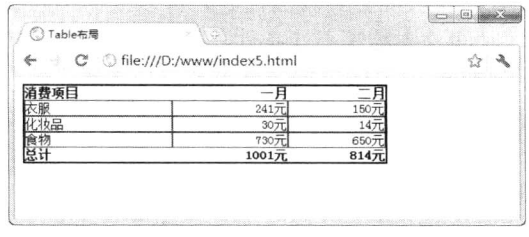


图 11.7　表格布局方式

【示例 2】以示例 1 为基础，下面是通过 DIV+CSS 方式来呈现一个表格的示例。

Note

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>Table 布局</title>
<style type="text/css">
/*定义表格的宽和字体样式*/
.table {
    width: 400px;
    font-size: 14px;
    font-family: "迷你简黑体";
}
/*定义行宽和对齐方式*/
.tr {
    width: 33.3%;
    float: left;
}
/*定义单元的边框样式*/
.td { border: #039 solid 1px; }
/*定义单元的边框样式和文本对齐方式*/
.td2 {
    border: #039 solid 1px;
    text-align: right;
}
</style>
</head>
<body>
<div class="table">
    <div class="tr">
        <div class="td">消费项目</div>
        <div class="td">衣服</div>
        <div class="td">化妆品</div>
        <div class="td">食物</div>
        <div class="td">总计</div>
    </div>
    <div class="tr">
        <div class="td2">一月</div>
        <div class="td2">241 元</div>
        <div class="td2">30 元</div>
        <div class="td2">730 元</div>
        <div class="td2">1001 元</div>
    </div>
    <div class="tr">
        <div class="td2">二月</div>
        <div class="td2">150 元</div>
        <div class="td2">14 元</div>
        <div class="td2">650 元</div>
        <div class="td2">814 元</div>
    </div>
</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.8 所示。通过显示的表格，可以看到该方式与 Table+CSS 方式呈现的效果大致相同，而 DIV 的应用方面更加方便。

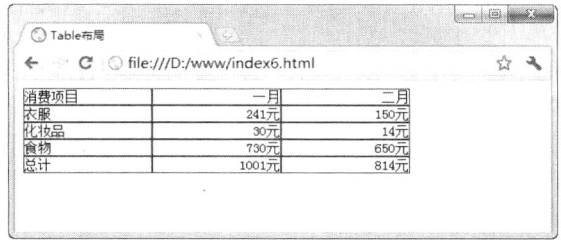


图 11.8　DIV+CSS 布局表格方式

11.1.4　CSS 编码规范

CSS 的编码规范是业内人士比较通用的一种 CSS 代码书写方法，这种规范对 CSS 代码并没有强制性的要求。按照 CSS 编码规范来书写 CSS，会使代码更加便于阅读。使用基本相同的 CSS 编码规范书写 CSS，可以方便其他人的使用。基本的 CSS 代码书写规范如下：

- ☑ 尽量不缩写，除非一看就明白的单词。
- ☑ 尽量统一用英文、英文简写或者统一使用拼音。
- ☑ 在对 CSS 代码进行注释时，可以使用/**/（斜杠和星号）或者使用<!-- -->进行注释。其中，/**/（斜杠和星号）进行多行注释时使用，而<!-- -->用于进行单行注释。
- ☑ 浏览器不同或者版本不同，则 CSS 代码的要求不同，一定要符合浏览器兼容标准。
- ☑ 全部小写，且每一项 CSS 定义写成一行。
- ☑ 一般不允许将样式的定义写在标签中（如 style="font-color:red;"）。
- ☑ 所有装饰性的图片（如背景图），要定义在 CSS 中。
- ☑ ID 必须是唯一的，且用在结构的定义中。
- ☑ CSS 代码命名不建议使用中杠连接符。
- ☑ 建议单位使用相对长度单位。如 em 和%。
- ☑ CSS 中谨慎使用 expression，尽量不要使用。
- ☑ CSS 中尽量不要使用滤镜，因为滤镜会影响代码读取速度和兼容性。

在设计 Web 页面时，尤其需要对 DIV 标记进行标识的时候，最直接方法是使用可以描述元素所在页面位置的词汇来对其命名。这种方法使用户阅读代码时，直观了解 Class 和 ID 对页面块定义的位置。因此，常见的 CSS 命令的名称如表 11.5 所示。

表 11.5　CSS 常见命令规则

名　称	含　义	名　称	含　义
header	页头	content/container	内容
footer	页脚	nav	导航
sidebar	侧栏	column	栏目
wrapper 或 wrap	页面外围控制整体布局宽度	left	左侧
right	右侧	center	中间
loginbar	登录条	logo	标志
banner	广告	main	页面主体

续表

名　称	含　义	名　称	含　义
hot	热点	news	新闻
download	下载	subnav	子导航
menu	菜单	submenu	子菜单
search	搜索	friendlink	友情链接
copyright	版权	scroll	滚动
content	内容	tab	标签页
list	文章列表	msg	提示信息
title	栏目标题	joinus	加入
guild	指南	service	服务
regsiter	注册	status	状态
vote	投票	partner	合作伙伴

Note

11.2 了解 CSS3 新增特性

早在 2001 年，W3C 就着手开始准备开发 CSS 第三版规范。CSS3 规范一个新的特点是规范被分为若干个相互独立的模块。一方面分成若干较小的模块较利于规范及时更新和发布，及时调整模块的内容。另外一方面，由于受支持设备和浏览器厂商的限制，设备或者厂商可以有选择地支持一部分模块，支持 CSS3 的一个子集。

CSS3 使很多以前需要使用图片和脚本来实现的效果，只需要短短几行代码就能完成，如圆角、图片边框、文字阴影和盒阴影等。CSS3 不仅能简化前端开发人员的设计过程，还能加快页面的载入速度。

目前，主流浏览器都已迫不及待地开始支持 CSS3 的部分特性了。虽然这部分特性还相对较少，但是这些以前很难使用 JavaScript 和图片实现的效果依然令人兴奋不已。下面就简单介绍一下主流浏览器实现的一些 CSS3 新特性。

11.2.1 属性选择器

在 CSS3 中，新添加了 3 个属性选择器：[att^="value"]、[att$="value"]、[att*="value"]，用来匹配属性中包含的某些特定的值，如表 11.6 所示。属性选择器是基于属性来定位一个元素。可以只指定该元素的某个属性，这样所有使用该属性的元素都将被定位。

表 11.6　CSS3 新增属性选择器

属 性 名 称	含　义
[att^="value"]	该属性的值以指定的值开始
[att$="value"]	该属性的值以指定的值结束
[att*="value"]	该属性的值包含指定的值，而无论其位置

【示例 1】下面通过一个示例，对两个 标记添加不同属性值，然后使用[att$="value"]属性选择器设置文本的变化。

Note

```
<!DOCTYPE html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>属性选择器</title>
<style type="text/css">
/*指定<div>标记 class 属性名称开始的内容样式*/
div[class$="title"] {
    /*定义文本颜色*/
    color: #30F;
    /*定义字体和文字大小*/
    font-family: "迷你简黑体";
    font-size: 16px;
}
</style>
</head>
<body>
<div class="title">该属性的值包含指定的值（而无论其位置）。</div>
<div class="ti">除了 Internet Explorer 6，所有的主流浏览器都支持属性选择器。</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.9 所示。从显示效果来看，可以确定 class="title" 将符合 div[class$="title"]属性选择器的要求，所以该 标记中的内容将应用该属性选择器中的样式。而 class="ti"不符合属性选择器的要求，所以其文本没有发生改变。

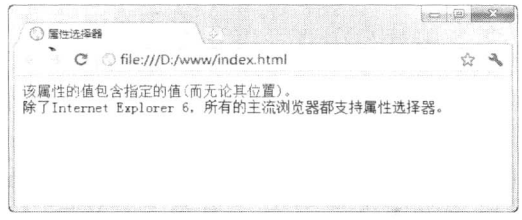


图 11.9　属性选择器

CSS3 中唯一新引入的连字符是通用的兄弟选择器（同级）。它针对一个元素有相同父级节点的所有兄弟级别元素。例如，给某个特定的 标记的同级的图片添加一个灰色的边框（ 标记和图片应该有相同的父级节点）。

```
div~img {
    //add your property here
}
```

目前，除了 IE 6.0 浏览器，其他主流浏览器都支持上述选择器。

【示例 2】下面通过示例在网页中添加两张图片，并使用同级选择器设置边框效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
<style type="text/css">
```

Note

```
img { height:200px;}
/*定义兄弟标记样式*/
span~img { /*定义边框样式*/
    border: 10px solid #ccc;
}
</style>
</head>
<body>
<span><img src="images/bg.jpg"/></span>
<img src="images/bg1.jpg"/>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.10 所示。通过显示效果，可以了解到与标记同级的<img>标记中的图片添加了边框效果。

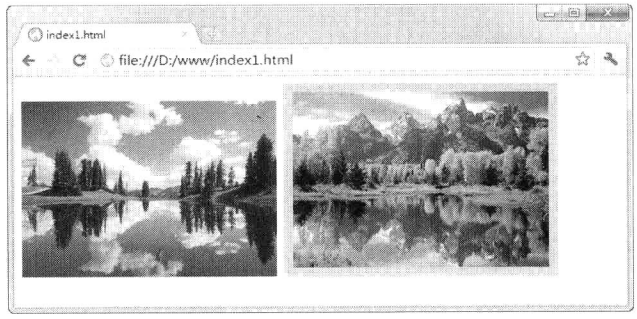


图 11.10　添加边框

11.2.2　RBGA 透明度

对经常处理图片的用户来说，RGB（红色 R+绿色 G+蓝色 B）并不陌生。而 CSS3 中的 RGBA 是什么呢？其实，这就是在 RGB 的基础上加进了一个通道 Alpha。

当设定一个 RGBA 色彩的时候，参数依次设定为红、蓝、和绿色的颜色值，可以是 0~255 或百分数。最后一个 Alpha 参数为透明值，应该在 0~1 之间。

【示例】本示例演示了如何使用 RBGA 方法设置透明度的效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
<style type="text/css">
/*定义单元样式的外观及文本样式*/
li.rgba {
    float: left;
    width: 100px;
    height: 80px;
    border: #CCC solid 1px;
    text-align: center;
    font-size: 12px;
```

```
}
/*定义背景颜色效果*/
li.rgba1 { background: rgba(255，0，0，1); }
li.rgba2 { background: rgba(255，0，0，0.8); }
li.rgba3 { background: rgba(255，0，0，0.6); }
li.rgba4 { background: rgba(255，0，0，0.4); }
li.rgba5 { background: rgba(255，0，0，0.2); }
li.rgba6 { background: rgba(255，0，0，0); }
</style>
</head>
<body>
<div class="example-opacity">
    <p>CSS3 的 RGBA 效果</p>
    <li class="rgba rgba1">Alpha 参数为 1</li>
    <li class="rgba rgba2">Alpha 参数为 0.8</li>
    <li class="rgba rgba3">Alpha 参数为 0.6</li>
    <li class="rgba rgba4">Alpha 参数为 0.4</li>
    <li class="rgba rgba5">Alpha 参数为 0.2</li>
    <li class="rgba rgba6">Alpha 参数为 0</li>
</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.11 所示。用户通过 RGBA 不仅可以设定色彩，还能设定元素的透明度。

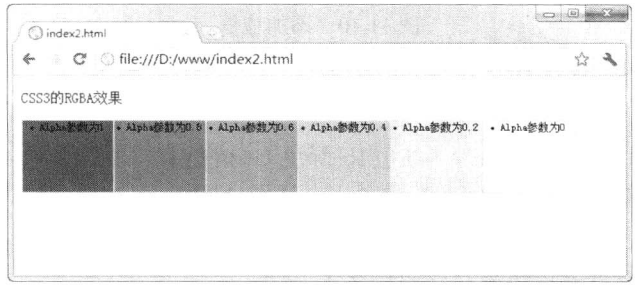


图 11.11　设置透明度

RGBA 和 Opacity 之间的不同点是前者只会应用到指定的元素上，而后者会影响指定的元素及其子元素。目前，支持 RBGA 颜色的浏览器有 Weskit 核心系列浏览器、Firefox 3+和 Opera 9.5+浏览器，而 IE 全系列浏览器都不支持。

11.2.3　多栏布局

在 CSS3 版本之前，如果要设置双栏、三栏排列的文本内容，其设置相当麻烦。用户不仅需要先创建一个容器，还得分别设置文本内容的位置、高度等。

而 CSS3 选择器不需要使用多个 标记就能实现多栏布局。浏览器解释这个属性并生成多栏。用户通过选择器定义 3 部分内容，如栏数（column-count）、栏宽（column-width 例子中没有用到）和各栏之间的空白和间距（column-gap）。如果 column-count 未设定，浏览器会在允许的宽度内容纳尽可能多的栏目。有了这个新的多栏布局样式，用户可以给文本呈现类似“报纸”排版的效果。

提示：目前 CSS3 多数属性只支持 Webkit 核心系列浏览器和 Firefox 2+浏览器，在编写 CSS3 代码时，需要在属性前添加“-webkit”前缀。对于 Firefox 浏览器等，需要在代码之前添加“-moz”前缀。Webkit 是开源的 Web 浏览器引擎，苹果的 Safari、谷歌的 Chrome 浏览器都是基于这个框架来开发的。

Note

【示例】本示例通过上述属性实现 3 栏效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
<style type="text/css">
/*定义<div>标记容器的样式*/
#bk {
    font-family: Arial，Helvetica，sans-serif;
    font-size: 12px;
    background-color: #CC9;
    border: #F60 solid 2px;
}
/*定义分栏的样式*/
#my_CSS3_id {
    margin: 3px 3px 3px 3px;
    text-align: justify;
    /*定义分栏的总数、栏之间的间距、分隔线等*/
    -webkit-column-count: 3;
    -webkit-column-gap: 1.5em;
    -webkit-column-rule: 1px solid #F00;
}
</style>
</head>
<body>
<div id="bk">
    <div id="my_CSS3_id">
        <div>“总之，这幸福的家庭一定须在 A，无可磋商。家庭中自然是两夫妇，就是主人和主妇，自由结婚的。他们订有四十多条条约，非常详细，所以非常平等，十分自由。而且受过高等教育，优美高尚……。东洋留学生已经不通行，——那么，假定为西洋留学生罢。主人始终穿洋服，硬领始终雪白；主妇是前头的头发始终烫得蓬蓬松松像一个麻雀窠，牙齿是始终雪白的露着，但衣服却是中国装，……”</div>
        <div>“不行不行，那不行！二十五斤！”他听得窗外一个男人的声音，不由的回过头去看，窗幔垂着，日光照着，明得眩目，他的眼睛昏花了；接着是小木片撒在地上的声响。“不相干，”他又回过头来想，“什么‘二十五斤’？——他们是优美高尚，很爱文艺的。但因为都从小生长在幸福里，所以不爱俄国的小说……。俄国小说多描写下等人，实在和这样的家庭也不合。‘二十五斤’？不管他。那么，他们看看什么书呢？——裴伦的诗？吉支〔7〕的？不行，都不稳当。——哦，有了，他们都爱看《理想之良人》〔8〕。我虽然没有见过这部书，但既然连大学教授也那么称赞他，想来他们也一定都爱看，你也看，我也看，——他们一人一本，这家庭里一共有两本，……”他觉得胃里有点空虚了，放下笔，用两只手支着头，教自己的头像地球仪似的在两个柱子间挂着。</div>
    </div>
</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.12 所示。用户通过浏览器所显示的效果，可以非常清楚看到，两个 标记被分为 3 栏效果，并且两栏之间用竖线隔开。

图 11.12　分栏效果

11.2.4　多背景图片

为了吸引浏览者，提高访问量，许多设计人员都想在网页中加入一些背景图片或特效效果。而在 CSS3 样式表中，用户可以使用多个属性，如 background-image、background-repeat,、background-size、background-position、background-originand 和 background-clip 等，在一个元素上添加多层背景图片。

【示例】本示例就来简单介绍一下在一个 标记中添加 4 张不同的图片的方法。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
<style type="text/css">
/*定义边框样式和背景图片*/
#my_CSS3_id {
    border: 5px solid #9e9aab;
    background: url(images/a.jpg) top left no-repeat，url(images/b.jpg) bottom left no-repeat，url(images/c.jpg) top right repeat-y，url(images/d.jpg) bottom  right repeat-x;
    width: 600px;
    height: 300px;
}
</style>
</head>
<body>
<div id="my_CSS3_id"></div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.13 所示。用户通过浏览器所显示的效果，可以看到两张图片分别位于左上角和左下角，而另外两张图片位于右上角和右下角，分别沿 y 轴和 x 轴重复平铺。

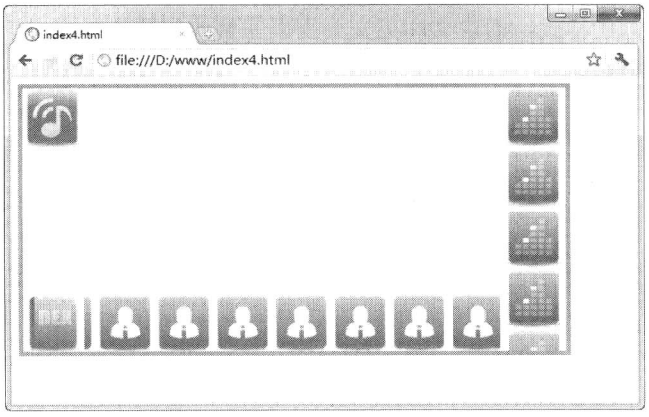


图 11.13 添加多图片背景

11.2.5 字符串溢出

word-wrap 属性用于防止太长的字符串溢出，其中包含 normal 和 break-word 两个属性值。normal 值（默认的）只在允许的断点截断文字，如连字符。

【示例】本示例通过 3 个 标记中的内容，来查看中/英文状态下，字符串溢出的效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
<style type="text/css">
/*定义 break-word 值的字符串溢出样式*/
.c2 {
    width: 300px;
    word-wrap: break-word;
    border: 1px solid #F00;
}
/*定义 break-word 值和 break-all 值的字符串溢出样式*/
.c3 {
    width: 300px;
    word-wrap: break-word;
    word-break: break-all;
    border: 1px solid #F90;
}
</style>
</head>
<body>
<div class=c2>This is all English. This is all English. This is all English.</div>
<div class=c2>全是中文的情况。全是中文的情况。全是中文的情况。</div>
<div class=c3>中英文混排的情况。Chinese and English. 中英文混排的情况。Chinese and English.</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.14 所示。用户通过浏览器中的内容，可以看到

前两个框中所使用的相同的样式，而中文和英文分别以完整的字符内容溢出。在第 3 个框中应用 c3 样式，则以边框的边界溢出。

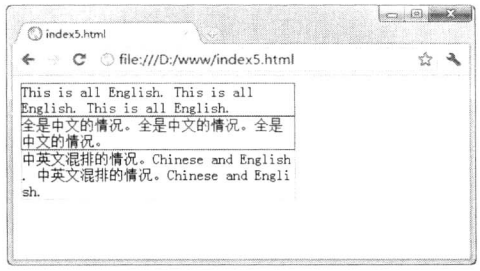


图 11.14　字符溢出

如果使用了 break-word 值，文字可以在任何需要的地方截断以匹配分配的空间并防止溢出。使用 break-word 时，是将强制换行，中文没有任何问题，英文语句也没问题，但是对于长串的英文，就不起作用了。break-all 是断开单词，在单词到边界时，下个字母自移动到下一行。它主要解决了长串英文的问题，而不用 word-wrap，中文就不会换行了（英文语句正常）。

11.2.6　块阴影与圆角阴影

box-shadow 和 text-shadow 是两个未被广泛应用的 CSS 属性，它们将在 CSS3 中被广泛采用。这两个属性在浏览器工具中，添加一个维度以使文字和边框更加醒目、具有立体感。这样，用户就可以不再需要 Photoshop 去处理很多复杂零碎的图片了。块阴影的效果可以伴随前面讲到的圆角效果产生圆角阴影。

【示例】本示例分别给图片、 标记和文本文字添加阴影效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
<style type="text/css">
/*定义图片的阴影效果样式*/
img {
    -webkit-box-shadow: 2px 2px 10px #06C;
    height: 240px;
}
/*定义<div>标记的阴影效果样式*/
div {
    padding: 5px;
    margin: 5px;
    height: 50px;
    width: 150px;
    background-color: #CCC;
    -webkit-box-shadow: 2px 2px 20px #06C;
}
/*定义文本的阴影效果样式*/
p { text-shadow: 10px 10px 20px #06C; }
</style>
```

```
</head>
<body>
<img src="images/bg.jpg"/>
<div></div>
<p>添加阴影效果</p>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行效果如图 11.15 所示。用户通过浏览器中的内容，可以看图片、 标记和文本所添加的阴影效果。

图 11.15　添加阴影效果

11.2.7　圆角

CSS3 新功能中最常用的一项就是圆角效果，border-radius 属性无需背景图片就能给 HTML 元素添加圆角。这个方案是清晰、有效的，而且可以让用户免于花费很多时间寻找精巧的浏览器方案和基于脚本或图片去实现圆角效果。

【示例】本示例使用 border-radius 属性实现一个 标记框上两个角的圆角效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
<style type="text/css">
/*定义<div>标记的大小和圆角样式*/
#circular_bead {
      width: 300px;
      height: 250px;
      border: #999 solid 2px;
      -webkit-border-top-right-radius: 20px;
      -webkit-border-top-left-radius: 20px;
}
</style>
</head>
```

```
<body>
<div id="circular_bead"></div>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行效果如图 11.16 所示。用户通过浏览器中的内容，可以看图片、 标记和文本所添加的阴影效果。

图 11.16　圆角效果

11.2.8　边框图片

border-image 属性，看起来很像 background-image，它们确实有些类似，但也有很大的区别，通过 border-image 可以定义设置 border 的背景图，允许用户在元素的边框上设定图片。border-image 属性的基本用法如下：

border-image: none | <image> [<number> | <percentage>]{1,4} [/ <border-width>{1,4}]? [stretch | repeat | round]{0,2}

在表 11.7 中，描述了上述各属性值的含义。

表 11.7　border-image 属性值

属　性　值	含　义
none	默认值。无背景图
<image>	使用绝对或相对 url 地址指定背景图像
<number>	边框宽度用固定像素值表示
<percentage>	边框宽度用百分比表示
[stretch | repeat | round]	描述了拉伸/重复/平铺效果，其中 stretch 是默认值

【示例】本示例是一个简单的添加一张相同图片，而实现 标记边框的图片效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>添加边框图片</title>
<style type="text/css">
/*定义边框图片效果样式*/
```

```
div {
    border: double green 12px;
    height:200px;
    -webkit-border-image: url("images/b.jpg") 124;
}
</style>
</head>
<body>
<div>在 Chrome 浏览器里能看到边框背景图</div>
</body>
</html>
```

Note

以上代码在 Chrome 浏览器中的运行效果如图 11.17 所示。用户通过浏览器中的效果，可以看到由于框较宽，所以上、下边框的图片被拉伸，而左、右边框和 4 个角的图片大小为原始图片。

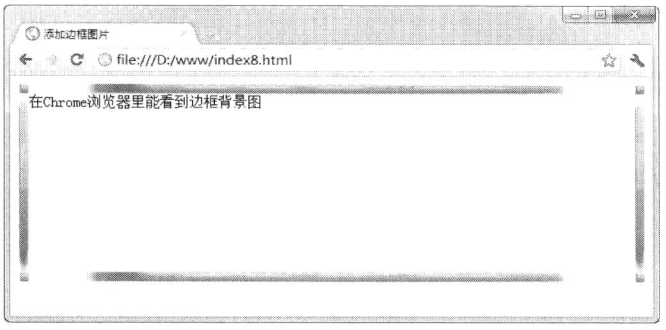


图 11.17　边框图片效果演示

11.2.9　形变

在 Web 中，想要实现文字和图片的施加或者形变，一般需要通过 Photoshop 或者 Flash 实现。而在 CSS3 中，用户可以通过 rotate 属性实现旋转、scale 属性实现伸缩，skew 属性实现倾斜等效果。rotate 属性的基本语法如下所示。

```
transform : none | <transform-function> [ <transform-function> ]
```

在表 11.8 中，描述了上述各属性值的含义。

表 11.8　transform 属性值

属　性　值	含　义
none	指定一个身份转变
matrix(<number>, <number>, <number>, <number>, <number>, <number>)	以一个含 6 值的（a,b,c,d,e,f）变换矩阵的形式指定一个 2D 变换，相当于直接应用一个[a b c d e f]变换矩阵
translate(<translation-value>[, <translation-value>])	通过矢量[tx,ty]指定一个 2D translation，tx 是第一个过渡值参数，ty 是第二个过渡值参数选项。如果<ty>未被提供，则 ty 以 0 作为其值
translateX(<translation-value>)	通过给定一个 X 方向上的数目指定一个 translation
translateY(<translation-value>)	通过给定 Y 方向的数目指定一个 translation
scale(<number>[, <number>])	提供执行[sx,sy]缩放矢量的两个参数指定一个 2D scale（2D 缩放）。如果第二个参数未提供，则取与第一个参数一样的值

续表

属 性 值	含 义
scaleX(<number>)	使用[sx,1]缩放矢量执行缩放操作，sx 为所需参数
scaleY(<number>)	使用[i,sy]缩放矢量执行缩放操作，sy 为所需参数
rotate(<angle>)	通过指定的角度参数对原元素指定一个 2D rotation（2D 旋转），需先有 transform-origin 属性的定义
skewX(<angle>)	按给定的角度沿 X 轴指定一个 skew transformation（斜切变换）
skewY(<angle>)	按给定的角度沿 Y 轴指定一个 skew transformation（斜切变换）
skew(<angle> [, <angle>])	X 轴 Y 轴上的 skew transformation（斜切变换）。第一个参数对应 X 轴，第二个参数对应 Y 轴。如果第二个参数未提供，则值为 0，也就是 Y 轴方向上无斜切

Note

【示例】本示例中将简单地添加同一张图片，而实现 标记边框的图片效果。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title> </title>
<style type="text/css">
.text-list {
    width: 500px;
    height: 460px;
    padding-top: 15px;
    border:solid 1px #CCCCCC;
    background: #fff;
    box-shadow: 5px 5px 6px #000;
}
.text-list li { line-height: 28px; }
.ul-one { -webkit-transform: rotate(5deg); }
/*针对 ul 设置，则整体都会有所倾斜*/
.one { -webkit-transform: rotate(5deg) scale(1.00，1.50); }
/*设置倾斜和缩放*/
.two a { -webkit-transform: rotate(3deg); } /*设置 li 的倾斜*/
.three { -webkit-transform: rotate(1deg); }
.three span {
    color: #f00;
    -webkit-transform: rotate(-5deg);
}
/*内联元素的倾斜度*/
.four a { -webkit-transform: skew(7deg，8deg); }
.p-con {
    font-weight: bold;
    color: #f00;
    -webkit-transform: rotate(15deg) scale(1.5，1.5) skew(5deg，8deg) translate(5px，-2px);
    -webkit-transform-origin: 0 5px;
}
</style>
```

Note

```
</head>
<body>
<div class="text-list">
    <ul class="ul-one">
        <li>li 的列表</li>
        <li><a href="#">缩放大小比例</a></li>
        <li><a href="#">旋转角度</a><span>缩放大小比例</span></li>
        <li><a href="#"><span>倾斜比例</span></a></li>
    </ul>
    <ul>
        <li class="one">设置字体的倾斜度 rotate(5deg)和缩放比例 scale(1.00，1.50); </li>
        <li class="two"><a href="#">倾斜度</a></li>
        <li class="three"><a href="#">这个外层元素已经设倾斜了</a><span>对 span 有设置了倾斜 两者没
有影响</span></li>
        <li class="four"><a href="#">字体的斜切变换</a></li>
    </ul>
    <p class="p-con">transform 的各个属性的用法</p>
</div>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.18 所示。用户通过浏览器中的效果，可以看到通过添加不同的样式，则可以实现不同的文字变形效果。

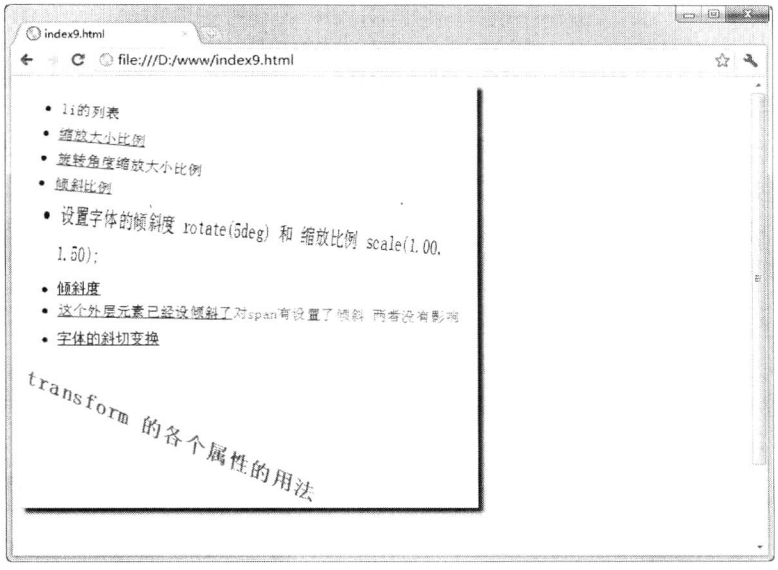


图 11.18　文本变形

11.3　CSS3 前景展望

号称“CSS 之父”的 Hakon Wium Lie 认为，除了 HTML5 外，CSS3 将是互联网发展的另一个趋势。开发 CSS3 这种样式表的最主要原因，是让用户能更加快速地访问互联网。

在 CSS 样式下，不需要图片，仅用文字即可。这让网页的文字在变得越来越漂亮的同时，不会影响浏览速度，并且还能更容易被搜索引擎搜索。程序人员不需要再基于操作系统底层代码编写程序，只要基于网络应用编写程序即可。

Note

11.3.1 CSS3 的应用范围

对于前端设计人员和开发人员来说，CSS 一直是 Web 设计过程中重要的一部分。而 CSS 的应用脱离了 Web 页面（即在 Web 页面中对不同页面组成部分的应用），以达到预想的效果。

随着 CSS3 的出现，以及浏览器对它的支持，可以实现各种各样酷似 Flash 动画的效果。下面列举几个常用网页设计，CSS3 可以实现的效果。

1. 按钮效果

一直以来，都是图片或是 JavaScript 用来创建网页中的按钮，但随着 CSS3 出现，用户可以创建一些大小、颜色不同元素的按钮，而不必每次都准备一个背景图片，如图 11.19 所示。

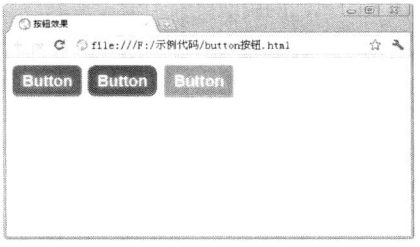


图 11.19　CSS3 样式的按钮

2. 制作柱形图

在 Web 页面上显示柱形、饼形等统计图表，构建直观的页面展示效果，通过 CSS3 样式，可以在 HTML5 中轻松地实现 3D 效果的柱形图，如图 11.20 所示。Ben Lister 选列了很多实用的 CSS3 技巧，包括这个看起来相当不错的 3D 柱形图。通过使用-webkit-transform 或-moz-transform 属性可以设置偏移效果。

3. 设计下拉菜单

Web 中的导航菜单，对一个网站来说是必不可少的组成部分。通过 CSS3 样式，可以设计出更漂亮、更具有动感效果的菜单。用户可通过 WebDesignerWall 的 Nick La 创建一个漂亮的 CSS3 弹出式下拉菜单，如图 11.21 所示。

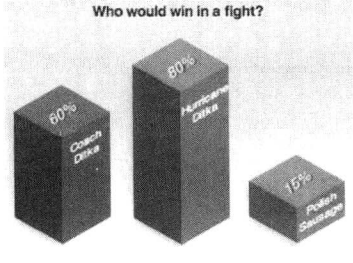


图 11.20　制作柱形图

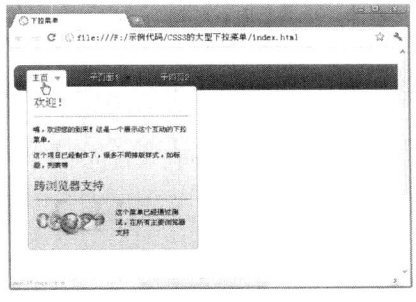


图 11.21　制作菜单

4. 图片阴影效果

从上述的新增特性中已经了解图片可以添加阴影效果，但除此之外，用户还可以在图片上添加其他的样式。例如，制作图片浏览器、产品图片展示等，可以在图片上添加边框和阴影效果，如图 11.22 所示。

Note

5. 文本框样式

表单中的对象总是给浏览者一种单调与沉闷的感觉，如按钮、文本框等。它们以一成不变的样式与颜色出现在网页中，或多或少都会破坏网页的整体美观程度。因此，通过 CSS3 代码可以改变表单对象的样式及激活状态的样式等，如图 11.23 所示。

图 11.22　添加图片边框和阴影效果

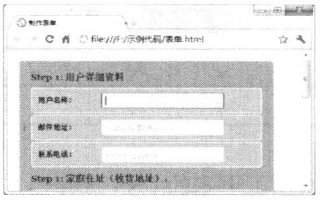


图 11.23　表单对象样式

6. 字体效果

过去需要 JavaScript 才能实现的复杂效果，现在使用 CSS3 即可简单地实现。例如，下面为通过 CSS 即可实现 3D 效果的文本内容，如图 11.24 所示。

7. CSS3 引用气泡

在 CSS3 之前，如果要实现对话气泡效果非常麻烦，需通过图片、JavaScript 等多种技术组合来实现。而 CSS3 的出现，使这一切变得相当简单，如图 11.25 所示。

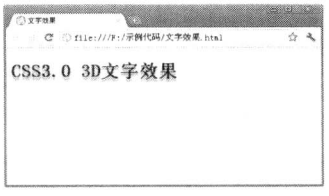


图 11.24　实现 3D 文本效果

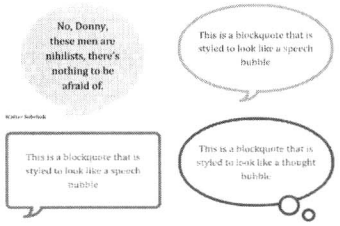


图 11.25　网页气泡

11.3.2　当前支持 CSS3 的浏览器

目前，支持 CSS3 和 HTML5 的浏览器越来越多，所谓支持仅仅是部分支持，因为 CSS3 和 HTML5 的 W3C 规范都尚未形成。如果用户现在使用 CSS3 和 HTML5 创建站点，至少要对各个浏览器对这两种新技术的支持情况有一个全面了解。

下面分别来介绍一下 CSS3 属性和 CSS3 选择器对浏览器的支持情况。一般支持 CSS3 属性的浏览器有 Chrome 和 Safari，而且不管是 Mac 平台还是 Windows 平台全支持，如表 11.9 所示。

表 11.9　浏览器对 CSS3 属性的支持

属　性	Windows							MAC		
	FIREFOX		SAFARI	IE		CHROME	OPERA	FIREFOX	SAFARI	OPERA
	3.6	4	5	9	10	10	11.1	4	5	11.1
Animations	×	×	√	×	×	√	×	×	√	×
Columns	√	√	√	×	√	√	√	√	√	√
Gradients	×	√	√	×	√	√	√	√	√	√
Reflections	×	×	√	×	×	√	×	×	√	×
Transforms	√	√	√	√	√	√	√	√	√	√
Transforms 3D	×	×	×	×	×	×	×	×	×	×
Transitions	×	√	√	×	×	√	√	√	√	√
FfontFace	√	√	√	√	√	√	√	√	√	√
FlexBox	√	√	√	×	√	√	×	√	√	×

CSS3 增加了更多的 CSS 选择器，可以实现更简单但是更强大的功能。除了 IE 家族和 Firefox 3 浏览支持外，几乎其他浏览器（如 Chrome、Safari、Firefox 3.6 和 Opera 10.5）全部支持，如表 11.10 所示。

表 11.10　CSS3 选择器与浏览器

属　性	MAC				Windows								
	CHROME	FIREFOX	OPERA	SAFARI	CHROME	FIREFOX		OPERA		SAFARI	IE		
	5	3.6	10	4	4	3.6	3	10	10.5	4	6	7	8
Begins with	√	√	√	√	√	√	√	√	√	√	×	√	√
Ends with	√	√	√	√	√	√	√	√	√	√	×	√	√
Matches	√	√	√	√	√	√	√	√	√	√	×	√	√
Root	√	√	√	√	√	√	√	√	√	√	×	×	×
Nth-child	√	√	√	√	√	√	×	√	√	√	×	×	×
Nth-last-child	√	√	√	√	√	√	×	√	√	√	×	×	×
Nth-of-type	√	√	√	√	√	√	×	√	√	√	×	×	×
Nth-last-of-type	√	√	√	√	√	√	×	√	√	√	×	×	×
Last-child	√	√	√	√	√	√	√	√	√	√	×	×	×
First-of-type	√	√	√	√	√	√	×	√	√	√	×	×	×
Last-of-type	√	√	√	√	√	√	×	√	√	√	×	×	×
Only-child	√	√	√	√	√	√	√	√	√	√	×	×	×
Only-of-type	√	√	√	√	√	√	×	√	√	√	×	×	×
Empty	√	√	√	√	√	√	√	√	√	√	×	×	×
Target	√	√	√	√	√	√	√	√	√	√	×	×	×
Enabled	√	√	√	√	√	√	√	√	√	√	×	×	×
Disabled	√	√	√	√	√	√	√	√	√	√	×	×	×
Checked	√	√	×	√	√	√	√	×	√	√	×	×	×
Not	√	√	√	√	√	√	√	√	√	√	×	×	×
General sibling	√	√	√	√	√	√	√	√	√	√	×	√	√

注：在表 11.9 和 11.10 表中，“对号”（√）表示浏览器支持该属性或者选择器。而“错号”（×）表示浏览器不支持该选择器。

11.4 案例实战：设计漂亮的表单

Note

在网页中，表单的作用不可小视，主要负责数据采集，比如可以采集访问者的名字和 E-mail 地址、调查表、留言簿等。精心设计的表单，能让用户感到心情舒畅，从而愉快地注册、付款或进行内容创建和管理，这是促成网上商业成功的秘密武器。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>制作表单</title>
<style type="text/css">
/*分别定义 HTML 中和标记中的距离样式*/
html，body，h1，form，fieldset，legend，ol，li {
    margin: 0;
    padding: 0;
}
/*定义<body>标记样式*/
body {
    background: #ffffff;
    color: #111111;
    font-family: Georgia，"Times New Roman"，Times，serif;
    padding: 20px;
}
/*定义付费内容的样式*/
form#payment {
    background: #9cbc2c;
    -webkit-border-radius: 5px;
    border-radius: 5px;
    padding: 20px;
    width: 400px;
}
form#payment fieldset {
    border: none;
    margin-bottom: 10px;
}
form#payment fieldset:last-of-type { margin-bottom: 0; }
form#payment legend {
    color: #384313;
    font-size: 16px;
    font-weight: bold;
    padding-bottom: 10px;
    text-shadow: 0 1px 1px #c0d576;
}
```

Note

```
form#payment > fieldset > legend:before {
    content: "Step " counter(fieldsets) ": ";
    counter-increment: fieldsets;
}
form#payment fieldset fieldset legend {
    color: #111111;
    font-size: 13px;
    font-weight: normal;
    padding-bottom: 0;
}
form#payment ol li {
    background: #b9cf6a;
    background: rgba(255，255，255，.3);
    border-color: #e3ebc3;
    border-color: rgba(255，255，255，.6);
    border-style: solid;
    border-width: 2px;
    -webkit-border-radius: 5px;
    line-height: 30px;
    list-style: none;
    padding: 5px 10px;
    margin-bottom: 2px;
}
form#payment ol ol li {
    background: none;
    border: none;
    float: left;
}
form#payment label {
    float: left;
    font-size: 13px;
    width: 110px;
}
form#payment fieldset fieldset label {
    background: none no-repeat left 50%;
    line-height: 20px;
    padding: 0 0 0 30px;
    width: auto;
}
form#payment fieldset fieldset label:hover { cursor: pointer; }
form#payment input:not([type=radio])，form#payment textarea {
    background: #ffffff;
    border: #FC3 solid 1px;
    -webkit-border-radius: 3px;
    font: italic 13px Georgia，"Times New Roman"，Times，serif;
    outline: none;
    padding: 5px;
```

Note

```
        width: 200px;
}
form#payment input:not([type=submit]):focus，form#payment textarea:focus {
        background: #eaeaea;
        border: #F00 solid 1px;
}
form#payment input[type=radio] {
        float: left;
        margin-right: 5px;
}
</style>
</head>
<body>
<form id=payment>
    <fieldset>
        <legend>用户详细资料</legend>
        <ol>
            <li>
                <label for=name>用户名称：</label>
                <input id=name name=name type=text placeholder="请输入用户名" required autofocus>
            </li>
            <li>
                <label for=email>邮件地址：</label>
                <input id=email name=email type=email placeholder="example@163.com" required>
            </li>
            <li>
                <label for=phone>联系电话：</label>
                <input id=phone name=phone type=tel placeholder="010-12345678" required>
            </li>
        </ol>
    </fieldset>
    <fieldset>
        <legend>家庭住址（收货地址）：</legend>
        <ol>
            <li>
                <label for=address>详细地址：</label>
                <textarea id=address name=address rows=1 required></textarea>
            </li>
            <li>
                <label for=postcode>邮政编码：</label>
                <input id=postcode name=postcode type=text required>
            </li>
            <li>
                <label for=country>国        家：</label>
                <input id=country name=country type=text required>
            </li>
        </ol>
```

```
    </fieldset>
    <fieldset>
        <legend>付费方式</legend>
        <ol>
            <li>
                <fieldset>
                    <legend>信用卡类型</legend>
                    <ol>
                        <li>
                            <input id=visa name=cardtype type=radio>
                            <label for=visa>中国工商银行</label>
                        </li>
                        <li>
                            <input id=amex name=cardtype type=radio>
                            <label for=amex>中国人民银行</label>
                        </li>
                        <li>
                            <input id=mastercard name=cardtype type=radio>
                            <label for=mastercard>中国建设银行</label>
                        </li>
                    </ol>
                </fieldset>
            </li>
            <li>
                <label for=cardnumber>银行卡号：</label>
                <input id=cardnumber name=cardnumber type=number required>
            </li>
            <li>
                <label for=secure>验  证  码：</label>
                <input id=secure name=secure type=number required>
            </li>
            <li>
                <label for=namecard>持  卡  人：</label>
                <input id=namecard name=namecard type=text placeholder="确定是否该卡用户！" required>
            </li>
        </ol>
    </fieldset>
    <fieldset>
        <button type=submit>现在购买</button>
    </fieldset>
</form>
</body>
</html>
```

以上代码在 Chrome 浏览器中的运行效果如图 11.26 所示。用户在浏览器中可以看到通过添加不同的样式实现的表单中对象及文字的样式效果。

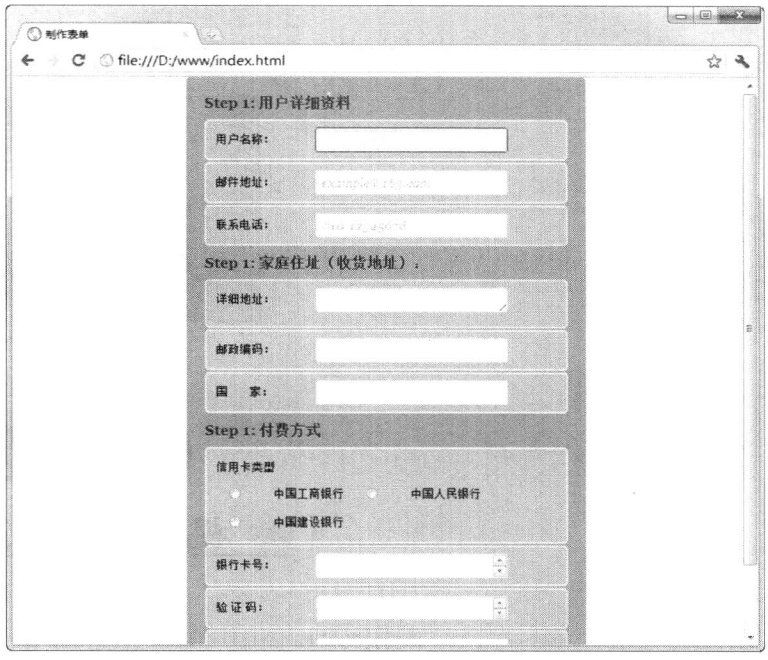


图 11.26　美化表单

Note

第12章

CSS 选择器

（视频讲解：34 分钟）

选择器是 W3C 在 CSS3 工作草案中独立设计的一个模块（http://www.w3.org/TR/css3-selectors/）。CSS1 和 CSS2 定义了大部分常用选择器，这些选择器能满足设计师常规设计需求。但是它们没有进行系统化，也没有形成独立的版块，不利于扩展。

选择器是 CSS 中一个重要的内容。使用它可以大幅度提高开发人员书写或修改样式表时的工作效率。在样式表中，一般会书写大量的代码，在大型网站中，样式表中的代码可能会达到几千行。麻烦的是，当整个网站或整个 Web 应用程序全部书写好之后，需要针对样式表进行修改的时候，在洋洋洒洒一大篇 CSS 代码之中，并没有说明什么样式服务于什么元素，只是使用了 class 属性，然后在页面中指定了元素的 class 属性。使用元素的 class 属性有两个缺点：第一，class 属性本身没有语义，它纯粹是用来为 CSS 样式服务的，属于多余属性。第二，使用 class 属性，并没有把样式与元素绑定起来，针对同一个 class 属性，文本框也可以使用，下拉框也可以使用，甚至按钮也可以使用，这样是非常混乱的，修改样式的时候也很不方便。在 CSS3 中，提倡使用选择器将样式与元素直接绑定起来，这样在样式表中什么样式与什么元素相匹配变得一目了然，修改起来也很方便。不仅如此，通过选择器还可以实现各种复杂的指定，同时也能大量减少样式表的代码书写量，最终书写出来的样式表也会变得简洁明了。为此，CSS3 增加并完善了选择器的功能，以便更灵活地匹配页面元素。

12.1　属性选择器

Note

在 HTML 中，通过各种各样的属性，可以给元素增加很多附加信息。例如，通过 width 属性，可以指定 div 元素的宽度，通过 id 属性，可以将不同的 div 元素进行区分，并且通过 JavaScript 来控制这个 div 元素的内容和状态。

12.1.1　认识属性选择器

CSS2 定义了 4 个属性选择器，如下所示。

- ☑ E[foo]：选择匹配 E 的元素，且该元素定义了 foo 属性。E 选择符可以省略，表示选择定义了 foo 属性的任意类型的元素，下面几个用法相同。
- ☑ E[foo="bar"]：选择匹配 E 的元素，且该元素定义了 foo 属性值为"bar"。
- ☑ E[foo~="bar"]：选择匹配 E 的元素，且该元素定义了 foo 属性，foo 属性值是一个以空格符分隔的列表，其中一个列表值为"bar"。例如，a[title~="bar1"]匹配，而不匹配。
- ☑ E[foo|="en"]：选择匹配 E 的元素，且该元素定义了 foo 属性，foo 属性值是一个以连字符（-）分隔的列表，值开头的字符为"en"。例如，[lang|="en"]匹配<body lang="en-us"><body>，而不匹配<body lang="fr-argot"><body>。

CSS3 在此基础上新增加了 3 个属性选择器，与已经定义的 4 个属性选择器构成了强大的标签属性过滤器。

- ☑ E[foo^="bar"]

选择匹配 E 的元素，且该元素定义了 foo 属性，foo 属性值包含前缀为"bar"的子字符串。

提示： E 选择符可以省略，表示匹配任意类型元素。

例如，body[lang^="en"]匹配<body lang="en-us"><body>，而不匹配<bodylang="fr-argot"><body>。

- ☑ E[foo$="bar"]

选择匹配 E 的元素，且该元素定义了 foo 属性，foo 属性值包含后缀为"bar"的子字符串。

提示： E 选择符可以省略，表示匹配任意类型元素。

例如，img[src$="jpg"]匹配，而不匹配。

- ☑ E[foo*="bar"]

选择匹配 E 的元素，且该元素定义了 foo 属性，foo 属性值包含"bar"的子字符串。

提示： E 选择符可以省略，表示匹配任意类型元素。

例如，img[src$="jpg"]匹配，而不匹配。

CSS3 遵循惯用编码规则，选用^、$和*这 3 个通用匹配运算符，其中^表示匹配起始符，$表示匹配终止符，*表示匹配任意字符，使用它们更符合编码习惯和惯用编程思维。CSS3 草案还保留了对 E[foo~="bar"]和 E[foo|="en"]选择器的支持。实际上，E[foo*="bar"]和 E[foo^="bar"]选择器更符合用户使用习惯，读者可以使用 E[foo*="bar"]选择器替换 E[foo~="bar"]和 E[foo|="en"]选择器，或者使用

E[foo^="bar"]选择器替换 E[foo|="en"]选择器，两者执行效率相差无几。

检测显示新增的 3 个属性选择器可以在实践中放心使用，不用担心浏览器兼容问题，也不用考虑 IE 浏览器的版本问题。

Note

12.1.2 案例实战

由于链接文档的类型不同，链接文件的扩展名也会不同，根据扩展名的不同，分别为不同链接文件类型的超链接增加不同的图标显示，这样能方便浏览者知道所选择的超链接类型。使用属性选择器匹配 a 元素中 href 属性值最后几个字符，即可设计为不同类型的链接添加不同的显示图标。完整代码如下，演示效果如图 12.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>上机练习</title>
<style type="text/css">
p { margin:4px; }
a[href^="http:"] { /*匹配所有有效超链接*/
      background: url(images/window.gif) no-repeat left center;
      padding-left: 18px;
}
a[href$="pdf"] { /*匹配 PDF 文件*/
      background: url(images/icon_pdf.gif) no-repeat left center;
      padding-left: 18px;
}
a[href$="xls"] { /*匹配 XML 样式表文件*/
      background: url(images/icon_xls.gif) no-repeat left center;
      padding-left: 18px;
}
a[href$="ppt"] { /*匹配演示文稿*/
      background: url(images/icon_ppt.gif) no-repeat left center;
      padding-left: 18px;
}
a[href$="rar"] { /*匹配压缩文件*/
      background: url(images/icon_rar.gif) no-repeat left center;
      padding-left: 18px;
}
a[href$="gif"] { /*匹配 GIF 图像文件*/
      background: url(images/icon_img.gif) no-repeat left center;
      padding-left: 18px;
}
a[href$="jpg"] { /*匹配 JPG 图像文件*/
      background: url(images/icon_img.gif) no-repeat left center;
      padding-left: 18px;
}
a[href$="png"] { /*匹配 PNG 图像文件*/
      background: url(images/icon_img.gif) no-repeat left center;
      padding-left: 18px;
```

Note

```
}
a[href$="txt"] { /*匹配记事本文件*/
    background: url(images/icon_txt.gif) no-repeat left center;
    padding-left: 18px;
}
</style>
</head>
<body>
<p><a href="http://www.baidu.com/name.pdf">PDF 文件</a> </p>
<p><a href="http://www.baidu.com/name.ppt">PPT 文件</a> </p>
<p><a href="http://www.baidu.com/name.xls">XLS 文件</a> </p>
<p><a href="http://www.baidu.com/name.rar">RAR 文件</a> </p>
<p><a href="http://www.baidu.com/name.gif">GIF 文件</a> </p>
<p><a href="http://www.baidu.com/name.jpg">JPG 文件</a> </p>
<p><a href="http://www.baidu.com/name.png">PNG 文件</a> </p>
<p><a href="http://www.baidu.com/name.txt">TXT 文件</a> </p>
<p><a href="http://www.baidu.com/#anchor">#锚点超链接</a></p>
<p><a href="http://www.baidu.com/">http://www.baidu.com/</a></p>
</body>
</html>
```

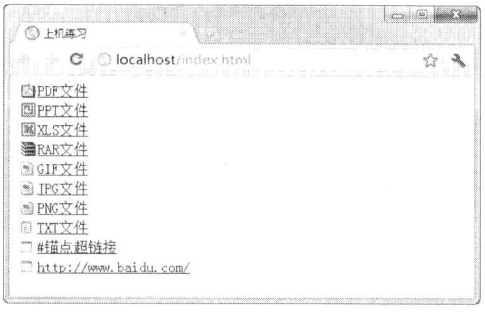


图 12.1　使用属性选择器设计链接类型

如果不借助 CSS3 中这些新添加的属性选择器，则用户可以使用脚本来实现，但会比较麻烦，代码如下。

```
<script type="text/javascript" src="images/jquery.js"></script>
<script type="text/javascript">
$(function(){
    $("a[href$=pdf]").addClass("pdf");
    $("a[href$=xls]").addClass("xls");
    $("a[href$=ppt]").addClass("ppt");
    $("a[href$=rar]").addClass("rar");
    $("a[href$=gif]").addClass("img");
    $("a[href$=jpg]").addClass("img");
    $("a[href$=png]").addClass("img");
    $("a[href$=txt]").addClass("txt");
    $("a:not([href*=http://www.])").not("[href^=#]")
        .addClass("external")
        .attr({ target: "_blank" });
});
</script>
```

12.2 结构伪类选择器

Note

在 CSS 中，可以使用类选择器把相同的元素定义成不同的样式。例如，针对一个 p 元素，可以做如下所示的定义。

```
p.right{text-align:right;}
p.center{text-align: centert;}
```

然后在页面上对 p 元素使用 class 属性，把定义好的样式指定给具体的 p 元素，代码如下所示。

```
<p class="right">测试文字</p>
<p class="center">测试文字</p>
```

在 CSS 中，除了上面所述的类选择器之外，还有一种伪类选择器，这种伪类选择器与类选择器的区别是，类选择 a 可以随便起名，如上面的 p.right 与 p.center，也可以命名为 p.class1 与 p.class2，然后在页面上使用 class= 'classl'和 class= 'class2'。但是伪类选择器是 CSS 中已经定义好的选择器，不能随便起名。在 CSS 中最常用的伪类选择器是 a（锚）元素上的几种选择器。

伪类选择器并不是针对真正的元素使用的选择器，而是针对 CSS 中已经定义好的伪类使用的选择器。

12.2.1 认识结构伪类选择器

结构伪类是 CSS3 新设计的选择器，它利用文档结构树实现元素过滤，通过文档结构的相互关系来匹配特定的元素，从而减少文档内 class 属性和 ID 属性的定义，使得文档更加简洁。

☑ E:root

选择匹配 E 所在文档的根元素。在（X）HTML 文档中，根元素就是 html 元素，此时该选择器与 html 类型选择器匹配内容相同。

☑ E:nth-child(n)

选择所有在其父元素中第 n 个位置的匹配 E 的子元素。

提示：参数 n 可以是数字（1、2、3）、关键字（odd、even）和公式（2n、2n+3），参数的索引起始值为 1，而不是 0。例如：

- tr:nth-child(3)匹配所有表格里排第 3 行的 tr 元素。
- tr:nth-child(2n+1)匹配所有表格的奇数行。
- tr:nth-child(2n)匹配所有表格的偶数行。
- tr:nth-child(odd)匹配所有表格的奇数行。
- tr:nth-child(even)匹配所有表格的偶数行。

☑ E:nth-last-child(n)

选择所有在其父元素中倒数第 n 个位置的匹配 E 的子元素。

提示：该选择器与 E:nth-child(n)是计算顺序相反的选择器，语法和用法相同。

☑ E:nth-of-type(n)

选择所有在其父元素中同类型第 n 个位置的匹配 E 的子元素。

提示：所有匹配 E 的子元素被分离出来单独排序。非 E 子元素不参与排序。参数 n 可以是数字（1、2、3）、关键字（odd、even）、公式（2n、2n+3），参数的索引起始值为 1，而不是 0。

例如，p:nth-of-type(2)匹配 <h1></h1> 片段中第 2 个 p 元素，但不匹配片段中位于第 2 个位置的 p 元素。

Note

☑ E:nth-last-of-type(n)

选择所有在其父元素中同类型倒数第 n 个位置的匹配 E 的子元素。

提示：该选择器与 E:nth-of-type(n)是计算顺序相反的选择器，语法和用法相同。

☑ E:last-child

选择位于其父元素中最后一个位置，且匹配 E 的子元素。

例如，h1:last-child 匹配 <h1></h1> 片段中 h1 元素。

☑ E:first-of-type

选择在其父元素中匹配 E 的第一个同类型的子元素。

提示：该选择器功能类似于 E:nth-of-type(1)选择器。

例如，p:first-of-type 匹配 <h1></h1> 片段中第一个 p 元素。

☑ E:last-of-type

选择在其父元素中匹配 E 的最后一个同类型的子元素。

提示：该选择器功能类似于 E:nth-last-of-type(1)选择器。

例如，p:last-of-type 匹配 <h1></h1> 片段中第二个 p 元素。

☑ E:only-child

选择其父元素只包含一个子元素，且该子元素匹配 E。例如，p:only-child 匹配 片段中的 p 元素，但不匹配 <h1></h1> 片段中的 p 元素。

☑ E:only-of-type

选择其父元素只包含一个同类型子元素，且该子元素匹配 E。例如，p:only-of-type 匹配 片段中的 p 元素，也匹配 <h1></h1> 片段中的 p 元素。

☑ E:empty

选择匹配 E 的元素，且该元素不包含子节点。

提示：文本也属于节点。

各主流浏览器对于结构伪类选择器的支持存在较大的分歧：IE8 及其以下版本浏览器完全不支持结构伪类选择器。Firefox 从 3.5 版本开始全面支持，在 Firefox 3.5 以前版本中支持不是很完善。Opera、Safari 和 Chrome 浏览器对于结构伪类选择器的支持比较完全。

当然读者也可以借助 JavaScript 脚本模拟类似的过滤功能，如 jQuery 等主流 JavaScript 类库中都提供了类似的脚本化结构伪类选择器。

12.2.2　案例实战

【示例 1】设计优雅的数据表格。通过结构伪类选择器设计隔行分色样式，这种样式比较易用，

能够提高浏览数据的速度和准确度。在传统设计方法中主要通过定义一个样式类，然后把该类应用到所有奇数行或偶数行内，当然这种原始方法会增加工作量，也给文档添加了很多不必要的属性。借助结构伪类选择器，使用 E:nth-child(n)选择器快速为偶数行或奇数行定义分色背景，详细代码如下，效果如图 12.2 所示。

Note

```
<!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=gb2312" />
<title>上机练习</title>
<style type="text/css">
h1 { font-size:16px; }
table {
    width:100%;
    font-size:12px;
    table-layout:fixed;
    empty-cells:show;
    border-collapse: collapse;
    margin:0 auto;
    border:1px solid #cad9ea;
    color:#666;
}
th {
    background-image: url(images/th_bg1.gif);
    background-repeat:repeat-x;
    height:30px;
    overflow:hidden;
}
td { height:20px; }
td, th {
    border:1px solid #cad9ea;
    padding:0 1em 0;
}
tr:nth-child(even) {
  background-color:#f5fafe;
}
</style>
</head>
<body>
<table>
    <tr>
        <th>排名</th><th>校名</th><th>总得分</th><th>人才培养总得分</th><th>研究生培养得分</th><th>本科生培养得分</th><th>科学研究总得分</th><th>自然科学研究得分</th><th>社会科学研究得分</th><th>所属省份</th><th>分省排名</th><th>学校类型</th>
    </tr>
    <tr>
        <td>1</td><td>清华大学 </td><td>296.77</td><td>128.92</td><td>93.83</td><td>35.09</td><td>167.85</td><td>148.47</td><td>19.38</td><td width="16">京 </td><td width="12">1 </td><td>理工 </td>
```

Note

```
        </tr>
        <tr>
            <td>2</td><td>北京大学 </td> <td>222.02</td>
<td>102.11</td><td>66.08</td><td>36.03</td><td>119.91</td><td>86.78</td><td>33.13</td><td>京 </td><td>2
</td><td>综合 </td>
        </tr>
        <tr>
            <td>3</td> <td>浙江大学 </td><td>205.65</td><td>94.67</td><td>60.32</td><td>34.35</td>
    <td>110.97</td><td>92.32</td><td>18.66</td><td>浙 </td><td>1</td><td>综合 </td>
        </tr>
        <tr>
            <td>4</td><td>上海交大</td><td>150.98</td><td>67.08</td><td>47.13</td><td>19.95</td><td>
83.89</td>
    <td>77.49</td><td>6.41</td><td>沪 </td><td>1</td><td>综合 </td>
        </tr>
        <tr>
            <td>5</td><td>南京大学</td><td>136.49</td><td>62.84</td><td>40.21</td><td>22.63</td>
    <td>73.65</td><td>53.87</td><td>19.78</td><td>苏</td><td>1</td><td>综合 </td>
        </tr>
        <tr>
            <td>6</td><td>复旦大学 </td><td>136.36</td><td>63.57</td><td>40.26</td><td>23.31</td><td>
72.78</td><td>51.47</td><td>21.31</td>
    <td>沪</td><td>2</td><td>综合 </td>
        </tr>
        <tr>
            <td>7</td><td>华中科大</td><td>110.08</td><td>54.76</td><td>30.26</td><td>24.50</td>
    <td>55.32</td><td>47.45</td><td>7.87</td><td>鄂 </td><td>1</td><td>理工 </td>
        </tr>
        <tr>
            <td>8</td><td>武汉大学 </td><td>103.82</td><td>50.21</td><td>29.37</td><td>20.84</td>
    <td>53.61</td><td>36.17</td><td>17.44</td><td>鄂 </td><td>2</td><td>综合</td>
        </tr>
        <tr>
            <td>9</td><td>吉林大学 </td><td>96.44</td><td>48.61</td><td>25.74</td><td>22.87</td><td>
47.83</td>
    <td>38.13</td><td>9.70</td><td>吉 </td><td>1</td><td>综合</td>
        </tr>
        <tr>
            <td>10</td><td>西安交大</td><td>92.82</td><td>47.22</td><td>24.54</td><td>22.68</td><td>
45.60</td><td>35.47</td>
    <td>10.13</td><td>陕</td><td>1</td><td>综合</td>
        </tr>
    </table>
    </body>
    </html>
```

整个表格以恬淡浅蓝色为主色调，边框为细线，使用背景装饰列表头可以让表格看起来更别致，显式定义单元格高度，适当撑起单元格，让数据看起来更轻松。设计浅色线分隔数据行和列，让表格看起来不是那么生硬，通过结构伪类选择器为表格内偶数行定义背景色，以实现隔行分色的显示效果。

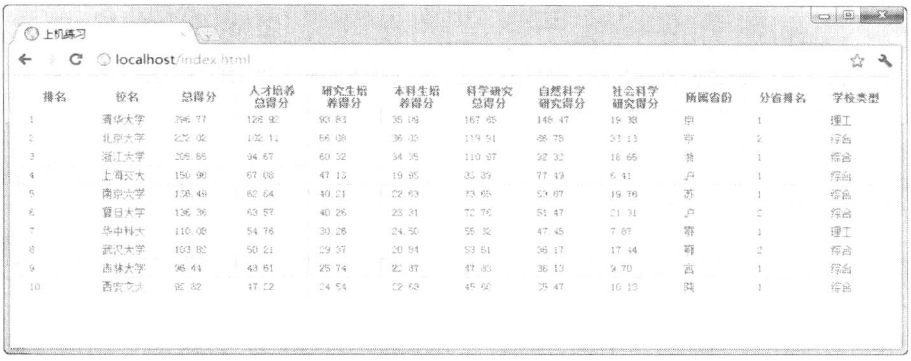


图 12.2　设计优雅的数据表格

【示例 2】下面借助结构伪类选择器设计个性导航菜单样式，本示例模拟综合门户网站的导航列表样式，演示如何使用结构伪类控制多列表样式。在这个导航菜单中包含了众多列表项，为了方便浏览者快速找到需要的链接，通过加粗和变色样式来强调不同菜单的导向作用。示例代码如下，演示效果如图 12.3 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>上机练习</title>
<style type="text/css">
h1 { font-size:16px; }
#menu {
    width:965px;
    height:126px;
    background:url(images/bg6.jpg) no-repeat right bottom;
}
ul, li { /*清除列表默认样式*/
    padding:0;
    margin:0;
    list-style:none;
}
ul { /*通过外框定位列表位置和大小*/
    float:right;
    margin-right:0px;
    margin-top:55px;
    width:790px;
    font-size:12px;
}
li {
    float:left;
    width:36px;
    padding:0 0 4px 0;
    text-align:center;
    background:url(images/line1.gif) no-repeat left center;
}
```

Note

```
li:nth-child(3n+1) { /*匹配 1、4、7、10、13……（步长为 3）项列表项*/
    font-weight:bold;
    background:none; /*清除加粗列表项的背景*/
}
li:nth-child(55) {font-size:11px;} /*为特定位置列表项定义样式，缩小显示*/
li:nth-child(22n+1){ margin-left:-1px;} /*微调每行第一个列表项，以便对齐*/
li:nth-child(20){ color:red;} /*为特定位置列表项定义样式，醒目显示*/
</style>
</head>
<body>
<div id="menu">
    <ul>
        <li> 新闻</li><li>军事</li><li>社会 </li><li> 财经</li><li>股票 </li><li> 基金</li><li> 科技
</li><li> 手机</li><li> 数码 </li><li> 体育 </li><li> 中超</li><li> NBA </li><li> 娱乐 </li><li> 明星
</li><li> 音乐 </li><li> 汽车 </li><li> 图库 </li><li> 车型 </li><li> 博客</li><li> 微博 </li><li> 草根
</li><li> 视频 </li><li> 播客 </li><li> 大片 </li><li> 房产 </li><li> 地产 </li><li> 家居 </li><li> 读书
</li><li> 教育 </li><li> 健康 </li><li> 女性 </li><li> 星座 </li><li> 育儿 </li><li> 乐库 </li><li> 尚品
</li><li> 宠物 </li><li> 空间</li><li> 邮箱 </li><li> 出国</li><li> 论坛</li><li> SHOW </li><li> UC </li><li>
游戏</li><li> 玩玩</li><li> 交友</li><li> 城市</li><li> 广东 </li><li> 上海</li><li> 生活</li><li> 旅游
</li><li> 电商</li><li> 短信 </li><li> 商城</li><li> 彩信</li><li> 健身</li><li> 下载</li><li> 导航</li><li>
商城 </li><li> 天气 </li><li> 爱问</li><li> 彩票 </li><li> 公益 </li><li> 世博 </li>
    </ul>
</div>
</body>
</html>
```

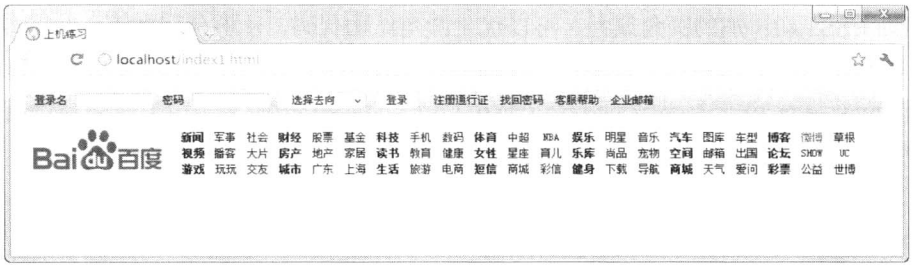


图 12.3　设计多列列表样式

【示例 3】设计圆角的方法有很多种，如一图设计圆角、二图设计圆角、四图设计圆角、山羊角、纯 CSS 圆角、JavaScript 圆角等。虽然圆角的效果相同，但是设计方法不尽一致。使用背景图模拟圆角，不是很灵活；如果使用纯 CSS 设计圆角，需要借助大量的无意义标签，如下所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>上机练习</title>
<style type="text/css">
.wrapper{width:98%;margin:0 auto;}
.sharp{width:100%;margin:20px auto 0;float:left;margin-right:2%;}
.content{height:400px;border-right:1px solid;border-left:1px solid;overflow:hidden;}
```

Note

```
.b1,.b2,.b3,.b4,.b5,.b6,.b7,.b8{height:1px; font-size:1px; overflow:hidden; display:block;}
.b1,.b8{margin:0 5px;}
.b2,.b7{margin:0 3px;border-right:2px solid; border-left:2px solid;}
.b3,.b6{margin:0 2px;border-right:1px solid; border-left:1px solid;}
.b4,.b5{margin:0 1px;border-right:1px solid; border-left:1px solid; height:2px;}
/*蓝色边框*/
/*边框色*/
.color1 .b2,.color1 .b3,.color1 .b4,.color1 .b5,.color1 .b6,.color1 .b7,.color1 .content{border-color:#96C2F1;}
.color1 .b1,.color1 .b8{background:#96C2F1;}
/*背景色*/
.color1 .b2,.color1 .b3,.color1 .b4,.color1 .b5,.color1 .b6,.color1 .b7,.color1 .content{background:#EFF7FF;}
</style>
</head>
<body>
<div class="wrapper">
    <div class="sharp color1">
        <b class="b1"></b><b class="b2"></b><b class="b3"></b><b class="b4"></b>
        <div class="content">
            <img src="images/bg1.jpg" />
        </div>
        <b class="b5"></b><b class="b6"></b><b class="b7"></b><b class="b8"></b>
    </div>
</div>
</body>
</html>
```

如果借助 CSS3 新添加的圆角属性，可以优化圆角结构代码。示例代码如下，演示效果如图 12.4 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>上机练习</title>
<style type="text/css">
.wrapper{width:98%;margin:0 auto;}
.sharp{width:100%;margin:20px auto 0;float:left;margin-right:2%;}
.content{height:400px;overflow:hidden; border:solid 1px;}
.content{
    border-radius:10px;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
}
.sharp b:empty {
    display:none;
}
/*蓝色边框*/
.color1 .content{background:#EFF7FF; border-color:#96C2F1;}
</style>
</head>
```

```
<body>
<div class="wrapper">
    <div class="sharp color1">
        <div class="content">
            <img src="images/bg1.jpg" />
        </div>
    </div>
</div>
</body>
</html>
```

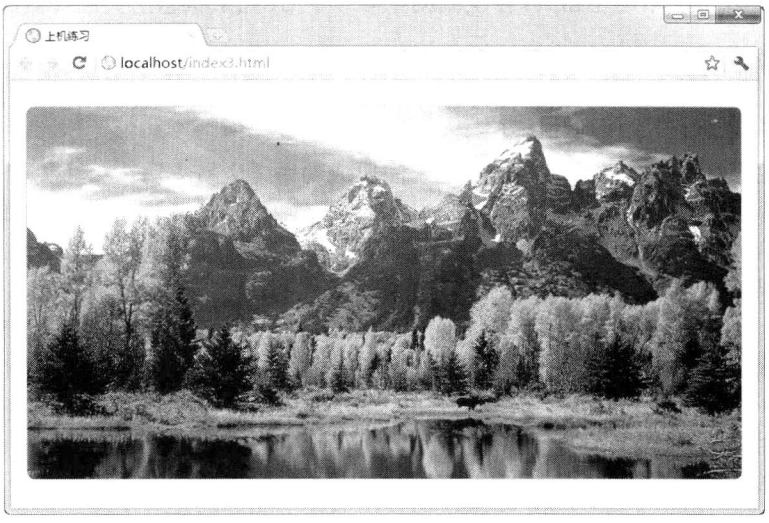


图 12.4　设计圆角样式

12.3　UI 伪类选择器

在 CSS3 的选择器中，除了结构性伪类选择器外，还有一种 UI 元素状态伪类选择器。这些选择器的共同特征是：指定的样式只有当元素处于某种状态下时才起作用，在默认状态下不起作用。

CSS3 共定义了 11 种 UI 元素状态伪类选择器，分别是 E:hover、E:active、E:focus、E:enabled、E:disabled、E:read-only、E:read-write、E:checked、E:default、E: indeterminate 和 E: selection。

12.3.1　认识常用 UI 伪类选择器

UI 元素状态伪类是 CSS3 新设计的选择器，其中 UI 是 User Interface（用户界面）的简写，UI 元素的状态一般包括可用、不可用、选中、未选中、获取焦点、失去焦点、锁定和待机等。下面简单介绍 CSS3 定义的 3 种常用状态伪类选择器。

☑　E:enabled

选择匹配 E 的所有可用 UI 元素。在网页中，UI 元素一般是指包含在 form 元素内的表单元素。

例如，input:enabled 匹配<form><input type="text" /><input type="button" disabled="disabled" /></form>片段中文本框，但不匹配该片段中按钮。

☑ E:disabled

选择匹配 E 的所有不可用 UI 元素。

例如，input:disabled 匹配<form><input type="text" /><input type="button" disabled="disabled" /></form>片段中按钮，但不匹配该片段中文本框。

Note

☑ E:checked

选择匹配 E 的所有可用 UI 元素。

例如，input:checked 匹配<form><input type="checkbox" /><input type="radio" checked="checked" /></form>片段中的单选按钮，但不匹配该片段中的复选框。

除了 IE 8 及其以下版本浏览器外，各主流浏览器对于 UI 元素状态伪类选择器的支持都非常优秀。

12.3.2 案例实战

在网页设计中经常需要用到表单，如注册、登录、回复、留言、购买等交互行为，都需要表单来辅助。表单设计的好坏，直接关系到用户体验、注册成功率、交易成功率和回头率。表单的设计应该追求简洁、美观、可用性、可访问和符合 Web 标准。要确保表单可用，首先设计师应该从 HTML 结构开始，设计符合语义的表单结构，然后使用 CSS 呈现表单效果。下面这个示例可以帮助读者认识如何设计合理的结构，以及如何设计简洁的表单样式。

【示例】设计合理的表单结构，离不开表单辅助元素的配合，其中 label 元素应该通过 for 属性绑定到表单域上，for 的属性值应设置为表单域的 name 属性值。当用户登录完毕，不妨通过脚本把文本框设置为不可用（disabled="disabled"）状态，这时设计师就可以通过 E: disabled 选择器让文本框显示为灰色，以告诉用户该文本框不可用了。这样就不用设计“不可用”样式类，并把该类添加到 HTML 结构中。本示例的源代码如下，演示效果如图 12.5 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>上机练习</title>
<style type="text/css">
h1 { font-size:20px; }
#login {
    width:400px;
    padding:1em 2em 0 2em;
    font-size:12px;
}
label {
    line-height:26px;
    display:block;
    font-weight:bold;
}
#name, #password {
    border:1px solid #ccc;
    width:160px;
    height:22px;
    padding-left:20px;
```

Note

```
    margin:6px 0;
    line-height:20px;
}
#name { background:url(images/name.gif) no-repeat 2px 2px; }
#password { background:url(images/password.gif) no-repeat 2px 2px; }
.button { margin:6px 0; }
#login input:disabled#name {
    background:#ddd url(images/name1.gif) no-repeat 2px 2px;
    border:1px solid #fff;
}
#login input:disabled#password {
    background:#ddd url(images/password1.gif) no-repeat 2px 2px;
    border:1px solid #fff;
}
</style>
</head>
<body>
<fieldset id="login">
    <legend>用户登录</legend>
    <form action="" method="POST" class="form">
        <label for="name">姓名
            <input name="name" type="text"    id="name" value="" />
        </label>
        <label for="password">密码
            <input name="password" type="text" id="password" value="" disabled="disabled" />
        </label>
        <input type="image"    class="button" src="images/login1.gif" />
    </form>
</fieldset>
</body>
</html>
```

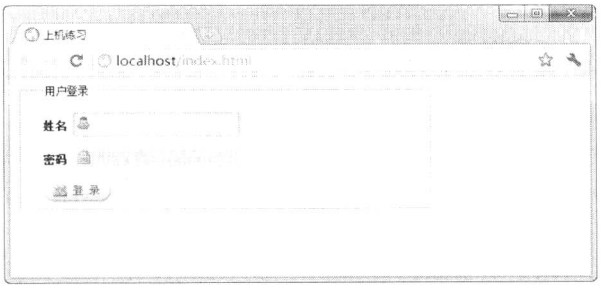


图 12.5　设计表单不可用样式

在上面示例中通过下面样式设计不可用状态效果。

```
#login input:disabled#name { /*姓名文本框不可用状态时的样式*/
background:#ddd url(images/name1.gif) no-repeat 2px 2px;
    border:1px solid #fff;
}
#login input:disabled#password { /*密码域不可用状态时的样式*/
background:#ddd url(images/password1.gif) no-repeat 2px 2px;
```

```
    border:1px solid #fff;
}
```

同时在文本框中补加 disabled 属性。

```
<label for="name">姓名
    <input name="name" type="text"    id="name" value="" />
</label>
<label for="password">密码
    <input name="password" type="text" id="password" value="" disabled="disabled" />
</label>
```

如果在上面示例基础上适当进行美化，则显示效果如图 12.6 所示。

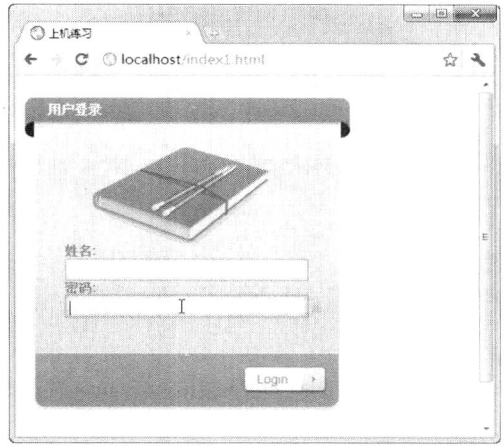


图 12.6　设计更漂亮的表单样式

在构建表单结构时，应该充分利用表单属性，为表单域设置这些属性，能极大地改善用户体验，让表单变得更好用。

- ☑ 使用 value 属性为输入型表单域设置默认值。
- ☑ 使用 title 属性帮助屏幕阅读器用户获得足够的提示信息。
- ☑ 如果标签（label 元素）中没有包含所关联的表单域时，使用 for 属性来匹配关联表单域（绑定 id 值），这样当用户单击标签时，当前表单域会自动获得焦点。
- ☑ 为每个表单域设置 tabindex 属性，方便用户使用 Tab 键切换表单项。
- ☑ 对于复选框和单选按钮来说，将标签（label）置于表单域前面，方便用户先阅读标签说明，后执行操作。
- ☑ 使用 outgroup 元素来组织 select 列表框中的众多选项，让下拉列表项目一目了然。
- ☑ 为表单域定义 accesskey 属性，方便用户使用快捷键快速访问。
- ☑ 使用 checked 属性，把常用的复选框和单选按钮定义为默认值，避免用户重复操作。
- ☑ 使用 selected 属性，在列表框中把常用选项定义为默认值，同样可以避免用户重复操作。

12.4　其他选择器

除了上面介绍的选择器外，CSS3 还增加了其他选择器，这些选择器用途不一，故放在一起进行

介绍。

☑　E~F

通用兄弟元素选择器类型。选择匹配 F 的所有元素，且匹配元素位于匹配 E 的元素后面。

提示：在 DOM 结构树中，E 和 F 所匹配的元素应该在同一级结构上。

例如，div~p 匹配 1 2 3 片段中 3 ，但是不匹配 1 和 2 。

☑　E:not(s)

否定伪类选择器类型。选择匹配 E 的所有元素，且过滤掉匹配 s 选择符的任意元素。s 是一个简单结构的选择器，不能够使用复合选择器。E:not(s)选择器相当于二次过滤，适合精确选择元素。

例如，div p:not(.red)匹配 1 2 3 片段中的 2 ，但是不匹配 1 和 3 。

在上面示例中，浏览器首先使用 div p 包含选择器，匹配所有 div 元素包含的 p 元素，此时匹配结果为前两个 p 元素。然后使用 class 选择器.red 匹配第一个 p 元素，并把它从最终结果集中过滤掉。最后，返回过滤后的匹配结果。

☑　E:target

目标伪类选择器类型。选择匹配 E 的所有元素，且匹配元素被相关 URL 指向。该选择器是动态选择器，只有当存在 URL 指向该匹配元素时，样式效果才能够有效。

【示例 1】针对下面文档，在浏览器地址栏中输入 URL，并附加“#red”，以锚点方式链接到 ，则该元素立即显示为红色背景，如图 12.7 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>上机练习</title>
<style type="text/css">
div:target { background:red; }
</style>
</head>
<body>
<div id="red">盒子 1</div>
<div id="blue">盒子 2</div>
</body>
</html>
```

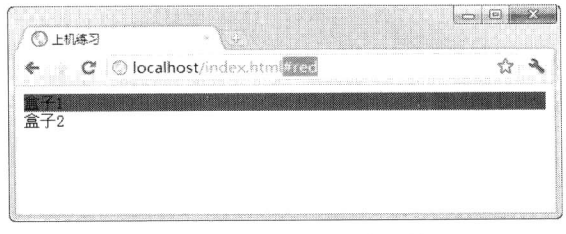


图 12.7　目标伪类样式应用效果

对于 E~F 选择器来说，目前主流浏览器都能够完全支持，IE8 及其以下版本浏览器不支持 E:not(s) 和 E:target 选择器。

Note

【示例 2】本示例演示如何设计一个分层表格样式。对于表格结构来说，table 表示一级结构，thead、tbody 和 tfoot 表示二级结构，tr 表示三级结构，th 和 td 表示四级结构。但是，如果希望数据表格也能够呈现层次结构关系，就应该借助 CSS 来模拟这种结构。这里主要借助否定伪类选择器和结构伪类选择器，配合 CSS 背景图像技术设计树形结构标志；借助伪类选择器设计鼠标经过时动态背景效果，利用 CSS 边框和背景色设计标题行的立体显示效果。示例完整代码如下，预览效果如图 12.8 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>上机练习</title>
<style type="text/css">
body { font-family:"宋体" arial, helvetica, sans-serif; }
    table {
    border-collapse: collapse; /*细线表格*/
    font-size: 75%;
    line-height: 1.1;
}
tr:hover, td:hover { background: #FF9; } /*鼠标经过时动态背景色*/
th, td {
    padding: .3em .5em;
    cursor:pointer;/*手形光标样式*/
}
th {
    font-weight: normal;/*清除加粗显示*/
    text-align: left;
    padding-left: 15px;
}
td:only-of-type {
    background:#eee url(images/arrow.gif) no-repeat 12px 50%;
    padding-left: 28px;
    font-weight:bold;
    color:#444;
}
thead th {
    background: #c6ceda;
    border-color: #fff #fff #888 #fff;
    border-style: solid;
    border-width: 1px 1px 2px 1px;
    padding-left: .5em;
}
tbody th:not(.end) {
    background: url(images/dots.gif) 15px 56% no-repeat;
    padding-left: 26px;
```

Note

```
}
tbody th.end {
    background: url(images/dots2.gif) 15px 56% no-repeat;
    padding-left: 26px;
}
</style>
</head>
<body>
<table>
    <thead>
        <tr>
            <th>排名</th><th>校名</th><th>总得分</th><th>人才培养总得分</th><th>研究生培养得分
</th><th>本科生培养得分</th><th>科学研究总得分</th><th>自然科学研究得分</th><th>社会科学研究得分
</th><th>所属省份</th><th>分省排名</th><th>学校类型</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td colspan="12">一类</td>
        </tr>
        <tr>
            <td>1</td><td>清华大学 </td><td>296.77</td><td>128.92</td><td>93.83</td><td>35.09</td>
<td>167.85</td><td>148.47</td><td>19.38</td><td width="16">京 </td><td width="12">1 </td><td>理工</td>
        </tr>
        <tr>
            <th class="end">2</th><td>北京大学 </td> <td>222.02</td> <td>102.11</td><td>66.08
</td><td>36.03</td><td>119.91</td><td>86.78</td><td>33.13</td><td>京 </td><td>2 </td><td>综合 </td>
        </tr>
        <tr>
            <td colspan="12">二类</td>
        </tr>
        <tr>
            <td>3</td> <td>浙江大学 </td><td>205.65</td><td>94.67</td><td>60.32</td><td>34.35
</td><td>110.97</td><td>92.32</td><td>18.66</td><td>浙 </td><td>1</td><td>综合 </td>
        </tr>
        <tr>
            <td>4</td><td>上海交大</td><td>150.98</td><td>67.08</td><td>47.13</td><td>19.95
</td><td>83.89</td><td>77.49</td><td>6.41</td><td>沪 </td><td>1</td><td>综合 </td>
        </tr>
        <tr>
            <td>5</td><td>南京大学</td><td>136.49</td><td>62.84</td><td>40.21</td><td>22.63
</td><td>73.65</td><td>53.87</td><td>19.78</td><td>苏</td><td>1</td><td>综合 </td>
        </tr>
        <tr>
            <td>6</td><td>复旦大学 </td><td>136.36</td><td>63.57</td><td>40.26</td><td>23.31
</td><td>72.78</td><td>51.47</td><td>21.31</td><td>沪</td><td>2</td><td>综合</td>
        </tr>
        <tr>
            <td>7</td><td>华中科大</td><td>110.08</td><td>54.76</td><td>30.26</td><td>24.50
```

Note

```
</td><td>55.32</td><td>47.45</td><td>7.87</td><td>鄂 </td><td>1</td><td>理工 </td>
            </tr>
            <tr>
                <td>8</td><td>武汉大学 </td><td>103.82</td><td>50.21</td><td>29.37</td><td>20.84
</td><td>53.61</td><td>36.17</td><td>17.44</td><td>鄂 </td><td>2</td><td>综合</td>
            </tr>
            <tr>
                <td>9</td><td>吉林大学 </td><td>96.44</td><td>48.61</td><td>25.74</td><td>22.87
</td><td>47.83</td><td>38.13</td><td>9.70</td><td>吉 </td><td>1</td><td>综合</td>
            </tr>
            <tr>
                <th class="end">10</th><td>西安交大</td><td>92.82</td><td>47.22</td><td>24.54
</td><td>22.68</td><td>45.60</td><td>35.47</td>
    <td>10.13</td><td>陕</td><td>1</td><td>综合</td>
            </tr>
        </tbody>
    </table>
    </body>
    </html>
```

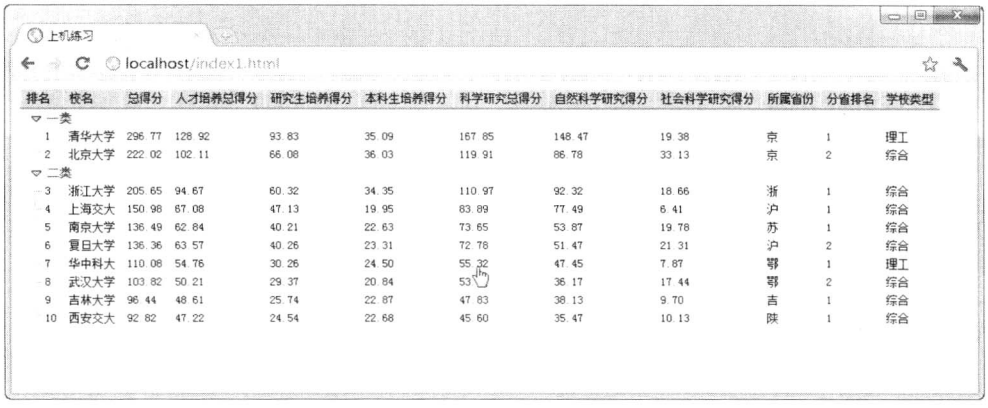


图 12.8　设计表格样式

使用结构伪类选择器选择合并单元格所在的行，以背景方式在行前定义指示图标。

```
td:only-of-type {
    background:#eee url(images/arrow.gif) no-repeat 12px 50%;
    padding-left: 28px;
    font-weight:bold;
    color:#444;
}
```

使用否定伪类选择器选择主体区域非最后一个 th 元素，以背景方式在行前定义结构路径线。

```
tbody th:not(.end) {
    background: url(images/dots.gif) 15px 56% no-repeat;
    padding-left: 26px;
}
```

使用类选择器选择主体区域非最后一个 th 元素，以背景方式在行前定义结构封闭路径线。

```
tbody th.end {
    background: url(images/dots2.gif) 15px 56% no-repeat;
    padding-left: 26px;
}
```

Note

使用 thead 元素把表头标题独立出来，这样对机器更友好，同时也方便 CSS 控制，避免定义过多的 class 属性。th 元素有两种显示形式，一种用来定义列标题，另一种是定义行标题。

第13章

文本、字体与颜色

（视频讲解：49 分钟）

在文本样式控制方面，CSS3 新增了几个文本属性，同时完善了颜色控制，实现不透明效果的支持。可以说，颜色的不透明度支持是 CSS3 革新的最大亮点。CSS3 定义的文本属性说明如下：

- text-shadow 文本定义文本阴影或模糊效果。
- text-overflow 文本定义省略文本的处理方式。
- word-wrap 文本定义文本超过指定容器的边界时是否断开转行。

CSS3 版本增强的颜色功能说明如下：

- HSL 颜色表示方式通过对色调（H）、饱和度（S）和亮度（L）三个颜色通道的变化以及它们相互之间的叠加来表示各式各样的颜色。
- HSLA 颜色表示方式 HSLA 是在 HSL 的基础上增加一个透明度（A）的设置。

13.1　设计文本阴影

Note

在 CSS3 中，可以使用 text-shadow 属性给页面上的文字添加阴影效果，到目前为止 Safari、Firefox、Chrome 和 Opera 等主流浏览器都支持该功能。text-shadow 属性是在 CSS2 中定义的，在 CSS 2.1 中被删除了，在 CSS3 的 Text 模块中又得以恢复。

13.1.1　定义 text-shadow 属性

text-shadow 属性的基本语法如下所示。

```
text-shadow:none|<length>none|[<shadow>,]*<shadow>
text-shadow:none|<color>[,<color>]*
```

text-shadow 属性的初始值为无，适用于所有元素。取值简单说明如下。

☑　<color>：表示颜色。

☑　<length>：表示由浮点数字和单位标识符组成的长度值，可为负值，指定阴影的水平延伸距离。

【示例】下面为段落文本定义一个简单的阴影效果，演示效果如图 13.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
p {
    text-align: center;
    font:bold 60px helvetica，arial，sans-serif;
    color: #999;
    text-shadow: 0.1em 0.1em #333;
}
</style>
</head>
<body>
<p>HTML5+CSS3</p>
</body>
</html>
```

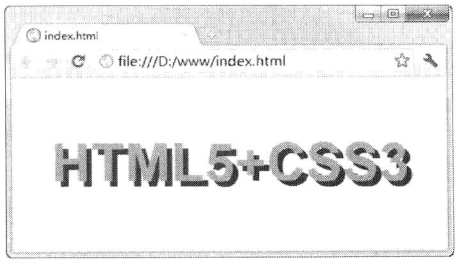


图 13.1　定义文本阴影

text-shadow: 0.1em 0.1em #333;声明了右下角文本阴影效果，如果把投影设置到左上角，则可以按以下方法声明，效果如图 13.2 所示。

Note

```
<style type="text/css">
p {text-shadow: -0.1em -0.1em #333;}
</style>
```

同理，如果设置阴影在文本的左下角，则可以设置如下样式，演示效果如图 13.3 所示。

```
<style type="text/css">
p {text-shadow: -0.1em 0.1em #333;}
</style>
```

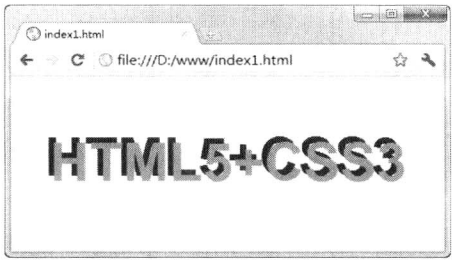


图 13.2　定义左上角阴影

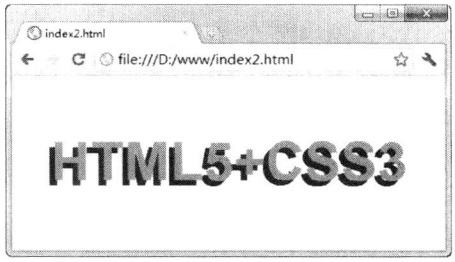


图 13.3　定义左下角阴影

也可以增加模糊效果的阴影，效果如图 13.4 所示。

```
<styletype="text/css">
p{ text-shadow: 0.1em 0.1em 0.3em #333; }
</style>
```

或者定义如下模糊阴影效果，效果如图 13.5 所示。

```
<styletype="text/css">
text-shadow: 0.1em 0.1em 0.2em black;
</style>
```

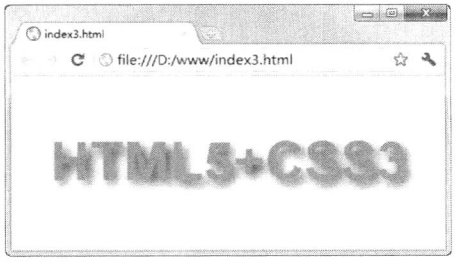


图 13.4　定义模糊阴影（1）

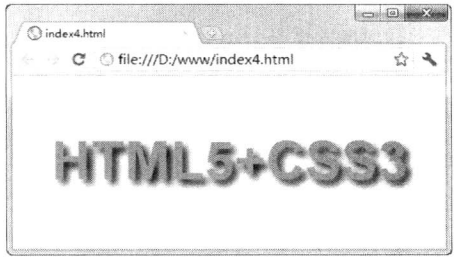


图 13.5　定义模糊阴影（2）

text-shadow 属性的第一个值表示水平位移，第二个值表示垂直位移，正值为偏右或偏下，负值为偏左或偏上，第三个值表示模糊半径，该值可选，第四个值表示阴影的颜色，该值可选。在阴影偏移之后，可以指定一个模糊半径。模糊半径是一个长度值，指出模糊效果的范围。如何计算模糊效果的具体算法并没有指定。在阴影效果的长度值之前或之后还可以选择指定一个颜色值。颜色值会被用作阴影效果的基础。如果没有指定颜色，那么将使用 color 属性值来替代。

13.1.2　应用阴影效果

Note

灵活使用 text-shadow 属性可以解决网页设计中很多实际问题，下面结合几个示例进行介绍。

☑　通过阴影增加前景色与背景色的对比度

【示例 1】在这个示例中通过阴影把文本颜色与背景色区分开来，让字体看起来更清晰，代码如下，演示效果如图 13.6 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
p {
    text-align: center;
    font:bold 60px helvetica，arial，sans-serif;
    color: #fff;
    text-shadow: black 0.1em 0.1em 0.2em;
}
</style>
</head>
<body>
<p>HTML5+CSS3</p>
</body>
</html>
```

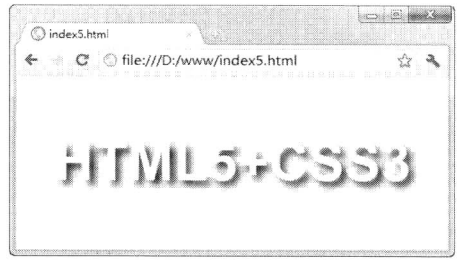


图 13.6　使用阴影增加前景色和背景色对比度

☑　定义多色阴影

text-shadow 属性可以接受一个以逗号分割的阴影效果列表，并应用到该元素的文本上。阴影效果按照给定的顺序应用，因此有可能出现互相覆盖，但是它们永远不会覆盖文本本身。阴影效果不会改变边框的尺寸，但可能延伸到它的边界之外。阴影效果的堆叠层次和元素本身的层次是一样的。

【示例 2】本示例演示了如何为红色文本定义 3 个不同颜色的阴影，演示效果如图 13.7 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
```

```
<style type="text/css">
p {
    text-align: center;
    font:bold 60px helvetica，arial，sans-serif;
    color: red;
    text-shadow: 0.2em 0.5em 0.1em #600,
        -0.3em 0.1em 0.1em #060,
        0.4em -0.3em 0.1em #006;
}
</style>
</head>
<body>
<p>HTML5+CSS3</p>
</body>
</html>
```

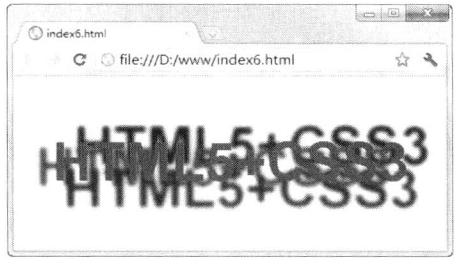


图 13.7　定义多色阴影（1）

提示：当使用 text-shadow 属性定义多色阴影时，每个阴影效果必须指定阴影偏移，而模糊半径、阴影颜色是可选参数。

【示例 3】本示例演示了如何把阴影设置到文本线框外面，代码如下，演示效果如图 13.8 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
p {
    text-align: center;
    font:bold 60px helvetica，arial，sans-serif;
    color: red;
    border:solid 1px red;
    text-shadow: 0.5em 0.5em 0.1em #600,
        -1em 1em 0.1em #060,
        0.8em -0.8em 0.1em #006;
}
</style>
</head>
<body>
<p>HTML5+CSS3</p>
```

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

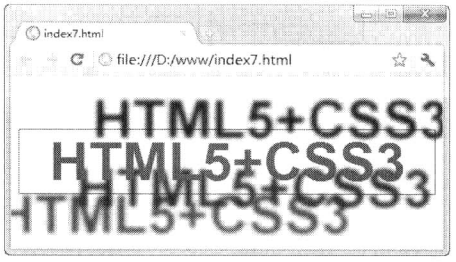


图 13.8　定义多色阴影（2）

☑　定义火焰文字

【示例 4】借助阴影效果列表机制，可以使用阴影叠加出燃烧的文字特效，代码如下，演示效果如图 13.9 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body {background:#000;}
p {
    text-align: center;
    font:bold 60px helvetica，arial，sans-serif;
    color: red;
    text-shadow: 0 0 4px white，
        0 -5px 4px #ff3，
        2px -10px 6px #fd3，
        -2px -15px 11px #f80，
        2px -25px 18px #f20;
}
</style>
</head>
<body>
<p>HTML5+CSS3</p>
</body>
</html>
```

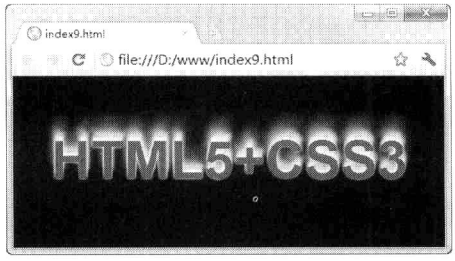


图 13.9　定义燃烧的文字阴影

读者还可以添加更多的阴影列表项，从而可以叠加各种复杂的特效。

☑ 定义立体文字

【示例 5】text-shadow 属性可以使用在:first-letter 和:first-line 伪元素上，同时还可以利用该属性设计立体文本。使用阴影叠加出的立体文本特效代码如下，演示效果如图 13.10 所示。

Note

```
<!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></title>.
<style type="text/css">
body { background: #000; }
p {
    text-align: center;
    padding: 24px;
    margin: 0;
    font-family: helvetica，arial，sans-serif;
    font-size: 80px;
    font-weight: bold;
    color: #D1D1D1;
    background: #CCC;
    text-shadow: -1px -1px white，
        1px 1px #333;
}
</style>
</head>
<body>
<p>HTML5+CSS3</p>
</body>
</html>
```

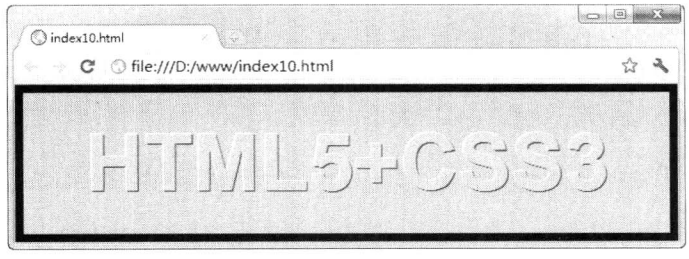


图 13.10 定义凸起的文字效果

通过左上和右下各添加一个 1 像素错位的补色阴影，营造一种淡淡的立体效果。反向思维，利用示例 5 的设计思路，也可以设计一种凹体效果，设计方法就是把示例 5 中左上和右下阴影颜色颠倒，主要代码如下，演示效果如图 13.11 所示。

```
<style type="text/css">
body { background: #000; }
p {
```

```
        text-shadow: 1px 1px white,
             -1px -1px #333;
    }
    </style>
```

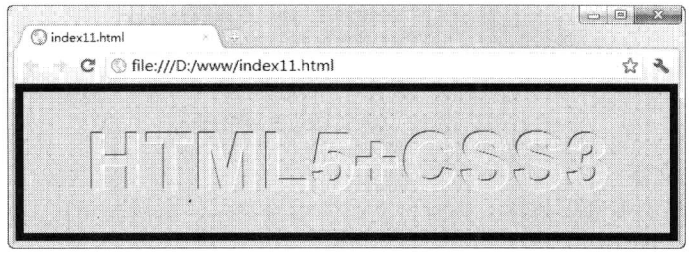


图 13.11　定义凹下的文字效果

☑　定义描边文字

【示例 6】使用 text-shadow 属性还可以为文本描边，设计方法是分别为文本的 4 条边添加 1 像素的实体阴影，代码如下，演示效果如图 13.12 所示。

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title></title>
    <style type="text/css">
    body { background: #000; }
    p {
        text-align: center;
        padding:24px;
        margin:0;
        font-family: helvetica，arial，sans-serif;
        font-size: 80px;
        font-weight: bold;
        color: #D1D1D1;
        background:#CCC;
        text-shadow: -1px 0 black，
             0 1px black,
             1px 0 black，
             0 -1px black;
    }
    </style>
    </head>
    <body>
    <p>HTML5+CSS3</p>
    </body>
    </html>
```

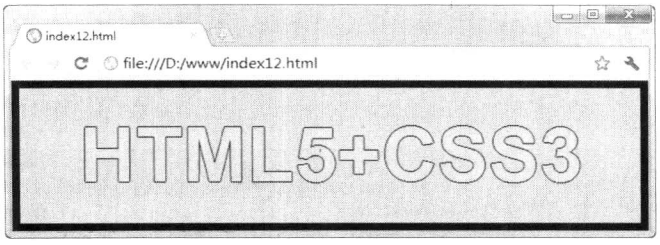


图 13.12　定义描边文字效果

☑　定义外发光文字

【示例 7】设计阴影不发生位移，同时定义阴影模糊显示，这样就可以模拟出文字外发光效果，代码如下，演示效果如图 13.13 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body { background: #000; }
p {
    text-align: center;
    padding:24px;
    margin:0;
    font-family: helvetica，arial，sans-serif;
    font-size: 80px;
    font-weight: bold;
    color: #D1D1D1;
    background:#CCC;
    text-shadow: 0 0 0.2em #F87,
        0 0 0.2em #F87;
}
</style>
</head>
<body>
<p>HTML5+CSS3</p>
</body>
</html>
```

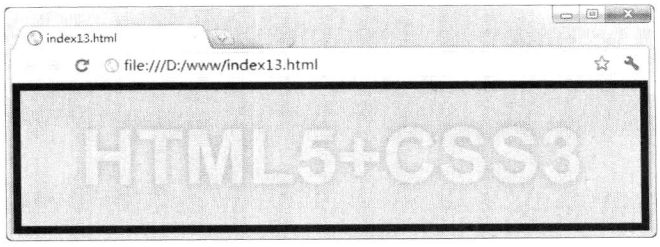


图 13.13　定义外发光文字效果

13.1.3　综合实战：设计黑客网站首页

Note

很多用户喜欢使用图形化首页引导浏览者的视线，富有冲击力的画面，极少的文字说明，都能够让浏览者有一种继续探知的冲动。本示例将模拟一个黑客网站的首页，借助 text-shadow 属性设计阴影效果，通过颜色的搭配，营造一种静谧而又神秘的氛围，使用两幅 PNG 图像对页面效果进行装饰和点缀，最后演示效果如图 13.14 所示。

图 13.14　设计黑客网站首页

示例完整代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body {
    padding: 0px;
    margin: 0px;
    background: black;
    color: #666;
}
/*设计包含框样式*/
#text-shadow-box {
    /*定义内部的定位元素以这个框为参照物*/
    position: relative;
    width: 598px;
    height: 406px;
    background: #666;
    /*禁止内容超过设定的区域*/
    overflow: hidden;
    border: #333 1px solid;
}
```

Note

```
/*设计背景墙样式*/
#text-shadow-box div.wall {
    position: absolute;
    width: 100%;
    top: 175px;
    left: 0px
}
/*设计导航文本样式*/
#text {
    text-align: center;
    line-height: 0.5em;
    margin: 0px;
    font-family: helvetica，arial，sans-serif;
    height: 1px;
    color: #999;
    font-size: 80px;
    font-weight: bold;
    text-shadow: 5px -5px 16px #000;
}
/*设计前面挡风板样式*/
div.wall div {
    position: absolute;
    width: 100%;
    height: 300px;
    top: 42px;
    left: 0px;
    background: #999;
}
/*设计覆盖在上面的探照灯效果图*/
#spotlight {
    position: absolute;
    width: 100%;
    height: 100%;
    top: 0px;
    left: 0px;
    background: url(images/spotlight.png) center -300px;
    font-size: 12px;
}
#spotlight a {
    color: #ccc;
    text-decoration: none;
    position: absolute;
    left: 45%;
    top: 58%;
    float: left;
    text-shadow: 1px 1px #999，-1px -1px #333;
}
#cat {
    position: absolute;
    top: 130px;
```

```
        left: 260px;
        z-index: 1000;
        opacity: 0.5;
    }
    #cat img { width: 80px; }
    </style>
    </head>
    <body>
    <div id="text-shadow-box">
        <div class="wall">
            <p id="text">黑客帝国</p>
            <div></div>
        </div>
        <div id="spotlight"><a href="index.htm">Hacker Home</a></div>
        <div id="cat"><img src="images/cat.png" /></div>
    </div>
    </body>
    </html>
```

Note

定义页面背景色为黑色，前景色为灰色，设计主色调，并清除页边距。设计右上偏移的阴影，适当进行模糊处理，产生色晕效果，阴影色为深色，营造静谧的主观效果。设计一个层，让其覆盖在页面上，使其满窗口显示，通过前期设计好的一个探照灯背景来营造神秘效果。通过 外罩，可以为页面覆盖一层桌纸，添加特殊的艺术效果。

13.2 定义文本样式

CSS3 文本模块（TextModule）把与文本相关的属性单独进行规范，以期形成一个独立的标准体系。文本模块的最早版本是在 2003 年制定的（http://www.w3.org/TR/2003/CR-css3-text-20030514/），2005 年对其进行了修订（http://www.w3.org/TR/2005/WD-css3-text-20050627/），2007 年又进行了系统更新（http://www.w3.org/TR/2007/WD-css3-text-20070306/），最后形成了一个较为完善的文本模型（http://www.w3.org/TR/css3-text/）。在最终版本的文本模块中，除了新增文本属性外，还对 CSS 2.1 版本中已定义的属性取值进行修补，增加了更多的属性值，以适应复杂环境中文本的呈现。

13.2.1 文本样式简介

CSS3 版本规范从起草到定型经历了漫长的演化过程，前后制订了 3 个主要版本的工作草案，最新版本的文本模型（http://www.w3.org/TR/css3-text/）与 2003 年版本（http://www.w3.org/TR/2003/CR-css3-text-20030514/）相比，进行了较大的改动，其中主要改动说明如下。

- ☑ line-break 和 word-break-cjk 属性被 word-break 属性替换。
- ☑ word-break-inside 属性被 hyphenate 属性替换。
- ☑ wrap-option 属性被 text-wrap 和 word-break 属性替换。
- ☑ linefeed-treatment、white-space-treatment 和 all-space-treatment 属性被 white-space-collapse 属性替换。
- ☑ min-font-size 和 max-font-size 属性被移到下一个 CSS3 版本中字体模块内。

Note

- ☑ 修改了 text-align 属性中 left 和 right 属性值在垂直文本中的行为。
- ☑ text-align-last 属性取消了 size 属性值。
- ☑ text-justify 属性取消了 newspaper 属性值。
- ☑ word-spacing 和 letter-spacing 属性增加了百分比取值。
- ☑ text-wrap 属性增加了 suppress 属性值。
- ☑ 删除了 linefeed-treatment 属性。
- ☑ text-align-last 属性取消了 size 属性值。
- ☑ text-justify 属性新增了 tibetan 属性值。
- ☑ punctuation-trim 属性新增了 end 属性值。
- ☑ kerning-mode:contextual 被 punctuation-trim:adjacent 替换，其他控制被移到字体模块中。
- ☑ text-shadow 属性现在可以继承。
- ☑ 新增 text-outline 属性。
- ☑ 新增 text-emphasis 属性，以替换 font-emphasis 属性。
- ☑ 重新定义了 text-indent 属性。
- ☑ 重新设计了 hanging-punctuation 属性。

最新版本的文本模型与 2005 年版本（http://www.w3.org/TR/2005/WD-css3-text-20050627/）相比，也进行了适当修订，其中增加 text-emphasis 和 text-outline 属性，移除了 font-emphasis 属性，其他更多改动细节请参阅工作文档。

为方便读者参考和学习，下面简单描述 CSS3 新增的文本属性。

☑ white-space-collapse

语法说明如下：

```
white-space-collapse:preserve|collapse|preserve-breaks|discard;
```

white-space-collapse 初始值为 collapse，适用于所有元素。该属性用于设置或检索如何处理对象内包含的空格字符，对应 CSS 2.1 版本中的 white-space 属性。取值简单说明如下。

- ➢ collapse：使用一个单一的字符序列呈现空白（或在某些情况下没有字符）。
- ➢ preserve：可以呈现所有空白，换行符将被保留。
- ➢ preserve-breaks：抛弃呈现所有空白，但保留换行符。
- ➢ discard：抛弃呈现所有空白。

☑ white-space

语法说明如下：

```
white-space:normal|pre|nowrap|pre-wrap|pre-line;
```

white-space 初始值为无，适用于所有元素。该属性设置或检索对象内空格字符的处理方式。它是 white-space-collapse 和 textwrap 属性的简便用法，但并没有包含 white-space-collapse 和 text-wrap 属性的所有功能。与 CSS 2.1 版本相比新增了两个属性值。取值简单说明如下。

- ➢ normal：类似 white-space-collapse:collapse;text-wrap:normal;。
- ➢ pre：类似 white-space-collapse:preserve;text-wrap:none;。
- ➢ nowrap：类似 white-space-collapse:collapse;text-wrap:none;。
- ➢ pre-wrap：类似 white-space-collapse:preserve;text-wrap:normal;。
- ➢ pre-line：类似 white-space-collapse:preserve-breaks;text-wrap:normal;。

☑ word-break

语法说明如下：

```
word-break:normal|keep-all|loose|break-strict|break-all;
```

word-break 属性初始值为 normal，适用于所有元素。该属性设置或检索对象内文本的字内换行行为，尤其在出现多种语言时。对于中文，应该使用 break-all。取值简单说明如下。

- normal：根据语言自己的规则，确定换行方式。
- keep-all：同 normal，对于中、日、韩字符不允许字断开。
- loose：类似 normal，但是允许中、日、韩字符在任意位置断开。
- break-strict：类似 normal，但是对于非中、日、韩字符可以允许在任意位置断开。
- break-all：类似 break-strict，除了中、日、韩字符都应遵循 loose 的规则。

☑ text-wrap

语法说明如下：

```
text-wrap:normal|unrestricted|none|suppress;
```

text-wrap 属性的初始值为 normal，适用于所有元素。CSS3 定义文本换行通过 text-wrap 和 word-wrap 两个属性来控制。text-wrap 属性设置或检索对象内文本的换行模式。取值简单说明如下。

- normal：自动换行模式。
- none：不换行模式。
- unrestricted：无限制模式。
- suppress：压制模式。

☑ word-wrap

该属性针对字符换行问题进行处理，设置或检索当前行超过指定容器的边界时是否断开转行。详细说明请参阅下面讲解。

☑ text-align

语法说明如下：

```
text-align:start|end|left|right|center|justify|<string>;
```

text-align 属性初始值为 start，适用于所有元素。该属性设置或检索对象中文本的对齐方式。与 CSS 2.1 版本相比，CSS3 增加了 start、end 和<string>属性值。其中 start 和 end 属性值主要是针对行内元素来说的，即在包含元素的开始位置或尾部位置显示。而<string>属性值主要应用于表格单元格中，将根据某个指定的字符进行对齐。

☑ text-align-last

语法说明如下：

```
text-align-last:start|end|left|right|center|justify;
```

text-align-last 属性初始值 start，适用于所有元素。该属性设置或检索对象中最后一行文本的对齐方式，当 text-align 设置为 justify 时，强制换行的文本对齐方式。

☑ text-justify

语法说明如下：

```
text-justify:auto|inter-word|inter-ideograph|inter-cluster|distribute|kashida|tibetan;
```

text-justify 初始值为 auto，适用于所有元素。该属性设置或检索对象内调整文本使用的对齐方式。只有当 text-align 设置为 justify 时，设置该属性才有效。CSS3 最新版本汲取了 IE 的私有属性 text-justify，

但是重新规划了取值。取值简单说明如下。

- auto：允许浏览器代理用户确定使用的两端对齐方式。
- inter-word：通过增加字之间的空格对齐文本。该行为是对齐所有文本行最快的方法。它的两端对齐行为对段落的最后一行无效。
- inter-ideograph：为表意字文本提供完全两端对齐，用于增加或减少表意字和词间的空格。
- inter-cluster：调整文本无词间空格的行。这种模式的调整是用于优化亚洲语言文档的。
- distribute：通过增加或减少字或字母之间的空格对齐文本。是用于拉丁文字母表两端对齐的最精确格式，适用于东亚文档，尤其是泰文。
- kashida：通过拉长选定点的字符调整文本。这种调整模式是专门为阿拉伯脚本语言提供的。
- tibetan：两端对齐行的方式与 distribute 相同，也同样不包含两段对齐段落的最后一行。适用于表意字文档。该值可能在未来修订中删除。

Note

☑ word-spacing

语法说明如下：

```
word-spacing:normal|<length>|<percentage>;
```

word-spacing 初始值为 normal，适用于所有元素。该属性检索或设置对象中的单词之间插入的空格。其中 percentage 表示根据空格字符（U+0020）的宽度进行计算。单词间距会受对齐调整的影响。

☑ letter-spacing

语法说明如下：

```
letter-spacing:normal|<length>|<percentage>;
```

letter-spacing 初始值为 normal，适用于所有元素。该属性检索或设置对象中字符之间的间隔。该属性将指定的间隔添加到每个文字之后，但最后一个字将被排除在外，字符间距会受对齐调整的影响。

☑ punctuation-trim

语法说明如下：

```
punctuation-trim:none|[start||end||adjacent];
```

punctuation-trim 初始值为 none，适用于所有元素及其内容。该属性检索或设置标点符号的修剪。取值简单说明如下。

- none：不修剪。
- start：根据开始位置的标点符号修剪另一半标点符号。
- end：根据结束位置的标点符号修剪另一半标点符号。
- adjacent：根据相邻位置的标点符号修剪另一半标点符号。

☑ text-emphasis

语法说明如下：

```
text-emphasis:none|[[accent|dot|circle|disc][before|after]?];
```

text-emphasis 初始值为 none，适用于所有元素及其内容。该属性检索或设置重点文本样式。取值简单说明如下。

- none：没有重点标记。
- accent：马克笔画标记。

Note

- dot：点标记。
- circle：空心圆标记。
- disc：实心圆标记。
- before：在顶部标记，或者右侧标记（针对垂直书写的文本）。
- after：在文本底部标记，或者左侧标记（针对垂直书写的文本）。

☑ text-shadow

该属性检索或设置文本阴影，详细说明请参阅下面讲解。

☑ text-outline

语法说明如下：

```
text-outline:none|[<color><length><length>?|<length><length>?<color>];
```

text-outline 初始值为 none，适用于所有元素及其内容。该属性检索或设置文本的外形轮廓。其中第一个长度值表示轮廓的厚度，第二个长度值是可选的，表示模糊半径。轮廓不会覆盖文本本身。

☑ text-indent

语法说明如下：

```
text-indent:[<length>|<percentage>]hanging?;
```

text-indent 初始值为 0，适用于块状元素、行内块状元素或者表格单元格。该属性检索或设置对象中的文本的缩进。其中<percentage>表示根据包含元素的宽度进行计算。

☑ hanging-punctuation

语法说明如下：

```
hanging-punctuation:none|[start||end||end-edge];
```

hanging-punctuation 属性初始值为 none，适用于块状元素、行内块状元素或者表格单元格。该属性检索或设置对象是否悬挂一个标点符号。取值简单说明如下。

- start：标点符号可以挂在第一行开始边缘。
- end：标点符号可以挂在最后一行的末尾的边缘。
- end-edge：标点可能会挂起以外的所有行结束边缘。

13.2.2　溢出文本

CSS3 新增了 text-overflow 属性，该属性可以设置超长文本省略显示。在信息列表中常会遇到栏目的宽度与列表项字符长度不一的矛盾，为了避免超长字符的信息项破坏栏目的布局，就可以使用该属性省略掉多出的字符，而在此之前，实现这种想法一般多需借助 JavaScript 脚本来实现。

text-overflow 属性的基本语法如下所示。

```
text-overflow:clip|ellipsis|ellipsis-word;
```

text-overflow 属性初始值为无，适用于块状元素或行内元素。该属性取值简单说明如下。

☑ clip：属性值表示不显示省略标记（…），而是简单的裁切。

☑ ellipsis：属性值表示当对象内文本溢出时显示省略标记（…），省略标记插入的位置是最后一个字符。

☑ ellipsis-word：表示当对象内文本溢出时显示省略标记（…），省略标记插入的位置是最后一个词（word）。

实际上，text-overflow 属性仅是内容注解，表明当文本溢出时是否显示省略标记，并不具备样式定义的特性。要实现溢出时产生省略号的效果，读者应该再定义两个样式：强制文本在一行内显示（white-space:nowrap）和溢出内容为隐藏（overflow:hidden），只有这样才能实现溢出文本显示省略号的效果。

Note

在早期 W3C 文档（http://www.w3.org/TR/2003/CR-css3-text-20030514/#textoverflow-mode）中，text-overflow 被纳入规范，但是在最新修订的文档（http://www.w3.org/TR/css3-text/）中没有再包含 text-overflow 属性。

由于 W3C 规范放弃了对 text-overflow 属性的支持，所以，Mozilla 类型浏览器也放弃了对该属性的支持。不过 Mozilla developer center 推荐使用-moz-binding 的 CSS 属性进行兼容。Firefox 支持 XUL（XUL，一种 XML 的用户界面语言），这样就可以使用-moz-binding 属性来绑定 XUL 里的 ellipsis 属性了。

【示例】设计固定区域的新闻列表。在下面代码中使用 text-overflow 属性来实现在固定的版块中设计新闻列表有序显示，对于超出指定宽度的新闻项，则通过省略并附加省略号，来避免新闻换行或者撑开版块，演示效果如图 13.15 所示。

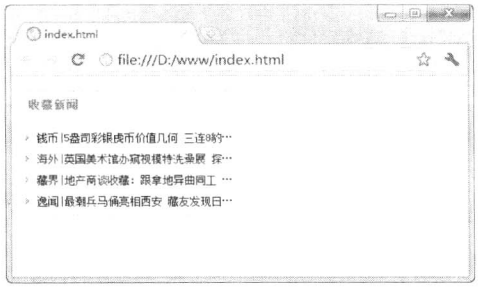


图 13.15　设计固定宽度的新闻栏目

```
<!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></title>
<style type="text/css">
/*定义新闻栏目外框，设置固定宽度*/
dl {
    width:240px;
    border:solid 1px #ccc;
}
/*设计新闻栏目标题行样式*/
dt {
    padding:8px 8px;
    background:#7FECAD url(images/green.gif) repeat-x;
    font-size:13px;
    text-align:left;
    font-weight:bold;
    color:#71790C;
    margin-bottom:12px;
    border-bottom:solid 1px #efefef;
```

Note

```
}
/*设新闻列表项样式*/
dd {
    font-size:0.78em;
    height:1.5em;
    width:220px;
    /*为添加新闻项目符号腾出空间*/
    padding:2px 2px 2px 18px;
    /*以背景方式添加项目符号*/
    background:url(images/icon.gif) no-repeat left 25%;
    margin:2px 0;
    /*为应用 text-overflow 作准备，禁止换行*/
    white-space: nowrap;
    /*为应用 text-overflow 作准备，禁止文本溢出显示*/
    overflow: hidden;
    -o-text-overflow: ellipsis;      /*兼容 Opera*/
    text-overflow: ellipsis;      /*兼容 IE，Safari（Webkit）*/
    -moz-binding: url('ellipsis.xml#ellipsis');      /*兼容 Firefox*/
}
</style>
</head>
<body>
<dl>
    <dt>收藏新闻</dt>
    <dd>钱币|5 盎司彩银虎币价值几何 三连 8 豹子钞市值达千元 </dd>
    <dd>海外|英国美术馆办窥视模特洗澡展 探秘荷兰人体博物馆</dd>
    <dd>藏界|地产商谈收藏：跟拿地异曲同工 拾垃圾的老外</dd>
    <dd>逸闻|最潮兵马俑亮相西安 藏友发现日军内参书籍</dd>
</dl>
</body>
</html>
```

13.2.3　文本换行

首先介绍一下，浏览器本身都自带着让文本自动换行的功能。在浏览器中显示文本时，会让文本在浏览器或 div 元素的右端自动实现换行。对于西方文字来说，浏览器会在半角空格或连字符的地方自动换行，而不会在单词的当中突然换行。对于中文来说，可以在任何一个中文字后面进行换行。

如果中文当中含有西方文字，浏览器也会在半角空格或连字符的地方进行换行，而不会在单词中间强制换行。当中文中含有标点符号时，浏览器总是不可能让标点符号位于一行文字的行首，通常将标点符号以及它前面的一个文字作为一个整体来统一换行。

在 CSS3 中，使用 word-break 属性定义文本自动换行。这原来是 IE 中独自发展出来的属性，在 CSS3 中被 Text 模块采用，现在也得到了 Chrome 和 Safari 浏览器的支持。实际上，IE 自定义了多个换行处理属性——line-break、word-break 和 word-wrap，另外 CSS1 定义了 white-space。这几个属性简单比较如下。

- ☑ line-break：专门负责控制日文换行。
- ☑ word-wrap：可以控制换行。当属性取值 break-word 时，将强制换行，对于中文文本没有任何问题，英文语句也没问题，但是对于长串的英文就不起作用，word-wrap:breakword 控制

是否断词，而不是断字符。

☑ word-break：主要针对亚洲语言和非亚洲语言进行控制换行。当属性取值 break-all 时，可以允许非亚洲语言文本行的任意字内断开。当属性值为 keep-all 时，表示对于中文、韩文、日文不允许字断开。

☑ white-space：具有格式化文本作用，当属性取值为 nowrap 时，表示强制在同一行内显示所有文本，而属性值为 pre 时，表示显示预定义文本格式。

word-wrap 属性的基本语法如下所示。

```
word-wrap:normal|break-word;
```

word-wrap 属性初始值为 normal，适用于所有元素。该属性取值简单说明如下：

☑ normal：值表示控制连续文本换行。

☑ break-word：值表示内容将在边界内换行。如果需要，词内换行（word-break）也会发生。

在 IE 浏览器下，使用 word-wrap:break-word;声明可以确保所有文本正常显示。在 Firefox 浏览器下，中文不会出任何问题，英文语句也不会出问题，但是长串英文会出问题。为了解决长串英文，一般使用 word-wrap:break-word;和 word-break:break-all;声明结合使用。但是，这种方法会导致普通的英文语句中的单词被断开显示（IE 下也是）。现在的问题主要是长串英文和英文单词被断开的问题。

为了解决这个问题，可使用 word-wrap:break-word;overflow:hidden;，而不是 wordwrap:break-word; word-break:break-all;。word-wrap:break-word;overflow:auto;在 IE 下没有任何问题，但是在 Firefox 下，长串英文单词就会被遮住部分内容。

word-wrap 属性没有被广泛地支持，特别是 Firefox 和 Opera 浏览器对其支持比较消极，这是因为在早期的 W3C 文本模型中（http://www.w3.org/TR/2003/CR-css3-text-20030514/）放弃了对其支持，而是定义了 wrap-option 属性代替 word-wrap 属性，但是在最新的文本模式中（http://www.w3.org/TR/css3-text/）继续支持该属性，并重定义了属性值。

【示例】在表格设计中，标题行常被撑开，影响了浏览体验。解决这个问题方法有很多种，可以固定表格的宽度，或者通过下面的方法来进行设计。一方面为 th 元素添加 nowrap 属性，同时借助 CSS 换行技术进行处理，演示效果如图 13.16 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title></title>
<style type="text/css">
h1 { font-size:16px; }
table {
    width:100%;
    font-size:12px;
    empty-cells:show;
    border-collapse:collapse;
    border-collapse: collapse;
    margin:0 auto;
    border:1px solid #cad9ea;
    color:#666;
    /*定义表格在浏览器端逐步解析逐步呈现*/
```

Note

```
        table-layout:fixed;
        /*禁止词断开显示*/
        word-break:keep-all;
        /*允许内容顶开指定的容器边界，如果声明 word-wrap:breakword;，则在 IE 浏览器中会出现换行显示，
破坏了整个标题行的样式*/
        word-wrap:normal;
        /*强迫在一行内显示*/
        white-space:nowrap;
    }
    th {
        background-image: url(images/th_bg1.gif);
        background-repeat:repeat-x;
        height:30px;
        overflow:hidden;
    }
    td { height:20px; }
    td，th {
        border:1px solid #cad9ea;
        padding:0 1em 0;
    }
    tr:nth-child(even) { background-color:#f5fafe;}
    </style>
    </head>
    <body>
    <table>
        <tr>
            <th nowrap="nowrap">排名</th>
            <th nowrap="nowrap">校名</th>
            <th nowrap="nowrap">总得分</th>
            <th nowrap="nowrap">人才培养总得分</th>
            <th nowrap="nowrap">研究生培养得分</th>
            <th nowrap="nowrap">本科生培养得分</th>
            <th nowrap="nowrap">科学研究总得分</th>
            <th nowrap="nowrap">自然科学研究得分</th>
            <th nowrap="nowrap">社会科学研究得分</th>
            <th nowrap="nowrap">所属省份</th>
            <th nowrap="nowrap">分省排名</th>
            <th nowrap="nowrap">学校类型</th>
        </tr>
        <tr>
            <td>1</td>
            <td>清华大学</td>
            <td>296.77</td>
            <td>128.92</td>
            <td>93.83</td>
            <td>35.09</td>
            <td>167.85</td>
            <td>148.47</td>
            <td>19.38</td>
            <td width="16">京</td>
```

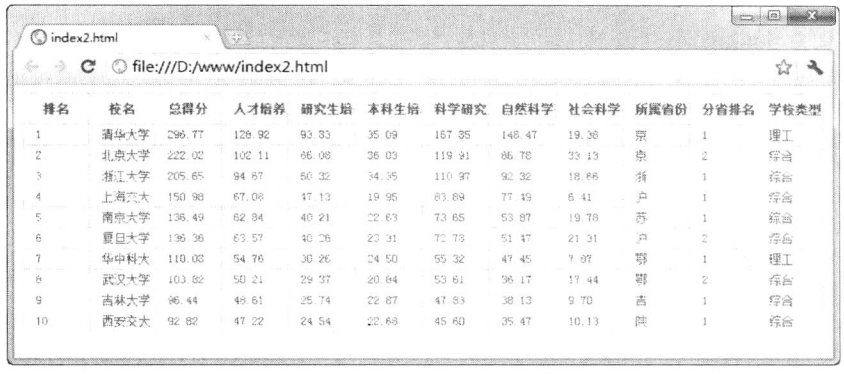
Note

```
        <td width="12">1 </td>
        <td>理工</td>
    </tr>
    <tr>
        <td>2</td>
        <td>北京大学</td>
        <td>222.02</td>
        <td>102.11</td>
        <td>66.08</td>
        <td>36.03</td>
        <td>119.91</td>
        <td>86.78</td>
        <td>33.13</td>
        <td>京</td>
        <td>2 </td>
        <td>综合</td>
    </tr>
    <tr>
        <td>3</td>
        <td>浙江大学</td>
        <td>205.65</td>
        <td>94.67</td>
        <td>60.32</td>
        <td>34.35</td>
        <td>110.97</td>
        <td>92.32</td>
        <td>18.66</td>
        <td>浙 </td>
        <td>1</td>
        <td>综合</td>
    </tr>
    <tr>
        <td>4</td>
        <td>上海交大</td>
        <td>150.98</td>
        <td>67.08</td>
        <td>47.13</td>
        <td>19.95</td>
        <td>83.89</td>
        <td>77.49</td>
        <td>6.41</td>
        <td>沪 </td>
        <td>1</td>
        <td>综合</td>
    </tr>
    <tr>
        <td>5</td>
        <td>南京大学</td>
        <td>136.49</td>
        <td>62.84</td>
```

Note

```
        <td>40.21</td>
        <td>22.63</td>
        <td>73.65</td>
        <td>53.87</td>
        <td>19.78</td>
        <td>苏</td>
        <td>1</td>
        <td>综合</td>
    </tr>
    <tr>
        <td>6</td>
        <td>复旦大学</td>
        <td>136.36</td>
        <td>63.57</td>
        <td>40.26</td>
        <td>23.31</td>
        <td>72.78</td>
        <td>51.47</td>
        <td>21.31</td>
        <td>沪</td>
        <td>2</td>
        <td>综合</td>
    </tr>
    <tr>
        <td>7</td>
        <td>华中科大</td>
        <td>110.08</td>
        <td>54.76</td>
        <td>30.26</td>
        <td>24.50</td>
        <td>55.32</td>
        <td>47.45</td>
        <td>7.87</td>
        <td>鄂 </td>
        <td>1</td>
        <td>理工</td>
    </tr>
    <tr>
        <td>8</td>
        <td>武汉大学</td>
        <td>103.82</td>
        <td>50.21</td>
        <td>29.37</td>
        <td>20.84</td>
        <td>53.61</td>
        <td>36.17</td>
        <td>17.44</td>
        <td>鄂</td>
        <td>2</td>
        <td>综合</td>
```

```
        </tr>
        <tr>
            <td>9</td>
            <td>吉林大学</td>
            <td>96.44</td>
            <td>48.61</td>
            <td>25.74</td>
            <td>22.87</td>
            <td>47.83</td>
            <td>38.13</td>
            <td>9.70</td>
            <td>吉</td>
            <td>1</td>
            <td>综合</td>
        </tr>
        <tr>
            <td>10</td>
            <td>西安交大</td>
            <td>92.82</td>
            <td>47.22</td>
            <td>24.54</td>
            <td>22.68</td>
            <td>45.60</td>
            <td>35.47</td>
            <td>10.13</td>
            <td>陕</td>
            <td>1</td>
            <td>综合</td>
        </tr>
    </table>
    </body>
    </html>
```



排名	校名	总得分	人才培养	研究生培	本科生培	科学研究	自然科学	社会科学	所属省份	分省排名	学校类型
1	清华大学	296.77	126.92	93.33	35.09	167.85	148.47	19.38	京	1	理工
2	北京大学	222.02	102.11	66.08	36.03	119.91	86.78	33.13	京	2	综合
3	浙江大学	205.65	94.67	60.32	34.35	110.97	92.32	18.66	浙	1	综合
4	上海交大	150.98	67.08	47.13	19.95	83.89	77.49	6.41	沪	1	综合
5	南京大学	136.49	62.84	40.21	22.63	73.65	53.87	19.78	苏	1	综合
6	复旦大学	136.36	63.57	40.26	23.31	72.78	51.47	21.31	沪	2	综合
7	华中科大	110.03	54.76	30.26	24.50	55.32	47.45	7.87	鄂	1	理工
8	武汉大学	103.82	50.21	29.37	20.84	53.61	36.17	17.44	鄂	2	综合
9	吉林大学	96.44	48.61	25.74	22.87	47.83	38.13	9.70	吉	1	综合
10	西安交大	92.82	47.22	24.54	22.68	45.60	35.47	10.13	陕	1	综合

图 13.16　禁止表格标题文本换行显示

通过手工添加这样一行属性，确保在不同浏览器中都能够很好地单行显示。如果 th 元素定义宽度，该属性将不再起作用。

13.3　设计颜色样式

Note

在 CSS3 之前，只能使用 RGB 模式定义颜色值，只能通过 opacity 属性设置元素的不透明度。CSS3 增加了 3 种颜色值定义模式：RGBA 颜色值、HSL 颜色值和 HSLA 颜色值，并且允许通过对 RGBA 颜色值和 HSLA 颜色值设定 Alpha 通道的方法来更加容易地实现将半透明文字与图像互相重叠的效果。

13.3.1　使用 RGBA 颜色值

RGBA 色彩模式是 RGB 色彩模式的扩展，它在红、绿、蓝三原色通道基础上增加了不透明度参数。其语法格式如下所示。

```
rgba(r,g,b,<opacity>)
```

其中 r、g、b 分别表示红色、绿色、蓝色 3 种原色所占的比重。r、g、b 的值可以是正整数或百分数。正整数值的取值范围为 0~255，百分数值的取值范围为 0.0%~100.0%。超出范围的数值将被截至其最接近的取值极限。注意，并非所有浏览器都支持使用百分数值。第四个参数<opacity>表示不透明度，取值在 0~1 之间。

【示例】设计带阴影边框的表单。在设计阴影或者其他效果的边框时，一般借助背景图片来实现，这是因为 CSS 无法实现这种效果。使用 CSS3 新增加的 box-shadow 属性，然后使用 RGBA 颜色模式为表单元素设置半透明度的阴影，从而实现一种润边形式阴影效果，预览效果如图 13.17 所示。rgba(0,0,0,0.1)表示不透明度为 0.1 的黑色，这里不宜直接设置为浅灰色，因为对于非白色背景来说，灰色发虚，而半透明效果可以避免这样情况。本示例代码如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
/*统一输入域样式*/
input，textarea {
    padding: 4px;
    border: solid 1px #E5E5E5;
    outline: 0;
    font: normal 13px/100% Verdana，Tahoma，sans-serif;
    width: 200px;
    background: #FFFFFF;
    /*设置边框阴影效果*/
    box-shadow: rgba(0,0,0,0.1) 0px 0px 8px;
    /*兼容 Mozilla 类型浏览器，如 FF*/
    -moz-box-shadow: rgba(0,0,0,0.1) 0px 0px 8px;
    /*兼容 Webkit 引擎，如 Chorme 和 Safari 等*/
    -webkit-box-shadow: rgba(0,0,0,0.1) 0px 0px 8px;
```

```
}
input:hover,textarea:hover,input:focus,textarea:focus { border-color: #C9C9C9; }
/*定义标签样式*/
label {
    margin-left: 10px;
    color: #999999;
    display:block; /*以块状显示，实现分行显示*/
}
.submit input {
    width:auto;
    padding: 9px 15px;
    background: #617798;
    border: 0;
    font-size: 14px;
    color: #FFFFFF;
}
</style>
</head>
<body>
<form>
    <p class="name">
        <label for="name">姓名</label>
        <input type="text" name="name" id="name" />
    </p>
    <p class="email">
        <label for="email">邮箱</label>
        <input type="text" name="email" id="email" />
    </p>
    <p class="submit">
        <input type="submit" value="提交" />
    </p>
</form>
</body>
</html>
```

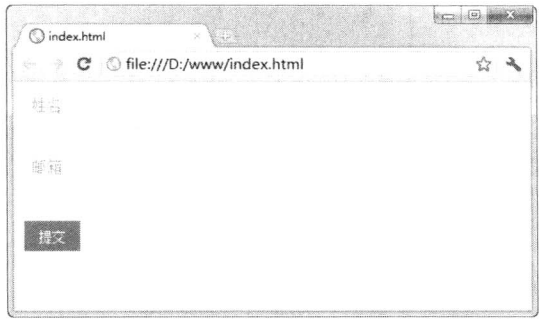


图 13.17　设计带有阴影边框的表单效果

到目前为止，Safari 浏览器、Firefox 浏览器、Chrome 浏览器及 Opera 浏览器都支持 RGBA 颜色。在 IE 等不支持 RGBA 颜色的浏览器中，将忽视对 RGBA 颜色值的指定。

13.3.2　使用 HSL 颜色值

Note

CSS3 新增加了 HSL 颜色表现方式（http://www.w3.org/TR/css3-color/）。HSL 色彩模式是工业界的一种颜色标准，它通过对色调（H）、饱和度（S）和亮度（L）3 个颜色通道的变化以及它们相互之间的叠加来获得各种颜色。这个标准几乎包括了人类视觉所能感知的所有颜色，在屏幕上可以重现 16777216 种颜色，是目前运用最广的颜色系统之一。

在 CSS3 中，HSL 色彩模式的表示语法如下：

```
hsl(<length>,<percentage>,<percentage>)
```

hsl()函数的 3 个参数说明如下。

- ☑ <length>：表示色调（Hue）。Hue 衍生于色盘，取值可以为任意数值，其中 0（或 360、-360）表示红色，60 表示黄色，120 表示绿色，180 表示青色，240 表示蓝色，300 表示洋红，当然可取设置其他数值来确定不同的颜色。
- ☑ <percentage>：表示饱和度（Saturation），也就是说该色彩被使用了多少，或者说颜色的深浅程度、鲜艳程度。取值为 0%~100%之间的值。其中 0%表示灰度，即没有使用该颜色；100%饱和度最高，即颜色最艳。
- ☑ <percentage>：表示亮度（Lightness）。取值为 0%~100%之间的值，其中 0%最暗，显示为黑色，50%表示均值，100%最亮，显示为白色。

【示例】设计颜色表。先选择一个色值，然后利用调整颜色的饱和度和亮度比重，分别设计不同的配色方案表。在网页设计中，利用这种方法就可以根据网页需要选择恰当的配色方案。使用 HSL 颜色表现方式，可以很轻松地设计网页配色方案表，模拟演示效果如图 13.18 所示。

```
<!DOCTYPEhtmlPUBLIC"-//W3C//DTDXHTML1.0Transitional//EN""http://www.w3.org/TR/xhtml1/
DTD/xhtml1-transitional.dtd">
<htmlxmlns="http://www.w3.org/1999/xhtml">
<head>
<metahttp-equiv="Content-Type"content="text/html;charset=utf-8"/>
<title></title>
<styletype="text/css">
table{
    border:solid1pxred;
    background:#eee;
    padding:6px;
}
th{
    color:red;
    font-size:12px;
    font-weight:normal;
}
td{
    width:80px;
    height:30px;
}
/*第 1 行*/
tr:nth-child(4) td:nth-of-type(1){background:hsl(0,100%,100%);}/*第 1 列*/
tr:nth-child(4) td:nth-of-type(2){background:hsl(0,75%,100%);}/*第 2 列*/
```

Note

```
tr:nth-child(4) td:nth-of-type(3){background:hsl(0,50%,100%);}/*第 3 列*/
tr:nth-child(4) td:nth-of-type(4){background:hsl(0,25%,100%);}/*第 4 列*/
tr:nth-child(4) td:nth-of-type(5){background:hsl(0,0%,100%);}/*第 5 列*/
/*第 2 行*/
tr:nth-child(5) td:nth-of-type(1){background:hsl(0,100%,88%);}/*第 1 列*/
tr:nth-child(5) td:nth-of-type(2){background:hsl(0,75%,88%);}/*第 2 列*/
tr:nth-child(5) td:nth-of-type(3){background:hsl(0,50%,88%);}/*第 3 列*/
tr:nth-child(5) td:nth-of-type(4){background:hsl(0,25%,88%);}/*第 4 列*/
tr:nth-child(5) td:nth-of-type(5){background:hsl(0,0%,88%);}/*第 5 列*/
/*第 3 行*/
tr:nth-child(6) td:nth-of-type(1){background:hsl(0,100%,75%);}/*第 1 列*/
tr:nth-child(6) td:nth-of-type(2){background:hsl(0,75%,75%);}/*第 2 列*/
tr:nth-child(6) td:nth-of-type(3){background:hsl(0,50%,75%);}/*第 3 列*/
tr:nth-child(6) td:nth-of-type(4){background:hsl(0,25%,75%);}/*第 4 列*/
tr:nth-child(6) td:nth-of-type(5){background:hsl(0,0%,75%);}/*第 5 列*/
/*第 4 行*/
tr:nth-child(7) td:nth-of-type(1){background:hsl(0,100%,63%);}/*第 1 列*/
tr:nth-child(7) td:nth-of-type(2){background:hsl(0,75%,63%);}/*第 2 列*/
tr:nth-child(7) td:nth-of-type(3){background:hsl(0,50%,63%);}/*第 3 列*/
tr:nth-child(7) td:nth-of-type(4){background:hsl(0,25%,63%);}/*第 4 列*/
tr:nth-child(7) td:nth-of-type(5){background:hsl(0,0%,63%);}/*第 5 列*/
/*第 5 行*/
tr:nth-child(8) td:nth-of-type(1){background:hsl(0,100%,50%);}/*第 1 列*/
tr:nth-child(8) td:nth-of-type(2){background:hsl(0,75%,50%);}/*第 2 列*/
tr:nth-child(8) td:nth-of-type(3){background:hsl(0,50%,50%);}/*第 3 列*/
tr:nth-child(8) td:nth-of-type(4){background:hsl(0,25%,50%);}/*第 4 列*/
tr:nth-child(8) td:nth-of-type(5){background:hsl(0,0%,50%);}/*第 5 列*/
/*第 6 行*/
tr:nth-child(9) td:nth-of-type(1){background:hsl(0,100%,38%);}/*第 1 列*/
tr:nth-child(9) td:nth-of-type(2){background:hsl(0,75%,38%);}/*第 2 列*/
tr:nth-child(9) td:nth-of-type(3){background:hsl(0,50%,38%);}/*第 3 列*/
tr:nth-child(9) td:nth-of-type(4){background:hsl(0,25%,38%);}/*第 4 列*/
tr:nth-child(9) td:nth-of-type(5){background:hsl(0,0%,38%);}/*第 5 列*/
/*第 7 行*/
tr:nth-child(10) td:nth-of-type(1){background:hsl(0,100%,25%);}/*第 1 列*/
tr:nth-child(10) td:nth-of-type(2){background:hsl(0,75%,25%);}/*第 2 列*/
tr:nth-child(10) td:nth-of-type(3){background:hsl(0,50%,25%);}/*第 3 列*/
tr:nth-child(10) td:nth-of-type(4){background:hsl(0,25%,25%);}/*第 4 列*/
tr:nth-child(10) td:nth-of-type(5){background:hsl(0,0%,25%);}/*第 5 列*/
/*第 8 行*/
tr:nth-child(11) td:nth-of-type(1){background:hsl(0,100%,13%);}/*第 1 列*/
tr:nth-child(11) td:nth-of-type(2){background:hsl(0,75%,13%);}/*第 2 列*/
tr:nth-child(11) td:nth-of-type(3){background:hsl(0,50%,13%);}/*第 3 列*/
tr:nth-child(11) td:nth-of-type(4){background:hsl(0,25%,13%);}/*第 4 列*/
tr:nth-child(11) td:nth-of-type(5){background:hsl(0,0%,13%);}/*第 5 列*/
/*第 9 行*/
tr:nth-child(12) td:nth-of-type(1){background:hsl(0,100%,0%);}/*第 1 列*/
tr:nth-child(12) td:nth-of-type(2){background:hsl(0,75%,0%);}/*第 2 列*/
tr:nth-child(12) td:nth-of-type(3){background:hsl(0,50%,0%);}/*第 3 列*/
tr:nth-child(12) td:nth-of-type(4){background:hsl(0,25%,0%);}/*第 4 列*/
```

Note

```
tr:nth-child(12) td:nth-of-type(5){background:hsl(0,0%,0%);}/*第 5 列*/
</style>
</head>
<body>
<table class="hslexample">
    <tbody>
        <tr>
            <th> </th>
            <th colspan="5">色相：H=0 Red </th>
        </tr>
        <tr>
            <th> </th>
            <th colspan="5">饱和度 (&rarr;)</th>
        </tr>
        <tr>
            <th>亮度 (&darr;)</th>
            <th>100% </th>
            <th>75% </th>
            <th>50% </th>
            <th>25% </th>
            <th>0% </th>
        </tr>
        <tr>
            <th>100 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>88 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>75 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>63 </th>
            <td> </td>
            <td> </td>
```

Note

```
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>50 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>38 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>25 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>13 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <th>0 </th>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
    </tbody>
</table>
</body>
</html>
```

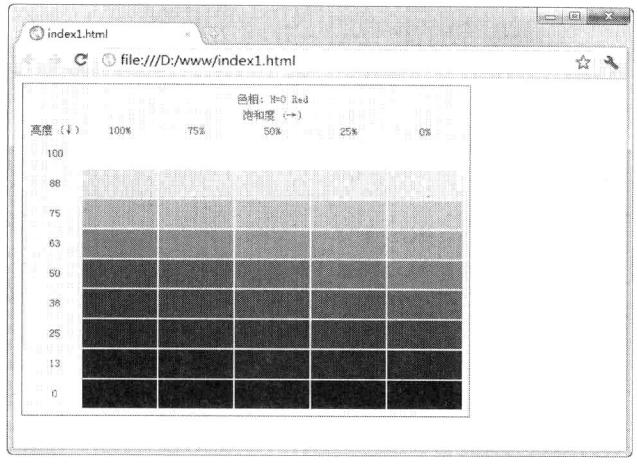


图 13.18　使用 HSL 颜色值设计颜色表

在上面代码中，tr:nth-child(4) td:nth-of-type(1)中的 tr:nth-child(4)子选择器表示选择行，而 td:nth-of-type(1)表示选择单元格（列）。其他行选择器结构依此类推。在 background:hsl(0,0%,0%);声明中，hsl()函数的第 1 个参数值 0 表示色相值，第 2 个参数值 0%表示饱和度，第 3 个参数值 0%表示亮度。

13.3.3　使用 HSLA 颜色值

HSLA 色彩模式是 HSL 色彩模式的扩展，在色相、饱和度、亮度三要素基础上增加了不透明度参数。使用 HSLA 色彩模式，可以定义不同透明效果。其语法格式如下。

```
hsla(<length>,<percentage>,<percentage>,<opacity>)
```

其中前 3 个参数与 hsl()函数参数含义和用法相同，第 4 个参数<opacity>表示不透明度，取值在 0~1 之间。

【示例】设计渐变色。本示例通过递减 HSLA 颜色值的不透明度，实现渐变色效果，如图 13.19 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>HSLA Color</title>
<style type="text/css">
li { height: 18px; }
li:nth-child(1) { background: hsla(120,50%,50%,0.1); }
li:nth-child(2) { background: hsla(120,50%,50%,0.2); }
li:nth-child(3) { background: hsla(120,50%,50%,0.3); }
li:nth-child(4) { background: hsla(120,50%,50%,0.4); }
li:nth-child(5) { background: hsla(120,50%,50%,0.5); }
li:nth-child(6) { background: hsla(120,50%,50%,0.6); }
li:nth-child(7) { background: hsla(120,50%,50%,0.7); }
li:nth-child(8) { background: hsla(120,50%,50%,0.8); }
```

```
li:nth-child(9) { background: hsla(120,50%,50%,0.9); }
li:nth-child(10) { background: hsla(120,50%,50%,1); }
</style>
</head>
<body>
<ol>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ol>
</body>
</html>
```

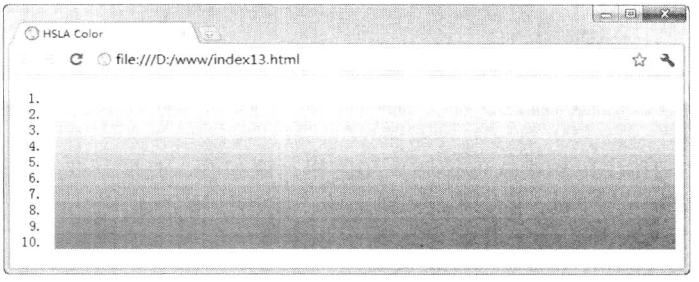


图 13.19　设计渐变色

13.3.4　定义 opacity 属性

在 CSS3 中，除了使用 Alpha 通道的方法来设定透明度外，也可以通过 opacity 属性来设定透明度。目前支持 opacity 属性的浏览器有 Firefox、Safari、Opera 和 Chrome。opacity 属性能够设置任何元素呈现为半透明效果，opacity 属性的基本语法如下所示。

```
opacity:<alphavalue>|inherit;
```

opacity 属性初始值为 1，适用于所有元素。取值简单说明如下：

- ☑ <alphavalue>为由浮点数字和单位标识符组成的长度值。不可为负值，默认值为 1。opacity 取值为 1 时，元素是完全不透明的，反之，取值为 0 时，元素是完全透明的。0~1 之间的任何值都表示该元素的不透明程度。
- ☑ inherit 表示继承，即继承父元素的不透明性。

【示例 1】设计灯箱广告背景布。通过设计 对象铺满整个窗口，设置黑色背景，不透明度为 0.7，实现覆盖整个页面，设计一种半透明的遮罩效果，然后通过 CSS 定位技术设计 对象显示在其上面。代码如下，演示效果如图 13.20 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Note

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body {
    margin:0;
    padding:0;
}
div { position:absolute; }
.bg {
    width:100%;
    height:100%;
    background:#000;
    opacity:0.7;
    filter:alpha(opacity=70);
}
.lightbox {
    left:50px;
    top:50px;
}
.lightbox img { border:solid 4px #fff; }
</style>
</head>
<body>
<div class="web"><img src="images/bg2.jpg" width="1259" height="630" /></div>
<div class="bg"></div>
<div class="lightbox"><img src="images/bg1.jpg" width="80%" /></div>
</body>
</html>
```

图 13.20　设计半透明的背景布效果

opacity 属性是 CSS 中专门用来指定透明度的一个属性，取值范围也在 0~1 之间，0 表示完全透明，1 表示不透明。使用 Alpha 通道对元素设定透明度时，可以单独针对元素的背景色和文字颜色等来指定透明度，而 opacity 属性只能指定整个元素的透明度。

Note

【示例 2】下面来看一个 Alpha 通道与 opacity 属性结合使用的示例，在该示例中，有 4 个 div 元素，其背景色均为绿色，其中第 1 个 div 元素不指定透明度，第 2 个 div 元素使用 Alpha 通道指定背景色的透明度为 0.5，第 3 个 div 元素使用 Alpha 通道指定背景色与文字颜色的透明度均为 0.5，第 4 个 div 元素使用 opacity 属性指定元素的透明度为 0.5。示例代码如下，演示效果如图 13.21 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body { background-image: url(images/bg3.jpg); }
div {
    width: 100%;
    height: 100px;
    color: white;
    font-size: 48px;
}
div#div1 {
    background-color: rgb(0,255,100);
    color: rgb(255,255,255);
}
div#div2 {
    background-color: rgba(0,255,100,0.5);
    color: rgb(255,255,255);
}
div#div3 {
    background-color: rgba(0,255,100,0.5);
    color: rgba(255,255,255,0.5);
}
div#div4 {
    background-color: rgb(0,255,100);
    color: rgb(255,255,255);
    opacity: 0.5;
}
</style>
</head>
<body>
<div id="div1">人生易老天难老，</div>
<div id="div2">岁岁重阳，今又重阳，战地黄花分外香。</div>
<div id="div3">一年一度秋风劲，</div>
<div id="div4">不似春光，胜似春光，寥廓江天万里霜。</div>
</body>
</html>
```

图 13.21　Alpha 通道与 opacity 属性混合使用

从图 13.21 中可以看出，第 2 个 div 元素的背景色使用 Alpha 通道时，并不会对文字产生影响，如果要让该元素的文字颜色也变成半透明，需要像第 3 个 div 元素那样同时对背景色和文字颜色使用 Alpha 通道。但是，在第 4 个 div 元素的样式代码中，因为使用一次 opacity 属性，文字颜色和背景色都变成了半透明的。

13.3.5　定义 transparent 颜色值

如果将颜色值指定为 transparent，则会将背景、文字或边框等的颜色设定为完全透明，相当于使用了值为 0 的 alpha 通道。在 CSS1 中，只能在 background-color 属性中指定 transparent 值。在 CSS2 中，可以在 background-color 及 border-color 属性中指定 transparent 值。在 CSS3 中，可以在一切指定颜色值的属性中指定 transparent 值。现在，transparent 值已经得到 Firefox、Safari、Opera 及 Chrome 等浏览器的支持。

【示例】本示例为 transparent 值的一个使用示例，该示例中有 3 个 div 元素，第 1 个 div 元素的背景色被设定为 transparent 值，第 2 个 div 元素的边框颜色被设定为 transparent 值，第 3 个 div 元素的文字颜色被设定为 transparent 值。3 个 div 元素的背景色均为白色，边框均为黄色，文字均为黑色，整个网页的背景颜色被指定为粉红色。示例代码如下，演示效果如图 13.22 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body { background-image: url(images/bg3.jpg); }
div {
    background-color: white;
    border: solid 3px yellow;
    width: 100%;
    height: 100px;
}
div#div1 { background-color: transparent; }
```

```
div#div2 { border-color: transparent; }
div#div3 { color: transparent; }
</style>
</head>
<body>
<div id="div1">人生易老天难老，</div>
<div id="div2">岁岁重阳，今又重阳，战地黄花分外香。</div>
<div id="div3">一年一度秋风劲，不似春光，胜似春光，寥廓江天万里霜。</div>
</body>
</html>
```

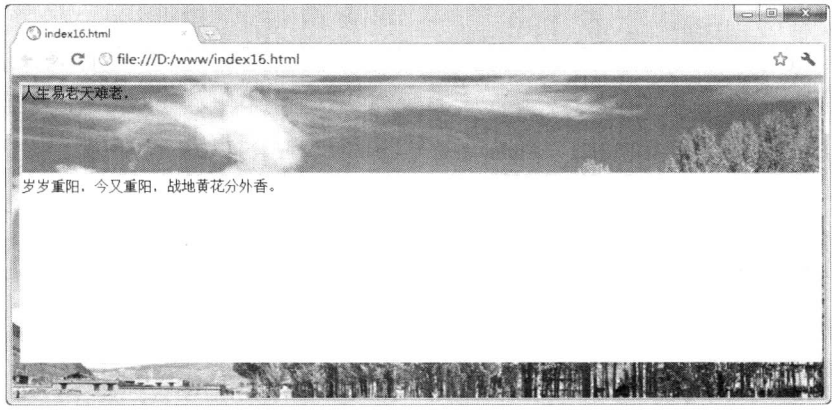


图 13.22　定义 transparent 颜色值

这段代码的运行到此为止，IE 对 transparent 值的支持并不完全，上面代码在 IE8 中运行时，div 元素的背景与边框将会被设定为透明，但是文字颜色不会被设定为透明。

第14章

背景和边框

（视频讲解：1小时09分钟）

CSS3 对原有的盒模型功能也进行了完善。例如，增强了元素边框和背景样式的控制能力（http://www.w3.org/TR/css3-background/），新增了不少 UI 特性（http://www.w3.org/TR/css3-ui/），即用户界面设置问题。

本章将介绍 CSS3 中与背景和边框相关的一些样式，其中包括与背景相关的几个属性，以及如何在一个元素的背景中使用多个图像文件，如何绘制圆角边框，如何给元素添加图像边框。

14.1 设计多色边框

Note

border-color 属性可以设置边框的颜色，CSS3 增强了这个属性的功能，使用它可以为边框设置更多的颜色，从而方便设计师设计渐变等炫丽的边框效果。border-color 属性的基本语法如下所示。

```
border-color:<color>;
```

border-color 属性适用于所有元素，初始值无。<color>可以为任意合法的颜色值或颜色值列表，支持不透明参数设置，与 CSS2 中的 border-color 属性可以混合使用，当为该属性设置一个颜色值时，则表示为边框设置纯色，如果设置 n 个颜色值，且边框宽度为 n 像素，那么就可以在该边框上使用 n 种颜色，每种颜色显示 1 像素的宽度。如果边框宽度是 10 个像素，但是只声明了 5 种颜色，那么最后一个颜色将被添加到剩下的宽度。

为了避免与 border-color 属性原功能（即定义边框颜色）发生冲突，CSS3 在这个属性基础上派生了 4 个边框颜色属性。

- ☑ border-top-color：定义指定元素顶部边框的色彩。
- ☑ border-right-color：定义指定元素右侧边框的色彩。
- ☑ border-bottom-color：定义指定元素底部边框的色彩。
- ☑ border-left-color：定义指定元素左侧边框的色彩。

14.1.1 用法详解

使用上述 4 个属性可分别定义元素 4 条边的多色边框。CSS3 对于 border-color 增强的功能并没有得到各大主流浏览器的响应支持，目前仅有 Mozilla Gecko 引擎支持-webkit-border-color 私有属性，且该属性只能够使用间接方法实现。

【示例】本示例演示了如何使用 border-color 属性定义渐变效果的边框效果，示例演示效果如图 14.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    border: 50px solid #dedede;
    height: 100px;
    width: 600px;
    /*兼容 Mozilla Gecko 引擎*/
    -moz-border-bottom-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
    -moz-border-top-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
    -moz-border-left-colors: #100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
    -moz-border-right-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
    /*标准用法*/
    border-bottom-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
```

```
        border-top-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
        border-left-colors: #100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
        border-right-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00;
    }
    </style>
    </head>
    <body>
    <div></div>
    </body>
    </html>
```

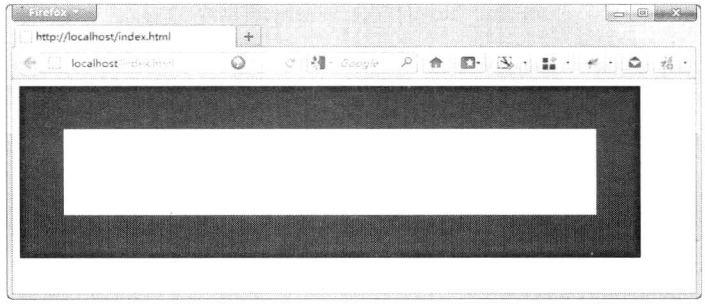


图 14.1　定义渐变边框效果

如果直接在 border-color 属性中进行定义，就会引发歧义，导致浏览器无法正确解析这项声明，最终显示为 border: 50px solid #dedede;声明的边框颜色，如图 14.2 所示。

```
<style type="text/css">
div {
    border: 50px solid #dedede;
    height: 100px;
    width: 600px;
    /*兼容 Mozilla Gecko 引擎*/
    -moz-border-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00 #100 #200 #300
      #400 #500 #600 #700 #800 #900 #a00 #100 #200 #300 #400 #500 #600 #700 #800 #900 #a00 #100
      #200 #300 #400 #500 #600 #700 #800 #900 #a00;
    /*标准用法*/
    border-colors:#100 #200 #300 #400 #500 #600 #700 #800 #900 #a00 #100 #200 #300 #400
      #500 #600 #700 #800 #900 #a00 #100 #200 #300 #400 #500 #600 #700 #800 #900 #a00 #100 #200
      #300 #400 #500 #600 #700 #800 #900 #a00;
}
</style>
```

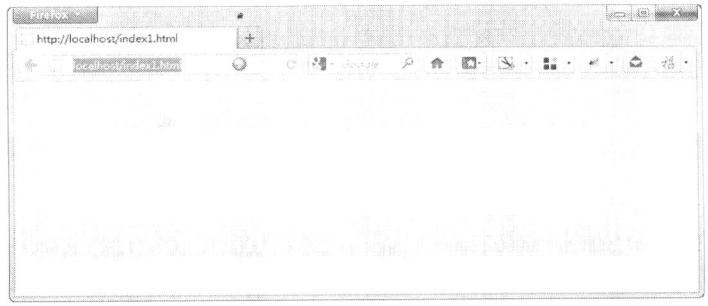


图 14.2　无效的用法

14.1.2 案例实战

Note

【示例】本示例借助 border-color 属性模拟立体效果的边框，使用深色和浅色交错设计，营造凸凹的立体效果。效果如图 14.3 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-color</title>
<style type="text/css">
div {
      border: 2px solid #dedede;
      height: 60px;
      width: 200px;
      background:url(images/001.gif);
      -moz-border-right-colors:#333 #aaa;
      -moz-border-bottom-colors:#333 #aaa;
      -moz-border-top-colors:#aaa #666;
      -moz-border-left-colors:#aaa #666;

}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

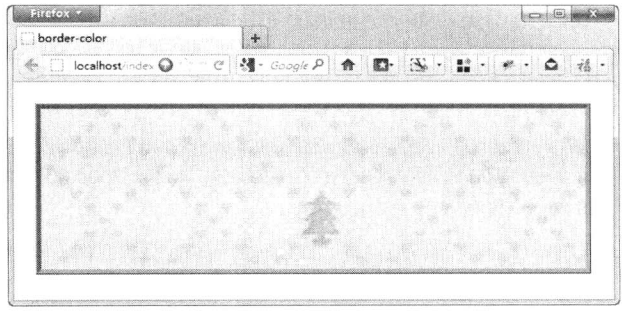


图 14.3　定义立体边框效果

14.2　设计边框背景

在 CSS3 之前，如果要使用图像边框，元素的长或宽是随时可变的，用户通常采用的做法是让元素的每条边单独使用一幅图像文件，但是这种做法也有缺点：一方面是比较麻烦，另一方面是页面上使用的元素比较多。

针对这种情况，CSS3 中增加了一个 border-image 属性，可以让处于随时变化状态的元素的长或宽的边框统一使用一个图像文件来绘制。使用 border-image 属性，会让浏览器在显示图像边框时，自动将所使用到的图像分割为 9 部分进行处理，这样就不需要用户再另外进行人工处理了。另外，页面中也不需要因此而使用较多的元素了。

Note

border-image 属性能够模拟 background-image 属性功能，且功能更加强大，该属性的基本语法如下所示。

```
border-image:none | <image> [ <number> | <percentage>]{1,4} [ / <border-width>{1,4}
```

取值说明如下。

- ☑ none：默认值，表示边框无背景图。
- ☑ <image>：使用绝对或相对 URL 地址指定边框的背景图像。
- ☑ <number>：设置边框宽度或者边框背景图像大小，使用固定像素值表示。
- ☑ <percentage>：设置边框背景图像大小，用百分比表示。

注意：border-image 属性适用于所有元素，除了 border-collapse 属性值为 collapse 的 table 元素。为了方便灵活使用，CSS3 允许 border-image 属性复合定义边框背景样式，同时还派生了众多子属性。一方面，CSS3 将 border-image 分成了 9 部分，使用 9 个子属性分别定义特定方位上边框的背景图像。

- ☑ border-top-image：定义顶部边框背景图像。
- ☑ border-right-image：定义右侧边框背景图像。
- ☑ border-bottom-image：定义底部边框背景图像。
- ☑ border-left-image：定义左侧边框背景图像。
- ☑ border-top-left-image：定义左上角边框背景图像。
- ☑ border-top-right-image：定义右上角边框背景图像。
- ☑ border-bottom-left-image：定义左下角边框背景图像。
- ☑ border-bottom-right-image：定义右下角边框背景图像。

另外，根据边框背景图像的处理功能，border-image 属性还派生了下面几个属性。

- ☑ border-image-source：定义边框的背景图像源，即图像 URL。
- ☑ border-image-slice：定义如何裁切背景图像，与背景图像的定位功能不同。
- ☑ border-image-repeat：定义边框背景图像的重复性。
- ☑ border-image-width：定义边框背景图像的显示大小（即边框显示大小）。虽然 W3C 定义了该属性，但是浏览器还是习惯使用 border-width 实现相同的功能。
- ☑ border-image-outset：定义边框背景图像的偏移位置。

Webkit 引擎支持-webkit-border-image 私有属性，Mozilla Gecko 引擎支持-moz-border-image 私有属性，Presto 引擎支持-o-border-image 私有属性，IE 浏览器暂时不支持 border-image 属性，也没有定义私有属性。

14.2.1 用法详解

border-image 属性与 border-image 属性的用法相似，包括图像源、剪裁位置和重复性。例如，border-image:url(01.jpg) 50 no-repeat;样式就表示设置边框背景图像为 01.jpg，剪裁位置为 50 像素，禁止重复。下面针对边框背景的图像源、剪裁和重复性进行分解。

☑ 边框图像源（border-image-source）

与 background-image 属性用法相同，border-image 属性使用 url()调用背景图像，图像可以是相对路径或是绝对路径字符串，也可以不使用图像，即 border-image:none;。

☑ 边框图像切片（border-image-slice）

Note

border-image-slice 属性值没有单位，默认单位为像素。支持百分比值，百分比值总是相对于边框图像而言。例如，边框图像大小为 400×300 像素，则 20%的实际效果就是剪裁了图像的 60 像素、80 像素、60 像素、80 像素的 4 条的边大小。

clip 是 CSS 专用剪裁属性，而 border-image-slice 属性虽然在语义上不是剪裁，但是从实现效果上来分析，与剪裁工具类似，它把边框图像打乱，再重新安置、变形。

border-image-slice 属性值包含 4 个参数。它遵循 CSS 方位规则（如 margin、padding 或 border 等属性的赋值），按照上、右、下、左的顺时针方向赋值剪裁。

【示例】为元素边框定义背景图像为 images/border1.png，然后设置 border-imageslice 属性值为（27 27 27 27），该属性值可以简写为 27。整个示例的代码如下，页面浏览效果如图 14.4 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-image</title>
<style type="text/css">
div {
    height:160px;
    border-width:27px;
    /*设置边框背景图像*/
    -webkit-border-image: url(images/border1.png) 27; /*兼容 Webkit 引擎*/
    -moz-border-image: url(images/border1.png) 27; /*兼容 Gecko 引擎*/
    -o-border-image: url(images/border1.png) 27; /*兼容 Presto 引擎*/
    border-image: url(images/border1.png) 27; /*兼容标准用法*/
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

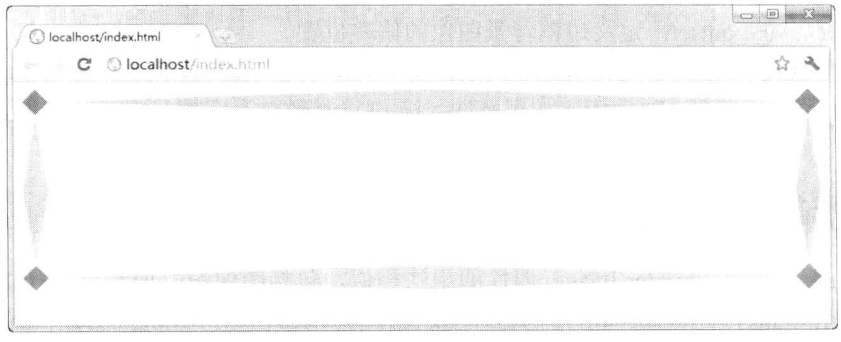


图 14.4 定义边框背景样式

Note

border-image 属性能够根据 border-image-slice 属性值把背景图像切分为 9 部分，然后分别把这 9 部分图像切片按顺序分别填充到边框上 4 边、4 角和内容区域。在上面示例中，使用了一个 71×71 像素大小的背景图，这个正方形的背景图被等分了 9 部分，每部分的高和宽都是 21×21 像素大小。当声明 border-image-slice 属性值为（27 27 27 27）时，第 1 个参数值表示从上向下裁切背景图像，然后显示在顶边。而第 3 个参数值正好相反，从下向上裁切背景图像，然后显示在底边。第 2 个参数值表示从右向左裁切背景图像，然后显示在右边。而第 4 个参数值正好相反，从左向右裁切背景图像，然后显示在左边。背景图像被 4 个参数值裁切为 9 部分，然后根据边框的大小进行自适应显示。例如，当分别设置边框为不同大小，则显示效果除了粗细之外，其他都是完全相同的。

另外，需要注意几个问题：

- ☑ 如果 border-image-slice 属性值都大于或等于背景图像的尺寸，即第 1、3 个参数值都大于或等于背景图像的高度，第 2、4 个参数值大于或等于背景图像的宽度。那么背景图像被缩放填充到边框的 4 个角中。
- ☑ 如果 border-image-slice 属性值中一对值（水平上或垂直上）都大于或等于背景图像的尺寸，即第 1、3 个参数值（或者第 2、4 个参数值）大于或等于背景图像的高度或者宽度，而另一对值之和等于背景图像的宽度或者高度，那么背景图像将被切分为两半，并按顺序分别缩放填充到边框的 4 个角中。
- ☑ 如果 border-image-slice 属性值中每一对值之和等于背景图像的宽度或者高度，那么背景图像将被切分为 4 部分，并按顺序分别缩放填充到边框的 4 个角中。
- ☑ 如果 border-image-slice 属性值中一对值或者两对值之和大于背景图像的宽度或者高度，但是参数值都不大于背景图像的宽度或者高度，那么背景图像将被切分为 4 部分，这 4 部分背景图像切片存在重叠部分，这时 border-image 仍然按顺序分别缩放它们，并填充到边框的 4 个角中。
- ☑ 如果 border-image-slice 属性值中一对值（水平上或垂直上）之和小于背景图像的尺寸，即第 1、3 个参数值之和（或者第 2、4 个参数值之和）小于背景图像的高度或者宽度，而另一对值之和等于背景图像的宽度或者高度。那么背景图像将被切分为 6 部分，并按顺序分别缩放填充到边框的 4 个角和对应的两条边中。
- ☑ 如果 border-image-slice 属性值中一对值都不大于背景图像的宽度或者高度，而另一对值之和小于背景图像的宽度或者高度。那么背景图像将被切分为 3 部分，这时 border-image 按顺序分别缩放它们，并填充到边框的一边上的两个角和边中，然后重复相同的方法填充另一边的两角和边上。
- ☑ 如果 border-image-slice 属性值中每一对值之和都小于背景图像的宽度或者高度。那么背景图像将被切分为 9 部分，这时 border-image 按顺序分别缩放它们，并填充到元素的 4 边、4 角和内容区域。
- ☑ 如果 border-image-slice 属性值单位为百分比，则它根据背景图像的大小进行百分比裁切，然后根据前面分析的规律填充边框。例如，四分背景图像，各填 4 个角效果。

border-image-repeat 属性包含 3 个值，简单说明如下。

- ☑ stretch：拉伸，为默认值。
- ☑ repeat：重复。
- ☑ round：平铺。

14.2.2 案例实战

Note

【示例 1】border-image 是一个非常实用的属性，它拓展了设计师的设计灵感，抛弃了借助背景图像实现边角设计的笨拙做法，提高了网页传输速度，降低了前期劳动量。本示例将演示如何设计局部或者全部圆角版块，演示效果如图 14.5 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-image</title>
<style type="text/css">
div {
    height:120px;
    border-width:10px;
    -moz-border-image: url(images/r2.png)  20;
    -webkit-border-image: url(images/r2.png) 20;
    -o-border-image:  url(images/r2.png) 20;
    border-image: url(images/r2.png) 20;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

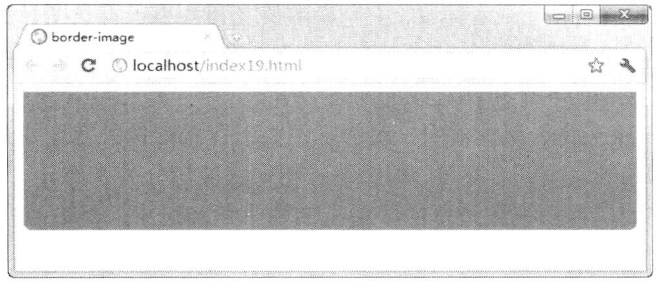


图 14.5　定义圆角样式

【示例 2】设计圆环边框版块。设计背景图像为 42×42 像素，圆环角为 20 像素，显示效果如图 14.6 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-image</title>
<style type="text/css">
```

Note

```
div {
    height:120px;
    border-width:10px;
    -moz-border-image: url(images/r3.png) 20;
    -webkit-border-image: url(images/r3.png) 20;
    -o-border-image:   url(images/r3.png) 20;
    border-image: url(images/r3.png) 20;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

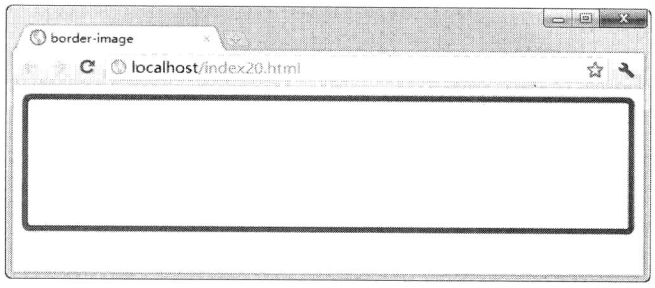


图 14.6　定义圆环边框样式

【示例 3】设计阴影效果。设计背景图像为 42×42 像素，圆环角为 20 像素，显示效果如图 14.7 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-image</title>
<style type="text/css">
img {
    height:400px;
    border-width:2px 5px 6px 2px;
    -moz-border-image: url(images/r4.png)   2 5 6 2;
    -webkit-border-image: url(images/r4.png)   2 5 6 2;
    -o-border-image:   url(images/r4.png)   2 5 6 2;
    border-image: url(images/r4.png)   2 5 6 2;
}
</style>
</head>
<body>
<img src="images/img2.jpg" />
</body>
</html>
```

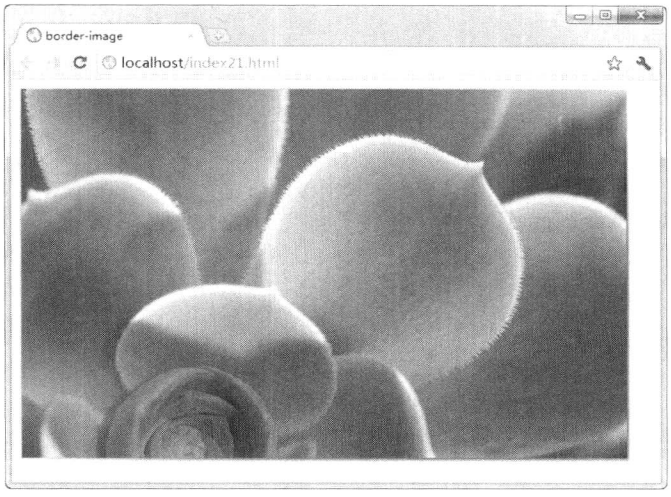


图 14.7　阴影效果

【示例 4】设计选项卡。设计背景图像为 12×27 像素，圆环角为 12 像素，显示效果如图 14.8 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-image</title>
<style type="text/css">
ul{
    margin:0;
    padding:0;
    list-style-type:none;
}
li {
    width:100px;
    height:20px;
    float:left;
    padding:4px 0;
    text-align:center;
    border-width:5px 5px 0px;
    -moz-border-image: url(images/r5.png) 5 5 0;
    -webkit-border-image: url(images/r5.png) 5 5 0;
    -o-border-image:   url(images/r5.png) 5 5 0;
    border-image: url(images/r5.png) 5 5 0;
}
</style>
</head>
<body>
<ul>
    <li>首页</li>
    <li>微博</li>
```

```
    <li>团购</li>
</ul>
</body>
</html>
```

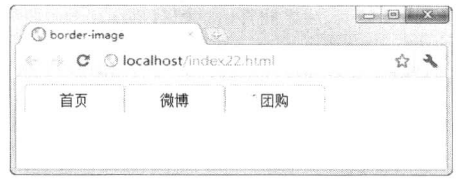


图 14.8　选项卡样式

注意： 如果 border-image-slice 属性值包含 3 个参数，则第 1 个参数表示顶部裁切值，第 2 个参数表示左右两侧裁切值，第 3 个参数表示底部裁切值。例如，border-image-slice:55 0;就等同于 border-image-slice:5 5 0 5;，它表示把边框背景图像分成 6 部分，底部没有裁切，然后分别填充到左上角、顶边、右上角、左边、中间内容区域和右边。border-image 属性的应用比较灵活，可以设计不同样式的背景图，然后设置不同的 border-image-slice 属性值，从而设计各种特殊边框样式。

14.3　设 计 圆 角

圆角边框的绘制也是 Web 网站或 Web 应用程序中经常用来美化页面效果的手法之一。在 CSS3 之前，需要使用图像文件才能达到同样效果，如果只靠样式就能完成圆角边框的绘制，对界面设计者来说无疑是一件可喜的事情。

CSS3 定义了 border-radius 属性，使用它可以设计元素以圆角样式显示。border-radius 属性的基本语法如下所示。

```
border-radius:none | <length>{1,4} [ / <length>{1,4} ]?;
```

border-radius 属性初始值为 none，适用于所有元素，除了 border-collapse 属性值为 collapse 的 table 元素。取值简单说明如下。

☑　none：默认值，表示元素没有圆角。

☑　<length>：由浮点数字和单位标识符组成的长度值，不可为负值。

为了方便定义元素的 4 个顶角为圆角，border-radius 属性派生了 4 个子属性。

☑　border-top-right-radius：定义右上角的圆角。

☑　border-bottom-right-radius：定义右下角的圆角。

☑　border-bottom-left-radius：定义左下角的圆角。

☑　border-top-left-radius：定义左上角的圆角。

目前，Webkit 引擎支持-webkit-borderradius 私有属性，Mozilla Gecko 引擎支持-moz-border-radius 私有属性，Presto 引擎和 IE9+支持 border-radius 标准属性。IE8 及其以前版本的浏览器暂时不支持 border-radius 属性。相应的，Webkit 和 Gecko 引擎还支持下面的私有属性。

☑　-moz-border-radius-bottomleft

☑　-webkit-border-bottom-left-radius

- ☑ -moz-border-radius-bottomright
- ☑ -webkit-border-bottom-right-radius
- ☑ -moz-border-radius-topleft
- ☑ -webkit-border-top-left-radius
- ☑ -moz-border-radius-topright
- ☑ -webkit-border-top-right-radius

Note

14.3.1 用法详解

border-radius 属性可包含两个参数值：第一个值表示圆角的水平半径，第二个值表示圆角的垂直半径，两个参数值通过斜线分隔。如果仅包含一个参数值，则第二个值与第一个值相同，它表示这个角就是一个 1/4 圆角。如果参数值中包含 0，则这个角就是直角，不会显示为圆角。

针对 border-radius 属性参数值，各种浏览器的处理方式并不一致。在 Chrome 和 Safari 浏览器中，会绘制出一个椭圆形边框，第一个半径为椭圆的水平方向半径，第二个半径为椭圆的垂直方向半径。在 Firefox 和 Opera 浏览器中，将第一个半径作为边框左上角与右下角的圆半径来绘制，将第二个半径作为边框右上角与左下角的圆半径来绘制。

【示例 1】在本示例中，给 border-radius 属性设置一个值，则圆角是一个 1/4 圆角，演示效果如图 14.9 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title> </title>
<style type="text/css">
div {
    height:100px;
    border:10px solid red;
    -moz-border-radius:10px; /*兼容 Gecko 引擎*/
    -webkit-border-radius:10px; /*兼容 Webkit 引擎*/
    border-radius:10px; /*标准用法*/
}
</style>
</head>
<body>
<div></div>
</body>
```

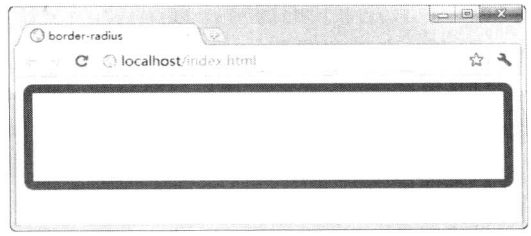


图 14.9 定义圆角样式

如果为 border-radius 属性设置两个参数，则效果如图 14.10 所示。

```
<style type="text/css">
div {
    height:100px;
    border:10px solid red;
    -moz-border-radius:20px/40px;
    -webkit-border-radius:20px/40px;
    border-radius:20px/40px;
}
</style>
```

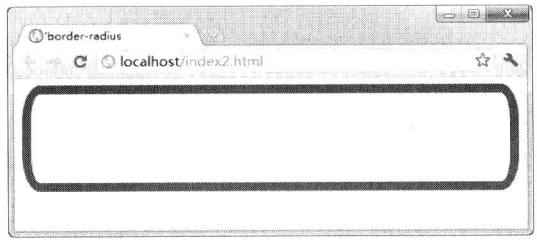


图 14.10　定义圆角样式

也可以为元素的 4 个角定义不同半径的圆角，实现的方法有两种：

一种方法是利用 border-radius 属性，为其赋一组值。当为 border-radius 属性赋一组值，将遵循 CSS 赋值规则，可以包含 2 个、3 个或者 4 个值集合，但是此时无法使用斜杠方式定义圆角水平和垂直半径。

如果是 4 个值，则这 4 个值将按照 top-left、top-right、bottom-right 和 bottom-left 的顺序来设置。如果 bottom-left 值省略，那么它等于 top-right。如果 bottom-right 值省略，那么它等于 top-left。如果 top-right 值省略，那么它等于 top-left。如果为 border-radius 属性设置 4 个值的集合参数，则每个值表示每个角的圆角半径。

【示例 2】下面的代码将定义不同角度的圆角半径，演示效果如图 14.11 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-radius</title>
<style type="text/css">
div {
    height:100px;
    border:10px solid red;
    -moz-border-radius:10px 30px 50px 70px;
    -webkit-border-radius:10px 30px 50px 70px;
    border-radius:10px 30px 50px 70px;
}
</style>
</head>
<body>
<div></div>
```

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

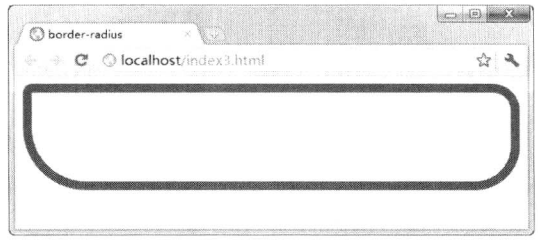


图 14.11　分别定义不同顶角的圆角样式

如果为 border-radius 属性设置 3 个值的集合参数，则第一个值表示左上角的圆角半径，第二个值表示右上和左下两个角的圆角半径，第三个值表示右下角的圆角半径。

如果为 border-radius 属性设置两个值的集合参数，则第一个值表示左上角和右下角的圆角半径，第二个值表示右上和左下两个角的圆角半径。

另一种方法是利用派生子属性进行定义，如 border-top-right-radius、border-bottomright-radius、border-bottom-left-radius 和 border-top-left-radius。注意，Gecko 和 Presto 引擎在写法上存在很大差异。

【示例 3】下面的代码定义 div 元素右上角为 50 像素的圆角，演示效果如图 14.12 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-radius</title>
<style type="text/css">
div {
    height:100px;
    border:10px solid red;
    -moz-border-radius-topright:50px;
    -webkit-border-top-right-radius:50px;
    border-top-right-radius:50px;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

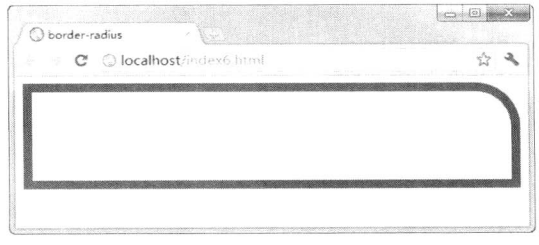


图 14.12　定义某个顶角的圆角样式

在 CSS3 中，如果使用了 border-radius 属性，但是把边框设定为不显示的时候，浏览器将把背景的 4 个角绘制为圆角。

```
<style type="text/css">
div {
    height:100px;
    border: none;
    -moz-border-radius:10px; /*兼容 Gecko 引擎*/
    -webkit-border-radius:10px; /*兼容 Webkit 引擎*/
    border-radius:10px; /*标准用法*/
}
</style>
```

使用 border-radius 属性后，不管边框是什么种类，都会将边框沿着圆角曲线进行绘制。

14.3.2　案例实战：设计椭圆图形

内边半径等于外边半径减去对应边的宽度。如果差值为负值，内边半径是 0，则会显示为内直角，而不是内圆角。所以内外边曲线的圆心并不必然是一致的。

如果角的两个相邻边宽度不同，那么这个角将会从宽的边圆滑过渡到窄的边。其中一条边甚至可以为 0。两条相邻边颜色和样式改变的中心点是在一个与两边宽度成正比的角上。如果两条边宽度相同，那么这个分界点应该就是在一个 45° 的角上。如果一条边是相邻另一条边的两倍，那么这个点就在一个 30° 的角上。界定这个改变的线就是毗连在内外曲线上的两个点的直线上。

圆角是不许彼此重叠的，所以当相邻两个圆角的半径之和大于元素的宽或高时，浏览器在解析时会强制缩小一个或多个圆角半径。

【示例】下面代码定义 div 元素显示为圆形，演示效果如图 14.13 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>border-radius</title>
<style type="text/css">
div {
    height:300px;
    width:300px;
    background:url(images/bg1.gif) no-repeat;
    border:1px solid red;
    -moz-border-radius:150px;
    -webkit-border-radius:150px;
    border-radius:150px;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

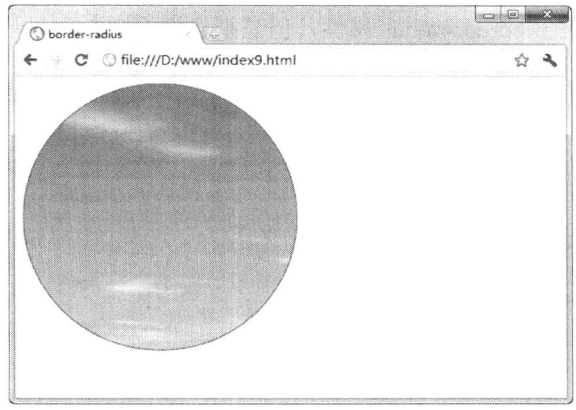


图 14.13　定义圆形显示的元素效果

在上面示例中，即使 border 属性值为 none，也会呈现圆形效果。如果 background-clip 属性值为 padding-box，那么背景会被曲线的圆角内边裁剪。如果 background-clip 属性值为 border-box，那么背景会被圆角外边裁剪。border 和 padding 属性定义的区域也一样会被曲线裁剪。另外，所有边框样式（如 solid、dotted、inset 等）都遵循边框圆角的曲线，即使是定义了 border-image 属性，曲线以外的边框背景也会被裁剪掉。如果角的两个相邻边宽度不同，那么这个角将会从宽的边圆滑过渡到窄的边。其中一条边甚至可以为 0。

两条相邻边颜色和样式改变的中心点是在一个与两边宽度成正比的角上。如果两条边宽度相同，那么这个分界点应该就是在一个 45° 的角上。如果一条边是相邻另一条边的两倍，那么这个点就在一个 30° 的角上。界定这个改变的线就是毗连在内外曲线上的两个点的直线上。

14.4　设 计 阴 影

box-shadow 属性定义元素的阴影，它与 text-shadow 属性功能是相同的，但是作用对象略有不同。该属性的基本语法如下所示。

```
box-shadow:none | <shadow> [ , <shadow> ]*;
```

box-shadow 属性的初始值是 none，该属性适用于所有元素。下面是取值的简单说明。

- ☑ none：默认值，表示元素没有阴影。
- ☑ <shadow>：该属性值可以使用公式表示为 inset? && [<length>{2,4} && <color>?]，其中 inset 表示设置阴影的类型为内阴影，默认为外阴影，<length>是由浮点数字和单位标识符组成的长度值，可取正负值，用来定义阴影水平偏移、垂直偏移，以及阴影大小、阴影扩展（即阴影模糊度）。<color>表示阴影颜色。

目前，Webkit 引擎支持-webkit-box-shadow 私有属性，Mozilla Gecko 引擎支持-moz-box-shadow 私有属性，Presto 引擎和 IE9+支持 box-shadow 标准属性。IE8 及其以前版本的浏览器暂时不支持 box-shadow 属性。

14.4.1　用法详解

box-shadow 属性值包含 6 个参数值：阴影类型、X 轴位移、Y 轴位移、阴影大小、阴影扩展和阴

影颜色，这 6 个参数值可以有选择性地省略。

如果不设置阴影类型时，默认为投影效果，当设置为 inset 时，则阴影效果为内阴影。X 轴位移和 Y 轴位移定义阴影的偏移距离。阴影大小、阴影扩展和阴影颜色是可选值，默认为黑色实影，box-shadow 属性值必须设置阴影的位移值，否则没有效果。如果定义了阴影大小，此时定义阴影位移为 0，才可以看到阴影效果。下面结合示例进行演示说明。

Note

【示例 1】定义简单的实影投影效果，演示效果如图 14.14 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>box-shadow</title>
<style type="text/css">
img{
    height:300px;
    -moz-box-shadow:5px 5px;
    -webkit-box-shadow:5px 5px;
    box-shadow:5px 5px;
}
</style>
</head>
<body>
<img src="images/img8.jpg"  />
</body>
</html>
```

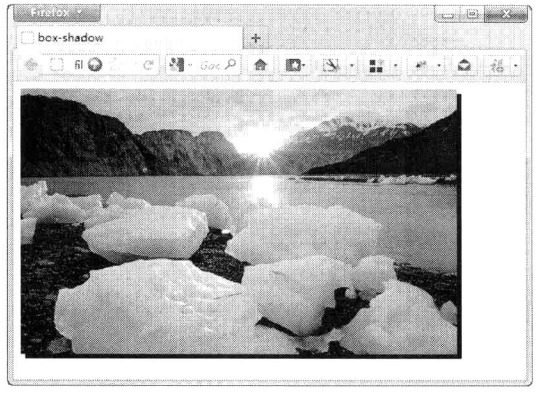


图 14.14　定义简单的阴影效果

【示例 2】定义位移、阴影大小和阴影颜色，演示效果如图 14.15 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>box-shadow</title>
<style type="text/css">
img{
```

```
        height:300px;
        -moz-box-shadow:2px 2px 10px #06C;
        -webkit-box-shadow:2px 2px 10px #06C;
        box-shadow:2px 2px 10px #06C;
    }
    </style>
    </head>
    <body>
    <img src="images/img8.jpg"/>
    </body>
    </html>
```

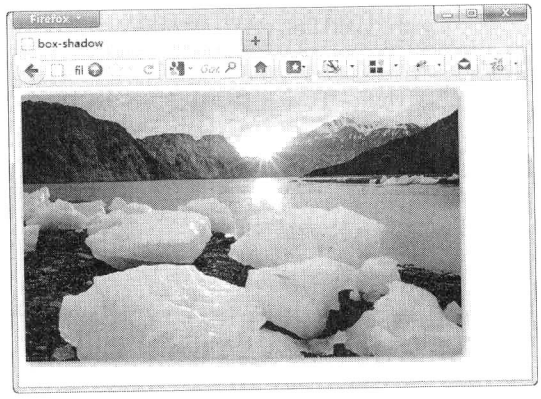


图 14.15　定义复杂的阴影效果

【示例 3】定义内阴影，位移 5 像素，阴影大小为 10 像素，颜色为#06C，则演示效果如图 14.16 所示。

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>box-shadow</title>
    <style type="text/css">
    body { margin:24px;}
    div {
        height:270px;
        width:310px;
        padding:16px;
        -moz-box-shadow:inset 2px 2px 10px #06C;
        -webkit-box-shadow:inset 2px 2px 10px #06C;
        box-shadow:inset 2px 2px 10px #06C;
    }
    </style>
    </head>
    <body>
    <div><img src="images/yybq.jpg" /></div>
    </body>
    </html>
```

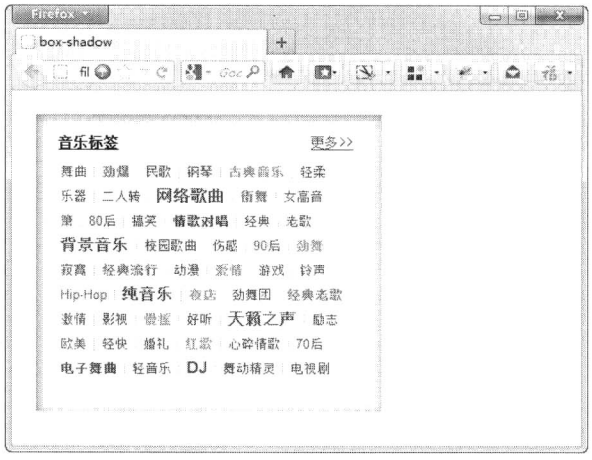


图 14.16　定义内阴影效果

【示例 4】通过设置多组参数值定义多色阴影，则演示效果如图 14.17 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>box-shadow</title>
<style type="text/css">
body {
    margin:24px;
}
img{
    height:300px;
    -moz-box-shadow:-10px 0 12px red,
      10px 0 12px blue,
      0 -10px 12px yellow,
      0 10px 12px green;
    -webkit-box-shadow:-10px 0 12px red,
      10px 0 12px blue,
      0 -10px 12px yellow,
      0 10px 12px green;
    box-shadow:-10px 0 12px red,
      10px 0 12px blue,
      0 -10px 12px yellow,
      0 10px 12px green;
}
</style>
</head>
<body>
<img src="images/img8.jpg"/>
</body>
</html>
```

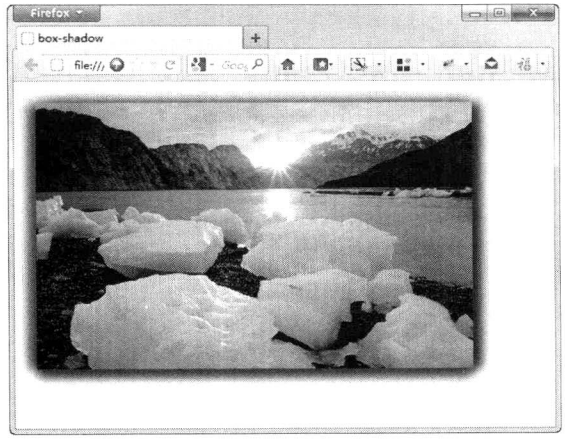


图 14.17　定义多色阴影效果

【示例 5】通过多组参数值还可以定义渐变阴影，演示效果如图 14.18 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>box-shadow</title>
<style type="text/css">
body {
    margin:24px;
}
img{
    height:300px;
    -moz-box-shadow:-10px 0 12px red,
      10px 0 12px blue,
      0 -10px 12px yellow,
      0 10px 12px green;
    -webkit-box-shadow:-10px 0 12px red,
      10px 0 12px blue,
      0 -10px 12px yellow,
      0 10px 12px green;
    box-shadow:-10px 0 12px red,
      10px 0 12px blue,
      0 -10px 12px yellow,
      0 10px 12px green;
}
</style>
</head>
<body>
<img src="images/img8.jpg"   />
</body>
</html>
```

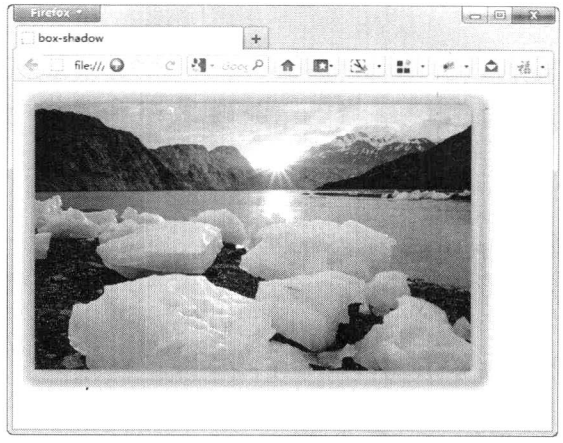


图 14.18　定义渐变阴影效果

当给同一个元素设计多个阴影时，需要注意它们的顺序，最先写的阴影将显示在最顶层。如在上面这段代码中，先定义一个 10 像素的红色阴影，再定义一个 10 像素大小、10 像素扩展的阴影。显示结果就是红色阴影层覆盖在黄色阴影层之上，此时如果顶层的阴影太大，就会遮盖底部的阴影。

14.4.2　案例实战：设计 Windows7 界面效果

下面利用 CSS3 新增的边框和背景样式来模拟 Windows7 界面效果。本例综合应用了 box-shadow、border-radius、text-shadow、border-color 和 border-image 等属性，同时还用到了渐变设计属性。整个案例的演示效果如图 14.19 所示。

图 14.19　设计 Windows7 界面效果

操作步骤：

第 1 步，设计页面结构。整个 UI 界面的结构比较简单，样式代码如下所示。

```
<div id="desktop">
    <div id="bgWindow" class="window secondary">
        <span>对话框</span>
        <div class="content"></div>
    </div>
    <div id="frontWindow" class="window">
        <span>用户反馈</span>
        <div id="winInput"><input type="text" value="姓名"><input type="text" value="联系方式"></div>
        <div id="winContent" class="content">请输入你的反馈意见……</div>
    </div>
    <div id="startmenu">
        <button id="winflag">开始</button>
        <span id="toolBtn"><!--任务栏图标-->
            <button class="application">♣</button>
            <button class="application">✱</button>
            <button class="application">☀</button>
            <button class="application">☯</button>
        </span>
    </div>
</div>
```

Note

第 2 步，设计桌面效果。在文档样式表中，先定制页面样式，然后设置桌面显示背景，样式代码如下所示。

```
html,body { /*页面样式定制，清除边距，显式定义高度*/
    padding:0;
    margin:0;
    height:100%;
}
#desktop { /*定制桌面背景效果*/
    background: #2c609b;
    height:100%; /*满窗口显示*/
    font: 12px "Segoe UI", Tahoma, sans-serif;
    position: relative; /*定义包含框，为后面的桌面定位元素提供参考*/
    /*定义桌面内阴影，使用一组 3 个内阴影设计梦幻效果*/
    -moz-box-shadow: inset 0 -200px 100px #032b5c,
        inset -100px 100px 100px #2073b5,
        inset 100px 200px 100px #1f9bb1;
    -webkit-box-shadow: inset 0 -200px 100px #032b5c,
        inset -100px 100px 100px #2073b5,
        inset 100px 200px 100px #1f9bb1;
    box-shadow: inset 0 -200px 100px #032b5c,
        inset -100px 100px 100px #2073b5,
        inset 100px 200px 100px #1f9bb1;
    overflow: hidden; /*隐藏超出的内容*/
}
```

第 3 步，设计开始菜单和任务栏。开始菜单和任务栏主要用到了圆角样式和盒子阴影，在设计任务栏中的图标时，还用到渐变效果，该技术将在后面章节中进行详细说明，该部分的样式代码如下所示。

Note

```
#startmenu { /*设置任务栏效果*/
    /*固定显示在页面底部*/
    position: absolute;
    bottom: 0;
    /*固定大小*/
    height: 40px;
    width: 100%;
    background: rgba(178, 215, 255, 0.25); /*增加半透明效果*/
    /*为任务栏设计顶部外阴影，以及在内部添加两道阴影效果*/
    -webkit-box-shadow: 0 -2px 20px rgba(0, 0, 0, 0.25);
    -moz-box-shadow: 0 -2px 20px rgba(0, 0, 0, 0.25),
        inset 0 1px #042754,
        inset 0 2px #5785b0;
    box-shadow: 0 -2px 20px rgba(0, 0, 0, 0.25),
        inset 0 1px #042754,
        inset 0 2px #5785b0;
    overflow: hidden;
}
#startmenu button {
    font-size: 1.6em;
    color: #fff;
    text-shadow: 1px 2px 2px #00294b; /*为按钮文字增加阴影效果*/
}
#startmenu #winflag { /*设计“开始”按钮样式*/
    float: left;
    margin: 2px;
    height: 34px;
    width: 80px;
    margin-right: 10px;
    border: none;
    background: #034a76;
    /*设计开始按钮圆角显示*/
    -moz-border-radius: 40px;
    -webkit-border-radius: 40px;
    border-radius: 40px;
    /*设计开始按钮内外阴影特效*/
    -moz-box-shadow: 0 0 1px #fff,
        0 0 3px #000,
        0 0 3px #000,
        inset 0 1px #fff,
        inset 0 12px rgba(255, 255, 255, 0.15),
        inset 0 4px 10px #cef,
        inset 0 22px 5px #0773b4,
        inset 0 -5px 10px #0df;
    -webkit-box-shadow: 0 0 1px #fff,
        0 0 3px #000,
        0 0 3px #000;
    box-shadow: 0 0 1px #fff,
        0 0 3px #000,
        0 0 3px #000,
```

Note

```
            inset 0 1px #fff,
            inset 0 12px rgba(255, 255, 255, 0.15),
            inset 0 4px 10px #cef,
            inset 0 22px 5px #0773b4,
            inset 0 -5px 10px #0df;
}
#startmenu .application { /*设计任务栏图标样式*/
    position: relative;
    bottom: 1px;
    height: 38px;
    width: 52px;
    background: rgba(14, 59, 103, 0.25);
    border: 1px solid rgba(0, 0, 0, 0.8);
    /*设计渐变特效*/
    -o-transition: .3s all;
    -webkit-transition: .3s all;
    -moz-transition: .3s all;
    /*设计任务栏图标圆角显示*/
    -moz-border-radius: 4px;
    -webkit-border-radius: 4px;
    border-radius: 4px;
    /*设计任务栏图标内外阴影特效*/
    -moz-box-shadow: inset 0 0 1px #fff,
        inset 4px 4px 20px rgba(255, 255, 255, 0.33),
        inset -2px -2px 10px rgba(255, 255, 255, 0.25);
    box-shadow: inset 0 0 1px #fff,
        inset 4px 4px 20px rgba(255, 255, 255, 0.33),
        inset -2px -2px 10px rgba(255, 255, 255, 0.25);
}
/*设计鼠标经过时，图标显示为半透明的色彩变化效果*/
#startmenu .application:hover { background-color: rgba(255, 255, 255, 0.25); }
```

第 4 步，设计窗口效果。窗口 UI 主要涉及圆角和半透明效果设计，样式代码如下所示。

```
/*设计窗口外框效果*/
.window {
    /*定位窗体大小和位置*/
    position: absolute;
    left: 150px;
    top: 75px;
    width: 400px;
    height: 400px;
    padding: 7px;
    /*设计半透明度效果的边框和背景效果*/
    border: 1px solid rgba(255, 255, 255, 0.6);
    background: rgba(178, 215, 255, 0.75);
    /*设计窗体外框圆角显示*/
    -webkit-border-radius: 8px;
    -moz-border-radius: 8px;
    border-radius: 8px;
    /*设计窗体外框的外阴影特效*/
```

Note

```
        -moz-box-shadow: 0 2px 16px #000,
                0 0 1px #000,
                0 0 1px #000;
        -webkit-box-shadow: 0 2px 16px #000,
                0 0 1px #000,
                0 0 1px #000;
        box-shadow: 0 2px 16px #000,
                0 0 1px #000,
                0 0 1px #000;
        /*设计晕边效果*/
        text-shadow: 0 0 15px #fff, 0 0 15px #fff;
}
.window span { display: block; }
.window input { /*文本输入框样式*/
        /*设计文本输入框圆角显示*/
        -webkit-border-radius: 2px;
        -moz-border-radius: 2px;
        /*设计文本输入框的内外阴影特效*/
        -moz-box-shadow: 0 0 2px #fff,
                0 0 1px #fff,
                inset 0 0 3px #fff;
        -webkit-box-shadow: 0 0 2px #fff,
                0 0 1px #fff;
        box-shadow: 0 0 2px #fff,
                0 0 1px #fff,
                inset 0 0 3px #fff
}
.window input + input { margin-left: 12px; }
.window.secondary { /*定位第二个窗体位置和不透明度*/
        left: 300px;
        top: 125px;
        opacity: 0.66;
}
.window.secondary span { margin-bottom: 85px; }
/*设计窗口内文本区域样式*/
.window .content {
        padding: 10px;
        height: 279px;
        background: #fff;
        border: 1px solid #000;
        /*设计文本区域圆角显示*/
        -webkit-border-radius: 2px;
        -moz-border-radius: 2px;
        border-radius: 2px;
        /*设计文本区域的内外阴影特效*/
        -moz-box-shadow: 0 0 5px #fff,
                0 0 1px #fff,
                inset 0 1px 2px #aaa;
        -webkit-box-shadow: 0 0 5px #fff,
                0 0 1px #fff;
```

```
    box-shadow: 0 0 5px #fff,
        0 0 1px #fff,
        inset 0 1px 2px #aaa;
    text-shadow: none; /*取消文本阴影*/
}
```

Note

14.5 设计背景

为了方便更灵活地设计网页效果，CSS3 增强了 background 属性的功能，允许在同一个元素内叠加多个背景图像。该属性的基本语法如下所示。

```
background:[ <bg-layer> , ]* <final-bg-layer>;
```

background 属性与 CSS2 中的用法基本相同，不过 CSS3 允许在该属性中添加多个背景图像组，背景图像之间通过逗号进行分隔。其中<bg-layer>表示一个背景图像层。每个背景图像层都可以包含下面的值：

```
[background-image] | [background-color] | [background-origin] | [background-clip]| [background-repeat] |
[background-size] | [background-position] |[background-attachment]
```

为了方便定义背景图像，background 属性又派生了 8 个子属性。

- ☑ background-image：定义背景图像。
- ☑ background-color：定义背景颜色。
- ☑ background-origin：指定背景的显示区域。
- ☑ background-clip：指定背景的裁剪区域。
- ☑ background-repeat：设置背景图像是否及如何重复铺排。
- ☑ background-size：定义背景图片的大小。
- ☑ background-position：设置背景图像位置。
- ☑ background-attachment：定义背景图像的显示方式。

在 CSS3 中，追加了 4 个与背景相关的属性。

- ☑ background-clip：指定背景的显示范围。
- ☑ background-origin：指定绘制背景图像时的起点。
- ☑ background-size：指定背景中图像的尺寸。
- ☑ background-break：指定内联元素的背景图像进行平铺时的循环方式。

在 Firefox 浏览器中，支持除了 background-size 属性之外的其他 3 个新属性，在书写样式代码时，需要在属性前面加上-moz-前缀，而在使用 background-break 属性时，在样式代码中不用-moz-background-break，而使用-moz-background-inline-policy，这一点需要注意。在 Safari、Chrome 和 Opera 浏览器中，支持除了 background-break 之外的其他 3 个新属性，在样式代码中需要在属性前面加上-webkit-前缀。

14.5.1 定义背景坐标

CSS3 为 background 新增了 3 个派生的子属性 background-origin、background-clip 和 background-size。其中 background-origin 属性定义 background-position 属性的参考位置。在默认情况下，

background-position 属性总是根据元素左上角为坐标原点进行定位背景图像。使用 background-origin 属性可以改变这种定位方式。该属性的基本语法如下所示。

```
background-origin:border | padding | content;
```

Note

background-origin 初始值是 padding，适用于所有元素。取值的简单说明如下。

☑ border：从边框区域开始显示背景。

☑ padding：从补白区域开始显示背景。

☑ content：仅在内容区域显示背景。

在最新版本的 CSS 背景模块规范中（http://www.w3.org/TR/css3-background/#background-origin），W3C 规定该属性取值为 padding-box、border-box 和 content-box，不过目前还没有得到主流浏览器的支持。目前，Webkit 引擎支持-webkit-backgroundorigin 私有属性，Mozilla Gecko 引擎支持-moz-background-origin 私有属性，Presto 引擎和 IE 浏览器暂不支持该属性。

【示例】background-origin 属性改善了背景图像定位的方式，更灵活地决定背景图像应该显示的位置。本示例利用 background-origin 属性重设背景图像的定位坐标，以便更好地控制背景图像的显示，演示效果如图 14.20 所示。

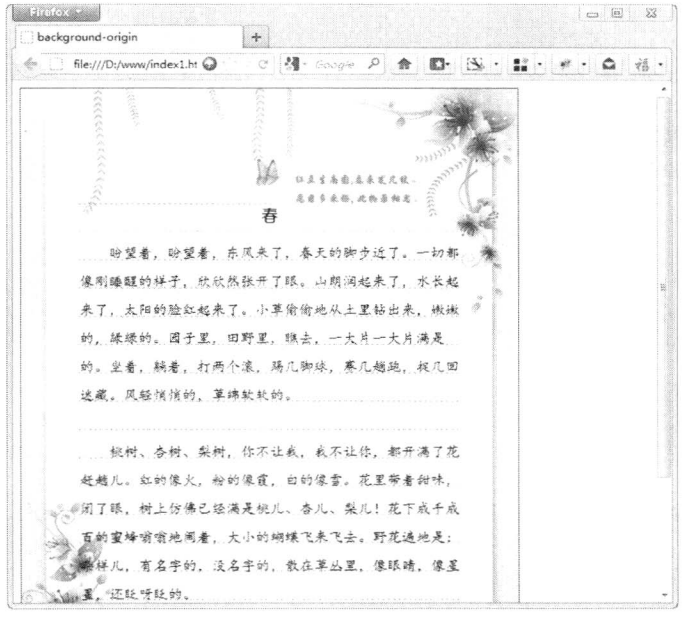


图 14.20　设计书信效果

实现本示例的代码如下所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>background-origin</title>
<style type="text/css">
div {
    height:600px;
```

Note

```
        width:416px;
        border:solid 1px red;
        padding:120px 4em 0;
        /*为了避免背景图像重复平铺到边框区域，应禁止它平铺*/
        background:url(images/p3.jpg) no-repeat;
        /*设计背景图像的定位坐标点为元素边框的左上角*/
        -moz-background-origin:border;
        -webkit-background-origin:border;
        background-origin:border;
        -moz-background-size:cover;
        -webkit-background-size:cover;
        background-size:cover;
        overflow:hidden;
    }
    div h1 {
        font-size:18px;
        font-family:"幼圆";
        text-align:center;
    }
    div p {
        text-indent:2em;
        line-height:2em;
        font-family:"楷体";
        margin-bottom:2em;
    }
    </style>
    </head>
    <body>
    <div>
        <h1>春</h1>
        <p>盼望着，盼望着，东风来了，春天的脚步近了。一切都像刚睡醒的样子，欣欣然张开了眼。山朗润起来了，水长起来了，太阳的脸红起来了。小草偷偷地从土里钻出来，嫩嫩的，绿绿的。园子里，田野里，瞧去，一大片一大片满是的。坐着，躺着，打两个滚，踢几脚球，赛几趟跑，捉几回迷藏。风轻悄悄的，草绵软软的。</p>
        <p>桃树、杏树、梨树，你不让我，我不让你，都开满了花赶趟儿。红的像火，粉的像霞，白的像雪。花里带着甜味，闭了眼，树上仿佛已经满是桃儿、杏儿、梨儿！花下成千成百的蜜蜂嗡嗡地闹着，大小的蝴蝶飞来飞去。野花遍地是：杂样儿，有名字的，没名字的，散在草丛里，像眼睛，像星星，还眨呀眨的。</p>
    </div>
    </body>
    </html>
```

14.5.2 定义背景裁剪区域

background-clip 属性定义背景图像的裁剪区域。background-clip 属性与 background-origin 属性有几分关联度。其中 background-clip 属性用来判断背景是否包含边框区域，而 background-origin 属性用来决定 background-position 属性定位的参考位置，它们的属性取值也很相似。该属性的基本语法如下所示。

```
background-clip:border | padding | content | no-clip;
```

background-clip 属性的初始值是 border-box，适用于所有元素。取值简单说明如下。

☑ border：从边框区域向外裁剪背景。

☑ padding：从补白区域向外裁剪背景。

☑ content：从内容区域向外裁剪背景。

☑ no-clip：从边框区域向外裁剪背景。

Note

目前，Webkit 引擎支持-webkit-backgroundclip 私有属性，Mozilla Gecko 引擎支持-moz-background-clip 私有属性，Presto 引擎和 IE9+浏览器支持该属性，但是 Firefox 不支持该属性。

background-clip 属性和 background-origin 属性实现的效果基本相同，但是它们的实现原理是不同的。在具体设计中，设计师可以配合 background-clip 属性和 background-origin 属性，以实现相同的效果，这样能够兼容各种主流浏览器。

根据 CSS 盒模型原理，任何一个元素都会包含 4 区域、4 边沿，即边界区域、边框区域、补白区域和内容区域，以及边界边缘、边框边缘、补白边缘和内容边缘，如图 14.21 所示。

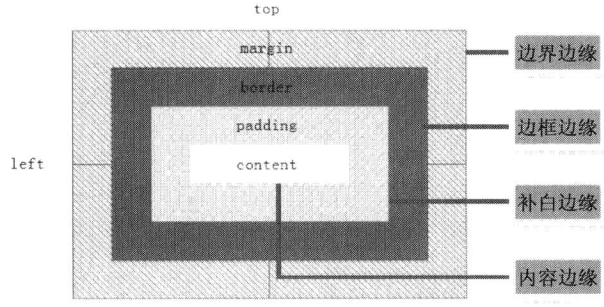


图 14.21　CSS 盒模型基本结构示意图

对于 background-clip 属性来说，如果取值为 padding，则 background-image 将忽略补白边缘，此时边框区域显示为透明；如果取值为 border，则 background-image 将包括边框区域；如果取值为 content，则 background-image 将只包含内容区域；如果 background-image 属性定义了多重背景，则 background-clip 属性值可以设置多个值，并用逗号分隔。对于 background-origin 属性来说，如果取值为 padding，则 background-position 相对于补白边缘进入定位，其中当 background-position 属性值为"0 0"时，定位点为补白边缘的左上角，而当 background-position 属性值为"100% 100%"时，定位点为补白边缘的为右下角。如果取值为 border，则 background-position 相对于边框边缘。如果取值为 content，则 backgroundposition 相对于内容边缘。与 background-clip 属性相同，多个值之间使用逗号分隔。

如果 background-clip 属性值为 padding，background-origin 属性取值为 border，且 background-position 属性值为"top left"（默认初始值），则背景图左上角将会被截取掉部分。

总之，background-clip 默认效果类似于 background-clip:border。background-origin 默认效果类似于 background-origin:padding。

【示例 1】设计内容区背景。background-clip 属性用法很简单，本示例演示如何设计背景图像仅在内容区域内显示，演示效果如图 14.22 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>background-clip</title>
```

Note

```
<style type="text/css">
div {
    height:50px;
    width:200px;
    border:solid 50px gray;
    padding:50px;
    background:url(images/bg2.jpg) no-repeat;
    -moz-background-size:cover;
    -webkit-background-size:cover;
    background-size:cover;
    -moz-background-clip:content;
    -webkit-background-clip:content;
    -o-background-clip:content;
    -khtml-background-clip:content;
    background-clip:content;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

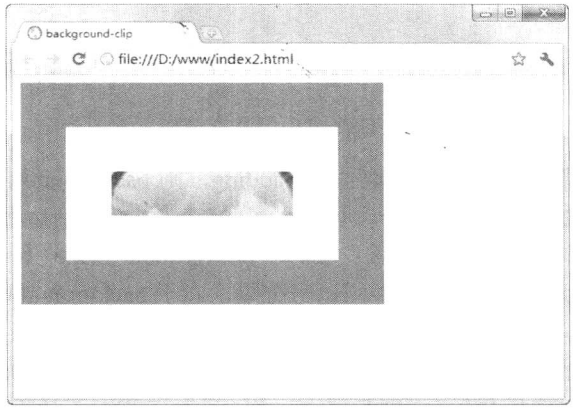


图 14.22　以内容边缘裁切背景图像效果

【示例 2】设计按钮效果。在本示例中同时定义 background-clip 和 background-origin 属性值为 content，可以设计比较特殊的按钮样式，演示效果如图 14.23 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>background-clip</title>
<style type="text/css">
button {
    height:40px;
    width:150px;
    padding:1px;
```

Note

```
        cursor:pointer;
        color:#fff;
        border:3px double #95071b;
        border-right-color:#650513;
        border-bottom-color:#650513;
        /*为了避免背景图像重复平铺到边框区域，应禁止它平铺*/
        background:url(images/img5.jpg) no-repeat;
        /*设计背景图像的定位坐标点为元素内容区域的左上角*/
        -moz-background-origin:content;
        -webkit-background-origin:content;
        background-origin:content;
        /*设计背景图像的以内容区域的边缘进行裁切背景图像*/
        -moz-background-clip:content;
        -webkit-background-clip:content;
        background-clip:content;
    }
    </style>
    </head>
    <body>
    <button>导航按钮 >></button>
    </body>
    </html>
```

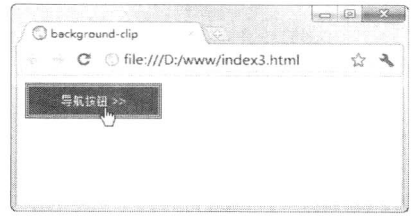


图 14.23　设计按钮效果

14.5.3　定义背景图像大小

在 CSS2 及其以前版本中，背景图像的大小是不可以控制的，如果要想使背景图像填充元素背景区域，则需要事先设计更大的背景图像，否则只能让背景图像以平铺的方式来填充元素。CSS3 新增的 background-size 属性可以控制背景图像的显示大小。background-size 属性的基本语法如下所示。

```
background-size:[ <length> | <percentage> | auto ]{1,2} | cover | contain;
```

background-size 属性的初始值为 auto，适用于所有元素。取值简单说明如下。

☑　<length>：由浮点数字和单位标识符组成的长度值，不可为负值。

☑　<percentage>：取值为 0%~100%之间的值，不可为负值。

☑　cover：保持背景图像本身的宽高比例，将图片缩放到正好完全覆盖所定义背景的区域。

☑　contain：保持图像本身的宽高比例，将图片缩放到宽度或高度正好适应所定义背景的区域。

background-size 属性可以设置一个或两个值，一个为必填，一个为可选。其中第一个值用于指定背景图像的 width，第二个值用于指定背景图像的 height，如果只设置一个值，则第二个值默认为 auto。目前，Webkit 引擎支持-webkit-backgroundsize 私有属性，Mozilla Gecko 引擎支持-moz- background-size 私有属性，Presto 引擎和 IE 9+浏览器支持该属性。

【示例】设计自适应模块大小的背景图像。借助 image-size 属性自由定制背景图像大小的功能，让背景图像自适应盒子的大小，从而可以设计与模块大小完全适应的背景图像，本示例效果如图 14.24 所示，只要背景图像长宽比与元素长宽比相同，就不用担心背景图像与模块区域脱节。

Note

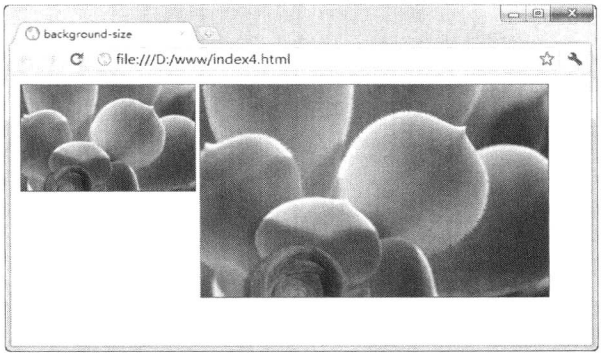


图 14.24　设计自适应背景大小效果

```
<!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>background-size</title>
<style type="text/css">
div {
    margin:2px;
    float:left;
    border:solid 1px red;
    background:url(images/img2.jpg) no-repeat center;
    /*设计背景图像完全覆盖元素区域*/
    -moz-background-size:cover;
    -webkit-background-size:cover;
    background-size:cover;
}
/*设计元素大小*/
.h1 { height:120px; width:192px; }
.h2 { height:240px; width:384px; }
</style>
</head>
<body>
<div class="h1"></div>
<div class="h2"></div>
</body>
</html>
```

14.5.4　定义背景图像循环方式

在 CSS3 中，可以使用 background-break 属性定义平铺内联元素背景图像时的循环方式。该属性包括 3 个值： bounding-box、each-box 和 continuous。这 3 种循环方式在使用 Firefox 浏览器的时候，

需要在样式代码中使用-moz-background-inline-policy 私有属性。到目前为止，该属性还没有得到其他浏览器的支持。

将 background-break 属性指定为 bounding-box 的时候，背景图像在整个内联元素中进行平铺。指定为 each-box 的时候，背景图像在每一行中进行平铺，指定为 continuous 的时候，下一行中的图像紧接着上一行中的图像继续平铺。

Note

【示例】在本示例中定义 3 个 div 元素，每一个 div 元素内部又包含了一个 span 元素，每一个 span 元素都具有背景图像，在样式代码中分别针对 3 个 span 元素指定 3 种 background-break 属性值，通过运行结果的显示可以看出这 3 种循环方式的区别。在浏览器中预览，则显示效果如图 14.25 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title></title>
</head>
<style type="text/css">
span {
    background-image: url(images/bg1.jpg);
    padding: 0.2em;
    line-height: 1.5;
    font-size: 1em;
    font-weight: bold;
}
div.div1 span { -moz-background-inline-policy: bounding-box; }
div.div2 span { -moz-background-inline-policy: each-box; }
div.div3 span { -moz-background-inline-policy: continuous; }
</style>
<body>
<div class="div1" ><span>让我怎么感谢您，我刚走向你的时候，原想收获一缕春风，你却给了我整个春天；你让我怎么感谢您，当我走向你的时候，我原想捧起一簇浪花，您却给了我整个海洋。</span></div>
<div class="div2"><span>让我怎么感谢您，我刚走向你的时候，原想收获一缕春风，你却给了我整个春天；你让我怎么感谢您，当我走向你的时候，我原想捧起一簇浪花，您却给了我整个海洋。</span></div>
<div class="div3"><span>让我怎么感谢您，我刚走向你的时候，原想收获一缕春风，你却给了我整个春天；你让我怎么感谢您，当我走向你的时候，我原想捧起一簇浪花，您却给了我整个海洋。</span></div>
</body>
</html>
```

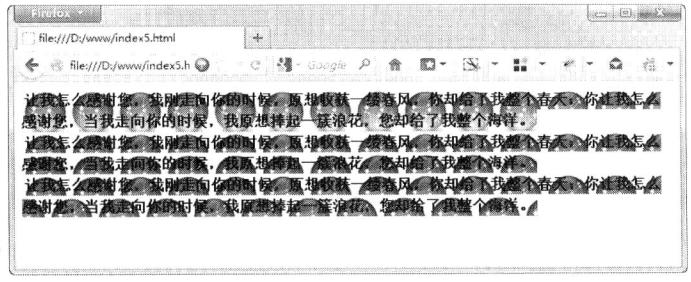


图 14.25　定义背景图像的不同显示方式

14.5.5 定义多背景图像

Note

在 CSS3 中可以在一个元素里显示多个背景图像，还可以将多个背景图像进行重复显示，从而使得背景图像中所用素材的调整变得更加容易。

【示例 1】在本示例中为一个 div 元素定义多个背景图像，在浏览器中预览，显示效果如图 14.26 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title></title>
</head>
<style type="text/css">
div {
    background-image: url(images/bg3.jpg), url(images/bg1.jpg), url(images/bg2.jpg);
    background-repeat: no-repeat, repeat-x, no-repeat;
    background-position: 3%, 98%, 85%, center center, top;
    width: 500px;
    padding: 160px 0px;
}
</style>
<body>
<div></div>
</body>
</html>
```

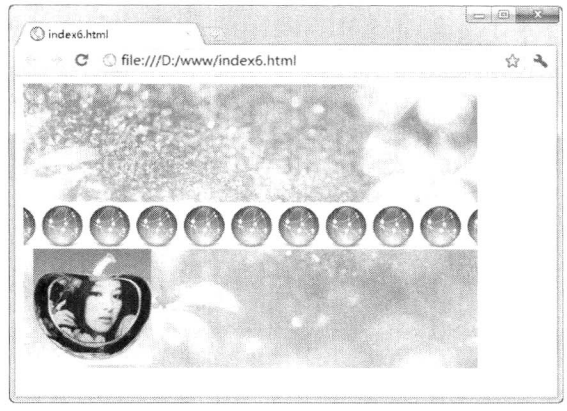


图 14.26 定义多背景图像

在 div 元素的样式代码中，上面示例用到了几个关于背景的属性：background-image、background-repeat 和 background-position 属性。这些属性都是 CSS1 中就有的属性，但是在 CSS3 中，通过利用逗号作为分隔符来同时指定多个属性的方法，可以指定多个背景图像，并且实现了在一个元素中显示多个背景图像的功能。

注意，在使用 background-image 属性来指定图像文件的时候，是按在浏览器中显示时图像叠放的顺序从上往下指定的，第一个图像文件是放在最上面的，最后指定的文件是放在最下面的。另外，通

过多个 background-repeat 属性与 background-position 属性的指定，可以单独指定背景图像中某个图像文件的平铺方式与放置位置。

通过指定多个 background-image、background-repeat 和 background-position 属性，实现了在一个元素的背景中显示多个图像文件的功能。具体来说，允许多重指定并配合多个图像文件一起使用的属性有如下几个：

Note

- ☑ background-image
- ☑ background-repeat
- ☑ background-position
- ☑ background-clip
- ☑ background-origin
- ☑ background-size

第15章

2D 变形

（视频讲解：31 分钟）

在 CSS3 中，如果使用动画功能，可以使页面上的文字或画像具有动画效果，可以使背景色从一种颜色平滑过渡到另一种颜色。CSS3 所提供的动画功能，主要包括变形、转换和动画技术。变形是最基本的动画形式，它主要通过 CSS 控制元素样式属性值的变化来实现。

利用 transform 功能来实现文字或图像的旋转、缩放、倾斜和移动这 4 种类型的变形处理，本章将结合具体应用案例以及浏览器支持情况，详细讲解 2D 变形动画的实现和实践。

15.1　认识 transform

Note

2009 年 3 月 W3C 组织正式发布 3D 变形动画标准草案（http://www.w3.org/TR/css3-3dtransforms/）。同年 12 月 W3C 在 3D 草案基础上，又发布了 2D 变形动画标准草案（http://www.w3.org/TR/css3-2d-transforms/），两个草案的核心内容基本相似，但是针对的主体不同，一个是 3D 动画，另一个是 2D 动画。

CSS 2D Transform 获得了各主流浏览器的支持，但是 CSS 3D Transform 仅能够在 Mac 系统下的 Safari 4.0+版本浏览器中获得支持。

transform 实现了一些可用 SVG 实现的变形功能。它可用于内联（inline）元素和块级（block）元素。该属性可以旋转、缩放和移动元素。熟练使用 transform 属性可以控制文字的变形，这种纯 CSS 的方法，可以确保网页内的文字保持可选，这是 CSS 相对于使用图片（或背景图片）的一个巨大优势。transform 属性的基本语法如下所示。

```
transform:none | <transform-function> [ <transform-function> ]*;
```

transform 属性的初始值是 none，适用于块元素和行内元素。取值简单说明如下。

☑ <transform-function>：设置变形函数。可以是一个或多个变形函数列表。transform-function 函数包括 matrix()、translate()、scale()、scaleX()、scaleY()、rotate()、skewX()、skewY()和 skew()等。关于这些常用变形函数的功能简单说明如下。

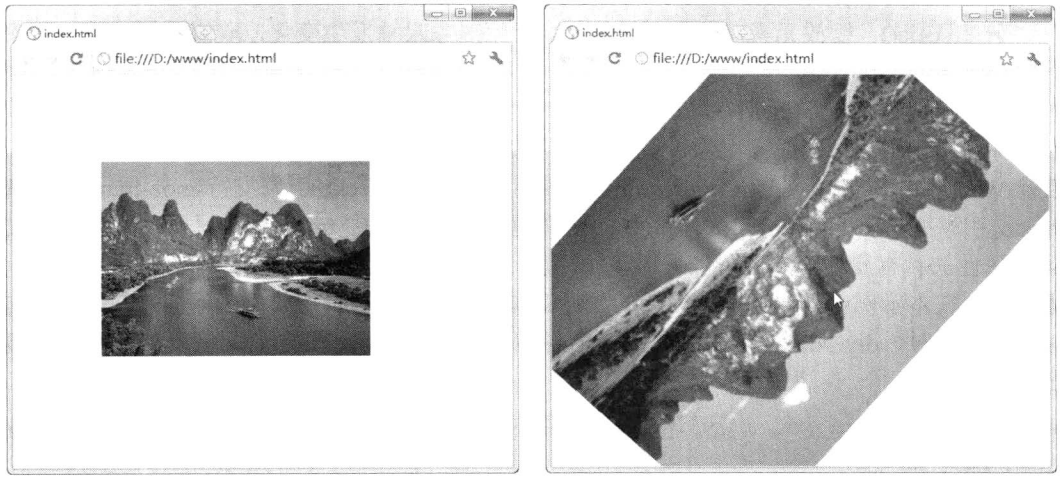
- matrix()：定义矩阵变换，即基于 X 和 Y 坐标重新定位元素的位置。
- translate()：移动元素对象，即基于 X 和 Y 坐标重新定位元素。
- scale()：缩放元素对象，可以使任意元素对象尺寸发生变化，取值包括正数、负数以及小数。
- rotate()：旋转元素对象，取值为一个度数值。
- skew()：倾斜元素对象，取值为一个度数值。

目前 Webkit 引擎支持-webkit-transform 私有属性，Mozilla Gecko 引擎支持-moz-transform 私有属性，Presto 引擎支持-o-transform 私有属性，IE 浏览器暂时不支持 transform 属性，也没有定义支持 transform 属性的私有属性，不过可以采用 IE 滤镜来进行兼容。

【示例】在本示例中，通过一个简单的鼠标动画体验 CSS3 动画是如何工作的。在这个动画中，当鼠标经过图像时，div 元素会放大，并旋转渐隐显示蓝色背景，移开之后又恢复默认效果，演示效果如图 15.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    position: absolute;
    left: 100px;
    top: 100px;
```

```
        width: 306px;
        height: 226px;
        /*使用背景图像覆盖 div 元素盒子*/
        background: #0FF url(images/bg2.jpg) no-repeat;
        -webkit-background-size: cover ;
        -o-background-size: cover ;
        background-size: cover ;
        /*定义动画的过程*/
        -webkit-transition:-webkit-transform .5s ease-in, background .5s ease-in;
        -moz-transition:-moz-transform .5s ease-in, background .5s ease-in;
        -o-transition:-o-transform .5s ease-in, background .5s ease-in;
        transition:transform .5s ease-in, background .5s ease-in;
}
div:hover {
        /*定义动画的状态*/
        /*旋转并放大 div 元素对象*/
        -webkit-transform: rotate(180deg) scale(2);
        -moz-transform: rotate(180deg) scale(2);
        -o-transform: rotate(180deg) scale(2);
        -transform: rotate(180deg) scale(2);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```



默认状态　　鼠标经过时逐步被旋转、放大

图 15.1　定义简单的 2D 动画效果

15.2　2D 变形

在 CSS3 中，Transitions 功能通过将元素的某个属性从一个属性值在指定的时间内平滑过渡到另

一个属性值来实现动画功能，可通过 transitions 属性来使用 Transitions 功能。但是如果设计更加细致的变形效果，还需要调用变形函数，这样用户只需要设置关键帧的样式属性，CSS 会自动补全动画过程中样式属性变化。

Note

15.2.1　旋转动画

rotate()函数能够旋转指定的元素对象，主要在二维空间内进行操作，接受一个角度参数值，用来指定旋转的幅度。元素对象可以是内联元素和块级元素。语法格式如下。

```
rotate(<angle>)
```

rotate()函数用法及其动画演示示意图如图 15.2 所示。

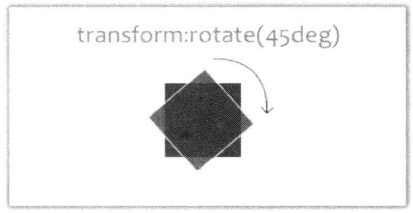


图 15.2　旋转动画示意图

针对旋转动画，IE 浏览器虽然不支持，但是可以考虑使用图形旋转滤镜进行兼容。语法格式化如下。

```
filter:progid:DXImageTransform.Microsoft.BasicImage(enabled=bEnabled,grayScale=bGray, mirror=bMirror,
opacity=fOpacity,XRay=bXRay )
```

参数说明如下。

- ☑ enabled：可选项，布尔值。设置或检索滤镜是否激活。默认值 true 表示滤镜激活，取值 false 表示滤镜被禁止。
- ☑ grayScale：可选项，布尔值。设置或检索是否以灰度显示对象内容。取值为 1 表示以灰度效果显示对象内容。0 为默认值，表示显示对象的原始色彩。
- ☑ mirror：可选项，布尔值。设置或检索是否反转显示对象内容。取值为 1 表示反转显示对象内容。0 为默认值，表示正常显示对象内容。
- ☑ opacity：可选项，浮点数。设置对象内容的透明度。取值范围为 0.0~1.0，默认值为 1.0，表示不透明黑色，0.0 表示为完全透明。
- ☑ XRay：可选项，布尔值。设置或检索是否以 X 光效果显示对象内容。取值为 1 表示以 X 光效果显示对象内容。0 为默认值，表示正常显示对象内容。

IE 图形旋转滤镜可以有 4 个旋转值：0、1、2 和 3。它无法像 Webkit 和 Gecko 引擎那样更精密地控制旋转的角度，不过可以在一定程度上保持兼容。

【示例】在本示例中演示了设置 div 元素在鼠标经过时如何逆时针旋转 90°，演示效果如图 15.3 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Note

```
<title></title>
<style type="text/css">
div {
    margin:100px 0;
    width: 400px;
    height: 100px;
    background:url(images/bg1.jpg) center;
}
div:hover {
    /*定义动画的状态*/
    -webkit-transform: rotate(-90deg);
    -moz-transform: rotate(-90deg);
    -o-transform: rotate(-90deg);
    -transform: rotate(-90deg);
    filter: progid:DXImageTransform.Microsoft.BasicImage(rotation=3);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

默认状态　　鼠标经过时被旋转

图 15.3　定义旋转动画效果

注意，IE 浏览器在怪异模式下不支持 filter 属性，应该使用 IE 私有属性进行定义，即-ms-filter，而不是 filter。例如，在上面示例中可以添加这样一个声明：

```
-ms-filter:progid:DXImageTransform.Microsoft.BasicImage(rotation=3);
```

15.2.2　缩放动画

scale()函数能够缩放元素大小，该函数包含两个参数值，分别用来定义宽和高缩放比例。语法格式如下：

```
scale(<number>[, <number>])
```

<number>参数值可以是正数、负数和小数。正数值基于指定的宽度和高度将放大元素。负数值

不会缩小元素，而是翻转元素（如文字被反转），然后再缩放元素。使用小于 1 的小数（如 0.5）可以缩小元素。如果第二个参数省略，则第二个参数等于第一个参数值。

scale ()函数用法及其动画演示示意图如图 15.4 所示。

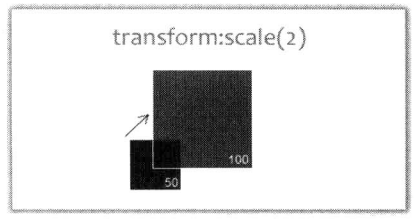


图 15.4　缩放动画示意图

scale()函数目前只有 Firefox、Safari、Chrome 以及 Opera 10.50+支持，到目前为止没有 IE 版本支持。缩放对象是相当有意义的功能，使用它可以渐进增强:hover 可用性。

【示例】在导航菜单中添加缩放功能，让导航菜单更好用，演示效果如图 15.5 所示。注意，当为 scale()函数传递不同参数值时，缩放动画的效果是不同的。

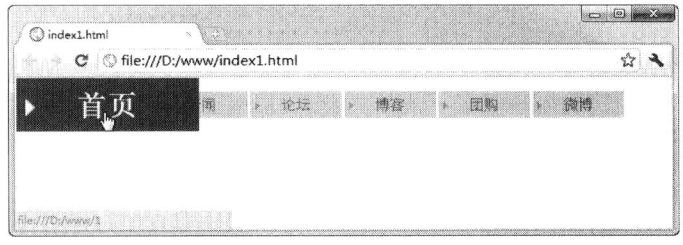


图 15.5　缩放动画效果

```
<!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></title>
<style type="text/css">
.test ul { list-style:none; }
.test li {
    float:left;
    width:100px;
    background:#CCC;
    margin-left:3px;
    line-height:30px;
}
.test a {
    display:block;
    text-align:center;
    height:30px;
}
.test a:link {
    color:#666;
    background:url(images/icon1.jpg) #CCC no-repeat 5px 12px;
```

```
        text-decoration:none;
    }
    .test a:visited {
        color:#666;
        text-decoration:underline;
    }
    .test a:hover {
        color:#FFF;
        font-weight:bold;
        text-decoration:none;
        background:url(images/icon2.jpg) #F00 no-repeat 5px 12px;
        /*设置 a 元素在鼠标经过时放大 2 倍尺寸进行显示*/
        -webkit-transform: scale(2);
        -moz-transform: scale(2);
        -o-transform: scale(2);
    }
    </style>
    </head>
    <body>
    <div class="test">
        <ul>
            <li><a href="1">首页</a></li>
            <li><a href="2">新闻</a></li>
            <li><a href="3">论坛</a></li>
            <li><a href="4">博客</a></li>
            <li><a href="5">团购</a></li>
            <li><a href="6">微博</a></li>
        </ul>
    </div>
    </body>
    </html>
```

15.2.3 移动动画

translate()函数能够重新定位元素的坐标，该函数包含两个参数值，分别用来定义 X 轴和 Y 轴坐标。语法格式如下：

```
translate(<translation-value>[,<translation-value>])
```

<translation-value>参数表示坐标值，第一个参数表示相对于原位置的 X 轴偏移距离，第二个参数表示相对于原位置的 Y 轴偏移距离，如果省略了第二个参数，则第二个参数值默认值为 0。

translate ()函数用法及其动画演示示意图如图 15.6 所示。

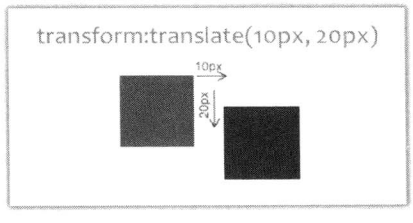


图 15.6 移动动画示意图

Firefox、Safari、Chrome 以及 Opera 10.50 支持 translate()函数，到目前为止没有 IE 版本支持。

【示例】在本示例中给导航菜单添加定位功能，让导航菜单更富动感，演示效果如图 15.7 所示。

Note

```
<!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>translate()函数</title>
<style type="text/css">
.test ul { list-style:none; }
.test li {
    float:left;
    width:100px;
    background:#CCC;
    margin-left:3px;
    line-height:30px;
}
.test a {
    display:block;
    text-align:center;
    height:30px;
}
.test a:link {
    color:#666;
    background:url(images/icon1.jpg) #CCC no-repeat 5px 12px;
    text-decoration:none;
}
.test a:visited {
    color:#666;
    text-decoration:underline;
}
.test a:hover {
    color:#FFF;
    font-weight:bold;
    text-decoration:none;
    background:url(images/icon2.jpg) #F00 no-repeat 5px 12px;
    /*设置 a 元素在鼠标经过时向右下角位置偏移 4 个像素*/
    -moz-transform: translate(4px,4px);
    -webkit-transform: translate(4px,4px);
    -o-transform: translate(4px,4px);
}
</style>
</head>
<body>
<div class="test">
    <ul>
        <li><a href="1">首页</a></li>
        <li><a href="2">新闻</a></li>
        <li><a href="3">论坛</a></li>
```

```
            <li><a href="4">博客</a></li>
            <li><a href="5">团购</a></li>
            <li><a href="6">微博</a></li>
        </ul>
    </div>
    </body>
    </html>
```

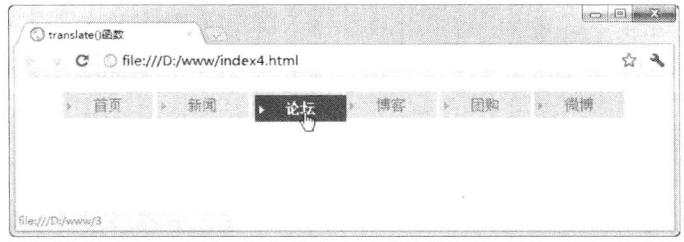


图 15.7　移动动画效果

注意： 当为 translate()函数传递一个参数值时，则表示水平偏移，如果垂直偏移，则应设置第一个参数值为 0，第二个参数值为垂直偏移值。如果设置为负数，则表示反向偏移，但是参考距离不同。

15.2.4　倾斜动画

skew()函数能够让元素倾斜显示，该函数包含两个参数值，分别用来定义 X 轴和 Y 轴坐标倾斜的角度。语法格式如下：

```
skew(<angle> [,<angle>])
```

<angle>参数表示角度值，第一个参数表示相对于 X 轴进行倾斜，第二个参数表示相对于 Y 轴进行倾斜，如果省略了第二个参数，则第二个参数值默认值为 0。

skew()也是一个很有用的变形函数，它可以将一个对象围绕着 X 轴和 Y 轴按照一定的角度倾斜。这与 rotate()函数的旋转不同，rotate()函数只是旋转，而不会改变元素的形状，skew()函数会改变元素的形状。

【示例】在本示例中给导航菜单添加倾斜变形功能，让导航菜单更富情趣，演示效果如图 15.8 所示。

```
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>translate()函数</title>
    <style type="text/css">
    .test ul { list-style:none; }
    .test li {
        float:left;
        width:100px;
        background:#CCC;
        margin-left:3px;
```

Note

```
        line-height:30px;
    }
    .test a {
        display:block;
        text-align:center;
        height:30px;
    }
    .test a:link {
        color:#666;
        background:url(images/icon1.jpg) #CCC no-repeat 5px 12px;
        text-decoration:none;
    }
    .test a:visited {
        color:#666;
        text-decoration:underline;
    }
    .test a:hover {
        color:#FFF;
        font-weight:bold;
        text-decoration:none;
        background:url(images/icon2.jpg) #F00 no-repeat 5px 12px;
        /*设置 a 元素在鼠标经过时向左下角位置倾斜*/
        -moz-transform: skew(30deg,-10deg);
        -webkit-transform: skew(30deg,-10deg);
        -o-transform: skew(30deg,-10deg);
    }
    </style>
    </head>
    <body>
    <div class="test">
        <ul>
            <li><a href="1">首页</a></li>
            <li><a href="2">新闻</a></li>
            <li><a href="3">论坛</a></li>
            <li><a href="4">博客</a></li>
            <li><a href="5">团购</a></li>
            <li><a href="6">微博</a></li>
        </ul>
    </div>
    </body>
    </html>
```

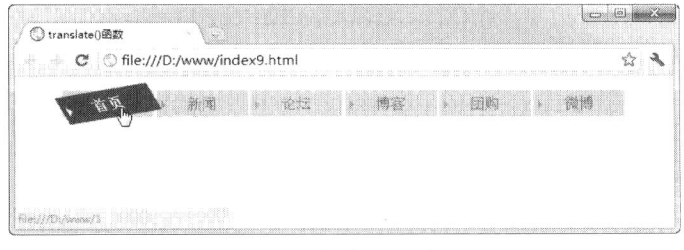


图 15.8　倾斜动画效果

15.2.5 变形动画

Note

matrix()是矩阵函数，调用该函数可以非常灵活地实现各种变形效果。matrix()函数包括 6 个参数（a、b、c、d、e 和 f），实际上 matrix()函数是一个 3×3 矩阵，经过该矩阵将旧的参数转换成新值。

$$\begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

该函数的矩阵方程可以参考 http://www.w3.org/TR/SVG/coords.html#TransformMatrixDefined，这是 SVG 的一个文档，但是对于 matrix()函数的变换原理是通用的。matrix()函数的语法格式如下。

```
matrix(<number>,<number>,<number>,<number>,<number>,<number>)
```

IE 支持 matrix 滤镜，使用这个滤镜基本上也可以实现相同的效果，不过读者需要理解矩阵运算原理。

【示例 1】在本示例中利用 matrix()函数的矩阵变形给导航菜单添加动态变形效果，演示效果如图 15.9 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
.test ul { list-style:none; }
.test li {
    float:left;
    width:100px;
    background:#CCC;
    margin-left:3px;
    line-height:30px;
}
.test a {
    display:block;
    text-align:center;
    height:30px;
}
.test a:link {
    color:#666;
    background:url(images/icon1.jpg) #CCC no-repeat 5px 12px;
    text-decoration:none;
}
.test a:visited {
    color:#666;
    text-decoration:underline;
}
.test a:hover {
    color:#FFF;
```

Note

```
        font-weight:bold;
        text-decoration:none;
        background:url(images/icon2.jpg) #F00 no-repeat 5px 12px;
        /*设置 a 元素在鼠标经过时矩阵变形*/
        -moz-transform: matrix(1,0.4,0,1,0,0);
        -webkit-transform: matrix(1,0.4,0,1,0,0);
        -o-transform: matrix(1,0.4,0,1,0,0);
    }
    </style>
    </head>
    <body>
    <div class="test">
        <ul>
            <li><a href="1">首页</a></li>
            <li><a href="2">新闻</a></li>
            <li><a href="3">论坛</a></li>
            <li><a href="4">博客</a></li>
            <li><a href="5">团购</a></li>
            <li><a href="6">微博</a></li>
        </ul>
    </div>
    </body>
    </html>
```

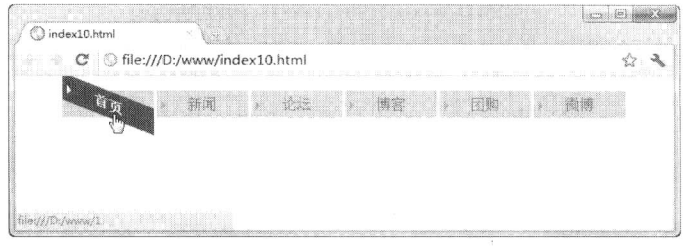


图 15.9　变形动画效果

在实战中，可能需要同时使用多种变形，以设计复杂的变形效果。为了方便代码编写，CSS3 支持缩写形式。

【示例 2】在下面示例中同时对 div 元素对象执行缩放、旋转和移动操作，演示效果如图 15.10 所示。

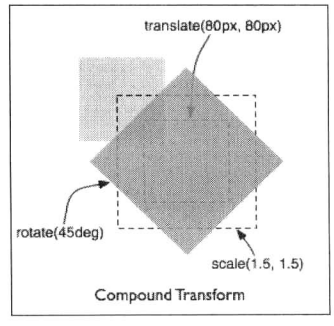


图 15.10　复杂的变形操作

其中变形样式代码如下：

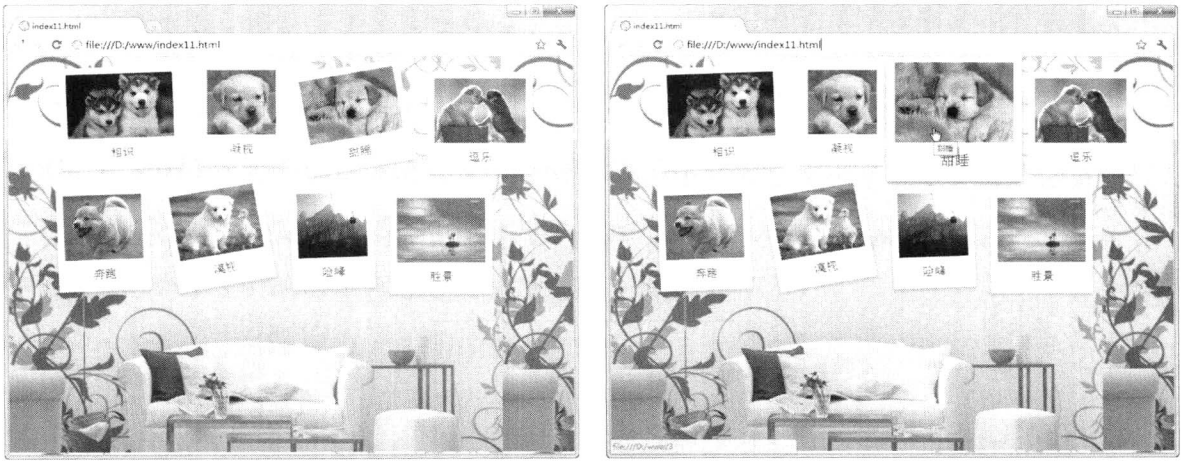
```
div {
    -moz-transform: translate(80,80);
    -webkit-transform: translate(80,80);
    -o-transform: translate(80,80);
    -moz-transform: rotate(45deg);
    -webkit-transform: rotate(45deg);
    -o-transform: rotate(45deg);
    -moz-transform: scale(1.5,1.5);
    -webkit-transform: scale(1.5,1.5);
    -o-transform: scale(1.5,1.5);
}
```

上面的样式可以缩写为：

```
div {
    -moz-transform: translate(80,80) rotate(45deg) scale(1.5,1.5);
    -webkit-transform: translate(80,80) rotate(45deg) scale(1.5,1.5);
    -o-transform: translate(80,80) rotate(45deg) scale(1.5,1.5);
}
```

15.2.6 案例实战：设计涂鸦墙

综合使用移动、旋转、缩放等函数控制元素而不使用 JavaScript，就可以创建更丰富、更轻量的界面和应用。在这个综合案例中，将用到 CSS3 阴影、透明效果，以及变形动画，让图片随意贴在墙上，当鼠标移动到图片上时，会自动放大并垂直摆放，演示效果如图 15.11 所示。在默认状态下，图片被随意显示在墙面上，鼠标经过图片时，图片会竖直摆放，并被放大显示。



默认状态　　鼠标经过

图 15.11　设计涂鸦墙效果

```
<!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>
```

Note

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title> </title>
<style type="text/css">
body { background:#fff url(images/bg3.JPG); }
ul.polaroids li { display: inline; }
ul.polaroids a {
    background: #fff;
    display: inline;
    float: left;
    margin: 0 0 27px 30px;
    width: auto;
    padding: 10px 10px 15px;
    text-align: center;
    font-family: "Marker Felt",sans-serif;
    text-decoration: none;
    color: #333;
    font-size: 18px;
    /*为图片外框设计阴影效果*/
    -webkit-box-shadow: 0 3px 6px rgba(0,0,0,.25);
    -moz-box-shadow: 0 3px 6px rgba(0,0,0,.25);
    /*顺时针旋转 2° */
    -webkit-transform: rotate(-2deg);
    -webkit-transition: -webkit-transform .15s linear;
    -moz-transform: rotate(-2deg);
}
ul.polaroids img {
    display: block;
    height: 100px;
    margin-bottom: 12px;
}
ul.polaroids a:after { content: attr(title); }
ul.polaroids li:nth-child(even) a {
    /*逆时针旋转 5° */
    -webkit-transform: rotate(5deg);
    -moz-transform: rotate(5deg);
}
ul.polaroids li:nth-child(2n) a {
    position: relative;
    top: -5px;
    /*不选择对象*/
    -webkit-transform: none;
    -moz-transform: none;
}
ul.polaroids li:nth-child(3n) a {
    position: relative;
    right: 5px;
    /*顺时针旋转 10° */
    -webkit-transform: rotate(-10deg);
    -moz-transform: rotate(-10deg);
}
```

```
ul.polaroids li:nth-child(4n) a {
    position: relative;
    right: 5px;
    top: 8px;
}
ul.polaroids li:nth-child(5n) a {
    position: relative;
    left: -5px;
    top: 3px;
}
ul.polaroids li a:hover {
    /*放大对象 1.25 倍*/
    -webkit-transform: scale(1.25);
    -moz-transform: scale(1.25);
    -webkit-box-shadow: 0 3px 6px rgba(0,0,0,.5);
    -moz-box-shadow: 0 3px 6px rgba(0,0,0,.5);
    position: relative;
    z-index: 5;
}
</style>
</head>
<body>
<ul class="polaroids">
    <li> <a href="1" title="相识"> <img src="images/11.jpg" alt="相识" /> </a> </li>
    <li> <a href="2" title="凝视"> <img src="images/22.jpg" alt="凝视" /> </a> </li>
    <li> <a href="3" title="甜睡"> <img src="images/33.jpg" alt="甜睡" /> </a> </li>
    <li> <a href="4" title="逗乐"> <img src="images/44.jpg" alt="逗乐" /> </a> </li>
    <li> <a href="5" title="奔跑"> <img src="images/55.jpg" alt="奔跑" /> </a> </li>
    <li> <a href="6" title="漠视"> <img src="images/66.jpg" alt="漠视" /> </a> </li>
    <li> <a href="7" title="险峰"> <img src="images/image-07.jpg" alt="险峰" /> </a> </li>
    <li> <a href="8" title="胜景"> <img src="images/image-08.jpg" alt="胜景" /> </a> </li>
</ul>
</body>
</html>
```

Note

15.3 自定义变形

CSS 变形的原点默认为对象的中心点，如果要改变这个中心点，可以使用 transform-origin 属性进行定义。例如，rotate 变形的默认原点是对象的中心点，使用 transform-origin 属性可以将原点设置在对象左上角，或者左下角，这样 rotate 变形的结果就不相同了。transform-origin 属性的基本语法如下所示。

```
transform-origin:[ [ <percentage> | <length> | left | center | right ] [ <percentage>| <length> | top | center | bottom ]? ]
| [ [ left | center | right ]|| [ top | center | bottom ] ]
```

transform-origin 属性的初始值为"50% 50%"，适用于块状元素和内联元素。transform-origin 接受两个参数，它们可以是百分比、em、px 等具体的值，也可以是 left、center、right，或者 top、middle、

bottom 等描述性关键字。

目前 Webkit 引擎支持-webkit-transformorigin 私有属性，Mozilla Gecko 引擎支持-moz-transform-origin 私有属性，Presto 引擎支持-o-transform-origin 私有属性，IE 浏览器暂时不支持 transform-origin 属性，也没有定义支持 transform-origin 属性的私有属性。

Note

【示例】通过改变变形对象的原点，可以实现不同的变形效果。在本示例中分别以图像的 4 个角点为原点来旋转图像，演示效果如图 15.12 所示。

图 15.12　自定义图片旋转

```
<!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></title>
<style type="text/css">
body { background:#fff url(images/bg3.JPG); }
ul.polaroids { margin-left:120px; }
ul.polaroids li { display: inline; }
ul.polaroids a {
    background: #fff;
    display: inline;
    float: left;
    margin: 0 0 27px 30px;
    width: auto;
    padding: 10px 10px 15px;
    text-align: center;
    text-decoration: none;
    color: #333;
```

```
        -webkit-box-shadow: 0 3px 6px rgba(0,0,0,.25);
        -moz-box-shadow: 0 3px 6px rgba(0,0,0,.25);
    }
    ul.polaroids img {
        display: block;
        height: 140px;
        margin-bottom: 12px;
    }
    ul.polaroids a:after { content: attr(title); }
    /*变形第 1 个对象*/
    ul.polaroids li:nth-child(1) a {
        /*以左上角为原点*/
        -moz-transform-origin: 0 0;
        -webkit-transform-origin: 0 0;
        -o-transform-origin: 0 0;
        /*顺时针旋转 30° */
        -webkit-transform: rotate(30deg);
        -moz-transform: rotate(30deg);
        -o-transform: rotate(30deg);
    }
    /*变形第 2 个对象*/
    ul.polaroids li:nth-child(2) a {
        /*以右上角为原点*/
        -moz-transform-origin: top right;
        -webkit-transform-origin: top right;
        -o-transform-origin: top right;
        /*逆时针旋转 30° */
        -webkit-transform: rotate(-30deg);
        -moz-transform: rotate(-30deg);
        -o-transform: rotate(-30deg);
    }
    </style>
    </head>
    <body>
    <ul class="polaroids">
        <li> <a href="1" title="亲亲狗狗"> <img src="images/44.jpg" /> </a> </li>
        <li> <a href="2" title="狗狗生气了"> <img src="images/66.jpg" /> </a> </li>
    </ul>
    </body>
    </html>
```

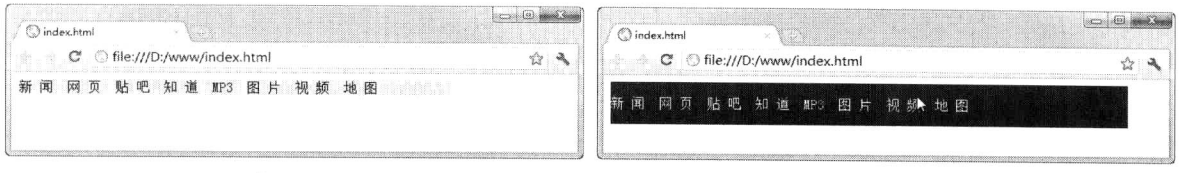
15.4 定义复杂变形

使用 Transitions 功能可以同时对多个属性值进行平滑过渡，通过这种方式可以设计复杂的变形效果。

【示例 1】在本示例中有一个 div 元素，元素的背景色为黄色，字体色为黑色，宽度为 300px，通过 hover 属性指定当鼠标指针停留在 div 元素上时的背景色为深蓝色，字体为白色，宽度为 400px。

通过 transitions 属性指定当鼠标指针移动到 div 元素上时在 1 秒钟内完成这几个属性值的平滑过渡。预览效果如图 15.13 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    background-color: #ffff00;
    color: #000000;
    width: 500px;
    -webkit-transition: background-color 1s linear,color 1s linear,width 1s linear;
    -moz-transition: background-color 1s linear,color 1s linear,width 1s linear;
    -o-transition: background-color 1s linear,color 1s linear,width 1s linear;
}
div:hover {
    background-color: #003366;
    color: #ffffff;
    width: 600px;
    height:50px;
    line-height:50px;
}
</style>
</head>
<body>
<div>新 闻　网 页　贴 吧　知 道　MP3　图 片　视 频　地 图</div>
</body>
</html>
```



默认效果　　鼠标经过时动画效果

图 15.13　定义复杂变形

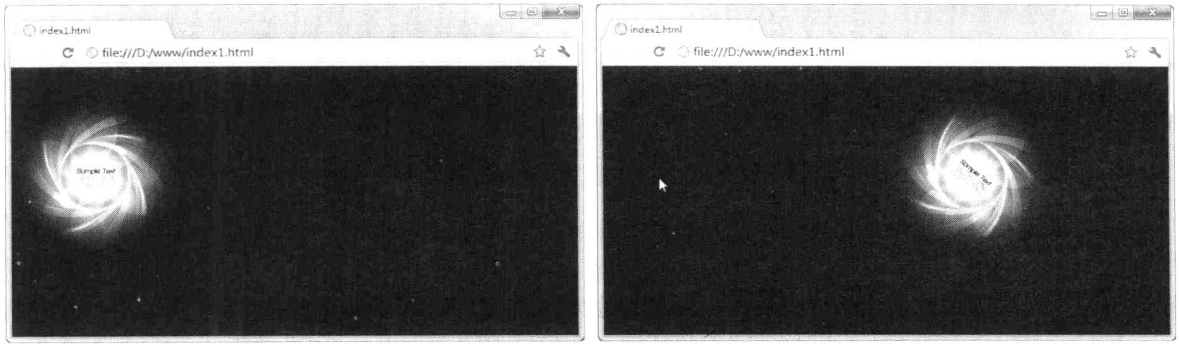
上面代码在浏览器中预览之后，当鼠标指针没有停留在 div 元素上时，页面显示如图 15.13 所示的默认效果。当鼠标指针停留在 div 元素上，该 div 元素的几个属性的属性值处于变化状态，最后 div 元素的几个属性的属性值变化结束后的页面显示为如图 15.13 所示的动画效果。

另外，可以通过改变元素的位置属性值或实现变形处理的 transform 属性值来让元素实现移动、旋转等动画效果。

【示例 2】本示例使用 Transitions 功能实现元素的移动与旋转动画，该示例中有一个 div 元素，当鼠标指针停留在 div 元素上，div 元素的属性值不断发生变化，在示例样式中调用了变形函数 rotate()，以便设计出更加细腻的变形效果，预览效果如图 15.14 所示。

Note

```
<!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></title>
<style type="text/css">
body{ background:#000; }
img {
    position: absolute;
    top: 40px;
    left: 0;
    height:200px;
    -webkit-transform: rotate(0deg);
    -webkit-transition: left 1s linear,-webkit-transform 1s linear;
    -moz-transform: rotate(0deg);
    -moz-transition: left 1s linear,-moz-transform 1s linear;
    -o-transform: rotate(0deg);
    -o-transition: left 1s linear,-o-transform 1s linear;
}
div:hover img {
    position: absolute;
    left: 500px;
    -webkit-transform: rotate(3000deg);
    -moz-transform: rotate(3000deg);
    -o-transform: rotate(3000deg);
}
</style>
</head>
<body>
<div> <img src="images/bg.jpg" alt="*" title="" /> </div>
</body>
</html>
```



默认效果　　鼠标经过时动画效果

图 15.14　定义复杂移动变形

示例 2 的运行结果分为如下 3 种情况：当鼠标指针没有停留在图像上时，页面显示如图 15.14 中的默认效果；当鼠标指针停留在图像上，图像正在向右移动和旋转时的页面显示如图 15.14 所示的动

画效果；当鼠标指针移开图像，图像会自动恢复默认显示效果。

使用 Transitions 功能实现动画效果的缺点是只能指定属性的开始值与终点值，然后在这两个属性值之间实现平滑过渡，不能实现更为复杂的效果。在 CSS3 中，除了使用 Transitions 功能外，还可以使用 Animations 功能来实现动画效果，它允许通过关键帧的指定来在页面上产生更复杂的动画效果，第 16 章中将针对 Animations 功能进行详细讲解。

Note

第16章

设计动画

（视频讲解：1小时20分钟）

CSS3动画分为Transition和Animations两种，二者均可以通过改变CSS中的属性值来产生动画效果。例如，通过改变background-color属性的属性值设计背景色从一种颜色平滑过渡到另一种颜色。到目前为止，Transitions功能支持从一个属性值平滑过渡到另一个属性值，Animations功能支持通过关键帧的指定在页面上产生更复杂的动画效果。

本章将对Transitions和Animations功能进行详细介绍，让读者能够使用Transitions功能来实现在属性值的开始值与结束值之间进行平滑过渡的动画。掌握CSS3中Animations功能的使用方法，能够在样式中创建多个关键帧、在这些关键帧中编写样式，并且能够在页面中创建结合这些关键帧所运行的较为复杂的动画。

16.1　平 滑 过 渡

Note

CSS Transformation 呈现的是一种变形结果，而 CSS Transition 呈现的是一种过渡，就是一种动画转换过程，如渐显、渐弱、动画快慢等。CSS Transformation 和 CSS Transition 是两种不同的动画模型，因此 W3C 为动画过渡定义了单独的模块，详细信息请参阅 http://www.w3.org/TR/css3-transitions/页面。

过渡可以与变形同时使用。例如，触发:hover 或者:focus 事件后创建动画过程，如淡出背景色，滑动一个元素以及让一个对象旋转都可以通过 CSS 转换实现。

transition 属性是一个复合属性，可以同时定义 transition-property、transitionduration、transition-timing-function、transition-delay、transition-property、transition-duration、transition-timing-function 和 transition-delay 子属性值。

transition 属性的基本语法如下所示，其初始值根据各个子属性的默认值而定。

```
transition:[<'transition-property'> || <'transition-duration'> ||<'transition-timing-function'> || <'transition-delay'>
[,[<'transition-property'> || <'transition-duration'> ||<'transition-timing-function'> || <'transition-delay'>]]*
```

目前 Webkit 引擎支持-webkit-transition 私有属性，Mozilla Gecko 引擎支持-moz-transition 私有属性，Presto 引擎支持-o-transition 私有属性，IE 浏览器暂时不支持 transition 属性，也没有定义支持 transition 属性的私有属性。

16.1.1　定义过渡属性

transition-property 属性用来定义转换动画的 CSS 属性名称，如 background-color 属性。该属性的基本语法如下所示。

```
transition-property:none | all | [ <IDENT> ] [',' <IDENT> ]*;
```

transition-property 属性初始值为 all，适用于所有元素以及:before 和:after 伪元素。取值简单说明如下。

- ☑ none：表示没有元素。
- ☑ all：表示针对所有元素。
- ☑ IDENT：指定 CSS 属性列表。

【示例】在本示例中，指定动画的属性为背景颜色。这样当鼠标光标经过 div 对象时，会自动从红色背景过渡到蓝色背景，演示效果如图 16.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    background-color:red;
    width:400px;
    height:200px;
```

```
}
div:hover {
    background-color:blue;
    /*指定动画过渡的 CSS 属性*/
    -webkit-transition-property:background-color;
    -moz-transition-property:background-color;
    -o-transition-property:background-color;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

Note

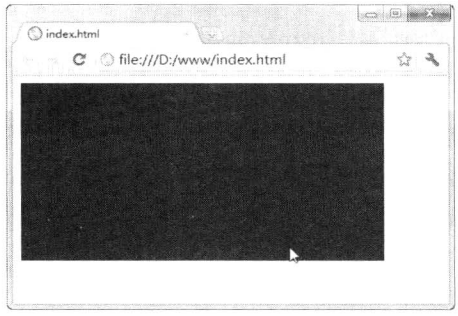


图 16.1　简单的背景色切换动画

16.1.2　定义过渡时间

transition-duration 属性用来定义转换动画的时间长度，即设置从旧属性换到新属性花费的时间，单位为秒。该属性的基本语法如下所示。

```
transition-duration:<time> [, <time>]*;
```

transition-duration 属性初始值为 0，适用于所有元素以及:before 和:after 伪元素。在默认情况下，动画过渡时间为 0 秒，所以当指定元素动画时，会看不到过渡的过程，直接看到结果。

【示例】在本示例中，设置动画过渡时间为 2 秒，则当鼠标光标移过 div 对象时，会看到背景色从红色逐渐过渡到蓝色，演示效果如图 16.2 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    background-color:red;
    width:400px;
    height:200px;
}
```

Note

```
div:hover {
    background-color:blue;
    /*指定动画过渡的 CSS 属性*/
    -webkit-transition-property:background-color;
    -moz-transition-property:background-color;
    -o-transition-property:background-color;
    /*指定动画过渡的时间*/
    -webkit-transition-duration:2s;
    -moz-transition-duration:2s;
    -o-transition-duration:2s;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

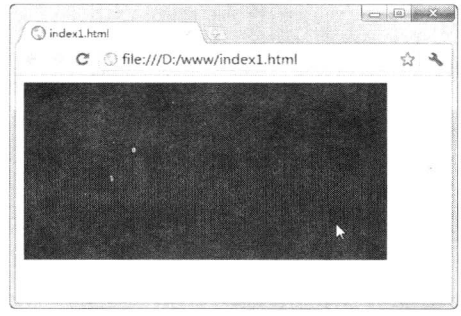


图 16.2　设置动画时间

16.1.3　定义过渡延迟时间

transition-delay 属性用来定义过渡动画的延迟时间。该属性的基本语法如下所示。

```
transition-delay:<time> [, <time>]*;
```

transition-delay 属性初始值为 0，适用于所有元素以及:before 和:after 伪元素。设置时间可以为正整数、负整数和 0，非零的时候必须设置单位为 s（秒）或者 ms（毫秒），为负数的时候，过渡的动作会从该时间点开始显示，之前的动作被截断。为正数的时候，过渡的动作会延迟触发。

【示例】在本示例中，设置过渡动画推迟 2 秒执行，则当鼠标光标移过 div 对象时，不会看到任何变化，过了 2 秒之后，才发现背景色从红色逐渐过渡到蓝色，演示效果如图 16.3 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    background-color:red;
```

```
        width:400px;
        height:200px;
    }
    div:hover {
        background-color:blue;
        /*指定动画过渡的 CSS 属性*/
        -webkit-transition-property:background-color;
        -moz-transition-property:background-color;
        -o-transition-property:background-color;
        /*指定动画过渡的时间*/
        -webkit-transition-duration:2s;
        -moz-transition-duration:2s;
        -o-transition-duration:2s;
        /*指定动画延迟触发*/
        -webkit-transition-delay:2s;
        -moz-transition-delay:2s;
        -o-transition-delay:2s;
    }
    </style>
    </head>
    <body>
    <div></div>
    </body>
    </html>
```

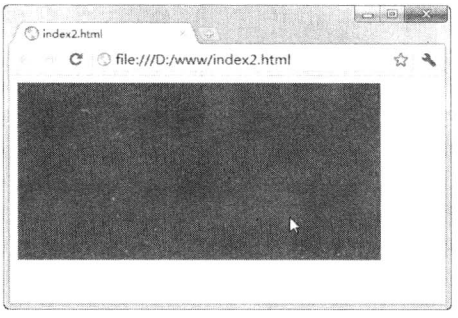


图 16.3　设置延迟动画开始时间

16.1.4　定义过渡效果

transition-timing-function 属性用来定义过渡动画的效果。该属性的基本语法如下所示。

```
transition-timing-function:ease | linear | ease-in | ease-out | ease-in-out | cubicbezier(<number>, <number>, <number>, <number>) [, ease | linear |
    ease-in | ease-out | ease-in-out | cubic-bezier(<number>, <number>,<number>, <number>)]*
```

transition-timing-function 属性初始值为 ease，适用于所有元素以及:before 和:after 伪元素。取值简单说明如下。

- ☑ ease：缓解效果，等同于 cubic-bezier(0.25, 0.1, 0.25, 1.0)函数，即立方贝塞尔。
- ☑ linear：线性效果，等同于 cubic-bezier(0.0, 0.0, 1.0, 1.0)函数。
- ☑ ease-in：渐显效果，等同于 cubic-bezier(0.42, 0, 1.0, 1.0)函数。
- ☑ ease-out：渐隐效果，等同于 cubic-bezier(0, 0, 0.58, 1.0)函数。

☑　ease-in-out：渐显渐隐效果，等同于 cubic-bezier(0.42, 0, 0.58, 1.0)函数。

☑　cubic-bezier：特殊的立方贝塞尔曲线效果。

【示例】在本示例中，设置动画渐变过程更加富有立体感，可以设置过渡效果为线性效果，代码如下，演示效果如图 16.4 所示。

Note

```
<!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></title>
<style type="text/css">
div {
    background-color:red;
    width:400px;
    height:200px;
}
div:hover {
    background-color:blue;
    /*指定动画过渡的 CSS 属性*/
    -webkit-transition-property:background-color;
    -moz-transition-property:background-color;
    -o-transition-property:background-color;
    /*指定动画过渡的时间*/
    -webkit-transition-duration:5s;
    -moz-transition-duration:5s;
    -o-transition-duration:5s;
    /*指定动画过渡为线性效果*/
    -webkit-transition-timing-function: linear;
    -moz-transition-timing-function: linear;
    -o-transition-timing-function: linear;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

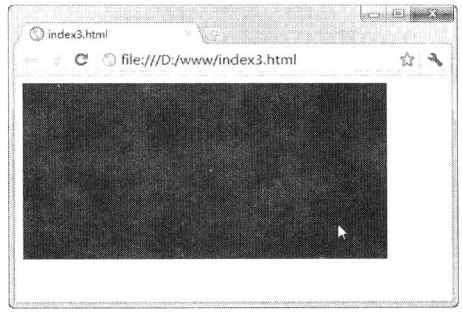


图 16.4　定义线性动画效果

16.1.5 案例实战：设计 Mac OS 导航器

Note

在下面这个案例中将借助过渡动画来模拟 Mac OS 系统桌面底部的导航，它类似于 Windows 桌面上的快捷图标，方便用户快速启动系统内软件，代码如下，演示效果如图 16.5 所示。

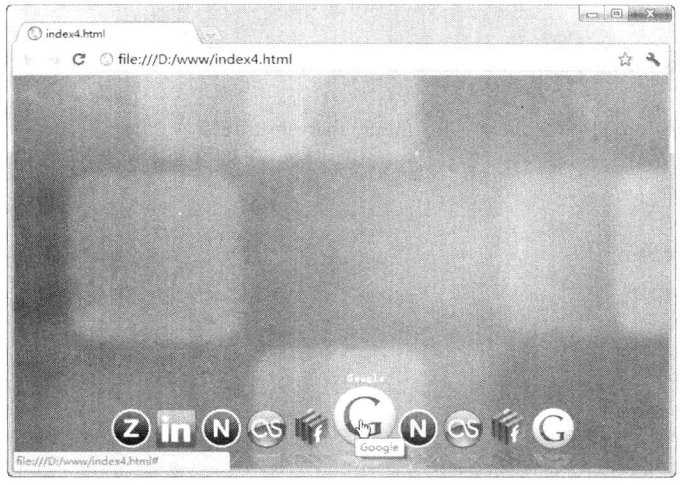


图 16.5 设计 Mac OS 导航器

```
<!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></title>
<style type="text/css">
body {/*设置页眉背景*/
    background: url(images/apple1.jpg)  center no-repeat;
    padding: 0;
    margin:0;
}
ul.osx-dock { /*定义外包含框的样式*/
    display: inline-block;
    height: 130px;
    padding: 0 40px 0 0;
    overflow: hidden;
    margin: 0 0 0 40px;
}
ul.osx-dock li {/*定义每个图标外框样式*/
    display: block;
    position: relative;
    float: left;
    width: 50px;
    height: 50px;
    margin: 60px 0 4px 0;
    /*设计线性过渡，过渡属性为-webkittransform 和 margin 属性*/
    -webkit-transition: 0.15s linear;
```

Note

```
        -webkit-transition-property: -webkit-transform margin;
        text-align: center;
    }
    ul.osx-dock li a {
        display: block;
        height: 50px;
        padding: 0 1px;
        /*设计线性过渡，过渡属性为-webkittransform 和 margin 属性*/

        -webkit-transition: 0.15s linear;
        -webkit-transition-property: -webkit-transform margin;
        margin: 0;
        /*设计倒影效果*/
        -webkit-box-reflect: below 2px -webkit-gradient(linear, left top, left bottom, from(transparent), color-stop(0.45,
transparent), to(rgba(255, 255, 255, 0.25)));
    }
    ul.osx-dock li a img { width: 48px; }
    ul.osx-dock li:hover {
        margin-left: 9px;
        margin-right: 9px;
        z-index: 200;
    }
    ul.osx-dock li:hover a {
        /*设置对象底部中心点为原点向上放大对象 1.5 倍*/
        -webkit-transform-origin: center bottom;
        -webkit-transform: scale(1.5);
    }
    ul.osx-dock li span {
        position: absolute;
        bottom: 80px;
        margin: 0 auto;
        display: none;
        width: auto;
        font-size: 11px;
        font-weight: bold;
        padding: 3px 6px;
        /*设计圆角*/
        -webkit-border-radius: 6px;
        color: #fff;
    }
    ul.osx-dock li:hover span { display: block; }
    div#dockContainer {
        position: fixed;
        bottom: 12px;
        height: 120px;
        padding: 50px 0 0;
        text-align: center;
        /*设计圆角*/
        -webkit-border-radius: 6px;
        -moz-border-radius: 6px;
        width: 100%;
```

```
        line-height: 1;
        z-index: 100;
    }
    div#dockWrapper {/*定义外包含框的样式*/
        width: auto;
        display: inline-block;
        position: relative;
        border-bottom: solid 2px rgba(255, 255, 255, 0.35);
        line-height: 0;
    }
    </style>
    </head>
    <body>
    <div id="dockContainer">
        <div id="dockWrapper">
            <ul class="osx-dock">
                <li class="active"> <span>ZURB</span> <a href="#" title="ZURB"><img src="images/zurb-icon
.png" /> </a> </li>
                <li> <span>LinkedIn</span> <a href="#" title="LinkedIn"><img src="images/linkedin-icon.png" />
</a> </li>
                <li> <span>Notable</span> <a href="#" title="Notable"><img src="images/notable-icon.png" />
</a> </li>
                <li> <span>last.fm</span> <a href="#" title="Last.fm"><img src="images/lastfm-icon.png" /></a>
</li>
                <li> <span>Facebook</span> <a href="#" title="Facebook"><img src="images/facebook-icon.png" />
</a> </li>
                <li> <span>Google</span> <a href="#" title="Google"><img src="images/google-icon.png" /></a>
</li>
                <li> <span>Notable</span> <a href="#" title="Notable"><img src="images/notable-icon.png" />
</a> </li>
                <li> <span>last.fm</span> <a href="#" title="Last.fm"><img src="images/lastfm-icon.png" /></a>
</li>
                <li> <span>Facebook</span> <a href="#" title="Facebook"><img src="images/facebook-icon.png" />
</a> </li>
                <li> <span>Google</span> <a href="#" title="Google"><img src="images/google-icon.png" /></a>
</li>
            </ul>
        </div>
    </div>
    </body>
    </html>
```

16.2 3D 动画

在 CSS3 中，除了可以使用 Transitions 功能实现动画效果之外，还可以使用 Animations 功能实现更为复杂的动画效果。到目前为止，Safari 4+和 Chrome 2+浏览器对该功能提供支持。

W3C 组织在 2009 年 3 月发布了 CSS Animations Module Level 3 草案，详细信息请参阅 http://www.w3.org/TR/css3-animations/，在这个草案中描述了 CSS 动画的基本实现方法和属性。

Note

animation 属性是一个复合属性，它包含了 animation-name、animation-duration、animation-timing-function、animation-delay、animation-iteration-count 和 animationdirection 子属性值。

animation:[<animation-name> || <animation-duration> || <animation-timingfunction>|| <animation-delay> || <animation-iteration-count> ||<animation-direction>] [, [<animation-name> || <animation-duration>|| <animation-timing-function> || <animation-delay> || <animationiteration-count> || <animation-direction>]]*;

animation 属性初始值根据各个子属性的默认值而定，适用于所有块状元素和内联元素。目前，只有少数浏览器支持 CSS 动画。2D CSS animations 可以在 Safari 4（Leopard）、Chrome 3、Safari Mobile 以及其他 Webkit 引擎浏览器上被支持。Safari 4（Snow Leopard）支持 3D 动画。

16.2.1　定义动画名称

Animations 功能与 Transitions 功能相同，都是通过改变元素的属性值来实现动画效果的。二者的区别在于，使用 Transitions 功能时只能通过指定属性的开始值与结束值，然后以在这两个属性值之间进行平滑过渡的方式来实现动画效果，因此不能实现比较复杂的动画效果，而 Animations 则通过定义多个关键帧以及定义每个关键帧中元素的属性值来实现更为复杂的动画效果。

animation-name 属性可以定义 CSS 动画的名称。该属性的基本语法如下所示。

animation-name:none | IDENT [, none | IDENT]*;

animation-name 属性初始值为 none，适用于所有块状元素和内联元素。animation-name 属性定义了一个适用的动画列表。每个名字是用来选择动画关键帧，提供动画的属性值。如果名称不符合任何一个定义的关键帧，则该动画将不执行。此外，如果动画的名称是 none，那么就不会有动画。该属性可用于覆盖任何动画。

16.2.2　定义动画时间

animation-duration 属性定义 CSS 动画的播放时间。该属性的基本语法如下所示。

animation-duration:<time> [, <time>]*;

animation-duration 属性初始值为 0，适用于所有块状元素和内联元素。该属性定义动画循环的时间，在默认情况下该属性值为 0，这意味着动画周期是直接的，即不会有动画。当值为负值时，则被视为 0。

16.2.3　定义动画播放方式

animation-timing-function 属性可以定义 CSS 动画的播放方式。该属性的基本语法如下所示。

animation-timing-function:ease | linear | ease-in | ease-out | ease-in-out | cubicbezier(<number>, <number>, <number>, <number>) [, ease | linear |ease-in | ease-out | ease-in-out | cubic-bezier(<number>, <number>,<number>, <number>)]*

animation-timing-function 属性初始值为 ease，适用于所有块状元素和内联元素，关于这些取值可以参阅 transition-timing-function 属性取值说明。

16.2.4　定义动画延迟时间

animation-delay 属性可以定义 CSS 动画延迟播放的时间，可以是延迟或者提前等。该属性的基本语法如下所示。

animation-delay:<time> [, <time>]*;

animation-delay 属性初始值为 0，适用于所有块状元素和内联元素。该属性定义动画的开始时间，允许一个动画开始执行一段时间后才被应用。当动画延迟时间为 0，即默认动画延迟时间，则意味着动画将尽快执行，否则该值指定将延迟执行的时间。

Note

16.2.5 定义动画播放次数

animation-iteration-count 属性定义 CSS 动画的播放次数。该属性的基本语法如下所示。

animation-iteration-count:infinite | <number> [, infinite | <number>]*;

animation-iteration-count 属性初始值为 1，适用于所有块状元素和内联元素。该属性定义动画的循环播放次数，默认值为 1，这意味着动画将从开始到结束播放一次。infinite 表示无限次，即 CSS 动画永远重复播放。如果取值为非整数，将导致动画结束一个周期的一部分。如果取值为负值，则将导致在交替周期内反向播放动画。

16.2.6 定义动画播放方向

animation-direction 属性定义 CSS 动画的播放方向。该属性的基本语法如下所示。

animation-direction:normal | alternate [, normal | alternate]*;

animation-direction 属性初始值为 normal，适用于所有块状元素和内联元素。该属性定义动画播放的方向，取值包括两个值，默认为 normal。当为默认值时，动画的每次循环都向前播放。另一个值是 alternate，设置该值则表示第偶数次向前播放，第奇数次向反方向播放。

16.2.7 案例实战：设计图片翻转特效

在本节案例中将借助 animation 属性来设计自动翻转的图片效果，该效果模拟在 2D 平面中实现 3D 翻转，演示效果如图 16.6 所示。在这个动画中，图片在 Y 轴上逐渐压缩，然后逐渐倒转图片，在 2D 平面中演示 3D 动画效果。

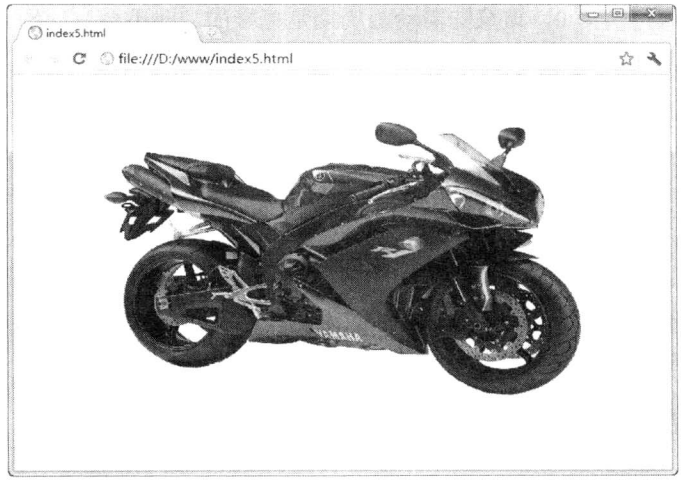


图 16.6 立体翻转动画效果

Note

```
<!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>animation</title>
<style type="text/css">
div {
    margin: 0 auto;
    width: 540px;
    height: 405px;
    background:url(images/moto1.gif) center no-repeat;/*定义 3D 空间*/
    -webkit-transform-style: preserve-3d;
    /*设计沿 X 轴旋转、20 秒线性过渡动画、无限次播放*/
    -webkit-animation-name: x-spin; /*定义动画名称为 x-spin*/
    -webkit-animation-duration: 20s; /*定义动画时间为 20 秒*/
    -webkit-animation-iteration-count: infinite; /*定义动画播放次数为无穷次*/
    -webkit-animation-timing-function: linear; /*定义动画过渡为线性转换*/
}
/*调用动画*/
@-webkit-keyframes x-spin { /*引用 x-spin  动画*/
    0% { /*设置第 1 个关键帧为开始位置*/
        -webkit-transform: rotateX(0deg); /*沿 X 轴开始旋转*/
    }
    50% { /*设置第 2 个关键帧为中间位置*/
        -webkit-transform: rotateX(180deg); /*沿 X 轴旋转 180°  */
    }
    100% { /*设置第 3 个关键帧为结束位置*/
        -webkit-transform: rotateX(360deg); /*沿 X 轴旋转 360°  */
    }
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

16.3　渐 变 效 果

W3C 将渐变定义为 CSS3 标准（参阅 http://dev.w3.org/csswg/css3-images/）。基于 CSS 的渐变与图片渐变相比，最大的优点是便于修改，同时支持无级缩放，过渡更加自然。但是，实现 CSS 渐变的只有基于 Webkit 和 Gecko 引擎的浏览器，基于 Presto 引擎的 Opera 浏览器暂时不支持渐变，基于 Trident 的 IE 虽然可以通过滤镜的方式实现，但并不提倡。

目前，CSS 渐变设计还没有统一的标准，用法差异化很大，不同渲染引擎实现渐变的语法不同，

均为私有属性，这给用户应用带来诸多不便。虽然 W3C 最后提出标准的渐变设计方法，但是还没有获得各主流浏览器的认可。Webkit 和 Gecko 引擎对于 CSS3 属性一般都采取同样的语法，但是对于渐变，目前还无法达成一致，最新推出的 W3C 标准语法也是千差万别，所以只能够分别进行讲解。

Note

16.3.1 设计 Webkit 渐变

Webkit 是第一个支持渐变的浏览器引擎（Safari 4 及其以上版本支持），读者可以访问 http://webkit.org/blog/175/introducing-css-gradients/网页参考更详细的信息。Webkit 引擎支持的渐变方法如下：

```
-webkit-gradient(<type>, <point> [, <radius>]?, <point> [, <radius>]? [,<stop>]*)
```

该函数的参数说明如下。

- ☑ <type>：定义渐变类型，包括线性渐变（linear）和径向渐变（radial）。
- ☑ <point>：定义渐变起始点和结束点坐标，即开始应用渐变的 X 轴和 Y 轴坐标，以及结束渐变的坐标。该参数支持数值、百分比和关键字，如（0 0）或者（left top）等。关键字包括 top、bottom、left 和 right。
- ☑ <radius>：当定义径向渐变时，用来设置径向渐变的长度，该参数为一个数值。
- ☑ <stop>：定义渐变色和步长。包括 3 个类型值，即开始的颜色，使用 from(colorvalue)函数定义；结束的颜色，使用 to(colorvalue)函数定义；颜色步长，使用 colorstop(value, color value)定义。color-stop()函数包含两个参数值，第一个参数值为一个数值或者百分比值，取值范围为 0~1.0（或者 0%~100%），第二个参数值表示任意颜色值。

【示例 1】本示例演示 Webkit 引擎的直线渐变实现方法，演示效果如图 16.7 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -webkit-gradient(linear, left top, left bottom, from(blue), to(red));
    -webkit-background-origin: padding-box;
    -webkit-background-clip: content-box;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

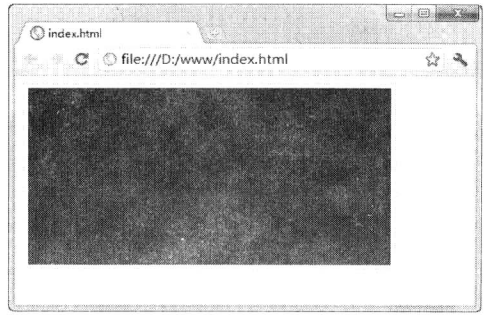


图 16.7　直线渐变效果

示例 1 演示了简单的线性渐变背景色，从顶部到底部，从蓝色向红色渐变显示。当然，也可以增加更多的中间渐变色。例如，下面代码是在上面示例基础上增加一种渐变色，从顶部到中间，再从中间到底部，从蓝色到绿色，再到红色渐变显示，演示效果如图 16.8 所示。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -webkit-gradient(linear, left top, left bottom, from(blue), to(red), color-stop(50%, green));
    -webkit-background-origin: padding-box;
    -webkit-background-clip: content-box;
}
</style>
```

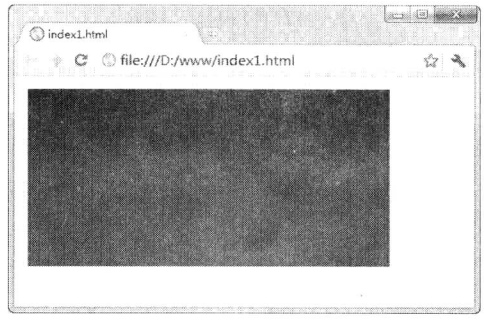


图 16.8　直线渐变效果

设计二重渐变，从顶部到底部，先是从蓝色到白色渐变显示，再从黑色到红色渐变显示，代码如下，演示效果如图 16.9 所示。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -webkit-gradient(linear, left top, left bottom, from(blue), to(red),color-stop(0.5, #fff),
color-stop(0.5, #000));
    -webkit-background-origin: padding-box;
```

```
    -webkit-background-clip: content-box;
}
</style>
```

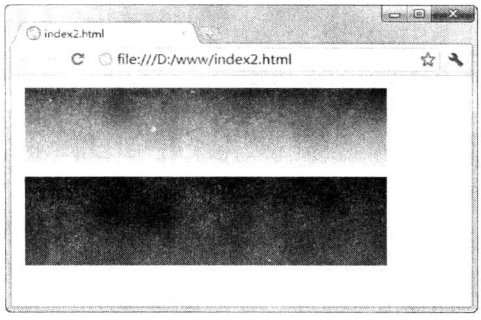


图 16.9　设计二重渐变效果

通过设置不同的步长值设计多重渐变效果，从顶部到底部，先是从蓝色到白色渐变，再从白色到黑色渐变，最后是黑色到红色渐变显示，则声明的代码如下。

```
background: -webkit-gradient(linear,
    left top, left bottom,
    from(blue), to(red),
    color-stop(0.4, #fff),
    color-stop(0.6, #000));
```

注意：color-stop()函数包含两个参数值，第一个参数值指定色标位置，第二个参数指定色标颜色。一个渐变可以包含多个色标，位置值为 0~1 之间的小数，或者 0%~100%之间的百分数，指定色标的位置比例，其用法与 Photoshop 中的直线渐变工具用法相似。

【示例 2】设计径向渐变。相对于直线渐变，径向渐变的用法稍显复杂。本示例设计一个同心圆，圆心坐标为（200,100），内圆半径为 10，外圆半径为 100，内圆小于外圆半径，从内圆红色到外圆绿色径向渐变，超出外圆半径显示为绿色，内圆显示红色，演示效果如图 16.10 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -webkit-gradient(radial,  200 100, 10, 200 100, 100, from(red), to(green));
    -webkit-background-origin: padding-box;
    -webkit-background-clip: content-box;
}
</style>
</head>
```

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

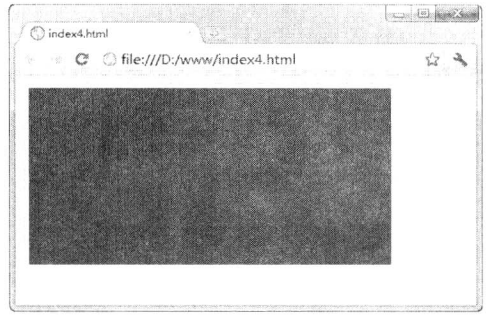


图 16.10　设计同心圆效果

设计同心圆时，在内圆到外圆中间 90%的位置，即距离外环内添加一个蓝色色标，设计多层径向渐变，还可以添加更多的色标，以设计多层径向渐变效果。

```
background: -webkit-gradient(radial,
    200 100, 10,
    200 100, 100,
    from(red), to(green)),
    color-stop(90%, blue);
```

也可以设计非同心圆。当内圆圆心和外圆圆心距离小于两圆半径的差，则显示如图 16.11 所示的效果，呈现锥形径向渐变效果。锥形尖锐性与两圆圆心距离成正比。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -webkit-gradient(radial,   120 100, 10, 200 100, 100, from(red), to(green));
    -webkit-background-origin: padding-box;
    -webkit-background-clip: content-box;
}
</style>
```

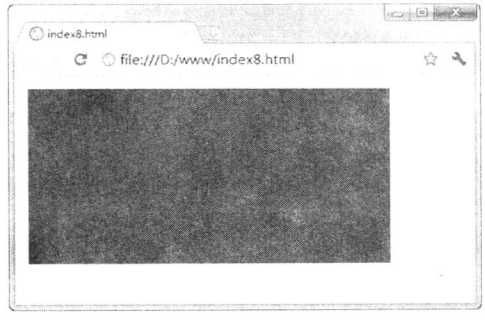


图 16.11　设计非同心圆效果

提示：径向渐变的起点坐标和终点坐标分别定义内圆和外圆坐标。在定义坐标的同时，还应该指定内圆和外圆的半径，当然也可以为径向渐变设置多个步长，用来添加多色径向渐变效果。

Note

【示例 3】通过设置 to()函数的颜色值为透明，同时设计相似色，则可以设计球形效果，演示效果如图 16.12 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -webkit-gradient(radial, 180 80, 10, 200 100, 90, from(#00C), to(rgba(1,159,98,0)), color-stop(98%, #0CF));
    -webkit-background-origin: padding-box;
    -webkit-background-clip: content-box;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

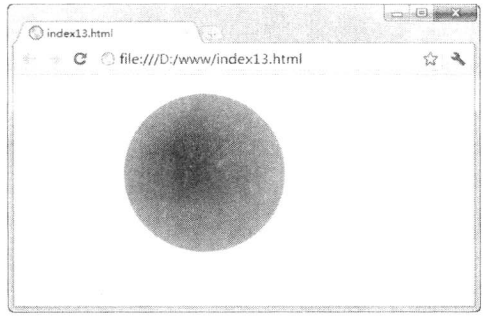


图 16.12　设计球形效果

如果为背景图定义多个径向渐变，则可以设计多个气泡效果，代码如下所示，演示效果如图 16.13 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
```

Note

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background:
    -webkit-gradient(radial, 60 45, 10, 52 50, 60, from(#A7D30C), to(rgba(1,159,98,0)), color-stop(90%,
#019F62)),
    -webkit-gradient(radial, 105 105, 20, 112 120, 80, from(#ff5f98), to(rgba(255,1,136,0)), color-stop(75%,
#ff0188)),
    -webkit-gradient(radial, 95 15, 15, 102 20, 60, from(#00c9ff), to(rgba(0,201,255,0)), color-stop(80%,
#00b5e2)),
    -webkit-gradient(radial, 300 110, 30, 300 100, 100, from(#f4f201), to(rgba(228, 199,0,0)), color-stop(80%,
#e4c700));
    -webkit-background-origin: padding-box;
    -webkit-background-clip: content-box;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

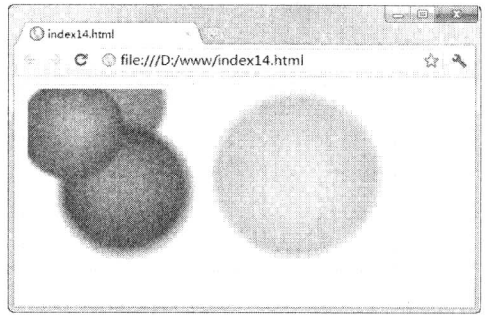


图 16.13　设计多个气泡效果

16.3.2　Webkit 案例实战

除了定义渐变背景外，还可以定义渐变边框、填充内容，以及设计图标等。下面分别结合示例进行说明。

☑　定义渐变效果的边框

【示例 1】本示例通过渐变特性，为-webkit-border-image 属性定义渐变边框图效果，演示效果如图 16.14 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Note

```
<title></title>
<style type="text/css">
div {
    border-width: 20px;
    width: 400px;
    height: 200px;
    margin: 20px;
    -webkit-border-image: -webkit-gradient(linear, left top, left bottom, from(#00abeb), to(#fff), color-stop(0.5,
#fff), color-stop(0.5, #66cc00)) 20;
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

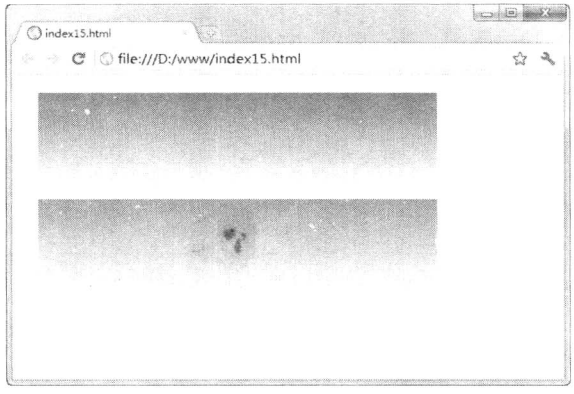


图 16.14　显示渐变效果的边框

☑　定义填充内容效果

【示例 2】本示例通过 content 属性，为 标签嵌入一个通过渐变设计的圆球，同时为这个包含框设计一个渐变背景，从而产生一种透视框的效果，如图 16.15 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
.div1 {
    width:400px;
    height:200px;
    border:10px solid #A7D30C;
    /*为 div 元素定义渐变背景*/
    background: -webkit-gradient(linear, left top, left bottom, from(#00abeb), to(#fff), color-stop(0.5, #fff),
color-stop(0.5, #66cc00));
    float:left;
```

Note

```
    }
    .div1::before {
        width:400px;
        height:200px;
        border:10px solid #019F62;
        /*在 div 元素前面插入一个内容对象，在该对象中绘制一个球体，并定义显示边框效果*/
        content: -webkit-gradient(radial, 200 100, 10, 200 100, 100, from(#A7D30C), to(rgba(1, 159, 98, 0)),
color-stop(90%, #019F62));
        display: block;
    }
    </style>
    </head>
    <body>
    <div class="div1"></div>
    </body>
    </html>
```

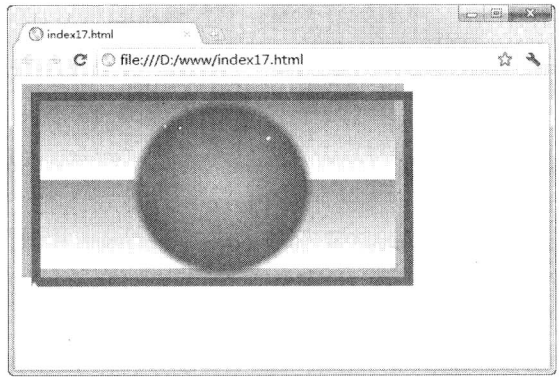


图 16.15　插入球形内容填充物并显示边框效果

☑　定义列表图标

【示例 3】本示例通过 list-style-image 属性，为 ul 元素定义自定义图标，该图标通过渐变特效进行绘制，从而产生一种精致的立体效果。

```
    <!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></title>
    <style type="text/css">
    ul {
        list-style-image: -webkit-gradient(radial, center center, 4, center center, 8, from(#ff0000), to(rgba(0, 0, 0, 0)),
color-stop(50%, #dd0000))
    }
    </style>
    </head>
    <body>
    <ul>
        <li>加入百度推广</li>
```

```
    <li>搜索风云榜</li>
    <li>关于百度</li>
    <li>About Baidu</li>
</ul>
</body>
</html>
```

Note

16.3.3 设计 Gecko 渐变

Firefox 浏览器从 3.6 版本开始支持渐变设计，详细信息请参阅 http://hacks.mozilla.org/2009/11/css-gradients-firefox-36/页面说明。但是 Gecko 引擎与 Webkit 引擎的用法不同。Gecko 引擎定义了两个私有函数，分别用来设计直线渐变和径向渐变。基本语法说明如下。

```
-moz-linear-gradient( [<point> || <angle>,]? <stop>, <stop> [, <stop>]* )
```

该函数的参数说明如下。

- ☑ <point>：定义渐变起始点，取值包含数值、百分比，也可以使用关键字，其中 left、center 和 right 关键字定义 X 轴坐标，top、center 和 bottom 关键字定义 Y 轴坐标。用法与 background-position 和-moz-transform-origin 属性中定位方式相同。当指定一个值时，则另一个值默认为 center。
- ☑ <angle>：定义直线渐变的角度。单位包括 deg（度，一圈等于 360deg）、grad（梯度、90°等于 100grad）、rad（弧度，一圈等于 2*PI rad）。
- ☑ <stop>：定义步长，用法与 Webkit 引擎的 color-stop()函数相似，但是该参数不需要调用函数，直接传递参数即可。其中第一个参数值设置颜色值，可以为任何合法的颜色值，第二个参数设置颜色的位置，取值为百分比（0%~100%）或者数值，也可以省略步长位置。-moz-linear-gradient() 函数用法的详细说明请参阅 https://developer.mozilla.org/en/CSS/-moz-linear-gradient 网页信息。

【示例 1】最简单的线性渐变，只需要指定开始颜色和结束颜色，则默认为从上到下实施线性渐变。本示例演示 Gecko 引擎的直线渐变实现方法，代码如下，演示效果如图 16.16 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -moz-linear-gradient(red, blue);
}
</style>
</head>
<body>
<div></div>
```

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

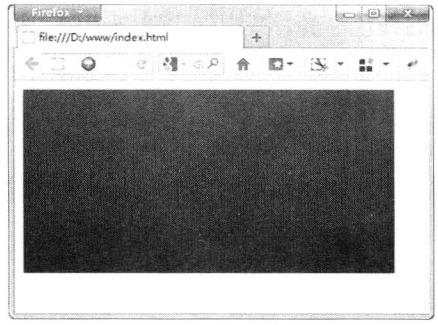


图 16.16　简单的线性渐变

如果是从左上角到右下角的线性渐变，其中 top 关键字设置起点 X 轴坐标，left 关键字设置起点 Y 轴坐标，则设计的样式代码如下。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background:
    -moz-linear-gradient(top left,red, blue);
}
</style>
```

如果设计从左到右的五彩渐变，其中 y 轴坐标默认为 center，多个色标按步长平均显示，则设计的样式代码如下，演示效果如图 16.17 所示。

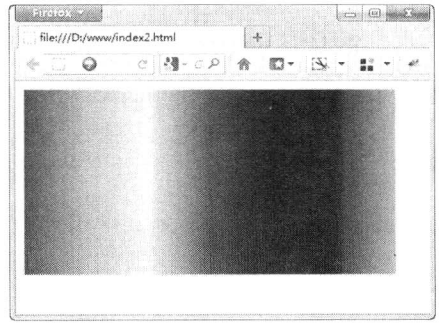


图 16.17　设计五彩渐变

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -moz-linear-gradient(left, red, orange, yellow, green, blue, indigo, violet);
```

```
}
</style>
```

如果设计从左上角到右下角的红色渐变，其中红色逐渐递弱，并最终显示为透明，则可以编写如下样式。

Note

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -moz-linear-gradient(top left, red, rgba(255,0,0,0));
}
</style>
```

当指定角度时，它是沿水平线按逆时针旋转定位的。因此，设置为 0°，将产生从左向右的水平渐变，而设置 90°，将创建一个从底部到顶部的渐变。

【示例 2】设计渐变半透明效果的背景图像。通过在背景图像上面覆盖一层从左到右的白色到透明的渐变填充层，从而实现渐显或者渐隐背景图像效果，如图 16.18 所示。

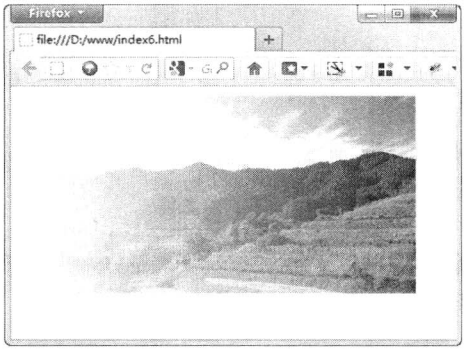


图 16.18　设计渐变背景图

```
<!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></title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -moz-linear-gradient(right, rgba(255,255,255,0), rgba(255,255,255,1)), url(images/bg7.jpg);
}
</style>
</head>
```

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

Note

Gecko 引擎定义的径向渐变基本语法说明如下。

```
-moz-radial-gradient( [<position> || <angle>,]? [<shape> || <size>,]? <stop>,<stop>[, <stop>]* )
```

该函数的参数说明如下。

- ☑ <position>：定义渐变起始点，取值包含数值、百分比，也可以使用关键字，其中 left、center 和 right 关键字定义 X 轴坐标，top、center 和 bottom 关键字定义 Y 轴坐标。用法与 background-position 和-moz-transform-origin 属性中定位方式相同。当指定一个值时，则另一个值默认为 center。
- ☑ <angle>：定义渐变的角度。单位包括 deg（度，一圈等于 360deg）、grad（梯度、90deg 等于 100grad）、rad（弧度，一圈等于 2*PI rad），默认值为 0deg。
- ☑ <shape>：定义径向渐变的形状，包括 circle（圆）和 ellipse（椭圆），默认值为 ellipse。
- ☑ <size>：定义圆半径，或者椭圆的轴长度。
- ☑ <stop>：定义步长，用法与 Webkit 引擎的 color-stop()函数相似，但是该参数不需要调用函数，直接传递参数即可。其中第一个参数值设置颜色值，可以为任何合法的颜色值，第二个参数设置颜色的位置，取值为百分比（0%~100%）或者数值，也可以省略步长位置。

-moz-radial-gradient()函数用法详细信息请参阅 https://developer.mozilla.org/en/CSS/-moz-radial-gradient 网页信息。

【示例 3】设计简单的径向渐变，从中间向外由红色、黄色到蓝色渐变显示，代码如下，演示效果如图 16.19 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" " http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -moz-radial-gradient(red, yellow, blue);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

Note

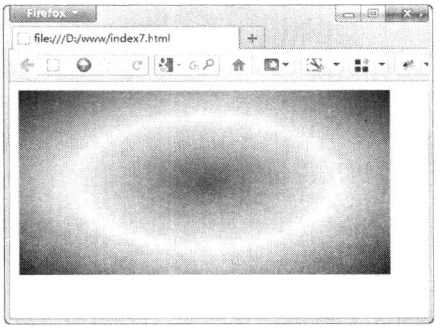


图 16.19　设计径向渐变

如果设计从中间向外由红色、黄色到蓝色渐变显示，并设置不同色标的显示位置，则样式代码如下。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background:
    -moz-radial-gradient(red 20%,yellow 30%, blue 40%);
}
</style>
```

设计径向渐变，从左下角向外由红色、黄色到蓝色渐变显示，则样式代码如下。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background:-moz-radial-gradient(bottom left,red, yellow, blue 80%);
}
</style>
```

如果设计径向渐变形状为圆形，从左侧中间向外由红色、黄色到蓝色渐变显示，则样式代码如下。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background:-moz-radial-gradient(left,circle, red, yellow, blue 50%);
}
</style>
```

提示：size 参数包含多个关键字：closestside、closest-corner、farthest-side、farthest-corner、contain 和 cover。使用这些关键字可以定义径向渐变的大小。

另外，Gecko 引擎还定义了-moz-repeating-linear-gradient 和-moz-repeating-radial-gradient 两个属性，用来定义重复直线渐变和重复径向渐变。

【示例 4】本示例分别调用上述两个属性，定义重复径向渐变和重复直线渐变，演示效果如图 16.20 和图 16.21 所示。

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -moz-repeating-radial-gradient(circle, black, black 10px, white 10px, white 20px);
}
</style>
```

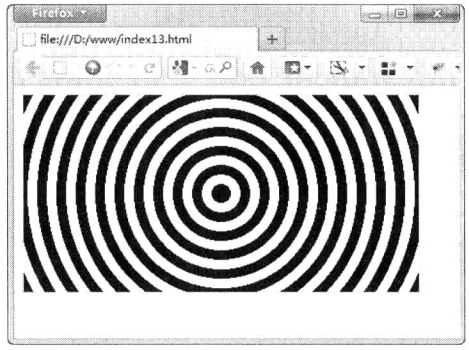


图 16.20　设计重复径向渐变

```
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: -moz-repeating-linear-gradient(top left 60deg,black, black 10px, white 10px, white 20px);
}
</style>
```

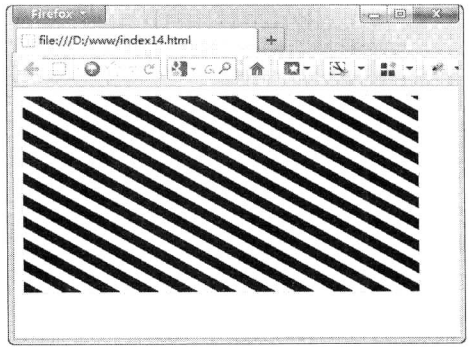


图 16.21　设计重复直线渐变

16.3.4 Gecko 案例实战

本节示例将综合应用 box-shadow、text-shadow 和 border-radius 等属性，定义一个包含阴影、圆角特效，同时利用 CSS 渐变、半透明特效设计精致的栏目效果，预览效果如图 16.22 所示。

Note

```
<!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></title>
<style type="text/css">
/*页面初始化*/
body {
    background-color: #454545;
    margin:1em;
    padding:0;
}
/*设计包含框样式*/
.box {
    border-radius: 10px; /*设计圆角*/
    box-shadow: 0 0 12px 1px rgba(205, 205, 205, 1); /*设计栏目阴影*/
    border: 1px solid black;
    padding: 10px;      max-width: 600px;
    margin: auto;
    text-shadow: black 1px 2px 2px;   /*设计包含文本阴影*/
    color: white;
    /*设计直线渐变背景*/
    background-image: -moz-linear-gradient(bottom, black, rgba(0, 47, 94, 0.2), white);
    background-color: rgba(43, 43, 43, 0.5);
}
/*设计光标经过时，放大阴影亮度*/
.box:hover { box-shadow: 0 0 12px 5px rgba(205, 205, 205, 1);}
h2 {
    font-size: 120%;
    font-weight:bold;
    text-decoration:underline;
}
/*在标题前面添加额外内容*/
h2:before { content: "观点：";}
p {
    padding: 6px;
    text-indent:2em;
    line-height:1.8em;
    font-size:14px;
}
</style>
</head>
<body>
<div   class="box">
```

```
    <h2>公关的弱智和弱智的公关</h2>
    <p>自媒体时代，公关发生了很大的变化，因为之前大众看到的形象是通过媒体传播出来的形象，而现在，大众可以直接看到企业领导人的言行，于是我小时候最爱的一句话又得以重现，“明星不接受采访看不出弱智”。</p>
    <p>……</p>
</div>
</body>
</html>
```

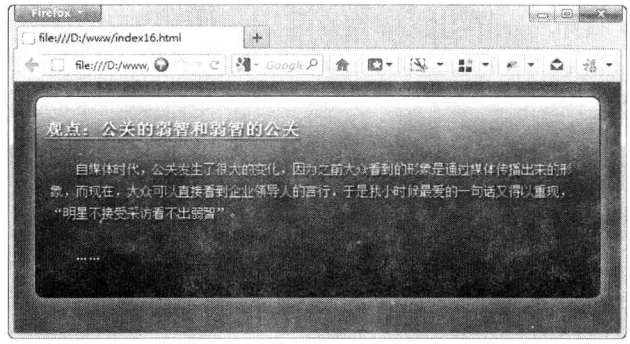


图 16.22　设计栏目板块

16.3.5　设计 IE 渐变

IE 不支持 CSS 渐变，但提供了渐变滤镜，可以实现简单的渐变效果。IE 浏览器渐变滤镜的基本语法说明如下。

```
filter:progid:DXImageTransform.Microsoft.Gradient(enabled=bEnabled,startColorStr=iWidth,endColorStr=iWidth)
```

该函数的参数说明如下。

- ☑ enabled：设置或检索滤镜是否激活。可选布尔值，包括 true 和 false，默认值为 true，激活状态。
- ☑ startColorStr：设置或检索色彩渐变的开始颜色和透明度。可选项，其格式为#AARRGGBB。AA、RR、GG、BB 为十六进制正整数，取值范围为 00~FF。RR 指定红色值，GG 指定绿色值，BB 指定蓝色值。AA 指定透明度，00 是完全透明，FF 是完全不透明。超出取值范围的值将被恢复为默认值。取值范围为#FF000000~#FFFFFFFF，默认值为#FF0000FF，即不透明蓝色。
- ☑ endColorStr：设置或检索色彩渐变的结束颜色和透明度。默认值为#FF000000，即不透明黑色。

IE 渐变滤镜的用法详细信息请参阅 http://msdn.microsoftcom/en-us/library/ms532997（VS.85,loband）.aspx 网页信息。注意，IE 渐变滤镜在 IE 5.5 及其以上版本浏览器中有效。

【示例】在本示例中，利用 IE 滤镜设计渐变背景，然后通过背景图像设计图文插画效果，预览如图 16.23 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
```

Note

```
<title></title>
<style type="text/css">
body {
    /*让渐变背景填满整个页面*/
    padding:1em;
    margin:0;
    /*为网页设计垂直渐变背景*/
    filter: progid:DXImageTransform.Microsoft.Gradient(gradientType=0, startColorStr=#9999FF,
endColorStr=#ffffff);
}
/*定义标题样式*/
h1 {
    color:white;
    font-size:18px;
    height:45px;
    padding-left:3em;
    line-height:65px; /*控制文本显示位置*/
    border-bottom:solid 2px #c72223;
    /*为标题插入一个装饰图标*/
    background:url(images/icon4.png) no-repeat left center;
}
.box {
    height:300px;
    background:url(images/bg6.png) no-repeat right bottom; /*设计插画背景*/
}
p {text-indent:2em;}
</style>
</head>
<body>
<div  class="box">
    <h1>公关的弱智和弱智的公关</h1>
    <p>自媒体时代，公关发生了很大的变化，因为之前大众看到的形象是通过媒体传播出来的形象，而现在，大众可以直接看到企业领导人的言行，于是我小时候最爱的一句话又得以重现，“明星不接受采访看不出弱智”。</p>
    <p>……</p>
</div>
</body>
</html>
```

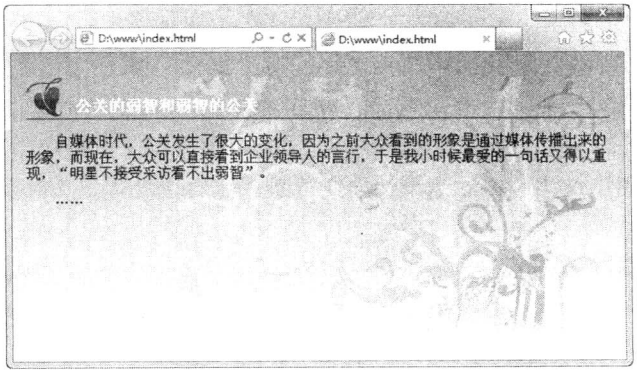


图 16.23　使用 IE 滤镜设计渐变背景

16.3.6　设计 W3C 渐变

W3C 于 2010 年 11 月份才正式发布支持渐变设计的工作草案，不过渐变设计并没有被单独列为一个模块，而是作为图像值和图像被替换内容模块的一部分，详细资料请参阅 http://dev.w3.org/csswg/css3-images/#gradients。标准草案沿袭 Gecko 引擎的渐变设计方法，语法和用法也基本相同，简单比较如下所示。

Note

☑　线性渐变

```
linear-gradient([[ [top | bottom] || [left| right] ]||<angle>,]?<color-stop>[, <color-stop>]+);
```

☑　径向渐变

```
radial-gradient([<bg-position> || <angle>,]?[[<shape> || <size>]|[<length>|<percentage>]{2},]?<color-stop>[,
<color-stop>]+);
```

【示例】本示例演示了如何使用标准方法设计一个直线渐变，从左上角开始显示从黄色到蓝色的过渡效果。

```
<!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></title>
<style type="text/css">
div {
    width:400px;
    height:200px;
    border:2px solid #FCF;
    padding: 4px;
    background: linear-gradient(top left, yellow, blue);
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

W3C 标准用法与 Gecko 引擎渐变用法基本相同，本节不再详解，请读者参阅前面章节对照学习，同时也可以访问 http://dev.w3.org/csswg/css3-images/#gradients 页面阅读标准渐变的基本用法。

16.4　案例综合实战

本节将通过几个综合案例帮助读者快速掌握 CSS3 动画设计技能。

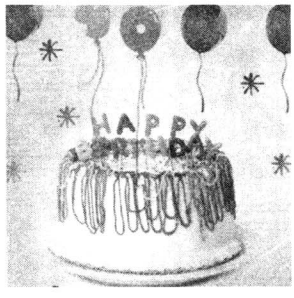
16.4.1 设计礼品盒

Note

本案例没有使用 JavaScript、图像或 SVG 等技术，仅利用第 15 章和本章介绍的各种 2D 转换属性进行设计，即采用变换属性中的倾斜，然后旋转定义边框的长方形，通过把 3 个旋转立方体面结合起来，形成一个三维对象。能够兼容 WebKit 和 Gecko 的浏览器，其中 Firefox 3.5+、Safari 3.2+，以及 Chrome 3.0+都可以进行很好的支持。演示效果如图 16.24 所示，其中礼品盒三面所用到的图像素材如图 16.25 所示。同时利用 3D 动画设计属性，设计当鼠标指针经过礼品盒时，产生自行移动效果。

图 16.24　设计礼品盒效果

top.jpg



left.jpg

right.jpg

图 16.25　礼品盒素材图

完整代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
.cube {/*固定定位盒子包含框*/
    position: fixed;
    left: 50%;
    top: 12px;
```

Note

```
}
.cube.one {
    top: 200px;
    left: 50%;
    margin-left: -200px;
}
.rightFace, .leftFace, .topFace div {/*统一盒子三立面的尺寸*/
    padding: 0px;
    width: 180px;
    height: 180px;
    border:solid 1px #ddd; /*定义边框线*/
}
.cube   img {/*定义礼品盒三面的贴图大小*/
    width: 180px;
    height: 180px;
}
/*绝对定位盒子三立面*/
.rightFace, .leftFace, .topFace { position: absolute; }
/*统一盒子三立面的背景色*/
.cube:hover .rightFace, .cube:hover .leftFace, .cube:hover .topFace div { background-color: #ffc; }
/*统一鼠标经过时，盒子三立面的背景色*/
.cube:hover .rightFace:hover, .cube:hover .leftFace:hover, .cube:hover .topFace:hover div { background-color: #ffa; }
/*变形左侧面*/
.leftFace {
    /*左侧面 Y 轴倾斜 30° */
    -webkit-transform: skew(0deg, 30deg);
    -moz-transform: skew(0deg, 30deg);
    background-color: #ccc;
}
/*变形正侧面*/
.rightFace {
    /*右侧面 Y 轴倾斜 30° ，并翻转*/
    -webkit-transform: skew(0deg, -30deg);
    -moz-transform: skew(0deg, -30deg);
    background-color: #ddd;
    left: 180px;
}
/*变形顶侧面*/
.topFace div {
    /*顶侧面 Y 轴倾斜 30° ，并翻转，同时放大显示*/
    -webkit-transform: skew(0deg, -30deg) scale(1, 1.16);
    -moz-transform: skew(0deg, -30deg) scale(1, 1.16);
    background-color: #eee;
    font-size: 0.862em;
}
/*旋转顶侧面*/
.topFace {
    /*顶侧面逆时针旋转 60° */
    -webkit-transform: rotate(60deg);
    -moz-transform: rotate(60deg);
    top: -142px;
    left: 89px;
```

```
}
/*仅兼容 Webkit 类型浏览器的移动动画*/
.cube { -webkit-transition: -webkit-transform 1s linear; }
.cube:hover { -webkit-transform: translate(202px, 115px); }
</style>
</head>
<body>
<div class="cube one">
    <div class="topFace">
        <div>
            <img src="images/top.jpg"  />
        </div>
    </div>
    <div class="leftFace">
        <img src="images/left.jpg"  />
    </div>
    <div class="rightFace">
        <img src="images/right.jpg"  />
    </div>
</div>
</body>
</html>
```

16.4.2 设计折叠面板

折叠面板是常用小部件，由于它比较实用，所以在网页上会经常看到设计精巧的折叠面板。尽管样式各异，但它们的设计思路和实现方法都基本相似，即利用 JavaScript 脚本动态控制每个选项卡盒子伸缩，实现动态显示和隐藏，从而实现鼠标操作的折叠面板效果。

本案例使用 CSS3 的目标伪类（:target）设计这种效果，没有使用 JavaScript 脚本，使用 Webkit 专有过渡属性设计的滑动效果，折叠动画效果如图 16.26 所示。

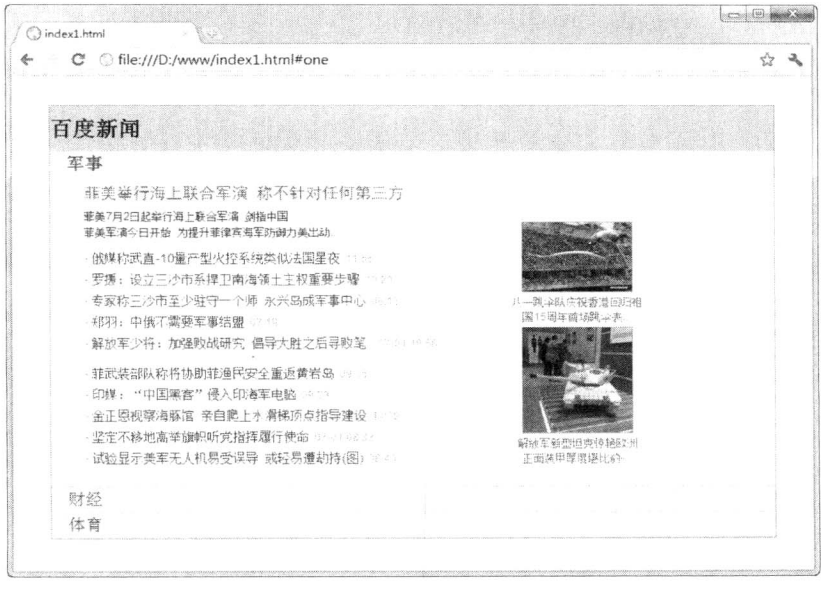


图 16.26 设计折叠面板

Note

完整代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
/*定义折叠框外框样式*/
.accordion {
    background: #eee;
    border: 1px solid #999;
    margin: 2em;
}
/*定义折叠框标题栏样式*/
.accordion h2 {
    margin: 0;
    padding: 12px 0;
    background:#CCC
}
/*定义折叠框内容框样式*/
.accordion .section {
    border-bottom: 1px solid #ccc;
    background: #fff;
}
/*定义折叠框选项标题栏样式*/
.accordion h3 {
    margin:0;
    padding:0;
    background: #eee;
    padding:3px 1em;
}
/*定义折叠框选项标题栏超链接样式*/
.accordion h3 a {
    font-weight: normal;
    text-decoration:none;
}
/*当获得目标焦点时，粗体显示选项标题栏文字*/
.accordion :target h3 a { font-weight: bold; }
/*选项栏标题对应的选项子框样式*/
.accordion h3 + div {
    height: 0;
    padding:0 1em;
    overflow: hidden;
    /*定义过渡对象为高度，过渡时间为 0.3 秒，渐显显示*/
    -webkit-transition: height 0.3s ease-in;
}
.accordion h3 + div img { margin:4px; }
/*当获得目标焦点时，子选项内容框样式*/
.accordion :target h3 + div {
    /*当获取目标之后，高度为 350 像素*/
    height:350px;
```

```
        overflow:auto;
    }
    </style>
    </head>
    <body>
    <div class="accordion">
        <h2>百度新闻</h2>
        <div id="one" class="section">
            <h3> <a href="#one">军事</a> </h3>
            <div><img src="images/1.jpg" /></div>
        </div>
        <div id="two" class="section">
            <h3> <a href="#two">财经</a> </h3>
            <div><img src="images/2.jpg" /></div>
        </div>
        <div id="three" class="section">
            <h3> <a href="#three">体育</a> </h3>
            <div><img src="images/3.jpg" /></div>
        </div>
    </div>
    </body>
    </html>
```

16.4.3 设计易拉罐

本案例使用多标签模拟像素来设计易拉罐立体视图效果，如图 16.27 所示，移动滚动条可以选择易拉罐。在设计过程中灵活使用 background-image、background-attachment 和 background-position 属性，设计二维位移的立体效果，同时当滚动窗口时，位移图将适用于不同部位的纹理（背景图像），从而模拟出旋转的易拉罐效果。利用一幅全景图片作为背景，如图 16.28 所示，通过 55 个 p 元素，然后进行拼接，从而设计出立体效果。定义过渡对象为高度，过渡时间为 0.3 秒，渐显显示。当获取目标之后，高度为 300 像素。虽然使用 CSS3 可以设计出三维立体效果，但本例所使用的方法设计出的三维立体效果比较简单，只要读者掌握光线的明暗，以及透视原理，在二维平面中可以使用任何方法呈现三维效果。

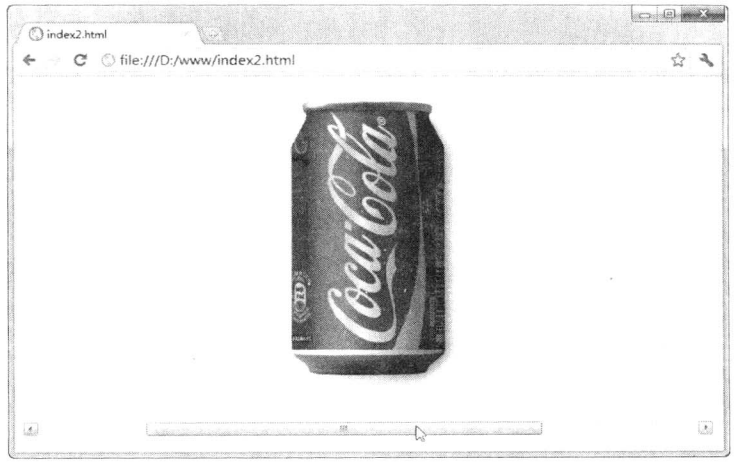


图 16.27　设计易拉罐

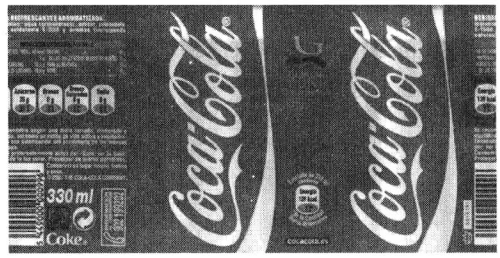


图 16.28　易拉罐全景图

完整代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
/*易拉罐外框样式*/
#coke {
    width: 760px;
    height: 400px;
    overflow: auto;
}
/*把易拉罐内胆嵌在底部*/
img {
    border: 0;
    margin-left: -172px;
}
a {/*把超链接设置为块显示*/
    display: block;
    padding-top: 19px;
    width: 194px;
}
a:hover img {/*鼠标经过后显示图标*/
    background-image: url(images/coke-title.png);
    background-repeat: no-repeat;
    background-position: 15px 100px;
}
/*加一个内框*/
div div {
    padding-left: 500px;
    width: 760px;
}
/*定义每个单元的样式*/
p {
    margin: 0;
    padding: 0;
    float: left;
    height: 336px;
```

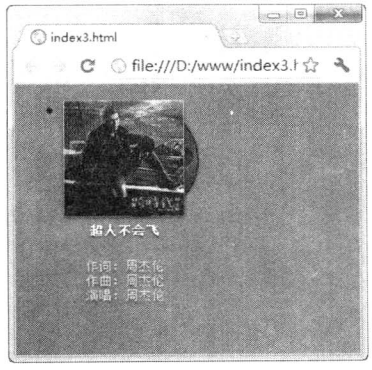
```
    width: 1px;
    background-image: url(images/coke-label.jpg); /*添加背景图像*/
    background-attachment: fixed; /*固定背景显示*/
    background-repeat: repeat-x; /*水平平铺*/
}
/*以下样式列表为设计每列背景图像，通过多条组合拼装出 3D 效果的易拉罐*/
#x1 { background-position: 5px 30px; }
#x2 { background-position: 0px 30px; }
#x3 { background-position: -3px 30px; }
#x4 { background-position: -6px 30px; }
#x5 { background-position: -8px 30px; }
#x6 { background-position: -10px 30px; }
#x7 { background-position: -12px 30px; }
#x8 { background-position: -14px 30px; }
#x9 { background-position: -15px 30px; }
#x10 { background-position: -16px 30px; }
#x11 { background-position: -17px 30px; }
#x12 { background-position: -18px 30px; }
#x13 { background-position: -19px 30px; }
#x14 { background-position: -20px 30px; }
#x15 { background-position: -21px 30px; }
#x16 { background-position: -22px 30px; width: 2px; }
#x17 { background-position: -23px 30px; }
#x18 { background-position: -24px 30px; width: 2px; }
#x19 { background-position: -25px 30px; width: 2px; }
#x20 { background-position: -26px 30px; width: 2px; }
#x21 { background-position: -27px 30px; width: 2px; }
#x22 { background-position: -28px 30px; width: 3px; }
#x23 { background-position: -29px 30px; width: 3px; }
#x24 { background-position: -30px 30px; width: 4px; }
#x25 { background-position: -31px 30px; width: 5px; }
#x26 { background-position: -32px 30px; width: 7px; }
#x27 { background-position: -33px 30px; width: 12px; }
#x28 { background-position: -34px 30px; width: 55px; }
#x29 { background-position: -35px 30px; width: 11px; }
#x30 { background-position: -36px 30px; width: 6px; }
#x31 { background-position: -37px 30px; width: 5px; }
#x32 { background-position: -38px 30px; width: 4px; }
#x33 { background-position: -39px 30px; width: 3px; }
#x34 { background-position: -40px 30px; width: 2px; }
#x35 { background-position: -41px 30px; width: 3px; }
#x36 { background-position: -42px 30px; width: 2px; }
#x37 { background-position: -43px 30px; width: 2px; }
#x38 { background-position: -44px 30px; }
#x39 { background-position: -45px 30px; width: 2px; }
#x40 { background-position: -46px 30px; }
#x41 { background-position: -47px 30px; }
#x42 { background-position: -48px 30px; }
#x43 { background-position: -49px 30px; }
#x44 { background-position: -50px 30px; }
```

Note

```
#x45 { background-position: -51px 30px; }
#x46 { background-position: -52px 30px; }
#x47 { background-position: -53px 30px; }
#x48 { background-position: -54px 30px; }
#x49 { background-position: -56px 30px; }
#x50 { background-position: -58px 30px; }
#x51 { background-position: -60px 30px; }
#x52 { background-position: -62px 30px; }
#x53 { background-position: -65px 30px; }
#x54 { background-position: -68px 30px; }
#x55 { background-position: -74px 30px; }
</style>
</head>
<body>
<div id="coke">
    <div id="y">
        <p id="x1"></p><p id="x2"></p><p id="x3"></p><p id="x4"></p><p id="x5"></p><p id="x6"></p>
<p id="x7"></p><p id="x8"></p><p id="x9"></p><p id="x10"></p><p id="x11"></p><p id="x12"></p><p id="x13">
</p><p id="x14"></p><p id="x15"></p><p id="x16"></p><p id="x17"></p><p id="x18"></p><p id="x19"></p><p
id="x20"></p><p id="x21"></p><p id="x22"></p><p id="x23"></p><p id="x24"></p><p id="x25"></p><p id="x26">
</p><p id="x27"></p><p id="x28"></p><p id="x29"></p><p id="x30"></p><p id="x31"></p><p id="x32"></p><p
id="x33"></p><p id="x34"></p><p id="x35"></p><p id="x36"></p><p id="x37"></p><p id="x38"></p><p id="x39">
</p><p id="x40"></p><p id="x41"></p><p id="x42"></p><p id="x43"></p><p id="x44"></p><p id="x45"></p><p
id="x46"></p><p id="x47"></p><p id="x48"></p><p id="x49"></p><p id="x50"></p><p id="x51"></p><p id="x52">
</p><p id="x53"></p><p id="x54"></p><p id="x55"></p>
        <a href="#"> <img src="images/coke-can.png" alt="Pure CSS Coke Can" /> </a> </div>
</div>
</body>
</html>
```

16.4.4　设计光盘滑动动画

本例将模拟光盘旋转出仓的动画效果，如图 16.29 所示。示例能够兼容 Webkit 类型的浏览器。旋转光盘动画效果主要通过旋转转换实现，光盘出仓的动画效果则主要通过移动转换实现。通过渐变和阴影设计光盘效果以及局部细节。



默认状态

滑出光盘

图 16.29　光盘出仓动画

完整代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body { background: #fff url(images/aple.jpg); }
/*设计外框向左移动*/
ul.tunes { margin-left: -20px; }
/*定义每个光盘项样式*/
ul.tunes li {
    position: relative; /*定义包含块*/
    width: 200px;
    height: 190px;
    float: left;
    margin-left: 20px;
}
/*音乐盒外框*/
ul.tunes li div.album {
    margin: 0 0 48px 0;
    display: inline;
    width: 200px;
    height: 120px;
    position: absolute;
    text-decoration: none;
    /*定义线性动画，0.15 秒，针对所包含的所有元素*/
    -webkit-transition: all .15s linear;
    color: #333;
    left: 0px;
    top: 0px;
}
ul.tunes img {
    width: 120px;
    position: relative;
    z-index: 30;
    float: left;
    -webkit-box-shadow: 0 3px 6px rgba(0, 0, 0, .5); /*音乐盒阴影*/
    -webkit-border-radius: 2px; /*音乐盒圆角*/
}
ul.tunes li div.album div {
    display: block;
    opacity: .95;
    text-align: center;
    /*线性动画，0.25 秒，针对所包含的所有元素*/
    -webkit-transition: all .25s linear;
    clear: left;
    width: 120px;
}
```

Note

```
ul.tunes li div.album div h5 { text-align: center; }
/*鼠标指针经过时外框的不透明度*/
ul.tunes li div.album:hover div { opacity: 1; }
/*光驱样式*/
ul.tunes li div.album span.vinyl {
    width: 116px;
    height: 116px;
    z-index: 1;
    display: block;
    /*线性动画，0.25 秒，针对所包含的所有元素*/
    -webkit-transition: all .25s linear;
    position: absolute;
    top: 2px;
    left: 2px;
    margin-left: 16px;
}
/*光驱内框样式*/
ul.tunes li div.album span.vinyl div {
    position: absolute;
    top: 2px;
    left: 2px;
    display: block;
    z-index: 10;
    width: 112px;
    height: 112px;
    /*定义圆角*/
    -webkit-border-radius: 59px;
    -moz-border-radius: 59px;
    /*线性动画，0.25 秒，针对所包含的所有元素*/
    -webkit-box-shadow: 0 0 6px rgba(0, 0, 0, .5);
    -webkit-transition: all .25s linear;
    overflow: hidden; /*禁止超出外框*/
    border: solid 1px black;
    /*设计光驱效果，通过直线渐变和径向渐变设计光泽效果*/
    background:
 -webkit-gradient( linear, left top, left bottom, from(transparent), color-stop(0.1, transparent), color-stop(0.5,
rgba(255, 255, 255, 0.25)), color-stop(0.9, transparent), to(transparent)), -webkit-gradient( radial, 56 56, 10, 56 56, 114,
from(transparent), color-stop(0.01, transparent), color-stop(0.021, rgba(0, 0, 0, 1)), color-stop(0.09, rgba(0, 0, 0, 1)),
color-stop(0.1, rgba(28, 28, 28, 1)), to(rgba(28, 28, 28, 1)));
    border-top: 1px solid rgba(255, 255, 255, .25);
}
/*鼠标经过时，设计出仓动画效果*/
ul.tunes li div.album:hover span.vinyl { -webkit-transform: translateX(60px); }
/*在出仓过程中旋转内框，营造立体动画效果*/
ul.tunes li div.album:hover span.vinyl div {
    -webkit-transform: rotate(120deg);
    border-top: 0;
    border-left: 1px solid rgba(255, 255, 255, .25);
}
/*设计掩饰物*/
```

```
ul.tunes li span.gloss {
    display: block;
    position: absolute;
    top: 0;
    left: 0;
    width: 120px;
    height: 120px;
    background: url(images/sheen3.png) no-repeat;
    z-index: 100;
}
/*设计音乐盒被激活时样式*/
ul.tunes li div.album ul.actions {
    display: block;
    position: absolute;
    width: 60px;
    /*定义圆角效果*/
    -moz-border-radius: 3px;
    -webkit-border-radius: 3px;
    left: 70px;
    top: 0px;
    height: 114px;
    z-index: 20;
    /*线性动画，0.25 秒，针对所包含的所有元素*/
    -webkit-transition: all 0.25s linear;
}
/*激活时光盘出仓动画*/
ul.tunes li div.album:hover ul.actions { -webkit-transform: translateX(60px); }
/*激活时光盘上按钮样式*/
ul.tunes li div.album ul.actions li {
    display: block;
    position: absolute;
    height: 20px;
    width: 20px;
    left: 10px;
    top: 22px;
    /*设计凹凸效果*/
    background: -webkit-gradient(linear, left top, left bottom, from(black), to(#333));
    /*设计圆角*/
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    /*添加阴影*/
    -webkit-box-shadow: 0 1px 0 rgba(255, 255, 255, .15);
}
/*被激活时改变光盘上按钮的凹凸效果*/
ul.tunes li div.album ul.actions li:hover { background: -webkit-gradient(linear, left top, left bottom, from(#333),
to(black)); }
ul.tunes li div.album ul.actions li.info {
    top: 48px;
    left: 19px;
}
```

Note

```
ul.tunes li div.album ul.actions li a {
    display: block;
    width: 20px;
    height: 20px;
}
/*播放按钮*/
ul.tunes li div.album ul.actions li.play-pause a { background: url(images/play-button.png) no-repeat center center; }
/*信息按钮*/
ul.tunes li div.album ul.actions li.info a { background: url(images/info.png) no-repeat center center; }
ul.tunes li { text-shadow: 0 2px 3px rgba(0, 0, 0, .75); }
ul.tunes h5 {
    padding-top: 8px;
    color: #fff;
}
ul.tunes small {
    color: #fff;
    opacity: .75;
}
</style>
</head>
<body>
<ul class="tunes">
    <li>
        <div class="album"> <a href=""><img src="images/222.JPG" /></a> <span class="vinyl">
            <div></div>
            </span>
            <ul class="actions">
                <li class="play-pause"><a href=""></a></li>
                <li class="info"><a href=""></a></li>
            </ul>
            <div>
                <h5>超人不会飞</h5>
                <small>作词：周杰伦<br />作曲：周杰伦<br />演唱：周杰伦</small></div>
            <span class="gloss"></span></div>
    </li>
</ul>
</body>
</html>
```

16.4.5　设计下拉菜单

多级菜单的设计方法有多种，一般使用 JavaScript 实现，也可以使用 CSS2 设计的多级菜单，但兼容性比较差，代码冗余度比较高，实战使用比较少。下面完全使用 CSS3 设计一个比较经典的下拉菜单，示例使用 CSS3 的 text-shadow、radius-border 和 boxshadow 等属性，巧妙设计出一个比较漂亮的多级渐变菜单，整个页面没有用到任何脚本和图片，演示效果如图 16.30 所示。本示例综合运用了 CSS3 渐变、文字阴影、圆角和盒子阴影等新技术，能够兼容 IE8+、Chrome 2+、Firefox 3.6+和 Safari 4+主流浏览器。

Note

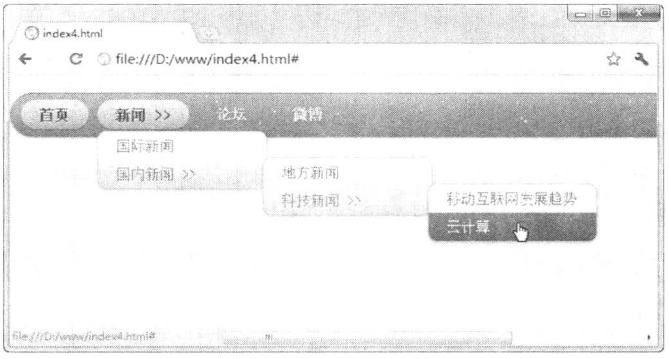


图 16.30　设计下拉菜单

```
<!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> </title>
<style type="text/css">
body { background: #ebebeb; width: 900px; margin: 20px auto; color: #666;}
a { color: #333; }
/*定义菜单外框样式*/
#nav {
    margin: 0;
    padding: 7px 6px 0;
    line-height: 100%; /*实现菜单项垂直居中*/
    /*设计圆角外框及其阴影效果*/
    border-radius: 2em;
    -webkit-border-radius: 2em;
    -moz-border-radius: 2em;
    -webkit-box-shadow: 0 1px 3px rgba(0, 0, 0, .4);
    -moz-box-shadow: 0 1px 3px rgba(0, 0, 0, .4);
    /*下面样式将为菜单外框添加渐变背景效果*/
    background: #8b8b8b; /*兼容不支持 CSS3 浏览器*/
    filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#a9a9a9',endColorstr='#7a7a7a'); /*兼容 IE*/
    background: -webkit-gradient(linear, left top, left bottom, from(#a9a9a9), to(#7a7a7a));
    /*兼容 Webkit 引擎浏览器*/
    background: -moz-linear-gradient(top, #a9a9a9, #7a7a7a);   /*兼容 Firefox 3.6+*/
    border: solid 1px #6d6d6d;
}
#nav li { /*定义子菜单列表项样式*/
    margin: 0 5px;
    padding: 0 0 8px;
    float: left;    /*实现水平并列显示*/
    position: relative; /*定义包含块，以便绝对定位子元素*/
    list-style: none;
}
/*主菜单项链接样式：默认效果*/
#nav a {
```

Note

```
        font-weight: bold;
        color: #e7e5e5;
        text-decoration: none;
        display: block;
        padding: 8px 20px;
        margin: 0;
        /*设计圆角及其字体阴影效果*/
        -webkit-border-radius: 1.6em;
        -moz-border-radius: 1.6em;
        text-shadow: 0 1px 1px rgba(0, 0, 0, .3);
}
/*主菜单项链接样式：鼠标经过时效果*/
#nav .current a, #nav li:hover > a {
        background: #d1d1d1; /*兼容不支持 CSS3 浏览器*/
        /*兼容 IE*/
        filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#ebebeb',endColorstr='#a1a1a1');
        background: -webkit-gradient(linear, left top, left bottom, from(#ebebeb), to(#a1a1a1));
        /*兼容 Webkit 浏览器*/
        background: -moz-linear-gradient(top, #ebebeb, #a1a1a1); /*兼容 Firefox 3.6+*/
        color: #444;
        border-top: solid 1px #f8f8f8;
        /*设计盒子阴影及其字体阴影效果*/
        -webkit-box-shadow: 0 1px 1px rgba(0, 0, 0, .2);
        -moz-box-shadow: 0 1px 1px rgba(0, 0, 0, .2);
        box-shadow: 0 1px 1px rgba(0, 0, 0, .2);
        text-shadow: 0 1px 0 rgba(255, 255, 255, .8);
}
/*子菜单链接样式：鼠标指针经过时效果*/
#nav ul li:hover a, #nav li:hover li a {
        background: none;
        border: none;
        color: #666;
        -webkit-box-shadow: none;
        -moz-box-shadow: none;
}
#nav ul a:hover {
        background: #0399d4 !important; /*兼容不支持 CSS3 浏览器*/
      /*兼容 IE*/
        filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#04acec',endColorstr='#0186ba');
        background: -webkit-gradient(linear, left top, left bottom, from(#04acec), to(#0186ba))
        !important; /*兼容 Webkit 浏览器*/
        background: -moz-linear-gradient(top, #04acec, #0186ba) !important; /*兼容 Firefox 3.6+*/
        color: #fff !important;
        -webkit-border-radius: 0;
        -moz-border-radius: 0;
        text-shadow: 0 1px 1px rgba(0, 0, 0, .1);
}
/*二级列表项样式*/
#nav ul {
        background: #ddd;    /*兼容不支持 CSS3 浏览器*/
```

Note

```
    /*兼容 IE*/
    filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#ffffff',endColorstr='#cfcfcf');
    /*兼容 Webkit 浏览器*/
    background: -webkit-gradient(linear, left top, left bottom, from(#fff), to(#cfcfcf));
    background: -moz-linear-gradient(top, #fff, #cfcfcf); /*兼容 Firefox 3.6+*/
    display: none; /*在默认状态下隐藏子菜单列表框*/
    margin: 0;
    padding: 0;
    width: 185px;
    position: absolute; /*通过绝对定位，能够更精确地控制子菜单的显示位置*/
    top: 35px;
    left: 0;
    border: solid 1px #b4b4b4;
    /*设计子菜单显示圆角边框，以及阴影效果*/
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
    -webkit-box-shadow: 0 1px 3px rgba(0, 0, 0, .3);
    -moz-box-shadow: 0 1px 3px rgba(0, 0, 0, .3);
    box-shadow: 0 1px 3px rgba(0, 0, 0, .3);
}
/*定义鼠标指针经过时显示子菜单*/
#nav li:hover > ul { display: block; }
#nav ul li {
    float: none;
    margin: 0;
    padding: 0;
}
#nav ul a {
    font-weight: normal;
    text-shadow: 0 1px 1px rgba(255, 255, 255, .9);
}
/*三级菜单列表框样式*/
#nav ul ul {
    left: 181px;
    top: -3px;
}
/*为第一个和最后一个子元素设置圆角效果*/
#nav ul li:first-child > a {    /*设置第一个子元素圆角效果*/
-webkit-border-top-left-radius: 9px;
    -moz-border-radius-topleft: 9px;
    -webkit-border-top-right-radius: 9px;
    -moz-border-radius-topright: 9px;
}
#nav ul li:last-child > a { /*设置最后一个子元素圆角效果*/
    -webkit-border-bottom-left-radius: 9px;
    -moz-border-radius-bottomleft: 9px;
    -webkit-border-bottom-right-radius: 9px;
    -moz-border-radius-bottomright: 9px;
}
```

Note

```
/*清除固定高度，让菜单框自动张开包含所有子元素*/
#nav:after {
    content: ".";
    display: block;
    clear: both;
    visibility: hidden;
    line-height: 0;
    height: 0;
}
#nav { display: inline-block; } /*标准用法*/
html[xmlns] #nav { display: block; } /*兼容 IE7*/
html #nav { height: 1%; } /*兼容 IE6*/
</style>
</head>
<body>
<ul id="nav">
    <li class="current"><a href="#">首页</a></li>
    <li><a href="#">新闻  >></a>
        <ul>
            <li><a href="#">国际新闻</a></li>
            <li><a href="#">国内新闻 >></a>
                <ul>
                    <li><a href="#">地方新闻</a></li>
                    <li><a href="#">科技新闻  >></a>
                        <ul>
                            <li><a href="#">移动互联网发展趋势</a></li>
                            <li><a href="#">云计算</a></li>
                        </ul>
                    </li>
                </ul>
            </li>
        </ul>
    </li>
    <li><a href="#">论坛</a></li>
    <li><a href="#">微博</a></li>
</ul>
</body>
</html>
```

16.4.6　设计精致按钮

设计精致的按钮一般会借助 Photoshop，这种方法较灵活，也很安全，但是适应能力比较差，重用性和扩展性不是很高。如果发挥 CSS3 新增的渐变、阴影、圆角等功能，就可以直接使用 CSS 快速设计各种精巧的按钮。纯 CSS 按钮可以根据字体大小自动伸缩，也可以通过修改 padding 和 font-size 属性值来调整按钮大小，同时还可以应用到任何 HTML 元素，如 div、span、p、a、button 和 input 等。本例设计的纯 CSS 按钮具有如下特点：

☑　不需要图片和 JavaScript。

☑　能够兼容 IE、Firefox 3.6+、Chrome 和 Safari 等主流浏览器。不兼容 Opera 浏览器。

☑ 支持 3 种按钮状态，如正常、悬停和激活。

☑ 可以应用到任何 HTML 元素，如 a、input、button、span、div、p 和 h3 等。

☑ 安全兼容不支持 CSS3 浏览器，如果不兼容 CSS3，则显示没有渐变和阴影的普通按钮。

Note

本案例设计的按钮效果如图 16.31 所示。按钮在正常状态下有边框的渐变和阴影，在光标经过时按钮会显示比较暗的渐变效果，当单击鼠标时会翻转渐变，并显示一个像素的下沉效果，按钮字体颜色加深。

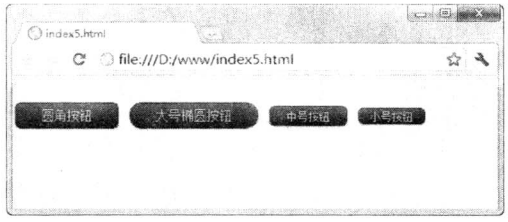


图 16.31 设计精致的按钮

```
<!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></title>
<style type="text/css">
body {
    background:#ededed;
    margin: 30px auto;
    color: #999;
}
.button { /*定义渐变按钮样式类*/
    display: inline-block;
    /*zoom 和 *display 属性都为了兼容 IE7，使其具有 display:inlineblock 特性*/
    zoom: 1;
    *display: inline;
    vertical-align: baseline;
    margin: 0 2px;
    outline: none;
    cursor: pointer;
    text-align: center;
    text-decoration: none;
    font: 14px/100% Arial, Helvetica, sans-serif;
    padding: .5em 2em .55em;
    /*设计按钮圆角、盒子阴影和文本阴影特效*/
    text-shadow: 0 1px 1px rgba(0, 0, 0, .3);
    -webkit-border-radius: .5em;
    -moz-border-radius: .5em;
    border-radius: .5em;
    -webkit-box-shadow: 0 1px 2px rgba(0, 0, 0, .2);
    -moz-box-shadow: 0 1px 2px rgba(0, 0, 0, .2);
    box-shadow: 0 1px 2px rgba(0, 0, 0, .2);
}
```

Note

```
.button:hover { text-decoration: none; }
.button:active {
    position: relative;
    top: 1px;
}
.bigrounded { /*定义大圆角样式类*/
    -webkit-border-radius: 2em;
    -moz-border-radius: 2em;
    border-radius: 2em;
}
.medium { /*定义大按钮样式类*/
    font-size: 12px;
    padding: .4em 1.5em .42em;
}
.small { /*定义小按钮样式类*/
    font-size: 11px;
    padding: .2em 1em .275em;
}
/*设计颜色样式类：黑色风格的按钮*/
/*通过设计不同颜色样式类，可以设计不同风格的按钮效果*/
.black { /*黑色样式类*/
    color: #d7d7d7;
    border: solid 1px #333;
    background: #333;
    background: -webkit-gradient(linear, left top, left bottom, from(#666), to(#000));
    background: -moz-linear-gradient(top, #666, #000);
    filter:  progid:DXImageTransform.Microsoft.gradient(startColorstr='#666666', endColorstr='#000000');
}
.black:hover { /*黑色鼠标经过样式类*/
    background: #000;
    background: -webkit-gradient(linear, left top, left bottom, from(#444), to(#000));
    background: -moz-linear-gradient(top, #444, #000);
    filter:  progid:DXImageTransform.Microsoft.gradient(startColorstr='#444444', endColorstr='#000000');
}
.black:active { /*黑色鼠标激活样式类*/
    color: #666;
    background: -webkit-gradient(linear, left top, left bottom, from(#000), to(#444));
    background: -moz-linear-gradient(top, #000, #444);
    filter:  progid:DXImageTransform.Microsoft.gradient(startColorstr='#000000', endColorstr='#666666');
</style>
</head>
<body>
<div>
    <a href="#" class="button black">圆角按钮</a>
    <a href="#" class="button black bigrounded">大号椭圆按钮</a>
    <a href="#" class="button black medium">中号按钮</a>
    <a href="#" class="button black small">小号按钮</a> <br />
</div>
</body>
</html>
```

第17章

网页布局

（视频讲解：40分钟）

网页布局是指在页面中如何对标题、导航栏、主要内容、脚注、表单等各种构成要素进行一个合理的编排。在CSS3之前，主要使用float属性或position属性进行页面布局，但是使用这些属性也存在一些缺点，两栏或多栏中如果元素的内容高度不一致则底部很难对齐。因此，在CSS3中增加了一些新的布局方式。使用这些新的布局方式，除了可以修改之前存在的问题之外，还可以进行更便捷、更复杂的页面布局。CSS3的Multiple Columns多列自动布局功能可以参考http://www.w3.org/TR/css3-multicol/页面，利用多列布局属性可以自动将内容按指定的列数排列，这种特性特别适合报纸和杂志类网页布局。

多列布局适合纯文档版式设计，不适合做网页结构布局。灵活使用多列布局特性，可以实现在多列中显示文字和图片，从而节省大量的网页空间。如果网页上的文本很长，多列布局特性就能够发挥它的用武之地。

17.1 设计多列布局

Note

使用 float 属性或 position 属性进行页面布局时有一个比较明显的缺点，就是多列的 div 元素间是各自独立的，因此如果在第一列 div 元素中加入一些内容，将会使得两列元素的底部不能对齐，导致页面中多出一块空白区域。这种情况在多列文章排版时显得更为明显，且问题更多，此时需要用户手动调整两列文本内容，这种操作会相当麻烦。

columns 是多列布局特性的基本属性，该属性可以同时定义列数和每列的宽度。columns 属性的基本语法如下所示。

```
columns:column-width> || <column-count>;
```

columns 属性初始值根据元素个别属性而定，适用于不可替换的块元素、行内块元素和单元格，但是表格元素除外。取值简单说明如下。

☑ <column-width>：定义每列的宽度。

☑ <column-count>：定义列数。

目前 Webkit 引擎支持-webkit-columns 私有属性，其他类型的浏览器暂时不支持 columns 属性，也没有定义支持 columns 的私有属性。

【示例】在网页中显示大量正文文字时，建议分列显示，以方便浏览器阅读。使用浮动布局实现这种效果往往会存在两个难题：一是多列浮动显示不容易进行控制；二是多列显示后各列内容无法互通，这样就为后期编辑带来诸多不便。如果使用 CSS3 多列布局特性，这些问题就可以轻松解决。本示例的演示效果如图 17.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css" media="screen">
/*设计网页文档分 3 栏显示，每栏宽度为 250 像素*/
body {
    -webkit-columns: 250px 3;
    columns: 250px 3;
}
h1 {
    color: #333333;
    padding: 5px 8px;
    font-size: 20px;
    text-align: center;
    padding: 12px;
}
h2 {
    font-size: 16px;
    text-align: center;
}
```

```
p {
    color: #333333;
    font-size: 14px;
    line-height: 180%;
    text-indent: 2em;
}
</style>
<title></title>
</head>
<body>
<h1>在斯坦福大学 2005 年毕业典礼上的演讲（选段）</h1>
<h2>史蒂夫 乔布斯</h2>
<p> 我今天很荣幸能和你们一起参加毕业典礼，斯坦福大学是世界上最好的大学之一。我从来没有从大学中毕业。说实话，今天也许是在我的生命中离大学毕业最近的一天了。今天我想向你们讲述我生活中的三个故事。不是什么大不了的事情，只是三个故事而已。</p>
<p>第一个故事是关于如何把生命中的点点滴滴串连起来。</p>
<p>我在 Reed 大学读了六个月之后就退学了，但是在十八个月以后——我真正的作出退学决定之前，我还经常去学校。我为什么要退学呢？</p>
<p>故事从我出生的时候讲起。我的亲生母亲是一个年轻的，没有结婚的大学毕业生。她决定让别人收养我，她十分想让我被大学毕业生收养。所以在我出生的时候，她已经做好了一切的准备工作。所以我的养父母突然在半夜接到了一个电话:“我们现在这儿有一个不小心生出来的男婴，你们想要他吗？”他们回答道:</p>
<p>“当然!”但是我亲生母亲随后发现，我的养母从来没有上过大学，我的养父甚至从没有读过高中。她拒绝签这个收养合同。只是在几个月以后，我的父母答应她一定要让我上大学，那个时候她才勉强同意。</p>
<p>在十七岁那年，我真的上了大学。但是我很愚蠢的选择了一个几乎和你们斯坦福大学一样贵的学校，我父母还处于蓝领阶层，他们几乎把所有积蓄都花在了我的学费上面。在六个月后，我已经看不到其中的价值所在。我不知道我真正想要做什么，我也不知道大学能怎样帮助我找到答案。但是在这里，我几乎花光了我父母这一辈子的全部积蓄。所以我决定要退学，我觉得这是个正确的决定。不能否认，我当时确实非常的害怕，但是现在回头看看，那的确是我这一生中最棒的一个决定。在我做出退学决定的那一刻，我终于可以不必去读那些令我提不起丝毫兴趣的课程了。然后我可以开始去修那些看起来有点意思的课程。</p>
</body>
</html>
```

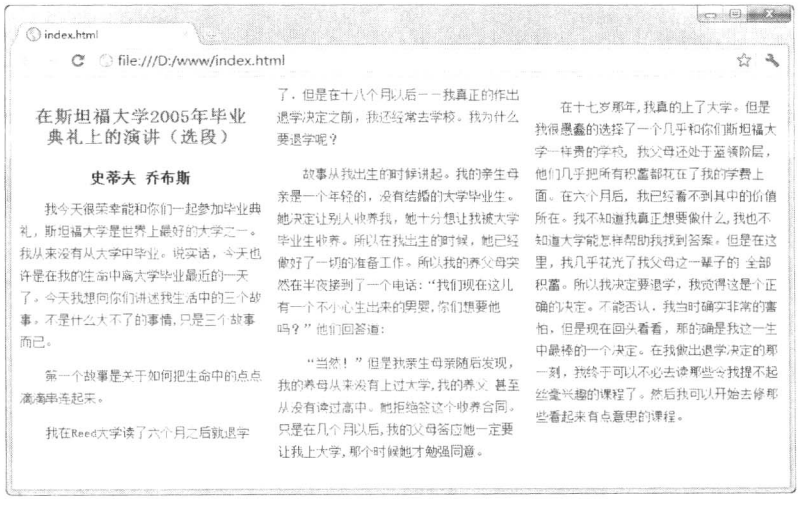


图 17.1　设计文章多栏显示演示效果

17.2　设置多列显示样式

Note

本节将详细介绍多列布局中基本属性定义和样式设置，主要包括列宽、列高、列数、列间距、列边框样式以及跨列显示。

17.2.1　定义列宽

column-width 属性可以定义单列显示的宽度，该属性可以与其他多列布局属性配合使用，也可以单独使用。column-width 属性的基本语法如下所示。

```
column-width:<length> | auto;
```

column-width 属性初始值为 auto，适用于不可替换的块元素、行内块元素和单元格，但是表格元素除外。取值简单说明如下。

☑　<length>：由浮点数字和单位标识符组成的长度值。不可为负值。

☑　auto：根据浏览器计算值自动设置。

目前 Webkit 引擎支持-webkit-column-width 私有属性，Mozilla Gecko 引擎支持-moz-column-width 私有属性，Presto 引擎和 IE 浏览器暂时不支持 column-width 属性，也没有定义支持 column-width 属性的私有属性。

column-width 可以与其他多列布局属性配合使用，设计指定固定列数、列宽的布局效果，也可以单独使用，限制模块的单列宽度，当超出宽度时，则会自动以多列进行显示。

【示例】以本示例为基础，演示 column-width 属性在多列布局中的应用。设计 body 元素的列宽度为 300 像素，如果网页内容能够在单列内显示，则会以单列显示；如果窗口足够宽，且内容很多，则会在多列中进行显示，演示效果如图 17.2 所示，根据窗口宽度自动调整为两栏显示，列宽度显示为 300 像素。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css" media="screen">
/*定义网页列宽为 300 像素，则网页中每个栏目的最大宽度为 300 像素*/
body {
    -webkit-column-width:300px;
    -moz-column-width:300px;
    column-width:300px;
}
h1 {color: #333333; padding: 5px 8px;font-size: 20px;text-align: center; padding: 12px;}
h2 {font-size: 16px; text-align: center;}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
</style>
<title></title>
</head>
<body>
```

```
<h1>在斯坦福大学 2005 年毕业典礼上的演讲（选段）</h1>
<h2>史蒂夫 乔布斯</h2>
<p> 我今天很荣幸能和你们一起参加毕业典礼，斯坦福大学是世界上最好的大学之一。</p>
…
</body>
</html>
```

Note

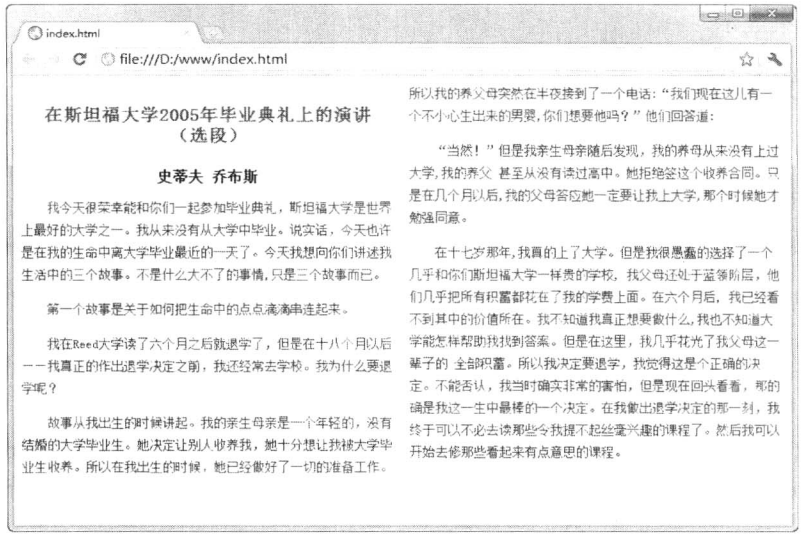


图 17.2　浏览器根据窗口宽度变化调整栏目的数量

17.2.2　定义列数

column-count 属性可以定义显示的列数，该属性的基本语法如下所示。

```
column-count:<integer> | auto;
```

column-count 属性初始值为 auto，适用于不可替换的块元素、行内块元素和单元格，但是表格元素除外。取值简单说明如下。

- ☑ <integer>：定义栏目的列数，取值为大于 0 的整数。如果 column-width 和 column-count 属性没有明确值，则该值为最大列数。
- ☑ auto：根据浏览器计算值自动设置。

目前 Webkit 引擎支持-webkit-column-width 私有属性，Mozilla Gecko 引擎支持-moz-column-width 私有属性，Presto 引擎和 IE 浏览器暂时不支持 column-width 属性，也没有定义支持 column-width 属性的私有属性。

【示例】在 17.2.1 节示例基础上，如果定义网页列数为 3，则不管浏览器窗口怎么调整，页面内容总是遵循 3 列布局，演示效果如图 17.3 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css" media="screen">
/*定义网页列数为 3，这样整个页面总是显示为 3 列*/
```

```
body {
    -webkit-column-count:3;
    -moz-column-count:3;
    column-count:3;
}
h1 {color: #333333; padding: 5px 8px;font-size: 20px;text-align: center; padding: 12px;}
h2 {font-size: 16px; text-align: center;}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
</style>
<title></title>
</head>
<body>
<h1>在斯坦福大学 2005 年毕业典礼上的演讲（选段）</h1>
<h2>史蒂夫 乔布斯</h2>
<p> 我今天很荣幸能和你们一起参加毕业典礼，斯坦福大学是世界上最好的大学之一。</p>
…
</body>
</html>
```

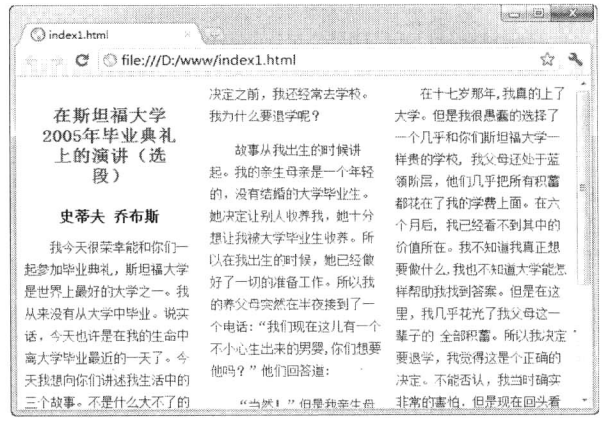


图 17.3 根据窗口宽度自动调整列宽，但是整个页面总是显示为 3 列

17.2.3 定义列间距

column-gap 属性可以定义两栏之间的间距，该属性的基本语法如下所示。

```
column-gap:normal | <length>;
```

column-gap 属性初始值为 normal，适用于多列布局元素。取值简单说明如下。

☑ normal：根据浏览器默认设置进行解析，一般为 1em。

☑ <length>：由浮点数字和单位标识符组成的长度值。不可为负值。

目前 Webkit 引擎支持-webkit-column-gap 私有属性，Mozilla Gecko 引擎支持-moz-column-gap 私有属性，Presto 引擎和 IE 浏览器暂时不支持 column-gap 属性，也没有定义支持 columngap 属性的私有属性。

【示例】在 17.2.2 节示例基础上，通过 column-gap 和 line-height 属性配合使用，把文档版面设计得疏朗大方，以方便阅读。其中列间距为 3em，行高为 2.5em，页面内文字内容看起来更明晰、轻松许多，演示效果如图 17.4 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css" media="screen">
body {
    /*定义页面内容显示为 3 列*/
    -webkit-column-count: 3;
    -moz-column-count: 3;
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    -webkit-column-gap: 3em;
    -moz-column-gap: 3em;
    column-gap: 3em;
    line-height: 2.5em; /*定义页面文本行高*/
}
h1 {color: #333333; padding: 5px 8px;font-size: 20px;text-align: center; padding: 12px;}
h2 {font-size: 16px; text-align: center;}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
</style>
<title></title>
</head>
<body>
<h1>在斯坦福大学 2005 年毕业典礼上的演讲（选段）</h1>
<h2>史蒂夫 乔布斯</h2>
<p> 我今天很荣幸能和你们一起参加毕业典礼，斯坦福大学是世界上最好的大学之一。</p>
…
</body>
</html>
```

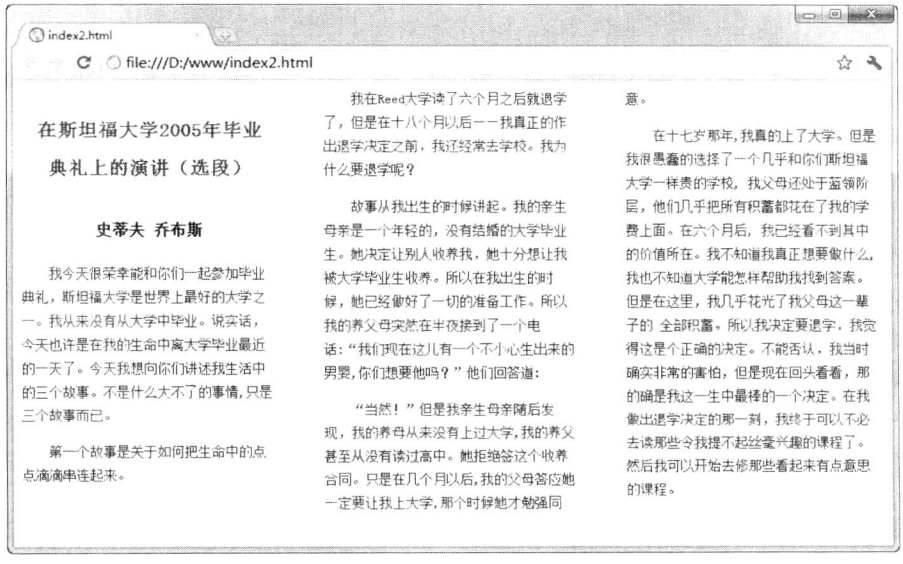


图 17.4　设计疏朗的页面布局

17.2.4 定义列边框样式

为列边框设计样式，能够有效区分各个栏目列之间的关系，阅读时比较清晰。column-rule 属性可以定义每列之间边框的宽度、样式和颜色。该属性的基本语法如下所示。

Note

```
column-rule:<length> | <style> | <color> | <transparent>;
```

column-rule 属性初始值根据个别属性而定，适用于多列布局元素。取值简单说明如下。

- ☑ <length>：由浮点数字和单位标识符组成的长度值，不可为负值。功能与 column-rule-width 属性相同。
- ☑ <style>：定义列边框样式。功能与 column-rule-style 属性相同。
- ☑ <color>：定义列边框的颜色。功能与 column-rule-color 属性相同。
- ☑ <transparent>：设置边框透明显示。

CSS3 在 column-rule 属性基础上派生了 3 个列边框属性。

- ☑ column-rule-color：定义列边框颜色。column-rule-color 属性接受所有的颜色。Webkit 引擎支持-webkit-column-rule-color 私有属性，Mozilla Gecko 引擎支持-moz-columnrule-color 私有属性。
- ☑ column-rule-width：定义列边框宽度。column-rule-width 属性接受任意浮点数，但不为负值。Webkit 引擎支持-webkit-column-rule-width 私有属性，Mozilla Gecko 引擎支持-moz-column-rule-width 私有属性。
- ☑ column-rule-style：定义列边框样式。column-rule-width 属性值与 border-style 属性值相同，包括 none、hidden、dotted、dashed、solid、double、groove、ridge、inset 和 outset。Webkit 引擎支持-webkit-column-rule-style 私有属性，Mozilla Gecko 引擎支持-moz-column-rule-style 私有属性。

目前 Webkit 引擎支持-webkit-column-rule 私有属性，Mozilla Gecko 引擎支持-moz-column-rule 私有属性，Presto 引擎和 IE 浏览器暂时不支持 column-rule 属性，也没有定义支持 column-rule 属性的私有属性。

【示例】在 17.2.3 节示例基础上，为每列之间的边框定义一个虚线分割线，线宽为 2 像素，灰色显示，演示效果如图 17.5 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css" media="screen">
body {
    /*定义页面内容显示为 3 列*/
    -webkit-column-count: 3;
    -moz-column-count: 3;
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    -webkit-column-gap: 3em;
    -moz-column-gap: 3em;
```

```
    column-gap: 3em;
    line-height: 2.5em;
    /*定义列边框为 2 像素宽的灰色虚线*/
    -webkit-column-rule: dashed 2px gray;
    -moz-column-rule: dashed 2px gray;
    column-rule: dashed 2px gray;
}
h1 {color: #333333; padding: 5px 8px;font-size: 20px;text-align: center; padding: 12px;}
h2 {font-size: 16px; text-align: center;}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
</style>
<title></title>
</head>
<body>
<h1>在斯坦福大学 2005 年毕业典礼上的演讲（选段）</h1>
<h2>史蒂夫 乔布斯</h2>
<p> 我今天很荣幸能和你们一起参加毕业典礼，斯坦福大学是世界上最好的大学之一。</p>
…
</body>
</html>
```

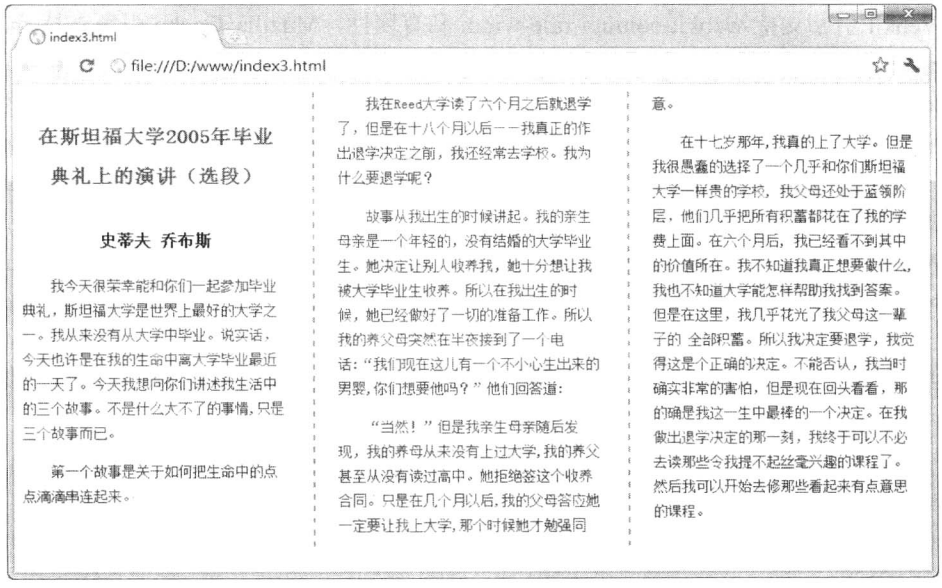


图 17.5　设计列边框效果

17.2.5　定义跨列显示

在纸质报刊杂志中，经常会看到文章标题跨列居中显示。column-span 属性可以定义跨列显示，也可以设置单列显示。该属性的基本语法如下所示。

```
column-span:1 | all;
```

column-span 属性初始值为 1，适用于静态的、非浮动元素。取值简单说明如下。

☑ 1：只在本栏中显示。

☑ all：将横跨所有列，并定位在列的 Z 轴之上。

目前 Webkit 引擎支持-webkit-column-span 私有属性，Mozilla Gecko 引擎、Presto 引擎和 IE 浏览器暂时不支持 column-span 属性，也没有定义支持 column-span 属性的私有属性。

Note

【示例】在 17.2.4 节示例基础上，使用 column-span 属性定义一级和二级标题跨列显示，演示效果如图 17.6 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css" media="screen">
body {
    /*定义页面内容显示为 3 列*/
    -webkit-column-count: 3;
    -moz-column-count: 3;
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    -webkit-column-gap: 3em;
    -moz-column-gap: 3em;
    column-gap: 3em;
    line-height: 2.5em;
    /*定义列边框为 2 像素宽的灰色虚线*/
    -webkit-column-rule: dashed 2px gray;
    -moz-column-rule: dashed 2px gray;
    column-rule: dashed 2px gray;
}
/*设置一级标题跨越所有列显示*/
h1 {
    color: #333333;
    font-size: 20px;
    text-align: center;
    padding: 12px;
    -webkit-column-span: all;
    -moz-column-span: all;
    column-span: all;
}
/*设置二级标题跨越所有列显示*/
h2 {
    font-size: 16px;
    text-align: center;
    -webkit-column-span: all;
    -moz-column-span: all;
    column-span: all;
}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
```

```
</style>
<title></title>
</head>
<body>
<h1>在斯坦福大学 2005 年毕业典礼上的演讲（选段）</h1>
<h2>史蒂夫 乔布斯</h2>
<p> 我今天很荣幸能和你们一起参加毕业典礼，斯坦福大学是世界上最好的大学之一。</p>
…
</body>
</html>
```

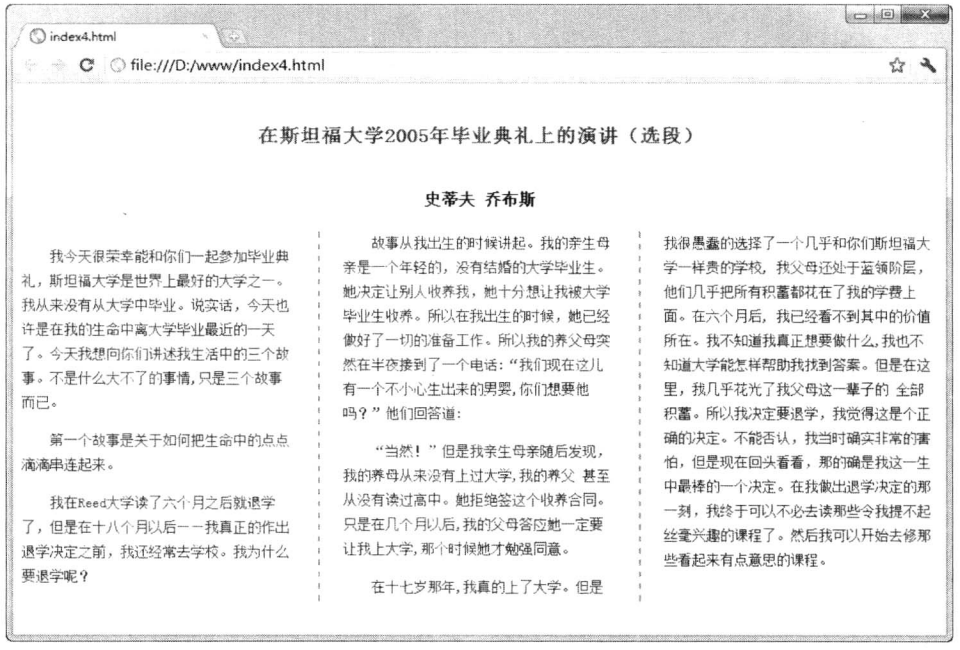


图 17.6　设计标题跨列显示效果

17.2.6　定义列高度

column-fill 属性可以定义栏目的高度是否统一。该属性的基本语法如下所示。

```
column-fill:auto | balance;
```

column-fill 属性初始值为 balance，适用于多列布局元素。取值简单说明如下。

☑　auto：各列的高度随其内容的变化而自动变化。

☑　balance：各列的高度将会根据内容最多的那一列的高度进行统一。

目前，Webkit 引擎支持-webkit-column-fill 私有属性，Mozilla Gecko 引擎、Presto 引擎（包括 Opera 浏览器等）和 IE 浏览器暂时不支持 column-fill 属性，也没有定义支持 column-fill 属性的私有属性。

【示例】在 17.2.5 节示例基础上，使用 column-fill 属性定义每列高度一致，演示效果如图 17.7 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Note

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<style type="text/css" media="screen">
body {
    /*定义页面内容显示为 3 列*/
    -webkit-column-count: 3;
    -moz-column-count: 3;
    column-count: 3;
    /*定义列间距为 3em，默认为 1em*/
    -webkit-column-gap: 3em;
    -moz-column-gap: 3em;
    column-gap: 3em;
    line-height: 2.5em;
    /*定义列边框为 2 像素宽的灰色虚线*/
    -webkit-column-rule: dashed 2px gray;
    -moz-column-rule: dashed 2px gray;
    column-rule: dashed 2px gray;
    /*设置各列高度自动调整*/
    -webkit-column-fill: auto;
    -moz-column-fill: auto;
    column-fill: auto;
}
/*设置一级标题跨越所有列显示*/
h1 {
    color: #333333;
    font-size: 20px;
    text-align: center;
    padding: 12px;
    -webkit-column-span: all;
    -moz-column-span: all;
    column-span: all;
}
/*设置二级标题跨越所有列显示*/
h2 {
    font-size: 16px;
    text-align: center;
    -webkit-column-span: all;
    -moz-column-span: all;
    column-span: all;
}
p {color: #333333; font-size: 14px; line-height: 180%; text-indent: 2em;}
</style>
<title></title>
</head>
<body>
<h1>在斯坦福大学 2005 年毕业典礼上的演讲（选段）</h1>
<h2>史蒂夫 乔布斯</h2>
<p> 我今天很荣幸能和你们一起参加毕业典礼，斯坦福大学是世界上最好的大学之一。</p>
```

```
……
</body>
</html>
```

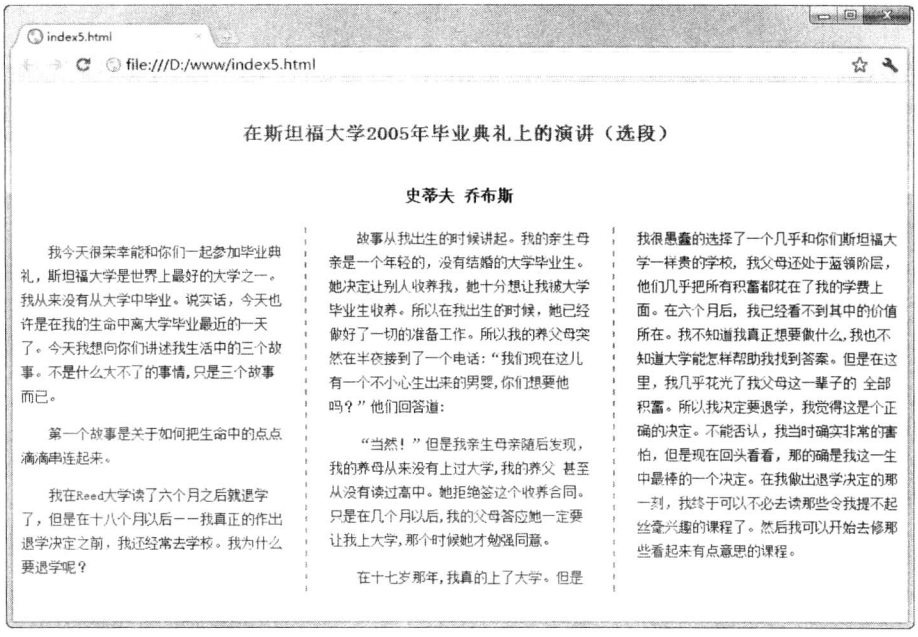


图 17.7　设计每列显示高度一致

17.2.7　定义打印列

break-before、break-after 和 break-inside 是新增的用来控制页面分列打印的 3 个属性，它们的功能与 CSS 2.1 规范中的 page-break-before、page-break-after 和 page-breakinside 3 个属性相同，用法也相同。这些属性的基本语法如下所示。

```
break-before:auto | always | avoid | left | right | page | column | avoid-page |avoid-column;
break-after:auto | always | avoid | left | right | page | column | avoid-page |avoid-column;
break-inside:auto | always | avoid | left | right | page | column | avoid-page |avoid-column;
```

取值简单说明如下。

- ☑ auto：不强迫也不禁止在生成框之前（之后或之间）分页。
- ☑ always：总是强迫在生成框之前（之后）分页。
- ☑ avoid：避免在生成框之前（之后或之间）分页。
- ☑ left：强迫在生成框之前（之后）分一个或两个页，使下一页成为一个左页。
- ☑ right：强迫在生成框之前（之后）分一个或两个页，使下一页成为一个右页。
- ☑ page：总是强迫在生成框之前（之后）分页。
- ☑ column：总是强迫在生成框之前（之后）分列。
- ☑ avoid-page：总是避免在生成框之前（之后）分页。
- ☑ avoid-column：总是避免在生成框之前（之后）分列。

一个可能的分页位置通常受到父元素的 break-inside 属性、前继元素的 break-after 属性或后续元素的 page-break-before 属性的影响。如果这些属性值不为 auto，则 always、left 和 right 属性值将优先

于 avoid。

目前，Webkit 引擎支持-webkit--break-before、-webkit--break-after 和-webkit--break-inside 私有属性，Mozilla Gecko 引擎、Presto 引擎（包括 Opera 浏览器等）和 IE 浏览器暂时不支持这些属性，也没有定义支持这些标准属性的私有属性。

Note

17.3　设计盒布局

CSS3 引入了新的盒模型——弹性盒模型，该模型决定一个盒子在其他盒子中的分布方式以及如何处理可用的空间。使用该模型可以很轻松地创建自适应浏览器窗口的流动布局或自适应字体大小的弹性布局。

传统的盒模型基于 HTML 流在垂直方向上排列盒子。使用弹性盒模型可以规定特定的顺序，也可以将其反转。要开启弹性盒模型，只需设置拥有子盒子的盒子的 display 属性值为 box（或 inline-box）即可。

```
display: box;
```

盒布局与多列布局的区别在于使用多列布局时，各列宽度必须是相等的，在指定每列宽度时，也只能为所有列指定一个统一的宽度。列与列之间的宽度不可能是不一样的。另外，使用多列布局时，也不可能具体指定什么列中显示什么内容，因此比较适用于显示文章内容时，不适合用于安排整个网页中由各元素组成的网页结构。

在 CSS3 中，除了多列布局之外，还可以使用盒布局解决使用 float 属性或 position 属性布局存在的问题。下面以示例比较的形式介绍盒布局的优点和应用技巧。

【示例 1】本示例使用 float 属性进行布局，该示例中有 3 个 div 元素，简单展示了网页中的左侧边栏、中间内容和右侧边栏，预览效果如图 17.8 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title></title>
<style type="text/css">
#left-sidebar {
    float: left;
    width: 160px;
    padding: 20px;
    background-color: orange;
}
#contents {
    float: left;
    width: 500px;
    padding: 20px;
    background-color: yellow;
}
```

Note

```
#right-sidebar {
    float: left;
    width: 160px;
    padding: 20px;
    background-color: limegreen;
}
#left-sidebar, #contents, #right-sidebar {
    box-sizing: border-box;
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
</style>
</head>
<body>
<div id="container">
    <div id="left-sidebar">
        <h2>站内导航</h2>
        <ul>
            <li><a href="">新闻</a></li>
            <li><a href="">博客</a></li>
            <li><a href="">微博</a></li>
            <li><a href="">社区</a></li>
            <li><a href="">关于</a></li>
        </ul>
    </div>
    <div id="contents">
        <h2>标题：JavaScript 是实现 HTML5 强大功能的重要语言</h2>
        <p> JavaScript 时代的大幕才刚刚拉开。在今年的 JSConf 上，一个核心主题就是“JavaScript 到 JavaScript 的编译器”，也被人们看成是未来的一个主要趋势。Google 在“编译生成 JavaScript 代码”方面是首开先河者。据我所知，GWT（Google Web Toolkit）应该是通过编译（从 Java 代码）生成 JavaScript 代码的第一个框架。以前我对 GWT 并没有太重视，只是觉得它是一个致力于拯救那些 Java 程序员的框架，好让他们不必因为（学习）编写 JavaScript 而浪费时间。可是，GWT 在编译过程中对 JavaScript 做了那么多的优化，简直是太神了。Closure 就是一个“JavaScript 到 JavaScript 的编译器”，能够实现同样级别的优化。Traceur，这是几个星期前才冒出来的一个框架，通过它能够试验 JavaScript 的新特性，换句话说，它可以把带有实验性语言特性的 JavaScript 代码编译成可以在所有现代平台中运行的 JavaScript 代码。</p>
    </div>
    <div id="right-sidebar">
        <h2>友情链接</h2>
        <ul>
            <li><a href="">百度</a></li>
            <li><a href="">谷歌</a></li>
            <li><a href="">360</a></li>
        </ul>
    </div>
</div>
</body>
</html>
```

图 17.8　使用 float 属性进行布局

通过图 17.8 可以看出，使用 float 属性或 position 属性时，左右两栏或多栏中 div 元素的底部并没有对齐。

如果使用盒布局，那么这个问题将很容易得到解决。在 CSS3 中，通过 box 属性来使用盒布局，目前 Webkit 引擎支持-webkit-box 私有属性，Mozilla Gecko 引擎支持-moz-box 私有属性。

【示例 2】以示例 1 为基础，为最外层的 标签定义 box 属性，并去除了代表左侧边栏 、中间内容 和右侧边栏 中 div 元素样式中的 float 属性，修改内部样式表代码如下，显示效果如图 17.9 所示。

```
<style type="text/css">
/*定义包含框为盒子布局*/
#container {
    display: box;
    display: -moz-box;
    display: -webkit-box;
}
#left-sidebar {
    width: 160px;
    padding: 20px;
    background-color: orange;
}
#contents {
    width: 500px;
    padding: 20px;
    background-color: yellow;
}
#right-sidebar {
    width: 160px;
    padding: 20px;
    background-color: limegreen;
}
/*绑定 3 列栏目为一个盒子整体布局效果*/
#left-sidebar, #contents, #right-sidebar {
    box-sizing: border-box;
```

```
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
</style>
```

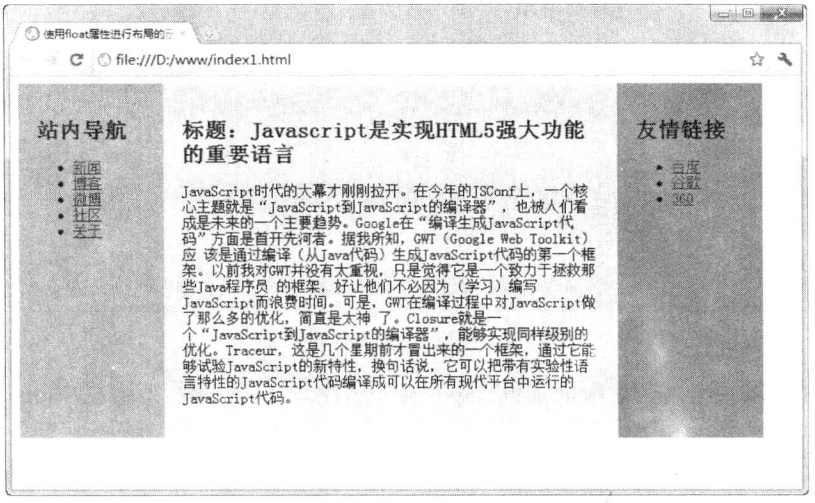


图 17.9　使用 box 属性进行布局

【示例 3】为了方便与多列布局进行比较，在本示例中将示例 2 修改为多列布局格式。将代码清单最外层的 标签样式改为通过 column-count 属性来定义，以便应用多栏布局，并去除代表左侧边栏 、中间内容 和右侧边栏 的 div 元素的 float 属性与 width 属性，修改代码如下所示，修改后重新运行该示例，则运行效果如图 17.10 所示。

```
<style type="text/css">
#container {
    column-count: 3;
    -moz-column-count: 3;
    -webkit-column-count: 3;
}
#left-sidebar {
    padding: 20px;
    background-color: orange;
}
#contents {
    padding: 20px;
    background-color: yellow;
}
#right-sidebar {
    padding: 20px;
    background-color: limegreen;
}
</style>
```

图 17.10 使用多列布局

通过图 17.10 可以看到，在多列布局中，3 列栏目融合在一起，因此多列布局不适合应用到网页结构控制方面，它仅适合于文章多列排版。

17.4 设置盒布局格式

17.3 节详细讲解了盒布局的基本方法，本节将介绍如何定义盒布局的基本格式，以便更灵活地控制页面布局效果。

17.4.1 定义自适应宽度

17.3 节介绍的盒布局中，如果对代表左侧边栏、中间内容和右侧边栏的 3 个 div 元素的宽度都进行了设定，希望这 3 个 div 元素的总宽度等于浏览器窗口的宽度，而且能够随着窗口宽度的改变而改变时，应该怎么设定呢？

在使用 float 属性或 position 属性的时候，一般使用包括负外边距在内的比较复杂的方法才能够达到这个要求，但是如果使用盒布局的话，只要使用一个 box-flex 属性，把盒布局变为弹性盒布局就可以了。Webkit 引擎支持-webkit-box-flex 私有属性，Mozilla Gecko 引擎支持-moz-box-flex 私有属性。

默认情况下，盒子并不具有弹性，如果 box-flex 的属性值至少为 1 时，则变得富有弹性。如果盒子不具有弹性，它将尽可能的宽使其内容可见，且没有任何溢出，其大小由 width 和 height 属性值，或者 min-height、min-width、max-width 和 max-height 属性值来决定。

如果盒子是弹性的，其大小将按下面的方式计算：

☑ 具体的大小声明（width、height、min-width、min-height、max-width 和 max-height）。

☑ 父盒子的大小和所有余下的可利用的内部空间。

如果盒子没有任何大小声明，那么其大小将完全取决于父盒子的大小，即子盒子的大小等于父盒子的大小乘以其 box-flex 在所有子盒子 box-flex 总和中的百分比，用如下公式表示：

子盒子的大小=父盒子的大小*子盒子的 box-flex/所有子盒子的 box-flex 值的和

如果一个或更多的盒子有一个具体的大小声明，那么其大小将计算其中，余下的弹性盒子将按照上面的原则分享剩下的可利用空间。

Note

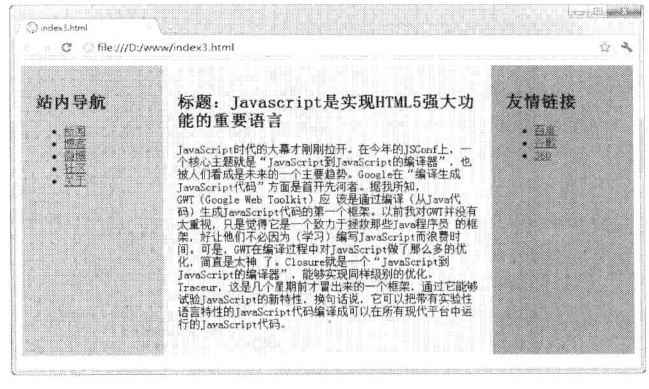
【示例】下面针对 17.3 节示例，在样式代码中添加 box-flex 属性，在样式代码中使用盒布局，并将表示左侧边栏与右侧边栏的两个 div 元素的宽度保留为 200 像素，在表示中间内容的 div 元素的样式代码中去除原来的指定宽度为 300 像素的样式代码，加入 box-flex 属性。详细代码如下所示，演示效果如图 17.11 所示，当调整窗口宽度时，中间列的宽度会自适应显示，使整个页面总是满窗口显示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title></title>
<style type="text/css">
#container{
    /*定义弹性盒布局样式*/
    display: -moz-box;
    display: -webkit-box;
}
#left-sidebar{
    width: 200px;
    padding: 20px;
    background-color: orange;
}
#contents{
    /*定义中间列宽度为自适应显示*/
    -moz-box-flex:1;
    -webkit-box-flex:1;
    padding: 20px;
    background-color: yellow;
}
#right-sidebar{
    width: 200px;
    padding: 20px;
    background-color: limegreen;
}
#left-sidebar, #contents, #right-sidebar{
    /*定义盒样式*/
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
</style>
</head>
<body>
<div id="container">
    <div id="left-sidebar">
        <h2>站内导航</h2>
        <ul>
```

Note

```
            <li><a href="">新闻</a></li>
            <li><a href="">博客</a></li>
            <li><a href="">微博</a></li>
            <li><a href="">社区</a></li>
            <li><a href="">关于</a></li>
        </ul>
    </div>
    <div id="contents">
        <h2>标题：JavaScript 是实现 HTML5 强大功能的重要语言</h2>
        <p> JavaScript 时代的大幕才刚刚拉开。在今年的 JSConf 上，一个核心主题就是“JavaScript 到 JavaScript 的编译器”，也被人们看成是未来的一个主要趋势。Google 在“编译生成 JavaScript 代码”方面是首开先河者。据我所知，GWT（Google Web Toolkit）应该是通过编译（从 Java 代码）生成 JavaScript 代码的第一个框架。以前我对 GWT 并没有太重视，只是觉得它是一个致力于拯救那些 Java 程序员 的框架，好让他们不必因为（学习）编写 JavaScript 而浪费时间。可是，GWT 在编译过程中对 JavaScript 做了那么多的优化，简直是太神了。Closure 就是一个“JavaScript 到 JavaScript 的编译器”，能够实现同样级别的优化。Traceur，这是几个星期前才冒出来的一个框架，通过它能够试验 JavaScript 的新特性，换句话说，它可以把带有实验性语言特性的 JavaScript 代码编译成可以在所有现代平台中运行的 JavaScript 代码。</p>
    </div>
    <div id="right-sidebar">
        <h2>友情链接</h2>
        <ul>
            <li><a href="">百度</a></li>
            <li><a href="">谷歌</a></li>
            <li><a href="">360</a></li>
        </ul>
    </div>
</div>
</body>
</html>
```

弹性盒布局窗口变窄



弹性盒布局窗口变宽

图 17.11　定义自适应宽度

17.4.2　定义列显示顺序

使用弹性盒布局的时候，可以用 box-ordinal-group 属性来改变各元素的显示顺序。可以在每个元素的样式中加入 box-ordinal-group 属性，该属性使用一个表示序号的整数属性值，浏览器在显示的时

候根据该序号从小到大来显示这些元素。

提示：如果使用 Firefox 浏览器，需要使用-moz-box-ordinal-group 私有属性，如果使用 Safari 浏览器或 Chrome 浏览器，则需要使用-webkit-box-ordinal-group 私有属性。

Note

【示例】继续针对 17.4.1 节示例，可以将该示例中的样式代码修改为如下所示的样式代码，在代表左侧边栏、中间内容和右侧边栏的 div 元素都加入一个 box-ordinal-group 属性，并在该属性中指定显示时的序号，这里将中间内容的序号指定为 1，右侧边栏的序号指定为 2，左侧边栏的序号指定为 3，则可以发现 3 列栏目的显示顺序发生了变化，演示效果如图 17.12 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title></title>
<style type="text/css">
#container{
    display: -moz-box;
    display: -webkit-box;
}
#left-sidebar{
    -moz-box-ordinal-group: 3;
    -webkit-box-ordinal-group: 3;
    width: 200px;
    padding: 20px;
    background-color: orange;
}
#contents{
    -moz-box-ordinal-group: 1;
    -webkit-box-ordinal-group: 1;
    -moz-box-flex:1;
    -webkit-box-flex:1;
    padding: 20px;
    background-color: yellow;
}
#right-sidebar{
    -moz-box-ordinal-group: 2;
    -webkit-box-ordinal-group: 2;
    width: 200px;
    padding: 20px;
    background-color: limegreen;
}
#left-sidebar, #contents, #right-sidebar{
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
</style>
</head>
<body>
```

Note

```
<div id="container">
    <div id="left-sidebar">
        <h2>站内导航</h2>
        <ul>…</ul>
    </div>
    <div id="contents">
        <h2>…</h2>
        <p>…</p>
    </div>
    <div id="right-sidebar">
        <h2>友情链接</h2>
        <ul>…</ul>
    </div>
</div>
</body>
</html>
```

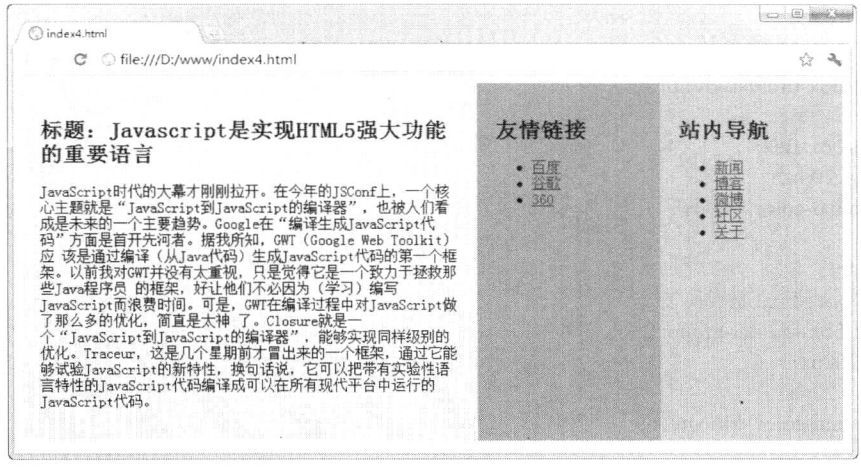


图 17.12　定义列显示顺序

从上面示例的演示效果可以看出，虽然没有改变 HTML5 的页面代码，但是通过使用弹性盒布局，使用-moz-box-ordinal-group 属性，同样可以改变元素的显示顺序，这样可以提高页面布局的工作效率。

17.4.3　定义列排列方向

使用弹性盒布局的时候，可以很简单地将多个元素的排列方向从水平方向修改为垂直方向，或者从垂直方向修改为水平方向。在 CSS3 中，使用 box-orient 来指定多个元素的排列方向。在样式代码中，如果使用 Firefox 浏览器，则需要使用-moz-box-orient 私有属性。如果使用 Safari 浏览器或 Chrome 浏览器，则需要使用-webkit-box-orient 私有属性。

【示例】针对 17.4.2 节示例，可以将样式代码修改为如下样式，在 标签样式中加入 box-orient 属性，并设定属性值为 vertical，即定义内容以垂直方向排列，则代表左侧边栏，中间内容和右侧边栏的 3 个 div 元素的排列方向将从水平方向改变为垂直方向，演示效果如图 17.13 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Note

```
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title></title>
<style type="text/css">
#container{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
}
#left-sidebar{
    -moz-box-ordinal-group: 3;
    -webkit-box-ordinal-group: 3;
    width: 200px;
    padding: 20px;
    background-color: orange;
}
#contents{
    -moz-box-ordinal-group: 1;
    -webkit-box-ordinal-group: 1;
    -moz-box-flex:1;
    -webkit-box-flex:1;
    padding: 20px;
    background-color: yellow;
}
#right-sidebar{
    -moz-box-ordinal-group: 2;
    -webkit-box-ordinal-group: 2;
    width: 200px;
    padding: 20px;
    background-color: limegreen;
}
#left-sidebar, #contents, #right-sidebar{
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
</style>
</head>
<body>
<div id="container">
    <div id="left-sidebar">
        <h2>站内导航</h2>
        <ul>…</ul>
    </div>
    <div id="contents">
        <h2>…</h2>
        <p>…</p>
    </div>
    <div id="right-sidebar">
        <h2>友情链接</h2>
        <ul>…</ul>
    </div>
```

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

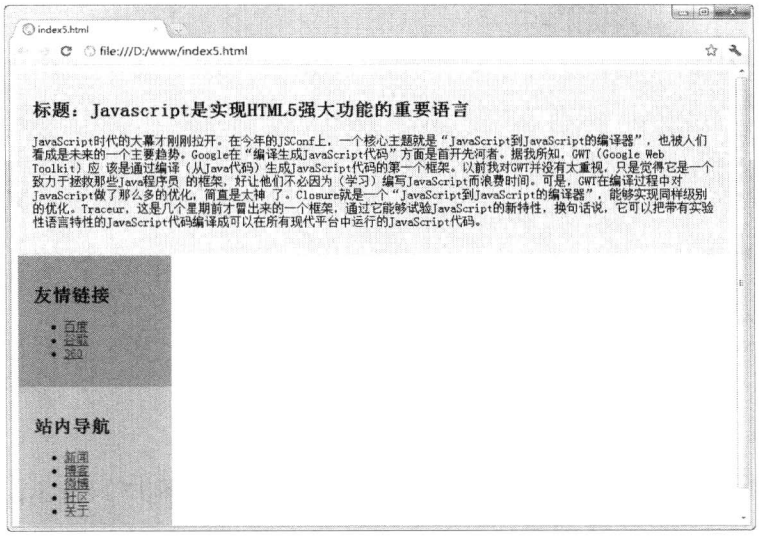


图 17.13　定义列显示方向

17.4.4　定义模块大小自适应

使用盒布局时，元素大小（包括宽和高）具有自适应性，即元素的宽度与高度可以根据排列方向的改变而改变。

【示例 1】在本示例中定义一个容器元素，其中包含 3 个 div 子元素，它们只对容器元素指定了宽度和高度。如果在浏览器中预览，则可以看到当排列方向被指定为水平方向时，3 个元素的宽度为元素中内容的宽度，高度自动变为容器的高度，当排列方向被指定为垂直方向时，3 个元素的高度为元素中内容的高度，宽度自动变为容器的宽度。

```
<!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=gb2312" />
<title></title>
<style type="text/css">
#container {
    display: -moz-box;
    display: -webkit-box;
    border: solid 1px red;
    -moz-box-orient: horizontal;
    -webkit-box-orient: horizontal;
    width: 800px;
    height: 200px;
}
#text-a { background-color: orange; }
#text-b { background-color: yellow; }
```

Note

```
#text-c { background-color: limegreen; }
#text-a, #text-b, #text-c {
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
    font-size: 1.5em;
    font-weight: bold;
    width:200px;
}
</style>
</head>
<body>
<div id="container">
    <div id="text-a">列 1</div>
    <div id="text-b">列 2</div>
    <div id="text-c">列 3</div>
</div>
</body>
</html>
```

在上面代码中，定义元素的排列方向为水平方向，运行效果如图 17.14 所示。

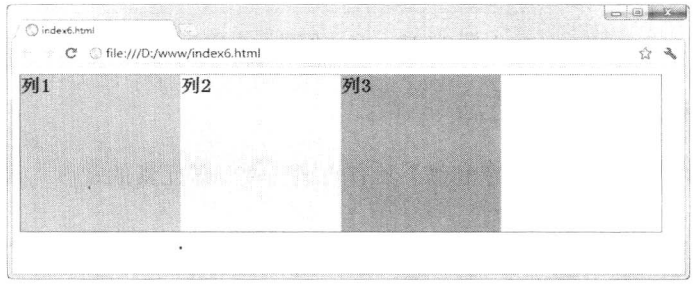


图 17.14　定义列水平方向显示

【示例 2】针对示例 1，不修改其他代码，如果只在容器元素的样式代码中把排列方向改变为垂直方向，删除 3 列样式中的宽度定义代码，添加固定高度声明，其中修改的样式代码如下，则在浏览器中的预览效果如图 17.15 所示。

```
#container {
    display: -moz-box;
    display: -webkit-box;
    border: solid 1px red;
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
    width: 800px;
    height: 500px;
}
#text-a, #text-b, #text-c {
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
    font-size: 1.5em;
    font-weight: bold;
    height:150px;
}
```

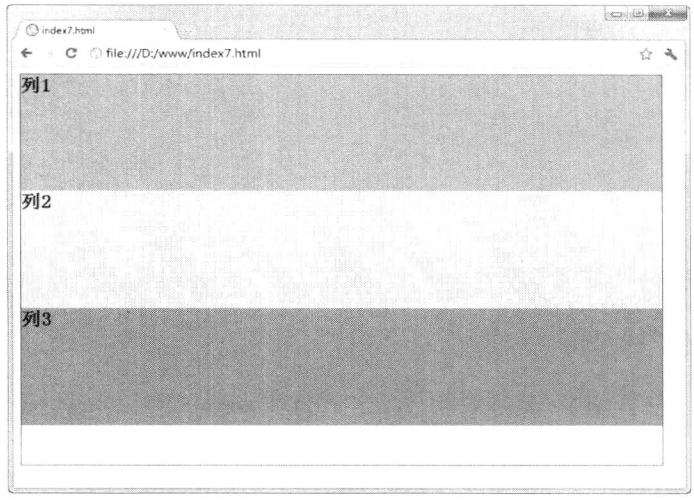


图 17.15 定义列垂直方向显示

17.4.5 消除空白

在 17.4.4 节示例中，可以看到网页布局中元素的大小不能够完全自适应的问题，不过如果改用弹性盒布局，就可以解决参与布局的元素的总宽度与总高度始终等于容器的宽度和高度。

【示例 1】以 17.4.4 节示例为基础，修改其中的样式代码，将排列方向定义为水平方向，在每一个 div 子元素的样式代码中添加 box-flex 属性，然后在浏览器中预览，则显示效果如图 17.16 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title></title>
<style type="text/css">
#container {
    display: -moz-box;
    display: -webkit-box;
    border: solid 1px red;
    -moz-box-orient: horizontal;
    -webkit-box-orient: horizontal;
    width: 800px;
    height: 400px;
}
#text-a { background-color: orange; width:200px; }
#text-b {
    background-color: yellow;
    -moz-box-flex: 1;
    -webkit-box-flex: 1;
}
#text-c { background-color: limegreen;   width:160px; }
#text-a, #text-b, #text-c {
    -moz-box-sizing: border-box;
```

```
    -webkit-box-sizing: border-box;
    font-size: 1.5em;
    font-weight: bold;
}
</style>
</head>
<body>
<div id="container">
    <div id="text-a">列 1</div>
    <div id="text-b">列 2</div>
    <div id="text-c">列 3</div>
</div>
</body>
</html>
```

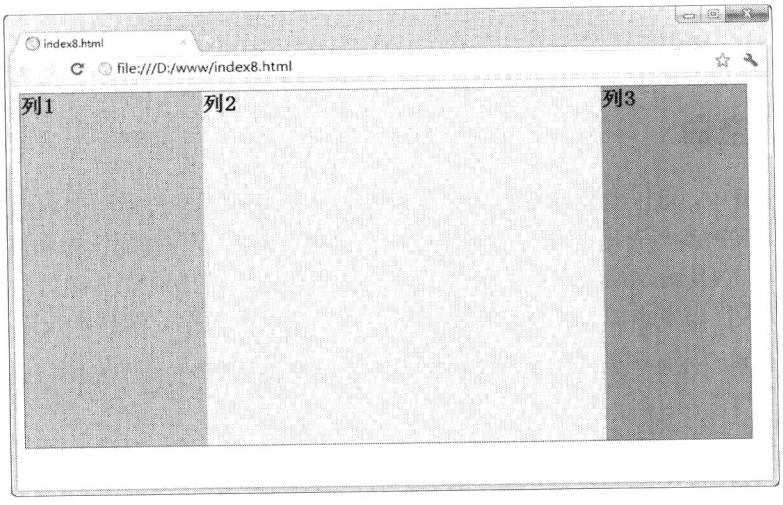


图 17.16　定义包含的元素能够同时自适应宽度和高度

【示例 2】如果修改 容器的样式代码，把排列方向修改为垂直方向，然后重新运行该示例，演示效果如图 17.17 所示。

```
<style type="text/css">
#container {
    display: -moz-box;
    display: -webkit-box;
    border: solid 1px red;
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
    width: 800px;
    height: 400px;
}
#text-a { background-color: orange; height:100px; }
#text-b {
    background-color: yellow;
    -moz-box-flex: 1;
    -webkit-box-flex: 1;
}
```

```
#text-c { background-color: limegreen;   height:60px; }
#text-a, #text-b, #text-c {
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
    font-size: 1.5em;
    font-weight: bold;
}
</style>
```

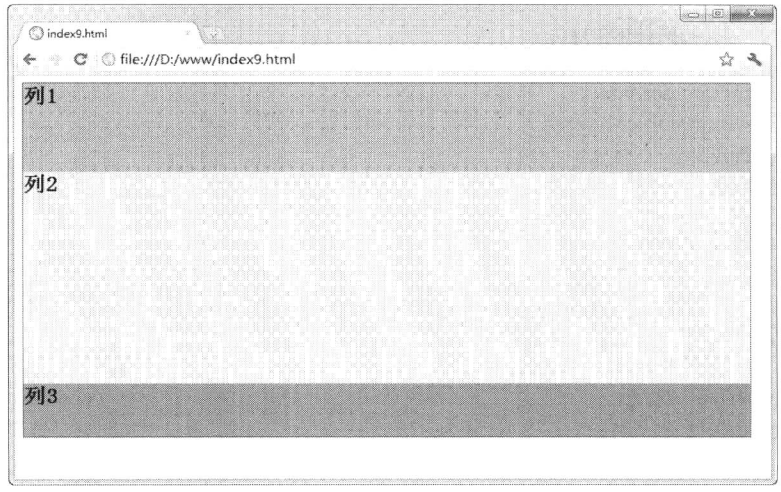


图 17.17　把排列方向修改为垂直方向

从图 17.17 可以看到，如果使用弹性盒布局，使用了 box-flex 属性的元素的宽度与高度总会自动扩大，使得参与排列的元素的总宽度与总高度始终等于容器元素的高度与宽度。

上面示例都是只对一个元素使用 box-flex 属性，使其宽度和高度自适应，让浏览器或容器中所有元素的总宽度或总高度等于浏览器或容器的宽度或高度。当然也可以为多个元素使用 box-flex 属性。

【示例 3】在示例 2 中把容器的前两个 div 元素的样式代码中都使用 box-flex 属性，元素排列方向为垂直排列，具体代码如下所示。在浏览器中预览，则显示效果如图 17.18 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title></title>
<style type="text/css">
#container {
    display: -moz-box;
    display: -webkit-box;
    border: solid 1px red;
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
    width: 800px;
    height: 400px;
}
#text-a {
```

```
        background-color: orange;
        -moz-box-flex: 1;
        -webkit-box-flex: 1;
    }
    #text-b {
        background-color: yellow;
        -moz-box-flex: 1;
        -webkit-box-flex: 1;
    }
    #text-c { background-color: limegreen; }
    #text-a, #text-b, #text-c {
        -moz-box-sizing: border-box;
        -webkit-box-sizing: border-box;
        font-size: 1.5em;
        font-weight: bold;
    }
    </style>
    </head>
    <body>
    <div id="container">
        <div id="text-a">列 1</div>
        <div id="text-b">列 2</div>
        <div id="text-c">列 3</div>
    </div>
    </body>
    </html>
```

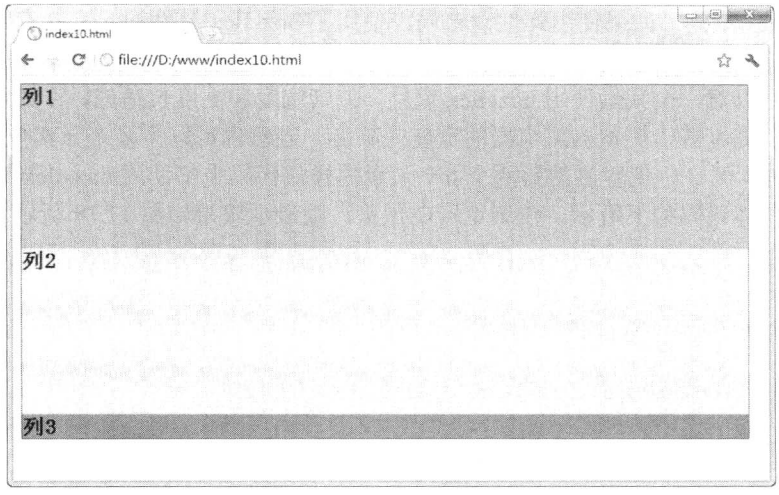


图 17.18　为多个子元素定义弹性布局样式

从运行结果中可以看出，前两个 div 元素的高度都自动扩大了，而且扩大后前两个 div 元素的高度保持相等，而第 3 个 div 元素的高度仍保持为元素内容的高度。如果 3 个 div 元素的样式中都使用 box-flex 属性，则每个 div 元素的高度就等于容器的高度除以 3 了。

到目前为止，在样式中所使用的 box-flex 属性的属性值都是 1，如果某个 div 元素的 box-flex 属性值大于 1，页面显示会自动成倍扩展，相当于 box-flex 属性定义了元素的自适应大小所需要的份额。如果 a 元素的 box-flex 属性值为 1，而 b 元素的 box-flex 属性值为 3，则 a 元素的宽（或者高度）应

该是容器宽度的 1/4，b 元素的宽度（或者高度）为 3/4。

17.4.6　定义对齐方式

Note

使用盒布局时，可以使用 box-pack 属性及 box-align 属性来指定元素中文字、图像及子元素水平方向或垂直方向的对齐方式。在 Firefox 浏览器中，需要使用-moz-box-pack 和-moz-box-align 私有属性，在 Safari 浏览器或 Chrome 浏览器中需要使用-webkit-pack 和-webkit-align 私有属性。

box-pack 和 box-align 属性包含的属性值含义说明如下。

- ☑ start：排列方式 horizontal。针对 box-pack 属性，表示左对齐，文字、图像或子元素被放置在元素最左边显示；针对 box-align 属性，表示顶部对齐，文字、图像或子元素被放置在元素最顶部显示。
- ☑ center：排列方式 horizontal。针对 box-pack 属性，表示中对齐，文字、图像或子元素被放置在元素中部显示；针对 box-align 属性，表示中部对齐，文字、图像或子元素被放置在元素中部显示。
- ☑ end：排列方式 horizontal。针对 box-pack 属性，表示右对齐，文字、图像或子元素被放置在元素最右边显示；针对 box-align 属性，表示底部对齐，文字、图像或子元素被放置在元素最底部显示。
- ☑ start：排列方式 vertical。针对 box-align 属性，表示左对齐，文字、图像或子元素被放置在元素最左边显示；针对 box-pack 属性，表示顶部对齐，文字、图像或子元素被放置在元素最顶部显示。
- ☑ center：排列方式 vertical。针对 box-align 属性，表示中对齐，文字、图像或子元素被放置在元素中部显示；针对 box-pack 属性，表示中部对齐，文字、图像或子元素被放置在元素中部显示。
- ☑ end：排列方式 vertical。针对 box-align 属性，表示右对齐，文字、图像或子元素被放置在元素最右边显示；针对 box-pack 属性，表示底部对齐，文字、图像或子元素被放置在元素最底部显示。

CSS3 版本以前，如果定义文字水平居中，只要使用 text-align 属性就可以了，但是如果要让文字垂直居中，由于 div 元素是不能使用 vertical-align 属性的，所以也就很难做到了。在 CSS3 中，只要让 div 元素使用 box-align 属性（排列方向默认为 horizontal）即可，文字就可以垂直居中了。

【示例】在本示例中有一个 div 元素，其中包含多个 div 子元素，子元素中有一些文字，使用 box-pack 属性及 box-align 属性让文字位于 div 容器的正中央。具体实现代码如下，预览效果如图 17.19 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=gb2312" />
<title></title>
<style type="text/css">
div#container {
    display: -moz-box;
    display: -webkit-box;
    -moz-box-align: center;
    -webkit-box-align: center;
```

```
        -moz-box-pack: center;
        -webkit-box-pack: center;
        width: 300px;
        height: 200px;
        background-color: pink;
    }
    #text-a { background-color: orange; }
    #text-b { background-color: yellow; }
    #text-c { background-color: limegreen; }
    </style>
    </head>
    <body>
    <div id="container">
        <div id="text-a">列 1</div>
        <div id="text-b">列 2</div>
        <div id="text-c">列 3</div>
    </div>
    </body>
    </html>
```

图 17.19　定义文本正中央显示

在 CSS3 之前，还有一种比较难处理的情况，就是如何让图像位于元素正中央，使用了 box-pack 和 box-align 属性，同样也使该问题很容易就得到了解决。

17.5　综合实战：设计多列网页

CSS Zen Garden（http://www.csszengarden.com/）是 Dave Shea 于 2003 年创建的 CSS 标准推广小站，整个页面通过左上、右下对顶角定义背景图像，这些荷花、梅花以及汉字形体修饰配合右上顶角的宗教建筑，完全用户带入富有禅意的后花园之中。

CSS 禅意花园整个页面的信息简洁，结构层次清晰。信息从上到下，按照网页标题、网页菜单、主体栏目信息、次要导航和页脚信息有顺序地排列在一起。所有导航菜单等功能信息全部放在结构的后面，以方便搜索引擎快速检索正文内容。

```
<body id="css-zen-garden">
<div id="container">
    <div id="intro">
        <div id="pageHeader" class="allcols">
```

```
            <h1><span>CSS 禅意花园</span></h1>
            <h2><span><acronym title="cascading style sheets">CSS</acronym>设计之美</span></h2>
        </div>
        <div id="quickSummary" class="allcols">
            <p class="p1"><span>展示以<acronym
    title="cascading style sheets">CSS</acronym>技术为基础，并提供超强的视觉冲击力。只要选择列表中任意一
个样式表，就可以将它加载到本页面中，并呈现不同的设计效果。</span></p>
            <p class="p2"><span>下载<a title="这个页面的 HTML 源代码不能够被改动。"
    href="http://www.csszengarden.com/zengarden-sample.html">HTML 文档</a> 和 <a
    title="这个页面的 CSS 样式表文件，你可以更改它。"
    href="http://www.csszengarden.com/zengarden-sample.css">CSS 文件</a>。</span></p>
        </div>
        <div id="preamble"><img src="images/page.jpg" />
            <h3><span>启蒙之路</span></h3>
            <p class="p1 first"><span>不同浏览器随意定义标签，导致无法相互兼容的<acronym
    title="document object model">DOM</acronym>结构，或者提供缺乏标准支持的<acronym
    title="cascading style sheets">CSS</acronym>等陋习随处可见，如今当使用这些不兼容的标签和样式时，设计
之路会一路坎坷。</span></p>
            <p class="p2"><span>现在，我们必须清除以前为了兼容不同浏览器而使用的一些过时的小技
巧。感谢<acronym
    title="world wide web consortium">W3C</acronym>、<acronym
    title="web standards project">WASP</acronym>等标准组织，以及浏览器厂家和开发师们的不懈努力，我们终
于能够进入 Web 设计的标准时代。</span></p>
            <p class="p3"><span>CSS Zen
                Garden（样式表禅意花园）邀请您发挥自己的想象力，构思一个专业级的网页。让我们用
慧眼来审视，充满理想和激情去学习 CSS 这个不朽的技术，最终使自己能够达到技术和艺术合而为一的最高境
界。</span></p>
        </div>
    </div>
    <div id="supportingText">
        <div id="explanation">
            <h3><span>这是什么？</span></h3>
            <p class="p1"><span>对于网页设计师来说应该好好研究<acronym
    title="cascading style sheets">CSS</acronym>。Zen Garden 致力于推广和使用 CSS 技术，努力激发和鼓励您
的灵感和参与。你可以从浏览高手的设计作品入门。只要选择列表中的任一个样式表，就可以将它加载到这个页
面中。<acronym
    title="hypertext markup language">HTML</acronym>文档结构始终不变，但是你可以自由的修改和定义
<acronym
    title="cascading style sheets">CSS</acronym>样式表。</span></p>
            <p class="p2"><span><acronym
    title="cascading style sheets">CSS</acronym>具有强大的功能，可以自由控制 HTML 结构。当然你需要拥有
驾驭 CSS 技术的能力和创意的灵感，同时亲自动手，用具体的实例展示 CSS 的魅力，展示个人的才华。截至目
前为止，很多 Web 设计师和程序员已经介绍过许多关于 CSS 应用技巧和兼容技术的各种技巧和案例。而平面设
计师还没有足够重视 CSS 的潜力。你是不是需要从现在开始呢？</span></p>
        </div>
        <div id="participation">
            <h3><span>邀您参与</span></h3>
            <p class="p1"><span>我们邀请广大设计师参与到这项计划中来，当然包括您。你需要修改这
个页面的样式表，所以要有良好的<acronym
    title="cascading style sheets">CSS</acronym>技术。不过本文档已经详尽地加上批注，即使是<acronym
    title="cascading style sheets">CSS</acronym>初学者，也能够在此基础上尝试设计出很棒的作品来。请参考
<a title=CSS 相关资源列表
```

Note

href="http://www.mezzoblue.com/zengarden/resources/"><acronym
title="cascading style sheets">CSS</acronym>资源，里面提供了 CSS 入门教程和使用技巧。
你虽然可以自由修改样式表，但是千万别改动<acronym
title="hypertext markup language">HTML</acronym>文档结构。如果你还感觉手比较生，没有思路。建议你先单击列表中的作品，通过深入学习和 研究别人的案例，以便从中找到设计的感觉。
请下载<a title=="这个页面的 HTML 结构请不要改动。"
href="http://www.csszengarden.com/zengarden-sample.html">HTML和<a
title=这个页面的 CSS 范例文件，你可以更改它。
href="http://www.csszengarden.com/zengarden-sample.css">CSS文件，然后尝试并努力在本地计算机中进行修改。当你设计完成之后，先将<acronym
title="cascading style sheets">CSS</acronym>样式表文件上传到你的网页服务器，再把样式表文件的网址<a
title=请用联络窗体给我们你的 CSS 档案
href="http://www.mezzoblue.com/zengarden/submit/">告诉我们。如果我们采用你的作品，就会自动下载相关图片，最后成品将放在我们的服务器上。

<h3>参与好处</h3>
为什么要邀请您参与这项计划呢？因为这里提供了一个平台，能够展现你的技术水平、激发你的设计灵感，并且所设计的<acronym
title="cascading style sheets">CSS</acronym>案例文档，将成为重要的学习资料。即使到了今天这些资料都是很珍贵的和重要的。越来越多主流网站开始采用 CSS 技术，但是还不够。虽然现在还不成气候，但总有一天这些资料会成为历史的宝藏。

<h3>参与要求</h3>
<acronym
title="cascading style sheets, version 1">CSS1</acronym>已经落伍了。<acronym
title="cascading style sheets, version 2">CSS2</acronym>现在成为网页设计的主要技术标准。CSS Zen Garden 采用的是可行、实用的<acronym title="cascading style sheets">CSS</acronym>语法，而不是那些仅能被个别超前浏览器所能解析的技术（如<acronym title="cascading style sheets, version 3">CSS3</acronym>）。实际上，我们唯一的要求就是你的<acronym
title="cascading style sheets">CSS</acronym>要符合标准。
即便如此，你还需要考虑<acronym
title="cascading style sheets">CSS</acronym>的兼容性。因为即使是完全符合标准的<acronym
title="cascading style sheets">CSS</acronym>语法，不同浏览器所呈现的效果也是不一致的。很多设计师整天忙于 CSS 兼容性处理，这实在是件让人恼火的事情。也请你参考相关<a
title=CSS 相关资源列表
href="http://www.mezzoblue.com/zengarden/resources/">资源了解一些 Bug 修正的信息。希望所有浏览器都能够完整、正确、统一的支持<acronym
title="cascading style sheets">CSS</acronym>是不切实际的，我们也不希望你用精准到像素级别 CSS 代码来实现在不同浏览器中具有相同的呈现效果。当然还是请你在各种主要浏览器中进行测试。如果你的设计无法在 IE 5 及以上版本浏览器中正常显示（因为超过 90%以上的用户都使用这些浏览器），我们无法采用你的作品。
<p
class="p3">我们要求你提交原创作品，因此你必须尊重别人的版权。请尽可能不要使用有争议的素材；我们可以接受高雅的裸体素材，但是拒绝低俗的色情图片。
这仅是一个示例，也是学习用的材料。你可以保留图片的完整版权，但我们要求你许可在<a
title="阅读 Zen Garden 的版权声明。"
href="http://creativecommons.org/licenses/sa/1.0/">本站免费发布你的<acronym
title="cascading style sheets">CSS</acronym>作品，这样其他读者可以从中学习。

Note

```
                <p class="p5"><span>带宽由<a
    href="http://www.dreamfirestudios.com/">dreamfire studios</a>提供。</span></p>
            </div>
            <div id="footer"><a title="检查网站的 XHTML 是否合乎标准"
    href="http://validator.w3.org/check/referer">XHTML</a>   <a
    title="检查网站的 CSS 是否合乎标准"
    href="http://jigsaw.w3.org/css-validator/check/referer">CSS</a>   <a
    title="阅读本站版权声明，由慷慨的 creative commons 提供。"
    href="http://creativecommons.org/licenses/by-nc-sa/1.0/">cc</a>   <a
    title="依据美国 508 条款，检查网站的亲和力"
    href="http://bobby.watchfire.com/bobby/bobbyservlet?url=http%3a%2f%2fwww.mezzoblue.com%2fzengarden%2
f&amp;output=submit&amp;gl=sec508&amp;test=">508</a>   <a title="依据 wai content accessibility guidelines
1，检查网站的亲和力"
    href="http://bobby.watchfire.com/bobby/bobbyservlet?url=http%3a%2f%2fwww.mezzoblue.com%2fzengarden%2
f&amp;output=submit&amp;gl=wcag1-aaa&amp;test=">aaa</a> </div>
        </div>
        <div id="linkList">
            <!-- 这是最具弹性的设计部分 - 需要你花费很大的心思来设计这个列表。-->
            <div id="linkList2">
                <!-- 你也许会因为链接而感到讶异，没有办法，这是为了配合 wcag 1 accessibility 使用的技
巧。-->
                <!-- 我也不喜欢这样，不过这是个视觉效果的练习。-->
                <div id="lselect">
                    <h3 class="select"><span>选择一项设计</span></h3>
                    <!-- 这是链接列表的起点，每页不会超过 8 个链接。-->
                    <ul>
                        <li><a title="快捷键: a" accesskey="a"
    href="http://www.csszengarden.com/">作品 #1</a> 创作 <a class=c
    href="http://www.mezzoblue.com/"> 作者 1 </a> 
                        <li><a title="快捷键: b" accesskey="b"
    href="http://www.csszengarden.com/">作品 #2</a> 创作 <a class=c
    href="http://www.mezzoblue.com/"> 作者 2 </a> 
                        ……
                    </ul>
                </div>
                <!--如果你的页面还没有塞满，这里都不会显示，只是准备内容。如果你为「选择一项设计」
的 h3 标签订制了不少样式，这里也必须订制好。
                <div id="lfavorites">
                    <h3 class="favorites"><span>最爱：</span></h3>
                    <ul>
                        <li><a href="#">filename</a> by <a href=" class="c">submitter</a> </li>
                        <li><a href="#">filename</a> by <a href=" class="c">submitter</a> </li>
                    </ul>
                        </div>
    -->
                <div id="larchives">
                    <h3 class="archives"><span>作品档案</span></h3>
                    <ul>
                        <li><a title="浏览下一个设计。快捷键：n" accesskey="n"
    href="http://www.csszengarden.com/tr/chinese/?CSSfile=/001/001-zh.css&amp;page=1">下一个设计
                            &raquo;</a>
```

Note

```
                    <li><a title="浏览 Zen Garden 上的所有设计。快捷键：w" accesskey="w"
href="http://www.mezzoblue.com/zengarden/alldesigns/">浏览所有设计</a> </li>
                </ul>
            </div>
            <div id="lresources">
                <h3 class="resources"><span>参考资源</span></h3>
                <ul>
                    <li><a title="检视这一个设计的 CSS 档案。快捷键：v" accesskey="v"
href="../001-zh.css">查看这个设计的样式表<acronym
title="cascading style sheets">CSS</acronym></a>
                    <li><a title="链接到提供 CSS 相关信息的好网站。快捷键：r" accesskey="r"
href="http://www.mezzoblue.com/zengarden/resources/"><acronym
title="cascading style sheets">CSS</acronym>参考资料</a>
                    ……
                </ul>
            </div>
        </div>
    </div>
</div>
<!-- 这些额外的标签，你可以发挥想象力进行其他的设计。-->
<!-- 你可以为它们加上背景图片，利用 width 和 height 控制它们的大小，最后使用绝对定位方式在页面进行
放置。-->
<!-- 这种做法会在 mozilla v1.0 到 v1.3 造成讨厌的透明 gif 色彩偏移问题，所以请务必使用这些浏览器测
试你的作品。-->
<div id="extraDiv1"><span></span></div>
<div id="extraDiv2"><span></span></div>
……
</body>
```

一般每个页面都会存在很多共同的信息模块，如标题（Logo）、广告（Banner）、导航（Mneu）、功能（Serve）、内容（Content）和版权（Copyright）等信息。而不同类型的网站有不同页面需求，对于各种公共信息模块的取舍和强弱，多少会略有不同，这时就应该具体情况具体分析。在设计网页基本结构时，应该根据信息需求的简单分析和信息的重要性来对页面各个模块进行适当排序，然后列出基本的结构。

```
<div id="c-xms"> <!-- 网页结构外套 -->
    <div id="header"></div> <!-- 网页标题模块 -->
    <div id="nav"></div> <!-- 网页菜单模块 -->
    <div id="content"></div> <!-- 网页信息模块 -->
    <div id="cnav"></div> <!-- 次要导航模块 -->
    <div id="footer"></div> <!-- 版权信息模块 -->
</div>
```

构建基本结构应该注意几个问题。

☑ 在设计基本结构时，可以不考虑标签的语义性，统一使用 div 元素来构建。

☑ 应该为基本结构的每一个 div 元素进行命名（设置 ID 属性），以便后期控制。

☑ 可以考虑为整个页面结构设计一个外套（即定义一个结构根元素），以方便控制。

在设计结构时，不要考虑后期如何显示，也不要顾虑结构的顺序是不是会影响页面的显示效果，应该完全抛弃页面效果和 CSS 样式等概念对结构的影响。

有了基本的框架结构，可以继续深入，这时不妨去完善主体区域的结构（即网页内容模块），这部分是整个页面的核心，也是工作的重点。应该说，在编辑网页结构的全部过程中，不去考虑页面显示效果问题，而是静下心来单纯考虑结构。但是在实际操作中，会不可避免地联想到页面的显示问题，例如，分几行几列显示（这里的行和列是指网页基本结构的走向问题）。不同的行列结构肯定都有适合自己的结构，所以当设计师在进入到这一步时，适当考虑页面显示问题也无可厚非，但是不要考虑太多。

Note

恰当的嵌套结构需要结合具体的信息来说，这里暂不详细分析。抽象地说，模块的结构关系可以分为 3 种基本模型。

☑　平行结构

```
<div id="A"></div>
<div id="B"></div>
<div id="C"></div>
```

☑　包含结构

```
<div id="A">
    <div id="B"></div>
    <div id="C"></div>
</div>
```

☑　嵌套结构

```
<div id="A"></div>
<div>
    <div id="B"></div>
    <div id="C"></div>
</div>
```

具体采用哪种结构都不重要，可以根据信息的结构关系来进行设计。如果 和 两个信息模块内容比较接近，而 模块与它们在内容上相差很远，不妨采用嵌套结构。如果这些栏目的信息类型雷同，使用并列式会更经济。

禅意花园的整个文章包含 3 部分：

☑　站点介绍。站点介绍部分犹如抒情散文，召唤网友赶紧加入到 CSS 标准设计中来，该部分包含 3 段，即网页标题信息（包括主副标题）、内容简介（呼唤网友赶紧加入进来）和启蒙之路（回忆和总结当前标准之路的艰巨性和紧迫性）。

☑　支持文本。支持文本部分犹如叙事散文，娓娓道来，详细介绍活动的内容，用户参与的条件、支持、好处等。该部分包含 4 段，即这是什么、邀您参与、参与好处、参与要求。另外末尾还包含了相关验证信息。

☑　链接列表。这部分很简洁地列出了所有超链接信息。该部分也包含 3 块链接信息。

基于上面的结构，使用 CSS3 多列布局特性设计网页内容显示为多列效果，预览效果如图 17.20 所示。

```
<style type=text/css media=all>
body {
    /*设计多重网页背景，并设置其显示大小*/
    background:url(images/page1.gif) no-repeat right 20px,
    url(images/bg.jpg) no-repeat right bottom,
```

Note

```
        url(images/page3.jpg) no-repeat left top;
        background-size:auto, 74% 79.5%, auto;
        color:#000;
        font-size:12px;
        font-family:"新宋体", Arial, Helvetica, sans-serif;
        /*定义页面内容显示为 3 列*/
        -webkit-column-count:3;
        -moz-column-count:3;
        column-count:3;
        /*定义列间距为 3em，默认为 1em*/
        -webkit-column-gap:3em;
        -moz-column-gap:3em;
        column-gap:3em;
        line-height:2em;
        /*定义列边框为 3 像素，宽的灰色虚线*/
        -webkit-column-rule:double 3px gray;
        -moz-column-rule:double 3px gray;
        column-rule:double 3px gray;
}
/*设计跨列显示类*/
.allcols {
        -webkit-column-span:all;
        -moz-column-span:all;
        column-span:all;
}
h1, h2, h3 {
        text-align:center;
        margin-bottom:1em;
}
h2 {
        color:#666;
        text-decoration:underline;
}
h3 {
        letter-spacing:0.4em;
        font-size:1.4em;
}
p {
        margin:0;
        line-height:1.8em;
}
#quickSummary .p2 { text-align:right; }
#quickSummary .p1 { color:#444; }
.p1, .p2, .p3 { text-indent:2em; }
#quickSummary { margin:4em; }
a { color:#222; }
a:hover {
        color:#000;
        text-decoration:underline;
}
/*设计报刊杂志的首字下沉显示类*/
```

Note

```
.first:first-letter {
    font-size:50px;
    float:left;
    margin-right:6px;
    padding:2px;
    font-weight:bold;
    line-height:1em;
    background:#000;
    color:#fff;
    text-indent:0;
}
#preamble img {
    height:260px;
    /*设计插图跨列显示，但实际浏览无效果*/
    -webkit-column-span:all;
    -moz-column-span:all;
    column-span:all;
}
/*设计栏目框半透明显示，从而设计网页背景半透明显示效果*/
#container {
    background:rgba(255, 255, 255, 0.8);
    padding:0 1em;
}
</style>
```

图 17.20　设计多列网页显示效果

提示：由于 CSS3 的多列布局特性并未得到各大主流浏览器的支持，同时支持浏览器的解析效果也存在差异，所以在不同浏览器中预览时，会看到效果存在一定的差异。

第18章

用户界面

（视频讲解：35分钟）

CSS3规范新增加了UI模块（User-interface样式模块），该模块用来控制与用户界面相关效果的呈现方式，详细资料请参阅http://www.w3.org/TR/css3-ui/。该模块定义了很多为提高用户体验而新增的属性和功能，本章将对这些新增的属性和功能进行详细介绍。

18.1 改变盒模型组成方式

Note

在 IE 5.x 以及 Quirks(怪异)模式的 IE6、IE7 浏览器中，border 和 padding 都包含在 width 或 height 之内。这为设计师平添了不少麻烦。而在符合标准的浏览器中，width 和 height 仅仅包含 content，刨去了 border 和 padding 区域。

为了解决这种问题，CSS3 对盒模型进行了改善，定义了 box-sizing 属性，该属性能够事先定义盒模型的尺寸解析方式。box-sizing 属性的基本语法如下所示。

```
box-sizing:content-box | border-box | inherit;
```

box-sizing 属性初始值为 content-box，适用于所有能够定义宽和高的元素。取值简单说明如下。

- ☑ content-box：该属性值将维持 CSS 2.1 盒模型的组成模式，即元素 width/height=border+padding+content。
- ☑ border-box：该属性值将重新定义 CSS 2.1 盒模型组成模式，即元素 width/height=content。此时浏览器对盒模型的解释与 IE6 解析相同。

IE 怪异模式对于盒模型的解释固然不符合 W3C 的规范，但是这种解析方法也有它的好处：无论如何改动元素边框或者补白大小，都不会使元素总尺寸发生变化，也就不会打乱页面整体布局。而在标准浏览器下，如果要改变一下 border 或者 padding 的值，就不得不重新计算元素的尺寸，从而影响整个页面的布局。虽然 IE 怪异模式不符合标准，但是这种做法还是值得设计师学习的。

目前，Webkit 引擎支持-webkit-box-sizing 私有属性，Mozilla Gecko 引擎支持-moz-box-sizing 私有属性，Presto 引擎和 IE 浏览器直接支持该属性。借助兼容方式，则各主流浏览器对 box-sizing 属性功能的实现分别进行说明。

18.2 调节元素尺寸

为了增强用户体验，CSS3 增加了很多新的属性，其中一个重要的属性就是 resize，它允许用户通过拖动的方式来修改元素的尺寸。到目前为止，主要用于可以使用 overflow 属性的任何容器元素中。

在此之前，设计师要实现相同的 UI 效果，则需要借助 JavaScript 编写大量的脚本才能够实现，这样做既费时费力，且执行效率也很低。

resize 是 CSS3 新增的一个非常实用的属性，该属性的基本语法如下所示。

```
resize:none | both | horizontal | vertical | inherit;
```

resize 属性初始值为 none，适用于所有 overflow 属性不为 visible 的元素。取值简单说明如下。

- ☑ none：浏览器不提供尺寸调整机制，用户不能操纵机制调节元素的尺寸。
- ☑ both：浏览器提供双向尺寸调整机制，允许用户调节元素的宽度和高度。
- ☑ horizontal：浏览器提供单向水平尺寸调整机制，允许用户调节元素的宽度。
- ☑ vertical：浏览器提供单向垂直尺寸的调整机制，允许用户调节元素的高度。
- ☑ inherit：默认继承。

注意：resize 属性属于 UI 模块，详细资料请参阅 http://www.w3.org/TR/css3-ui/。

目前仅有 Safari 和 Chrome 浏览器允许元素的缩放，但尚未完全支持，只允许双向调整。CSS3 允许将该属性应用到任意元素，这将使网页缩放功能拥有跨浏览器的支持。

【示例】在本示例中将演示如何使用 resize 属性设计可以自由调整大小的图片，演示效果如图 18.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
#resize {
    /*以背景方式显示图像，这样可以更轻松地控制缩放操作*/
    background:url(images/img10.jpg) no-repeat center;
    /*设计背景图像仅在内容区域显示，留出补白区域*/
    -moz-background-clip:content;
    -webkit-background-clip:content;
    background-clip:content;
    /*设计元素最小和最大显示尺寸，用户也只能够在该范围内自由调整*/
    width:200px;
    height:120px;
    max-width:800px;
    max-height:600px;
    padding:6px;
    border: 1px solid red;
    /*必须同时定义 overflow 和 resize，否则 resize 属性声明无效，元素默认溢出显示为 visible*/
    resize: both;
    overflow: auto;
}
</style>
</head>
<body>
<div id="resize"></div>
</body>
</html>
```

默认大小

鼠标拖动放大

图 18.1　调节元素尺寸

18.3　设 计 轮 廓

Note

有些时候，设计师可能希望在可视对象（如按钮、活动窗体域、图形地图等）周围添加一圈外部轮廓线，使对象突出显示。当然这里的外轮廓线与元素边框是不同的，外轮廓线不占用网页布局空间，且不一定是矩形，外轮廓线属于动态样式，只有当对象获取焦点或者被激活时呈现。

18.3.1　定义轮廓线

outline 属性可以定义块元素的外轮廓线，该属性在 CSS 2.1 规范中已被明确定义，但是并未得到各主流浏览器的广泛支持，CSS3 增强了该特性。

在元素周围绘制一条轮廓线，可以起到突出元素的作用。例如，可以在原本没有边框的 radio 单选框外围加上一条轮廓线，使其在页面上显得更加突出，也可以在一组 radio 单选框中只对某个单选框加上轮廓线，使其区别于别的单选框。

outline 属性的基本语法如下所示。

```
outline:[outline-color] || [outline-style] || [outline-width] || [outline-offset]| inherit
```

outline 属性初始值根据具体的元素而定，它适用于所有元素。取值简单说明如下。

- ☑ <outline-color>：定义轮廓边框颜色。
- ☑ <outline-style>：定义轮廓边框轮廓。
- ☑ <outline-width>：定义轮廓边框宽度。
- ☑ <outline-offset>：定义轮廓边框偏移位置的数值。
- ☑ inherit：默认继承。

注意：outline 属性创建的外轮廓线是画在一个框“上面”，也就是说，外轮廓线总是在顶上，不会影响该框或任何其他框的尺寸。因此，显示或不显示外轮廓线不会影响文档流，也不会破坏网页布局。

外轮廓线可能是非矩形的。例如，如果元素被分割在好几行，那么外轮廓线就至少是能要包含该元素所有框的外廓。和边框不同的是，外廓在线框的起讫端都不是开放的，它总是完全闭合的。

CSS3 在 outline 属性基础上派生了 4 个外轮廓线属性，简单说明如下。

- ☑ outline-color：定义轮廓边框颜色。outline-color 属性接受所有的颜色，还包括关键字 invert。invert 应该在屏幕上对像素点颜色进行一次反转。这个技巧保证焦点框可见，而不管背景颜色是什么。
- ☑ outline-style：定义轮廓边框轮廓。outline-style 属性与 border-style 属性接受值和用法相同，如 none、dotted、dashed、solid、double、groove、ridge、inset 和 outset。但是，hidden 属性值并不是一个合法的外轮廓样式。
- ☑ outline-width：定义轮廓边框宽度。outline-width 属性与 border-width 属性接受值和用法相同。
- ☑ outline-offset：定义轮廓边框偏移位置的数值。

outline 属性是个复合属性，可以设置 outline-style、outline-width 和 outline-color 属性值。注意，外轮廓线在各边都是一样的，这与边框不同，没有 outline-top 或 outline-left 属性。CSS 不支持定义多个互相覆盖的外轮廓线，也没有定义如何定义因在其他元素之后而有部分不可见的框的外轮廓线。

焦点是文档中用户交互的主题（如输入文本、选择一个按钮等），图形化的用户界面可以使用元素周围的外轮廓线来告诉用户页面上哪个元素获得了焦点。这些外轮廓线是不同于任何边框样式的，切换外轮廓线的显示和不显示都不会使文档流发生变化。浏览器如果支持交互媒介组，则必须跟踪焦点位置，而且必须显示焦点。这个可以通过动态外轮廓线和:focus 伪类的联合使用完成。

Note

【示例】在一个元素获得焦点时在周围画一个粗实线外轮廓线，而在它活动时也画一个不同色的粗实线外轮廓线，从而提高用户交互效果，如图 18.2 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
/*统一页面字体和大小*/
body {
    font-family:"Lucida Grande", "Lucida Sans Unicode", Verdana, Arial, Helvetica, sans-serif;
    font-size:12px;
}
/*清除常用元素的边界、补白、边框默认样式*/
p, h1, form, button {
    border:0;
    margin:0;
    padding:0;
}
/*定义一个强制换行显示类*/
.spacer {
    clear:both;
    height:1px;
}
/*定义表单外框样式*/
.myform {
    margin:0 auto;
    width:400px;
    padding:14px;
}
/*定制当前表单样式*/
#stylized {
    border:solid 2px #b7ddf2;
    background:#ebf4fb;
}
/*设计表单内 div 和 p 通用样式效果*/
#stylized h1 {
    font-size:14px;
    font-weight:bold;
    margin-bottom:8px;
}
#stylized p {
    font-size:11px;
```

Note

```
        color:#666666;
        margin-bottom:20px;
        border-bottom:solid 1px #b7ddf2;
        padding-bottom:10px;
    }
    #stylized label {/*定义表单标签样式*/
        display:block;
        font-weight:bold;
        text-align:right;
        width:140px;
        float:left;
    }
    /*定义小字体样式类*/
    #stylized .small {
        color:#666666;
        display:block;
        font-size:11px;
        font-weight:normal;
        text-align:right;
        width:140px;
    }
    /*统一输入文本框样式*/
    #stylized input {
        float:left;
        font-size:12px;
        padding:4px 2px;
        border:solid 1px #aacfe4;
        width:200px;
        margin:2px 0 20px 10px;
    }
    /*定义图形化按钮样式*/
    #stylized button {
        clear:both;
        margin-left:150px;
        width:125px;
        height:31px;
        background:#666666 url(images/button.png) no-repeat;
        text-align:center;
        line-height:31px;
        color:#FFFFFF;
        font-size:11px;
        font-weight:bold;
    }
    /*设计表单内文本框和按钮在被激活和获取焦点状态下时，外轮廓线的宽、样式和颜色*/
    input:focus, button:focus { outline: thick solid #b7ddf2 }
    input:active, button:active   { outline: thick solid #aaa }
    </style>
    </head>
    <body>
    <div id="stylized" class="myform">
```

```
    <form id="form1" name="form1" method="post" action="">
        <h1>登录</h1>
        <p>请准确填写个人信息...</p>
        <label>Name <span class="small">姓名</span> </label>
        <input type="text" name="textfield" id="textfield" />
        <label>Email <span class="small">电子邮箱</span> </label>
        <input type="text" name="textfield" id="textfield" />
        <label>Password <span class="small">密码</span> </label>
        <input type="text" name="textfield" id="textfield" />
        <button   type="submit">登 录</button>
        <div class="spacer"></div>
    </form>
</div>
</body>
</html>
```

默认状态　　激活状态

图 18.2　设计文本框的轮廓线

18.3.2　定义轮廓线宽度

outline-width 属性可以单独设置外轮廓线的宽度。该属性的基本语法如下所示。

```
outline-width:thin | medium | thick | <length> | inherit;
```

outline-width 属性初始值为 medium，适用于所有元素。取值简单说明如下。

- ☑ thin：定义细轮廓。
- ☑ medium：定义中等轮廓。
- ☑ thick：定义粗轮廓。
- ☑ <length> ：定义轮廓粗细的值。
- ☑ inherit：默认继承。

注意：outline-width 属性用于设置元素整个轮廓的宽度，只有当轮廓样式不是 none 时，该属性才会起作用。如果样式为 none，宽度实际上会重置为 0。不允许设置负长度值。

18.3.3　定义轮廓线样式

outline-style 属性可以设置外轮廓线的样式。该属性的基本语法如下所示。

outline-style:auto | <border-style> | inherit;

outline-style 属性初始值为 none，适用于所有元素。取值简单说明如下。

☑ auto：根据浏览器自动设置。

☑ <border-style>：沿用边框样式。包括 none、dotted、dashed、solid、double、groove、ridge、inset 和 outset。详细说明请参阅 CSS 2.1 中有关 border-style 属性值内容部分。

☑ inherit：默认继承。

该属性的浏览器兼容性与 outline-width 属性相同。

Note

18.3.4　定义轮廓线颜色

outline-color 属性可以单独设置外轮廓线的颜色。该属性的基本语法如下所示。

outline-color:<color> | invert | inherit;

outline-color 属性初始值为 invert，适用于所有元素。取值简单说明如下。

☑ <color>：可以是颜色名，如 red；函数值，如 rgb(255,0,0)；或者十六进制值，如#ff0000。

☑ invert：执行颜色反转（逆向的颜色）。这样可以确保外轮廓线在不同的背景颜色中都是可见的。

☑ inherit：默认继承。

注意：轮廓的样式不能是 none，否则轮廓不会出现。该属性的浏览器兼容性与 outline-width 属性相同。

18.3.5　定义轮廓线位移

outline-offset 属性可以单独设置外轮廓线的偏移位置。该属性的基本语法如下所示。

outline-offset:<length> | inherit;

outline-offset 属性初始值为 0，适用于所有元素。取值简单说明如下。

☑ <length>：定义轮廓距离容器的值。

☑ inherit：默认继承。

该属性的浏览器兼容性与 outline-width 属性相同。

【示例】在 18.3.1 节示例的基础上，通过 outline-offset 属性放大外轮廓线，使其看起来更大方，演示效果如图 18.3 所示。

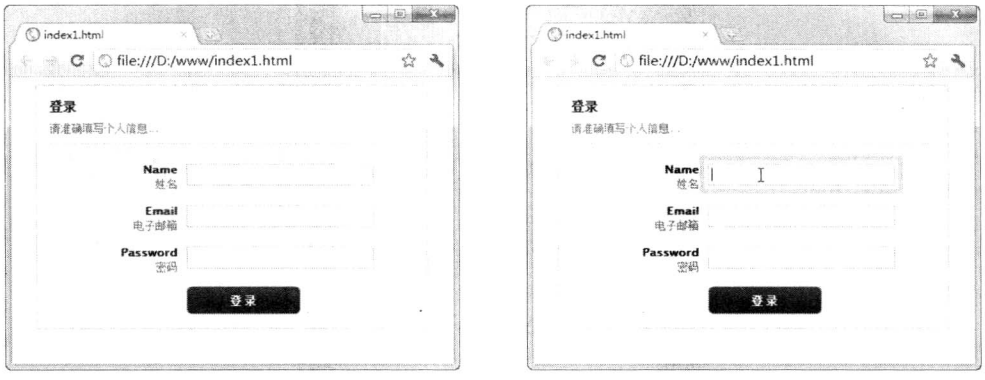
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
/*统一页面字体和大小*/
body {
    font-family:"Lucida Grande", "Lucida Sans Unicode", Verdana, Arial, Helvetica, sans-serif;
    font-size:12px;
}
```

Note

```
/*清除常用元素的边界、补白、边框默认样式*/
p, h1, form, button {
    border:0;
    margin:0;
    padding:0;
}
/*定义一个强制换行显示类*/
.spacer {
    clear:both;
    height:1px;
}
/*定义表单外框样式*/
.myform {
    margin:0 auto;
    width:400px;
    padding:14px;
}
/*定制当前表单样式*/
#stylized {
    border:solid 2px #b7ddf2;
    background:#ebf4fb;
}
/*设计表单内 div 和 p 通用样式效果*/
#stylized h1 {
    font-size:14px;
    font-weight:bold;
    margin-bottom:8px;
}
#stylized p {
    font-size:11px;
    color:#666666;
    margin-bottom:20px;
    border-bottom:solid 1px #b7ddf2;
    padding-bottom:10px;
}
#stylized label {/*定义表单标签样式*/
    display:block;
    font-weight:bold;
    text-align:right;
    width:140px;
    float:left;
}
/*定义小字体样式类*/
#stylized .small {
    color:#666666;
    display:block;
    font-size:11px;
    font-weight:normal;
    text-align:right;
    width:140px;
```

Note

```
}
/*统一输入文本框样式*/
#stylized input {
    float:left;
    font-size:12px;
    padding:4px 2px;
    border:solid 1px #aacfe4;
    width:200px;
    margin:2px 0 20px 10px;
}
/*定义图形化按钮样式*/
#stylized button {
    clear:both;
    margin-left:150px;
    width:125px;
    height:31px;
    background:#666666 url(images/button.png) no-repeat;
    text-align:center;
    line-height:31px;
    color:#FFFFFF;
    font-size:11px;
    font-weight:bold;
}
/*设计表单内文本框和按钮在被激活和获取焦点状态下时，外轮廓线的宽、样式和颜色*/
input:focus, button:focus { outline: thick solid #b7ddf2 }
input:active, button:active { outline: thick solid #aaa }
/*通过 outlineoffset 属性放大外轮廓线*/
input:active, button:active { outline-offset: 4px; }
input:focus, button:focus { outline-offset: 4px; }
</style>
</head>
<body>
<div id="stylized" class="myform">
    <form id="form1" name="form1" method="post" action="">
        <h1>登录</h1>
        <p>请准确填写个人信息...</p>
        <label>Name <span class="small">姓名</span> </label>
        <input type="text" name="textfield" id="textfield" />
        <label>Email <span class="small">电子邮箱</span> </label>
        <input type="text" name="textfield" id="textfield" />
        <label>Password <span class="small">密码</span> </label>
        <input type="text" name="textfield" id="textfield" />
        <button   type="submit">登  录</button>
        <div class="spacer"></div>
    </form>
</div>
</body>
</html>
```



默认状态　　激活状态

图 18.3　放大激活和焦点提示框

18.4　设 计 导 航

HTML 为表单标签定义 tabindex 属性，这个属性为当前元素指定了其在当前文档中导航的序列号。导航的序列号可以修改页面中元素通过键盘操作获得焦点的顺序。该属性可以存在于嵌套的页面元素当中。

18.4.1　定义导航顺序

CSS3 新增加了 nav-index 属性，用来代替标签属性 tabindex。为了使浏览器能按顺序获取焦点，页面元素需要遵循如下规则：

- ☑ 该元素支持 nav-index 属性，而被赋予正整数属性值的元素将会被优先导航。浏览器将按照 nav-index 属性值从小到大的顺序进行导航。属性值无须按次序，也无须以特定的值开始。拥有同一 nav-index 属性值的元素将以它们在文档流中出现的顺序进行导航。
- ☑ 对于那些不支持 nav-index 属性，或者 nav-index 属性值为 auto 的元素，将以它们在文档流中出现的顺序进行导航。
- ☑ 对那些禁用的元素，将不参与导航的排序。

导航和激活页面元素的快捷键依赖于浏览器的设置，例如，通常 Tab 键用于按顺序导航，而 Enter 键则用于激活选中的元素。浏览器一般也会定义反向导航的快捷键。当通过 Tab 键导航到序列的结束或者开始时，浏览器可能会循环到导航序列的开始或者结束。按 Shift+Tab 组合键通常用于反向导航。

nav-index 属性的基本语法如下所示。

```
nav-index: auto | <number> | inherit;
```

nav-index 属性初始值为 auto，适用于所有可用元素。取值简单说明如下。

- ☑ auto：浏览器默认的顺序。
- ☑ <number>：必须是正整数，该数字指定了元素的导航顺序。1 意味着最先被导航。当若干个元素的 nav-index 值相同时，则按照文档的先后顺序进行导航。
- ☑ inherit：默认继承。

【示例】调整表单输入框的键盘激活顺序。对于下面这个表单示例来说，传统做法是使用 tabindex

属性来改变表单输的键盘激活和响应顺序，代码如下，当在浏览器中预览输入时，按 Tab 键，会从“姓名”文本框直接跳转到“省/市”文本框，演示效果如图 18.4 所示。

Note

```
<!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=gb2312" />
<title></title>
</head>
<body
<div id="leftSide">
    <fieldset>
        <legend>个人信息登记表</legend>
        <form action="" method="POST" class="form">
            <label for="name">姓名</label>
            <div class="div_texbox">
                <input name="name" type="text" class="textbox" tabindex="1" id="name" value="" />
            </div>
            <label for="address">地址</label>
            <div class="div_texbox">
                <input name="address" type="text" class="textbox" tabindex="4" id="address" value="" />
            </div>
            <label for="city">区/县</label>
            <div class="div_texbox">
                <input name="city" type="text" class="textbox" tabindex="3" id="city" value="" />
            </div>
            <label for="country">省/市</label>
            <div class="div_texbox">
                <input name="country" type="text" class="textbox"   tabindex="2" id="country" value="" />
            </div>
            <div class="button_div">
                <input name="Submit" type="button" value="提交" tabindex="5" class="buttons" />
            </div>
        </form>
    </fieldset>
</div>
</body>
</html>
```

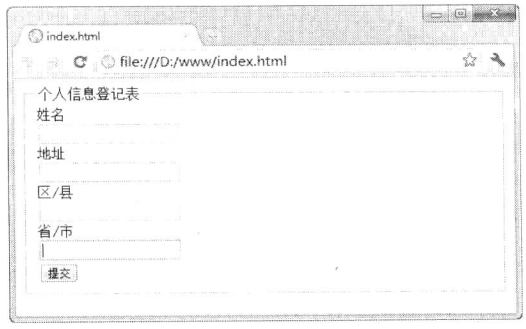


图 18.4　使用 tabindex 标签属性调整表单域的键盘焦点顺序

现在使用 nav-index 则直接在 CSS 样式表中调整表单域的键盘影响顺序，其中样式表代码如下，演示效果如图 18.5 所示。

```
<style type="text/css">
/*统一默认样式*/
body { background-color:#F7F7F7; }
fieldset {border:1px dashed #CCC; padding:10px;}
legend { font-family:Arial, Helvetica, sans-serif; color:#fff; background: #666; border: 1px
solid #333; padding: 2px 6px;}
label { width:142px; height:32px; margin:3px 2px 0 0; padding:11px 0 0 6px; float:left;
color:#666; text-align:right; }
/*清除表单默认样式*/
.form {margin:0; padding:0;}
/*定制表单外框样式*/
#leftSide { width:530px; padding-top:30px; float:left;}
/*定义表单域区块样式*/
.div_texbox { width:347px; float:right; background-color:#E6E6E6; height:35px; margintop:
3px; padding-top:5px; padding-bottom:3px; padding-left:5px;}
.textbox {background-image: url(images/16t.gif); background-repeat: no-repeat; backgroundposition:
left; width:285px; font:normal 18px Arial; color: #999999; padding:3px 5px 3px
19px;}
/*定义表单域获取焦点和鼠标指针经过时的样式*/
.username:focus, .username:hover { background-color:#F0FFE6; }
/*定义按钮样式*/
.button_div { width:287px; float:right; text-align:right; height:35px; margin-top:3px;
padding:5px 32px 3px;}
.buttons { padding:6px 14px; border:2px solid; border-color: #fff #d8d8d0 #d8d8d0 #fff;
background: #e3e3db; color: #989070; font-weight:bold;}
/*定义表单域的键盘焦点顺序*/
input.nav1 { nav-index:1; }
input.nav2 { nav-index:2; }
input.nav3 { nav-index:3; }
input.nav4 { nav-index:4; }
input.nav5 { nav-index:5; }
</style>
```

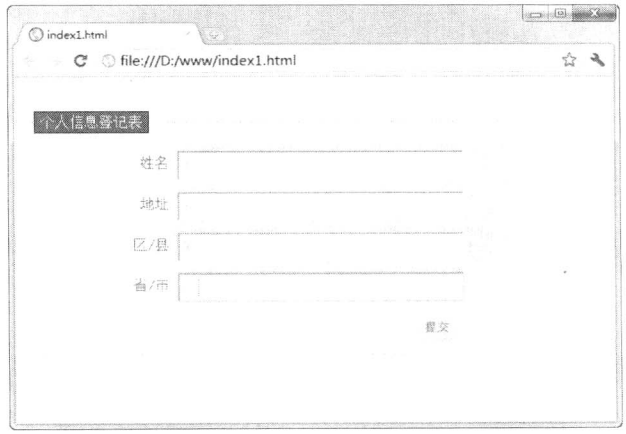


图 18.5　使 nav-index 样式属性调整表单域的键盘焦点顺序

18.4.2　定义方向键控制顺序

Note

输入设备默认 4 个方向键将根据文档流顺序依次控制元素的焦点切换，但为了有更好的用户体验，CSS3 定义了切换焦点的控制顺序方向键特性。它主要由 4 个属性来配合实现：

☑　nav-up：控制向上方向键。

☑　nav-right：控制向右方向键。

☑　nav-down：控制向下方向键。

☑　nav-left：控制向左方向键。

这些属性的基本语法如下所示。

```
nav-up:auto | <id> [ current | root | <target-name> ]? | inherit;
nav- right:auto | <id> [ current | root | <target-name> ]? | inherit;
nav- down:auto | <id> [ current | root | <target-name> ]? | inherit;
nav- left:auto | <id> [ current | root | <target-name> ]? | inherit;
```

这些属性的初始值为 auto，适用于所有可用元素。取值简单说明如下。

☑　auto：根据浏览器默认的顺序。

☑　<id>：要切换元素的 id 命名。

☑　root | <target-name>：该参数不能以"_"命名，指出 frameset 目标页面之间的元素焦点切换。如果指定的目标页面不存在，则被视为当前页面的焦点，意味着完全依赖框架页。该属性是以关键节点"root"标示，浏览器将把整个 frameset 框架页定为目标。

☑　inherit：继承。

【示例】本示例介绍了如何正确定义键盘控制键顺序，了解这些键盘方向键样式控制的一般方法。在浏览器中预览，由于页面比较大，当窗口显示滚动条时，如果要快速找到分别位于顶部、底部、左侧和右侧的按钮，则直接按方向键即可，当按不同的方向键时，则会自动切换位置，显示对应按钮，演示效果如图 18.6 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
body { background:url(images/img9.jpg) center no-repeat; }
button { position:absolute; }
button#b1 {/*顶部按钮*/
    top:0;
    left:50%;
    /*按向左或向上方向键，移动焦点到按钮 4 上，按向右或向下方向键，移动焦点到按钮 2 上*/
    nav-index:1;
    nav-right:#b2;
    nav-left:#b4;
    nav-down:#b2;
    nav-up:#b4;
}
button#b2 {/*右侧按钮*/
    top:50%;
```

Note

```
        left:100%;
        /*按向左或向上方向键，移动焦点到按钮 1 上，按向右或向下方向键，移动焦点到按钮 3 上*/
        nav-index:2;
        nav-right:#b3;
        nav-left:#b1;
        nav-down:#b3;
        nav-up:#b1;
    }
    button#b3 {/*底部按钮*/
        top:100%;
        left:50%;
        /*按向左或向上方向键，移动焦点到按钮 2 上，按向右或向下方向键，移动焦点到按钮 4 上*/
        nav-index:3;
        nav-right:#b4;
        nav-left:#b2;
        nav-down:#b4;
        nav-up:#b2;
    }
    button#b4 {/*左侧按钮*/
        top:50%;
        left:0;
        /*按向左或向上方向键，移动焦点到按钮 3 上，按向右或向下方向键，移动焦点到按钮 1 上*/
        nav-index:4;
        nav-right:#b1;
        nav-left:#b3;
        nav-down:#b1;
        nav-up:#b3;
    }
    </style>
    </head>
    <body>
    <div>
        <button id="b1">顶部</button>
        <button id="b2">右侧</button>
        <button id="b3">底部</button>
        <button id="b4">左侧</button>
    </div>
    </body>
    </html>
```

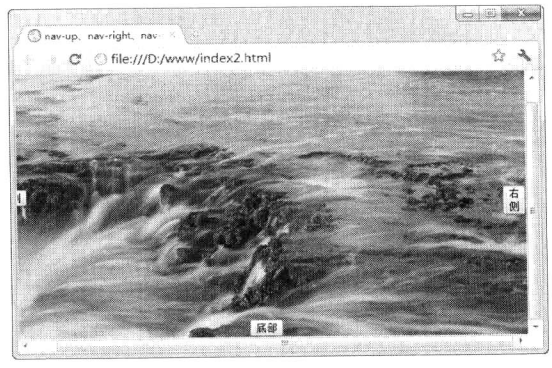


图 18.6　键盘控制键演示效果

18.5　添加显示内容

Note

content 属性属于内容生成和替换模块（http://www.w3.org/TR/css3-content/），该属性能够为指定元素添加内容。实际上内容生成和替换行为已经超越了 CSS 样式表的核心功能，这部分功能替代了原需 JavaScript 脚本来实现的角色任务。不过 content 属性比较实用，它能够满足样式设计中临时添加非结构性的样式服务标签，或者添加补充说明性内容等。

content 属性的基本语法如下所示。

```
content: normal | string | attr() | uri() | counter() | none;
```

content 属性初始值为 normal，适用于所有可用元素。取值简单说明如下。

- ☑ normal：默认值。
- ☑ string：插入文本内容。
- ☑ attr()：插入元素的属性值。
- ☑ uri()：插入一个外部资源，如图像、音频、视频或浏览器支持的其他任何资源。
- ☑ counter()：计数器，用于插入排序标识。
- ☑ none：无任何内容。

【示例 1】content 属性的简单用法就是在文档中添加内容，本示例为 div 元素添加一行提示文本，提示当前浏览器是否支持 content 特性，演示效果如图 18.7 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
p { text-align:center; color:red; }
/*在 p 元素内容尾部添加文字*/
p:after { content:"支持"; }
</style>
</head>
<body>
<p>当前浏览器是否支持 content 特性？<br /></p>
</body>
</html>
```

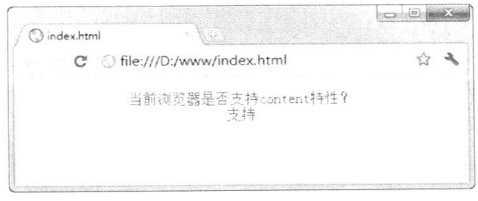


图 18.7　content 特性检测效果

【示例 2】下面使用 content 属性为页面对象添加外部图像，演示效果如图 18.8 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
div {
    padding: 50px;
    border: solid 1px red;
    content: url(images/xinzhi.png); /*在 div 元素内添加图片*/
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

Note

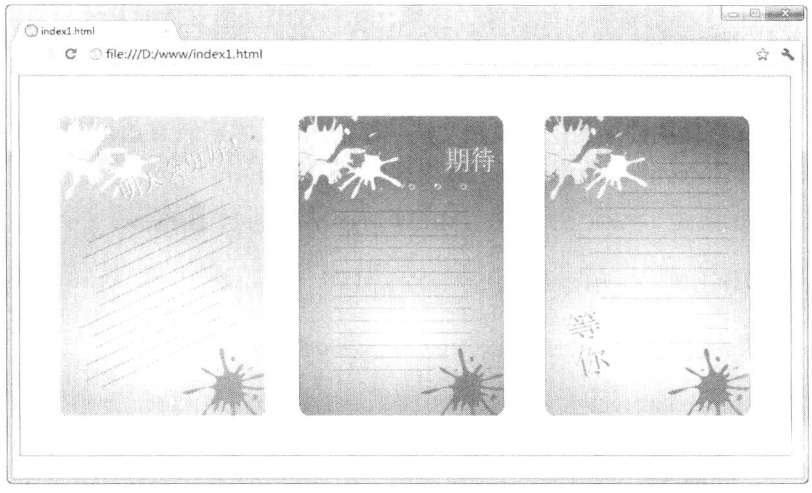


图 18.8　使用 content 属性在当前元素内插入图像演示效果

18.6　恢复默认样式

在 CSS3 中，可以利用 initial 属性值取消对元素的样式指定。目前有 Firefox、Safari 和 Chrome 等浏览器对 initial 属性值提供支持。

18.6.1　取消元素样式

要取消元素已定义的样式，可以通过多种方法来达到这个目的，其中最简单的方法是直接在样式表中删除设定该样式的代码。但是，在大多数情况下，一个样式写好以后会对很多页面中的元素指定这个样式。所以，如果对单个元素取消其样式指定时，这种做法是不可取的。

第二种方法是目前采用的使用 class 的方法，要取消对单个元素的样式指定，只要把这个元素的

class 属性取消就可以了，但是 class 属性本身是一个多余的、没有任何语义的属性。同时，如果多个元素使用同一个样式，还必须为每一个元素增加同样的 class 属性。如果要删除一个样式，还要逐个删除这些元素的 class 属性，所以很不实用。

在 CSS3 中已经不推荐使用它，取而代之的是将样式与元素或元素标记直接绑定的做法。所以，第二种方法在下一代 Web 平台中使用的机会也会越来越少，直到最终随着 class 属性一起被废弃掉。

Note

针对这种情况，CSS3 中新增了一个 initial 属性值，使用这个 initial 属性值可以直接取消对某个元素的样式指定。

【示例】在本示例中，页面中有 3 个 p 元素，然后在内部样式表中定义这些 p 元素的样式。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title></title>
</head>
<style type="text/css">
p{
    color:blue;
    font-family:宋体;
}
</style>
<body>
<p id="text01">有时，爱也是种伤害。残忍的人，选择伤害别人，善良的人，选择伤害自己。</p>
<p id="text02">有些事，我们明知道是错的，也要去坚持，因为不甘心；有些人，我们明知道是爱的，也要去放弃，因为没结局；有时候，我们明知道没路了，却还在前行，因为习惯了。</p>
<p id="text03">以为蒙上了眼睛，就可以看不见这个世界；以为捂住了耳朵，就可以听不到所有的烦恼；以为脚步停了下来，心就可以不再远行；以为我需要的爱情，只是一个拥抱。</p>
</body>
</html>
```

在浏览器中预览，显示效果如图 18.9 所示。

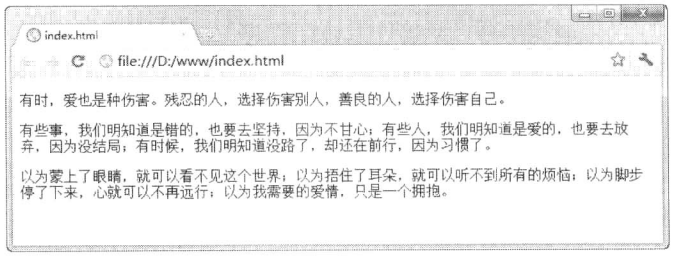


图 18.9　定义段落文本样式

3 个 p 元素的文字颜色都是蓝色，字体都是宋体。这时如果禁止<p id="text02">使用已定义的段落样式，只需在样式代码中为这个元素单独添加一个样式，然后把文字颜色的值设为 initial 值就可以了，具体代码如下所示。

```
<style type="text/css">
p {
    color: blue;
```

```
        font-family: 宋体;
    }
    p#text02 {
        color: initial;
        color: -moz-initial;
        font-family: 宋体;
    }
    </style>
```

把上面这段代码替换到示例样式代码中，然后运行该示例，运行结果如图 18.10 所示。

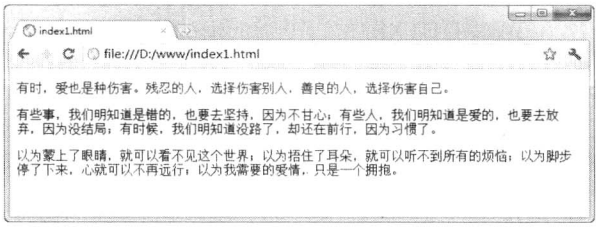


图 18.10　恢复段落文本样式

initial 属性值的作用是让各种属性使用默认值，在浏览器中文字颜色的默认值是黑色，所以第二段文本的字体颜色显示为黑色。

18.6.2　慎用 initial 的情况

在特殊情况下，对元素使用 initial 属性值后的显示效果并不等于将该元素的样式设定直接删除后的结果。

【示例】本示例显示了在默认情况下，一级标题和段落文本显示的效果是不同的，其中一级标题加粗、字号增大显示，如图 18.11 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title></title>
</head>
<body>
<h1>有时，爱也是种伤害。残忍的人，选择伤害别人，善良的人，选择伤害自己。</h1>
<p>有时，爱也是种伤害。残忍的人，选择伤害别人，善良的人，选择伤害自己。</p>
</body>
</html>
```

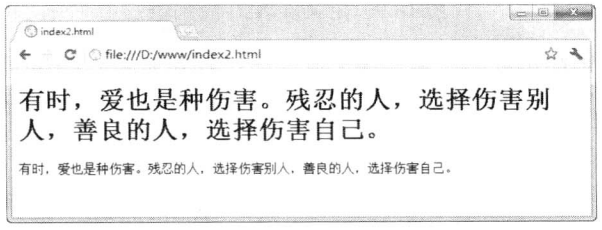


图 18.11　标题和文本默认样式

在浏览器中，为了使一些元素变得更容易阅读，浏览器可以自行对该元素使用一些样式。例如，对于 hl 元素来说，浏览器使用了如下所示的样式：

```
<style type="text/css">
h1 {
    font-size: 2em;
    font-weight: bold;
}
</style>
```

Note

可以在样式中对 hl 元素重新定义，然后在浏览器中预览，则显示效果如图 18.12 所示。

```
<style type="text/css">
h1 {
    font-size: 3em;
    font-weight:normal;
}
</style>
```

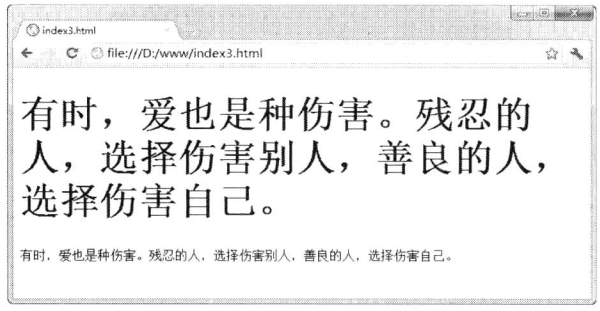


图 18.12　定义标题样式

接着，在这段样式后面追加一段 hl 元素使用的样式，对上面文字的字号和字体粗细均使用 initial 属性值。追加后 hl 元素的样式代码如下，然后在浏览器中预览，则显示效果如图 18.13 所示。

```
<style type="text/css">
h1 {
    font-size: 3em;
    font-weight:normal;
}
h1 {
    font-size: initial;
    font-size: -moz-initial;
    font-weight: initial;
    font-weight: -moz-initial;
}
</style>
```

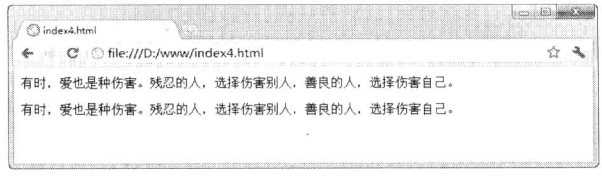


图 18.13　恢复标题默认样式

这个显示效果与不使用任何样式设定时的显示效果并不相同。因为追加了 initial 属性值的样式设定后，hl 元素的字号和字体粗细均使用 CSS 中对字号和字体粗细属性设定的默认值，并不考虑浏览器对 hl 元素定义了什么样式。而在 CSS 中，字号的默认值为 medium，字体粗细的默认值为 normal，与浏览器对 hl 元素使用的样式并不一致，如果要想让 hl 元素的字号和字体粗细的默认值使用浏览器的默认值，就不应该使用 initial 属性值，而是使用浏览器追加的默认样式中的属性值。在 http://www.w3.org/TR/CSS 21/sample.html 网页中可以查找到浏览器对 HTML4 中元素所定义的默认样式清单，目前各主流浏览器均遵照这个清单来定义元素的默认样式。

Note

第19章

CSS3 其他新特性

（视频讲解：40 分钟）

CSS3 以模块化进行开发，模块之间相互独立，大量新增的功能还在不断研发中，在 W3C 官方网站（http://www.w3.org/Style/CSS/）中可以了解各个开发项目的进度，新的工作草案不断被推出，新的模块不断被列上日程表。即便是 W3C 发布了各种标准草案，能够获得各主流浏览器的支持，也需要一定的时间。本章将对 CSS3 中的一些内容比较少，但又非常重要的样式和属性进行介绍。感兴趣的读者可以访问 W3C 官方网站查询更专业的资讯，其他细节限于篇幅就不再详细讲解。

19.1 溢 出 处 理

Note

overflow 是 CSS 2.1 规范中的特性，而 overflow-x 和 overflow-y 属性则是 CSS3 基础盒模型（http://www.w3.org/TR/css3-box/）草案中新加入的特性。overflow 属性定义当一个块级元素的内容溢出了元素的框（它作为内容的包含块）时是否剪切显示。overflow-x 属性定义了对左右边（水平方向）的剪切，而 overflow-y 属性定义了对上下边（垂直方向）的剪切。

overflow-x 和 overflow-y 属性的基本语法如下所示。

```
overflow-x:visible | hidden | scroll | auto | no-display | no-content;
overflow-y:visible | hidden | scroll | auto | no-display | no-content;
```

overflow-x 和 overflow-y 属性的初始值为 visible，适用于非替换的块元素，或者非替换的行内块元素。取值简单说明如下。

- ☑ visible：不剪切内容，也不添加滚动条。该属性值为默认值，元素将被剪切为包含对象的窗口大小，且 clip 属性设置将失效。
- ☑ auto：在需要时剪切内容并添加滚动条。该属性为 body 和 textarea 元素的默认值。
- ☑ hidden：不显示超出元素尺寸的内容。
- ☑ scroll：当内容超出元素尺寸时，overflow-x 显示为横向滚动条，而 overflow-y 显示为纵向滚动条。
- ☑ no-display：当内容超出元素尺寸时，则不显示元素，此时类似添加了 display:none 声明。该属性值是最新添加的，仅作为交流。
- ☑ no-content：当内容超出元素尺寸时，则不显示内容，此时类似添加了 visibility:hidden 声明。该属性值是最新添加的，仅作为交流。

目前，所有浏览器都能够正确解析该属性。但是部分浏览器在解析时，会存在一些细节差异。

根据 CSS3 基础盒模型草案规范，overflow-x 和 overflow-y 的计算值与所设置的值应该相等，除非这一对值不合理。如果其中一个属性值被设置成了 scroll 或 auto，而另一个属性值为 visible，那么 visible 会被设置成 auto。如果 overflow-x 和 overflow-y 属性值相同，则 overflow 的计算值与前两者的指定值相同，否则，它的值是一个 overflow-x 和 overflow-y 的计算值对。关于 overflow 的详细资料，请参考 CSS 2.1 规范（http://www.w3.org/TR/CSS2/visufx.html#overflow），关于 overflow-x 和 overflow-y 的详细资料，请参考 CSS3 基础盒模型草案（http://www.w3.org/TR/css3-box/#overflow）。

当 overflow-x 或 overflow-y 属性值为 hidden，另一个属性值为 visible 时，该元素最终渲染使用的 overflow-y 或 overflow-x 属性值不同。IE 浏览器使用 hidden，其他浏览器使用 auto。造成的影响可能是页面内容显示不完全，或在不同浏览器下最终显示效果不一致。

CSS3 草案中并没有说明，当 overflow-x 和 overflow-y 中的一个属性值为 hidden，另一个属性值为 visible 时，该 visible 值应该设置为什么，各浏览器有不同的实现方法。

【示例】本示例演示了当为 overflow-x 和 overflow-y 设置不同值后的效果，如图 19.1 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Note

```
<title></title>
<style type="text/css">
#cont1 div, #cont3 div, #cont5 div {
      width:300px;
      height:200px;
}
#cont2 div, #cont4 div, #cont6 div {
      width:100px;
      height:50px;
}
.cont {
      float:left;
      margin:4px;
      overflow-y:visible;
      padding:10px;
      width:200px;
      height:100px;
}
.cont, .cont div { border : solid 2px red; }
</style>
</head>
<body>
<div id="cont1" class="cont" style="overflow-x:scroll; ">
      <div>style="overflow-x:scroll; "</div>
</div>
<div id="cont2" class="cont" style="overflow-x:scroll; ">
      <div>style="overflow-x:scroll; "</div>
</div>
<div id="cont3" class="cont" style="overflow-x:auto; ">
      <div>style="overflow-x:auto; "</div>
</div>
<div id="cont4" class="cont" style="overflow-x:auto; ">
      <div>style="overflow-x:auto; "</div>
</div>
<div id="cont5" class="cont" style="overflow-x:hidden; ">
      <div>style="overflow-x:hidden; "</div>
</div>
<div id="cont6" class="cont" style="overflow-x:hidden; ">
      <div>style="overflow-x:hidden; "</div>
</div></body>
</html>
```

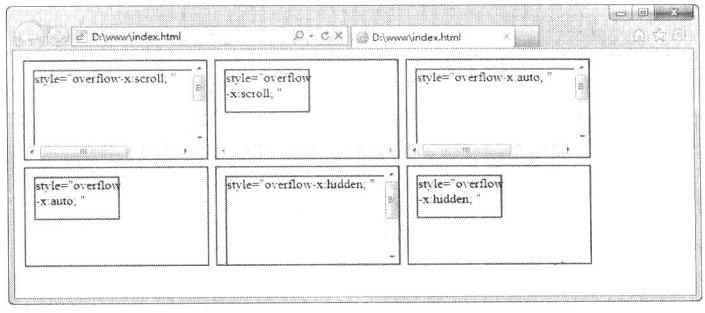


图 19.1　设置 overflow 演示效果

对于 overflow-x 和 overflow-y 的组合渲染，所有浏览器均依照规范处理。但是当 overflow-x:hidden 且 overflow-y:visible 时，IE9 及其以下版本浏览器将 overflow-y 渲染为 hidden，其他浏览器则渲染为 auto。也就是说，在 IE 浏览器中所有容器的 overflow-y 计算值都为 visible，而其他浏览器中其值却为 auto。

Note

要避免不同浏览器在解析上的差异，在使用时应该同时设置 overflow-x 和 overflow-y 的属性值，不要出现其中一个值为 hidden，而另一个值为 visible 的情况。另外，还要避免编写依赖指定值为 visible 的 overflow-x 和 overflow-y 属性的计算值的代码。

19.2 自定义字体类型

CSS3 允许用户自定义字体类型，通过@font-face 能够加载服务器端的字体文件，让客户端浏览器显示客户端所没有安装的字体。@font-face 规则在 CSS3 规范中属于字体模块（http://www.w3org/TR/css3-fonts/#font-face）。在传统网页设计中，如果要使用各种艺术字体类型，必须考虑每位浏览者的系统中是否安装了所用的字体。

19.2.1 使用@font-face 规则

使用@font-face 规则，可以不用担心用户系统字体库。只要在互联网上指定一种字体类型源，然后不管用户的计算机是否安装了该字体，设计的网页都能够正确显示。如果没有@font-face 规则，浏览器只能够在客户端系统中寻找指定字体，这就给网页设计带来很多限制，妨碍了设计师的创意设计，也无法展现丰富多彩的字体艺术。

@font-face 规则的语法格式如下：

```
@font-face { <font-description> }
```

@font-face 规则的选择符是固定的，用来引用服务器端的字体文件。

<font-description>是一个属性名值对，格式样式如下：

```
descriptor: value;
descriptor: value;
descriptor: value;
descriptor: value;
[...]
descriptor: value;
```

属性及其取值说明如下。

- ☑ font-family：设置文本的字体名称。
- ☑ font-style：设置文本样式。
- ☑ font-variant：设置文本是否大小写。
- ☑ font-weight：设置文本的粗细。
- ☑ font-stretch：设置文本是否横向拉伸变形。
- ☑ font-size：设置文本字体大小。
- ☑ src：设置自定义字体的相对路径或者绝对路径。注意，该属性只能在@font-face 规则里使用。事实上，IE5 已经开始支持该属性，但是只支持微软自有的.eot（Embedded Open Type）字体格式，

而其他浏览器直到现在都不支持这一字体格式。不过，从 Safari 3.1 开始，用户可以设置.ttf（TrueType）和.otf（OpenType）两种字体作为自定义字体了。考虑到浏览器的兼容性，在使用时建议同时定义.eot和.ttf，以便能够兼容所有主流浏览器。

Note

【示例】下面是一个简单的示例，帮助读者学会使用@font-face 规则。示例代码如下，演示效果如图 19.2 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
/*引入外部字体文件*/
@font-face {
    /*选择默认的字体类型*/
    font-family: "lexograph";
    /*兼容 IE*/
    src: url(http://randsco.com//fonts/lexograph.eot);
    /*兼容非 IE*/
    src: local("Lexographer"), url(http://randsco.com/fonts/lexograph.ttf) format("truetype");
}
h1 {
    /*设置引入字体文件中的 lexograph 字体类型*/
    font-family: lexograph, verdana, sans-serif;
    font-size:4em;
}
</style>
</head>
<h1>Cascading Style Sheet</h1>
<body>
</body>
</html>
```

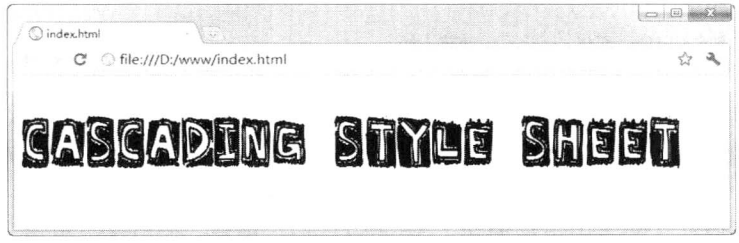


图 19.2　设置为 lexograph 字体类型的文字

注意，嵌入外部字体的做法对于中文网站来说不大适用，因为一个中文字体文件少的有几 MB，大的有十几 MB，这么大的字体文件下载过程是无法忍受的，同时服务器也不能忍受如此频繁的申请下载。所以对中文来说，如果只是想标题使用特殊字体，最好设计成图片。由于英文字体文件只有几十 KB，与一张图片的大小差不多，如果有大量的文字需要使用该字体，那么在存储、带宽这方面就合适得多了。

19.2.2 开放字体

Note

@font-face 规则存在诸多问题，催生了 Web 开发字体格式。Web 开放字体格式由 Mozilla 发起，旨在让设计者在 Web 中使用自己的个性字体。Web 开放字体格式允许设计者像链接图片那样链接自己的字体文件，并使用该字体显示文字。Web 开放字体格式是为了解决 CSS3 的@font-face 存在的功能问题，@font-face 允许浏览器下载指定的字体，但并没有指明字体的格式，使用@font-face 可能会在不同浏览器产生不同的效果。著名博客站点 Boing Boing 就曾使用@font-face 规则对站点进行重建，结果以失败告终。

Web 开放字体格式的另一个好处是它本身是经过压缩的，这样就避免了下载巨大字体文件的问题，Web 开放字体格式目前已经部署到 Firefox 3.6，Opera 和微软都表示将提供支持，该格式目前已经提交到 W3C，以希望其成为标准。不过，微软的 IE9 还不支持该格式，但微软是该标准的提议者之一，IE 产品博客中也对该格式有所提及，因此，我们可以期待 IE9 的最终发布时会包含对该功能的支持。Chrome 也支持 Web 开放字体格式，目前仅有苹果的 Safari 未表明支持该技术的浏览器。

除了浏览器的支持，另一个重要的问题是字体设计商的支持，因为 Web 开放字体格式并不安全，字体商有可能不愿对之提供支持，不过包括 Adobe、House Industries、Hoefler Frere-Jones，以及 ITC、LinoType 在内的字体商都表示支持该格式。

尽管 Web 开放字体格式并不能解决 Web 字体的所有问题，但是它朝着正确的方向迈进了一大步，随着 Firefox、Opera 和 Chrome 等浏览器对该技术的支持，设计者们有望很快在网页设计中任意使用自己喜欢的字体。关于 Web 安全字体的更多资料，可以参考 Mozilla Hack 博客。

19.3 定义设备类型

在 CSS 中，与媒体相关的样式定义是从 CSS 2.1 开始的。CSS 2.1 中定义了各种媒体类型，包括显示器、便携设备、电视机等。CSS3 中加入 Media Queries 模块，该模块中允许添加媒体查询（media query）表达式，用以指定媒体类型，然后根据媒体类型来选择应该使用的样式，允许在不改变内容的情况下在样式中选择一种页面的布局以精确地适应不同的设备，从而改善用户体验。

19.3.1 认识 Media Queries 模块

在不同的设备中，浏览器的窗口尺寸可能是不同的。如果只针对某种窗口尺寸来制作网页，在其他设备中呈现该网页时就会产生很多问题。如果针对不同的窗口尺寸制作不同的网页，则要制作的网页就会太多。

为了解决这个问题，CSS3 中单独增加了 Media Queries 模块，使用这个模块，网页制作者只需要针对不同的浏览器窗口尺寸来编写不同的样式，然后让浏览器根据不同的窗口尺寸来选择使用不同的样式即可。

到目前为止，Media Queries 模块得到了 Firefox 浏览器、Safari 浏览器、Chrome 浏览器以及 Opera 浏览器的支持。

CSS 2.1 支持定义设备类型，允许为不同样式表文件设置不同的设备类型，如打印样式表文件、手机样式表文件、计算机样式表文件等。

☑ 设置外部样式表文件的设备类型。

```
<link href="csss.css" rel="stylesheet" type="text/css" media="handheld" />
```

☑ 设置内部样式表文件的设备类型。

```
<style type="text/css" media="screen">
...
</style>
```

Note

W3C 于 2010 年 7 月候选推荐了 Media Queries 标准块，详细信息请参阅 http://www.w3.org/TR/css3-mediaqueries/网页内容。媒体查询比 CSS2 的 Media Type（媒体类型）更加实用，它可以帮助用户获取以下数据。

☑ 浏览器窗口的宽和高。
☑ 设备的宽和高。
☑ 设备的手持方向，横向还是竖向。
☑ 分辨率。

Media Queries 允许添加表达式用以确定媒体的情况，以此来应用不同的样式表。它允许在不改变内容的情况下，改变页面的布局以精确适应不同的设备，以此加强体验，这种设计灵感来源于 IE 的条件语句，不过 Media Queries 的功能更加强大。

```
<!--[if IE]>
CSS 样式、JavaScript 脚本、HTML 结构
<![endif]-->
```

如果用户有一个支持 Media Queries 的设备，就可以为该设备编写专门的 CSS，让网站适应这个设备的屏幕显示。所以 Media Queries 和 CSS 优化没有关系，甚至是矛盾的。

19.3.2　认识@media 规则

@media 规则的语法格式如下：

```
@media: <sMedia> { sRules }
```

参数简单说明如下。

☑ <sMedia>：指定设备名称。CSS 设备类型包括以下这些。
  - ➢ all：用于所有设备类型。
  - ➢ aural：用于语音和音乐合成器。
  - ➢ braille：用于触觉反馈设备。
  - ➢ embossed：用于凸点字符（盲文）印刷设备。
  - ➢ handheld：用于小型或手提设备。
  - ➢ print：用于打印机。
  - ➢ projection：用于投影图像，如幻灯片。
  - ➢ screen：用于计算机显示器。
  - ➢ tty：用于使用固定间距字符格的设备，如电传打字机和终端。
  - ➢ tv：用于电视类设备。

☑ {sRules}：定义样式表。

Media Queries 的使用方法可以简化成下面形式：

```
@media 设备类型 and (设备特性) {样式代码}
```

在代码的开头必须要书写@media，然后指定设备类型，也可以称之为媒体类型。CSS 2.1 中定义了 10 种设备类型，此处可以指定的值与该值所代表的设备类型如上面的参数说明所示。

设备特性的书写方式与样式的书写方式很相似，分为两个部分，当中由冒号分割，冒号前书写设备的某种特性，冒号后书写该特性的具体值。例如，如果需要指定浏览器的窗口宽度大于 400 像素时所使用的样式，书写方法如下所示：

Note

```
(min-widths:400px)
```

CSS 中的设备特性共有 13 种，是一个类似于 CSS 属性的集合。但与 CSS 属性不同的是，大部分设备特性的指定值接受 min/max 的前缀，用来表示大于等于或小于等于的逻辑，以此避免使用“<”和“>”这些字符。

使用 and 关键字来指定当某种设备类型的某种特性的值满足某个条件时所使用的样式，例如，以下语句指定了当设备窗口宽度小于 640 像素时所使用的样式：

```
@media screen and (max-width: 639px) {样式代码}
```

可以使用多条语句来将同一个样式应用于不同的设备类型和设备特性中，指定方式类似如下所示：

```
@media handheld and (min-width:360px),screen and (min-width:480px) {样式代码}
```

可以在表达式中加上 not 关键字或 only 关键字，not 关键字表示对后面的表达式执行取反操作，书写方法类似如下所示：

```
//样式代码将被使用在除便携设备之外的其他设备或非彩色便携设备中。
@media not handheld and (color) {样式代码}
//样式代码将被使用在所有非彩色设备中
@media all and (not color) {样式代码}
```

only 关键字的作用是让那些不支持 Media Queries，但是能够读取 Media Type 的设备的浏览器将表达式中的样式隐藏起来。例如，下面的语句，对于支持 Media Queries 的设备来说，将能够正确地应用样式，就像 only 不存在一样。对于不支持 Media Queries 但能够读取 Media Type 的设备（如 IE8 只支持@media screen）来说，由于先读取到的是 only 而不是 screen，将忽略这个样式。对于不支持 Media Queries 的浏览器（如 IE8 之前的浏览器）来说，无论是否有 only，都将忽略这个样式。另外，CSS3 中的 Media Queries 模块中也支持对外部样式表的引用。

```
@media only screen and (color) {样式代码}
```

通过判断设备的类型，然后使用不同的 CSS 样式呈现，这样就可以使 CSS 更精确作用于不同的设备类型或同一设备的不同条件（分辨率、色数等）详细用法如下：

```
media_query: [only | not]? <media_type> [ and <expression> ]*
expression: ( <media_feature> [: <value>]? )
media_type: all | aural | braille | handheld | print | projection | screen |tty | tv | embossed
media_feature: width | min-width | max-width
    | height | min-height | max-height
    | device-width | min-device-width | max-device-width
    | device-height | min-device-height | max-device-height
    | device-aspect-ratio | min-device-aspect-ratio | max-device-aspect-ratio
    | color | min-color | max-color
    | color-index | min-color-index | max-color-index
    | monochrome | min-monochrome | max-monochrome
```

```
| resolution | min-resolution | max-resolution
| scan | grid
```

在@media 规则中，主要参数说明如下。

- ☑ media_query：设置逻辑关键字，如 and（逻辑与）、not（排除某种设备）、only（限定某种设备） 等。
- ☑ media_type：设置设备类型，在 CSS 2.1 中已经定义了 10 种媒体类型，例如，screen 就是媒体类型里的一种，用来指定应用设备为计算机屏幕。
- ☑ mediafeature：定义媒体特性，其被放置在一对圆括号中，如(min-width:400px)。完整的特性说明如表 19.1 所示。

Note

表 19.1　Media Queries 媒体特性

媒 体 特 性	值	可用媒体类型	接受 min/max
width	length	visual、tactile	yes
height	length	visual、tactile	yes
device-width	length	visual、tactile	yes
device-height	length	visual、tactile	yes
orientation	portrait | landscape	bitmap	no
aspect-ratio	ratio	bitmap	yes
device-aspectratio-ratio	ratio	bitmap	yes
color	integer	visual	yes
color-index	integer	visual	yes
monochrome	integer	visual	yes
resolution	resolution	bitmap	yes
scan	progressive |interlace	tv	no
grid	integer	visual、tactile	no

媒体特性共 13 种，类似 CSS 属性的集合，但与 CSS 属性不同的是，媒体特性只接受单个的逻辑表达式作为其值，或者没有值。并且其中的大部分接受 min/max 的前缀，用来表示大于等于/小于等于的逻辑，以此避免使用<和>这些字符。

19.3.3　使用@media 规则

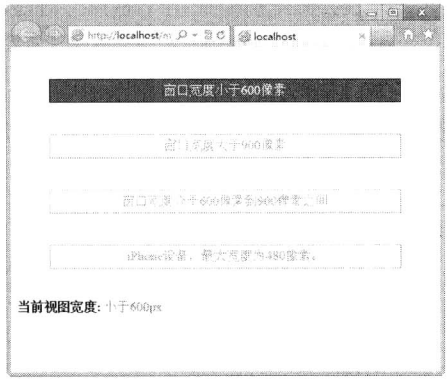
下面通过一个示例演示如何正确使用@media 规则。示例代码如下，演示效果如图 19.3 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title></title>
<style type="text/css">
.wrapper {
    padding: 5px 10px;
    margin: 40px;
    text-align:center;
```

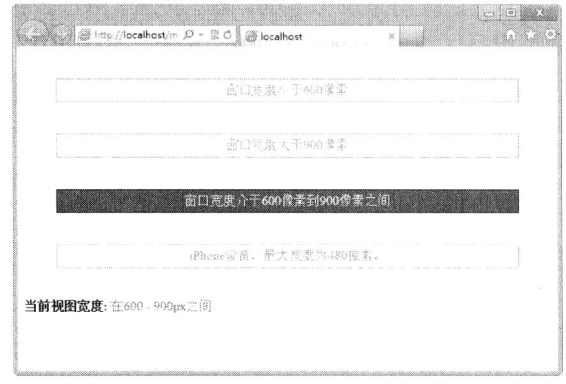
Note

```
        border: solid 1px #999;
        color:#999;
    }
    .viewing-area span {
        color: #666;
        display: none;
    }
    /*max-width:如果视图窗口的宽度小于 600 像素，则醒目显示，并显示提示性的文字*/
    @media screen and (max-width: 600px) {
        .one {
            background: red;
            border: solid 1px #000;
            color:#fff;
        }
        span.lt600 {
            display: inline-block;
        }
    }
    /*min-width:如果视图窗口的宽度大于 900 像素，则醒目显示，并显示提示性的文字*/
    @media screen and (min-width: 900px) {
        .two {
            background: red;
            border: solid 1px #000;
            color:#fff;
        }
        span.gt900 {
            display: inline-block;
        }
    }
    /*min-width & max-width：如果视图窗口的宽度小于 600 像素，则醒目显示，并显示提示性的文字*/
    @media screen and (min-width: 600px) and (max-width: 900px) {
        .three {
            background: red;
            border: solid 1px #000;
            color:#fff;
        }
        span.bt600-900 {
            display: inline-block;
        }
    }
    /*max device width：下面样式应用于 iPhone 设备，且设备最大宽度为 480 像素*/
    @media screen and (max-device-width: 480px) {
        .iphone {
            background: #ccc;
        }
    }
    </style>
    </head>
    <div class="wrapper one">窗口宽度小于 600 像素</div>
    <div class="wrapper two">窗口宽度大于 900 像素</div>
```

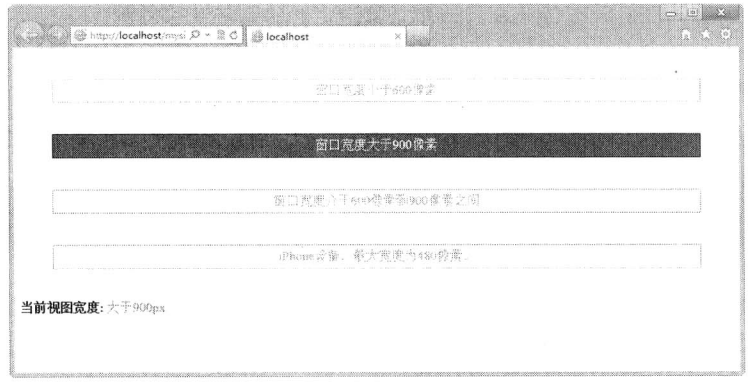
```
<div class="wrapper three">窗口宽度介于 600 像素到 900 像素之间</div>
<div class="wrapper iphone"> iPhone 设备，最大宽度为 480 像素。</div>
<p class="viewing-area">
    <strong>当前视图宽度:</strong>
    <span class="lt600">小于 600px</span>
    <span class="bt600-900">在 600 - 900px 之间</span>
    <span class="gt900">大于 900px</span>
</p>
<body>
</body>
</html>
```



窗口宽度小于 600 像素



窗口宽度介于 600 像素~900 像素之间



窗口宽度大于 900 像素

图 19.3　使用@media 规则

19.3.4　在网站中应用@media 规则

通过 19.3.3 节的示例可以看到，直接在网页内部样式表中使用移动设备样式很方便，但对于网站来说，这种做就显得非常麻烦。这时可以使用 Media Queries 链接外部 CSS 样式表文件，以便在独立的样式表文件中为不同设备编写 CSS 代码。具体用法如下。

```
<link rel="stylesheet" type="text/css" href="small-device.css"
    media="only screen and (max-device-width: 480px)" />
```

通过在<link>标签中设置 media 属性，添加 Media Queries 规则，此时 media 属性值的语法格式遵

循@media 规则的用法。media 属性类似于标签的 style 属性，即在标签内添加样式属性。

一个 Media Query 语句包含一种媒体类型，如果媒体类型没有指定，那么就是默认类型 all，例如：

```
<link rel="stylesheet" type="text/css" href="example.css"
    media="(max-width: 600px)">
```

Note

一个 Media Query 包含 0 到多个表达式，表达式又包含 0 到多个关键字，以及一种媒体特性，例如：

```
<link rel="stylesheet" type="text/css" href="example.css"
    media="handheld and (min-width:20em) and (max-width:50em)">
```

逗号(,)被用来表示并列，表示或者。例如，下面代码表示该 CSS 样式表被应用于宽度小于 20em 的手持设备，或者宽度小于 30em 的屏幕设备中。

```
<link rel="stylesheet" type="text/css" href="example.css"
    media="handheld and (max-width:20em), screen and (max-width:30em)">
```

not 关键字用来排除符合表达式的设备，例如：

```
<link rel="stylesheet" type="text/css" href="example.css"
    media="not screen and (color)">
```

再看下面这个示例。

```
<link rel="stylesheet" type="text/css" href="styleA.css"
    media="screen and (min-width: 800px)">
<link rel="stylesheet" type="text/css" href="styleB.css"
    media="screen and (min-width: 600px) and (max-width: 800px)">
<link rel="stylesheet" type="text/css" href="styleC.css"
    media="screen and (max-width: 600px)">
```

上面将设备分成 3 种，分别是宽度大于 800 像素时，应用 styleA，宽度为 600 像素~800 像素时应用 styleB，以及宽度小于 600 像素时应用 styleC。当宽度正好等于 800 像素时应用 styleB，因为前两条表达式都成立，后者覆盖了前者。

上面代码虽然正确，但是不标准。正常情况应该这样写：

```
<link rel="stylesheet" type="text/css" href="styleA.css"
    media="screen">
<link rel="stylesheet" type="text/css" href="styleB.css"
    media="screen and (max-width: 800px)">
<link rel="stylesheet" type="text/css" href="styleC.css"
    media="screen and (max-width: 600px)">
```

Media Queries 是 CSS3 对于 Media Type 的一个扩展，所以不支持 Media Queries 的浏览器，应该仍然要识别 Media Type，但是 IE 只是简单地忽略了样式。only 关键字可能显得有些多余，对支持 Media Queries 的浏览器来说确实是这样，因为加不加 only 都没有影响。only 的作用很多时候是用来对那些不支持 Media Queries 但是却读取 Media Type 的设备隐藏样式表的。例如：

```
<link rel="stylesheet" type="text/css" href="example.css"
    media="only screen and (color)">
```

支持 Media Queries 的设备，正确应用样式，不支持 Media Queries，但正确读取 Media Type 的设

备，由于先读取到 only 而不是 screen，将忽略这个样式，而不支持 Media Queries 的设备不论是否有 only，都忽略样式。

要在不同设备上测试 Media Queries 并非易事，用户需要准备各种设备，还要将代码上传到某个主机进行访问测试。读者可以访问 http://protofluid.com/，该网站提供在线测试服务，允许提供要测试的网站 URL，或者本机上的 URL，然后模拟 iPhone 等移动设备显示网页设计，用户也可以填写自己的窗口尺寸，来模拟特定的设备。

Note

【示例】本示例根据不同的窗口尺寸来选择使用不同样式，该示例中有 3 个 div 元素，当浏览器的窗口尺寸不同时，页面会根据当前窗口的大小选择使用不同的样式。当窗口宽度在 1000 像素以上时，将 3 个 div 元素分为 3 栏并列显示。当窗口宽度在 640 像素以上、999 像素以下时，3 个 div 元素分两栏显示。当窗口宽度在 639 像素以下时，3 个 div 元素从上往下排列显示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
<title></title>
<style type="text/css">
body { margin: 20px 0; }
#container {
    width: 960px;
    margin: auto;
}
#wrapper {
    width: 740px;
    float: left;
}
.height {
    line-height: 600px;
    text-align: center;
    font-weight: bold;
    font-size: 2em;
    margin: 0 0 20px 0;
}
#main {
    width: 520px;
    float: right;
    background: yellow; /*黄色*/
}
#sub01 {
    width: 200px;
    float: left;
    background: orange; /*橙色*/
}
#sub02 {
    width: 200px;
    float: right;
    background: green; /*緑色*/
}
```

Note

```
/*窗口宽度在 1000 像素以上*/
@media screen and (min-width: 1000px) {
/*3 栏显示*/
#container { width: 1000px; }
#wrapper {
    width: 780px;
    float: left;
}
#main {
    width: 560px;
    float: right;
}
#sub01 {
    width: 200px;
    float: left;
}
#sub02 {
    width: 200px;
    float: right;
}
}
/*窗口宽度在 640 像素以上、999 像素以下*/
@media screen and (min-width: 640px) and (max-width: 999px) {
/*2 栏显示*/
#container { width: 640px; }
#wrapper {
    width: 640px;
    float: none;
}
.height { line-height: 300px; }
#main {
    width: 420px;
    float: right;
}
#sub01 {
    width: 200px;
    float: left;
}
#sub02 {
    width: 100%;
    float: none;
    clear: both;
    line-height: 150px;
}
}
/*窗口宽度在 639 像素以下*/
@media screen and (max-width: 639px) {
/*1 栏显示*/
#container { width: 100%; }
#wrapper {
    width: 100%;
```

Note

```
        float: none;
}
body { margin: 20px; }
.height { line-height: 300px; }
#main {
        width: 100%;
        float: none;
}
#sub01 {
        width: 100%;
        float: none;
        line-height: 100px;
}
#sub02 {
        width: 100%;
        float: none;
        line-height: 100px;
}
}
</style>
</head>
<body>
<div id="container">
        <div id="wrapper">
                <div id="main" class="height">主栏目</div>
                <div id="sub01" class="height">次要栏目</div>
        </div>
        <div id="sub02" class="height">辅助栏目</div>
</div>
</body>
</html>
```

当窗口宽度在 1000 像素以上时，将 3 个 div 元素分为 3 栏并列显示，预览效果如图 19.4 所示。

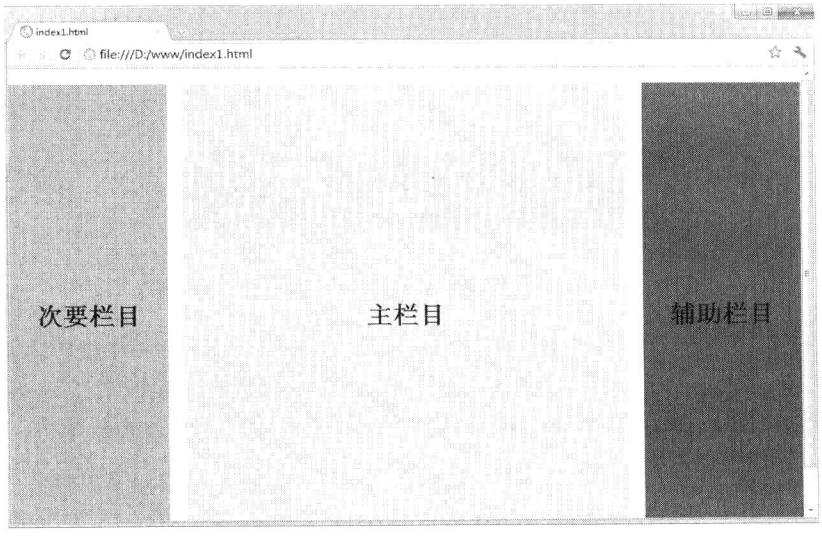


图 19.4　窗口宽度在 1000 像素以上时页面显示效果

当窗口宽度在 640 像素以上、999 像素以下时，3 个 div 元素分 2 栏显示，则预览效果如图 19.5 所示。

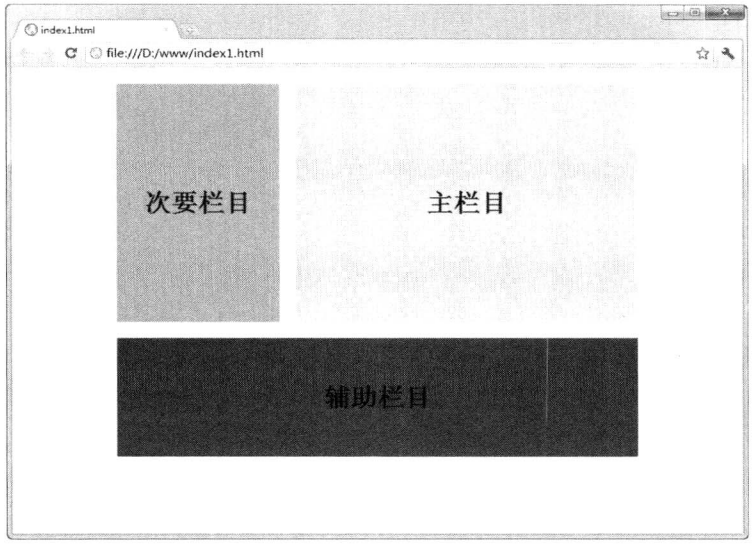


图 19.5　窗口宽度在 640 像素以上、999 像素以下时页面显示效果

当窗口宽度在 639 像素以下时，3 个 div 元素从上往下排列显示，预览效果如图 19.6 所示。

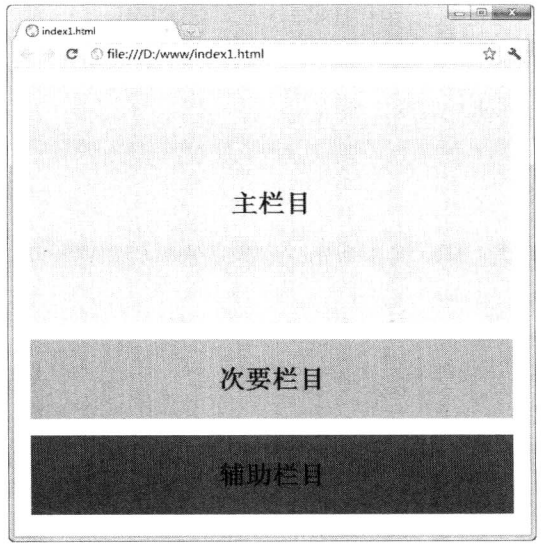


图 19.6　窗口宽度在 639 像素以下时页面显示效果

提示： 在 iPhone 和 iPod touch 中使用的 Safari 浏览器也对 CSS3 的媒体查询表达式提供支持。本示例中 iPhone 的分辨率是 320 × 480 像素，所以，示例页面中的 3 个 div 元素本应从上往下排列显示，但是真正运行的时候，浏览器中显示结果却为两栏显示。

这是因为在 iPhone 中使用的 Safari 浏览器在进行页面显示时是将窗口宽度作为 980 像素进行显示的。因为现在的网页大多是按照宽度为 800 像素左右的标准进行制作的，所以 Safari 浏览器如果按照 980 像素的宽度来显示，就可以正常显示绝大多数的网页了。

所以，即使在页面中已经写好了页面在小尺寸窗口中运行时的样式，iPhone 中的 Safari 浏览器也不会使用这个样式，而是选择窗口宽度为 980 像素时所使用的样式。在这种情况下，可以利用<meta>标签在页面中指定Safari浏览器在处理本页面时按照多少像素的窗口宽度来进行，指定方法如下所示：

```
<meta name="viewport" conten="width=600px" />
```

Note

这样在 iPhone 中正新运行该示例，Safari 浏览器将窗口宽度作为 600 像素来处理，将 3 个 div 元素从上往下并排显示。因此，如果在页面中已经准备好了在小尺寸的窗口中使用的样式，并且有可能在 iPhone 或 iPod touch 中被打开时，不要忘了加入<meta>标签并在标签中写入指定的窗口宽度。

19.4 添加语音功能

2004 年 12 月，CSS3 新推出了 CSS3 Speech Module 语音模块，详细信息请参阅 http://www.w3.org/TR/css3-speech/。CSS3 的 Speech 模块移除了一些旧属性，添加了新属性，这些都分配在 speech 媒体类型中。目前，Opera 是实现最多 CSS3 Speech 属性的最流行的浏览器。该浏览器通过-xv-前缀加以实现。

提示： CSS2 版本添加了听觉媒体类型的支持。这项支持用来定义在听觉设备中合成语音样式。当 CSS 2.1 草案发布后，新标准又提出了 media="speech"的特性，但是没有明确定义可以应用的属性。所以听觉媒体类型在当时没有被认可。

CSS3 Speech 定义了众多属性，详细说明如下所示。

- ☑ voice-volume：设置音量。该属性取值包括<number>、<percentage>、silent | x-soft、soft、medium、loud、x-loud 和 inherit，默认值为 medium。
- ☑ voice-balance：设置声音平衡。该属性取值包括<number>、left、center、right、leftwards、rightwards 和 inherit，默认值为 center。
- ☑ speak：置阅读类型。该属性取值包括 none、normal、spell-out、digits、literal-punctuation、no-punctuation 和 inherit，默认值为 normal。
- ☑ pause-before、pause-after：设置暂停时的效果。该属性取值包括<time>、none、x-weak、weak、medium、strong、x-strong 和 inherit，默认值为 implementation dependent。
- ☑ pause：设置暂停。该属性取值包括[<"pause-before">||<"pause-after">]和 inherit，默认值为 implementation dependent。
- ☑ rest-before、rest-after：设置休止时的效果。该属性取值包括<time>、none、x-weak、weak、medium、strong、x-strong 和 inherit，默认值为 implementation dependent。
- ☑ rest：设置休止。该属性取值包括[<"rest-before">||<"rest-after">]、inherit，默认值为 implementation dependent。
- ☑ cue-before、cue-after：设置提示时的效果。该属性取值包括<uri> [<number>、<percentage>、silent、x-soft、soft、medium、loud、x-loud]、none 和 inherit，默认值为 none。
- ☑ cue：设置提示。该属性取值包括[<"cue-before">||<"cue-after">]和 inherit，无默认值。
- ☑ mark-before、mark-after：设置标注时的效果。该属性取值为<string>，默认值为 none
- ☑ mark：设置标注。该属性取值包括[<"mark-before">||<"mark-after">]，无默认值。
- ☑ voice-family：设置语系。该属性取值包括[[<specific-voice>、[<age>]<generic-voice>]

[<number>],]* [<specific-voice>、[<age>]<generic-voice>] [<number>]、inherit，默认值为 implementation dependent。

- ☑ voice-rate：设置比率。该属性取值包括<percentage>、x-slow、slow、medium、fast、x-fast 和 inherit，默认值为 implementation dependent。
- ☑ voice-pitch：设置音调。该属性取值包括<number>、<percentage>、x-low、low、medium、high、x-high 和 inherit，默认值为 medium。
- ☑ voice-pitch-range：设置音调范围。该属性取值包括<number>、x-low、low、medium、high、x-high 和 inherit，默认值为 implementation dependent。
- ☑ voice-stress：设置重音。该属性取值包括 strong、moderate、none、reduced 和 inherit，默认值为 moderate。
- ☑ voice-duration：设置音乐持续时间。该属性取值为<time>，默认值为 implementation dependent。
- ☑ phonemes：设置音位。该属性取值为<string>，默认值为 implementation dependent。

【示例】下面通过一个简单的示例，体验 CSS3 Speech 的使用。

```
<!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></title>
<style type="text/css">
#speech-sample {
    voice-family: male;
    border: 1px solid #666;
    padding:5px;
    font-size:14px;
}
#voice-volume {
    -xv-voice-volume: x-soft;
    -xv-voice-balance: right;
}
#voice-balance { -xv-voice-balance: left; }
#speech-cue { cue-after: url(images/ding.wav); }
#voice-rate { -xv-voice-rate: x-slow; }
#voice-family { voice-family: female; }
#voice-pitch { -xv-voice-pitch: x-low; }
#speech-speak { speak: spell-out; }
</style>
</head>
<body>
<div>
    <div id="speech-sample">
        <span id="voice-volume">voice-volume、</span>
        <span id="voice-balance">voice-balance、</span>
        <span id="voice-rate">voice-rate、</span>
        <span id="voice-family">voice-family、</span>
        <span id="voice-pitch">voice-pitch、</span>
```

```
        <span id="speech-speak">speak</span> 和
        <span id="speech-cue">cue-after。</span>
    </div>
</div>
</body>
</html>
```

Note

19.5　设 计 倒 影

CSS3 新增了 CSS Reflections 板块，CSS Reflections 允许 CSS 设计投影效果，这种效果在传统设计中只能够通过 Photoshop 等图像编辑软件事先设计好，然后再导入到网页中。CSS Reflections 简化了这种操作，它允许用户使用 CSS 样式来实现超酷的视觉效果。

目前，CSS Reflections 仅获得 Webkit 引擎的支持，只能够在谷歌 Chrome 和 Safari 浏览器中进行测试。W3C 还没有推出 CSS Reflection 标准草案。

Webkit 引擎定义了-webkit-box-reflect 属性，该属性能够实现投影效果，具体语法格式如下。

```
-webkit-box-reflect: <direction> <offset> <mask-box-image>
```

属性取值说明如下。

☑ <direction>：定义反射方向，取值包括 above、below、left 和 right。

☑ <offset>：定义反射偏移的距离，取值包括数值或者百分比，其中百分比根据对象的尺寸进行确定，如果省略该参数值，则默认值为 0。

☑ <mask-box-image>：定义遮罩图像，该图像将覆盖投影区域。如果省略该参数值，则默认值为无遮罩图像。也可以设置渐变色或者纯色覆盖。

当对象源发生变化时，投影能够自动更新，当鼠标经过对象时，也能够在投影中看到鼠标效果，如果该属性应用到<video>标签上时，还可以看到视频以投影效果进行播放。投影的规模和反射偏移不影响页面的布局。

【示例 1】定义简单的倒影样式效果，在样式中定义倒影的方向，代码如下，演示效果如图 19.7 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>CSS Reflections</title>
<style type="text/css">
img {
    height:250px;
    border:1px solid red;
    /*定义倒影*/
    -webkit-box-reflect:below;
}
</style>
</head>
<body>
```

```
<img src="images/bg1.jpg" />
</body>
</html>
```

Note

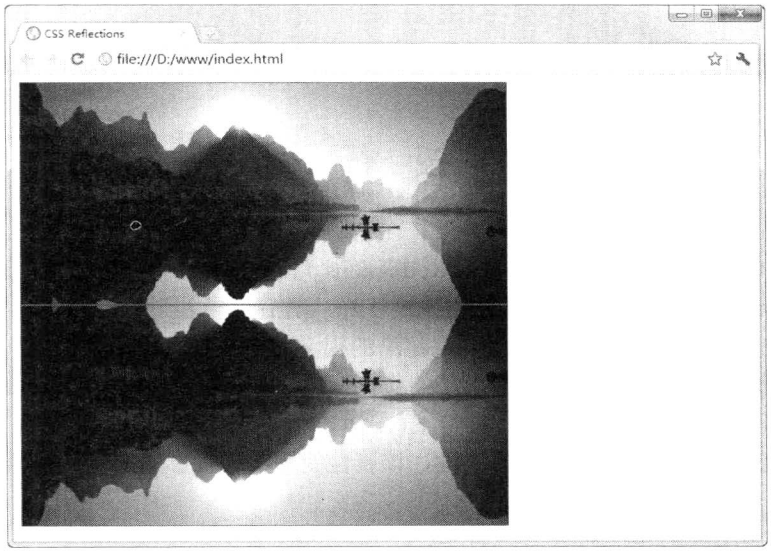


图 19.7　定义简单的倒影效果

在上面示例基础上，可以为倒影设置距离，向下偏移 10 像素，预览效果如图 19.8 所示。

```
<style type="text/css">
img {
    height:250px;
    -webkit-box-reflect:below 10px;
}
</style>
```

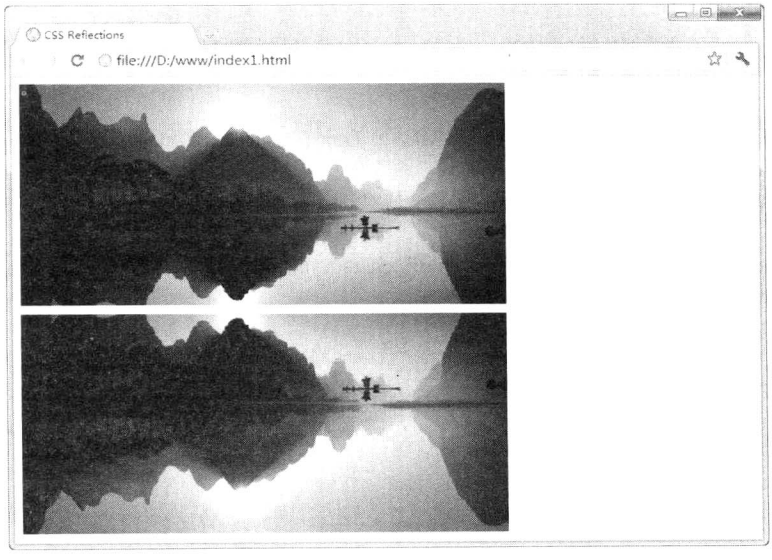


图 19.8　调整倒影距离后的效果

也可以设计渐变倒影，通过渐变遮罩逐渐盖住下面倒影，营造渐隐效果，样式代码如下，预览效果如图 19.9 所示。

Note

```
<style type="text/css">
img {
    height:250px;
    -webkit-box-reflect:below 5px -webkit-gradient(linear, left top, left bottom, from(transparent), color-stop(0.5, transparent), to(white));
}
</style>
```

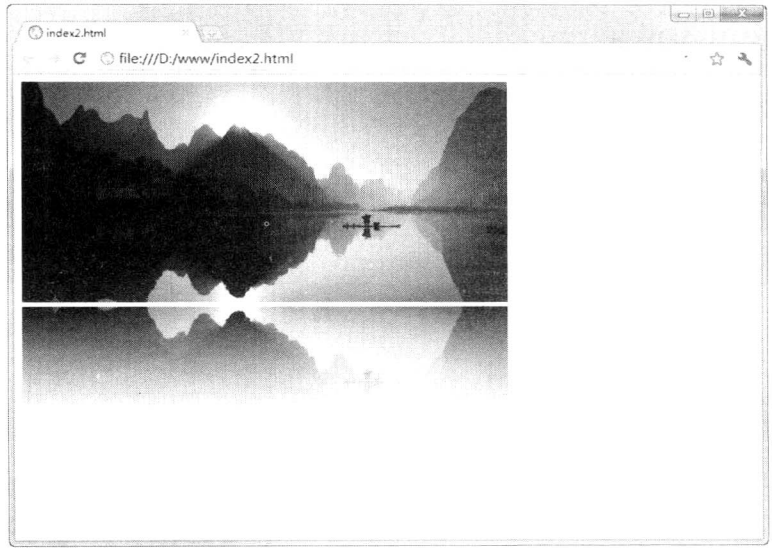


图 19.9　设置渐变倒影效果

【示例 2】除了图片可以设计倒影外，实际上网页上任何对象都可以设计倒影效果，本示例演示了如何为文本设置倒影，代码如下，预览如图 19.10 所示。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>CSS Reflections</title>
<style type="text/css">
img {
    height: 200px;
    position: absolute;
    right: 0;
    bottom: 0;
}
div {
    border: 1px solid #666;
    color: #666;
    -webkit-box-reflect: below 5px;
}
```

```
h1 { text-align: center; }
</style>
</head>
<body>
<div>
    <h1>《雨巷》--- 戴望舒</h1>
    <p>撑着油纸伞，独自<br />
        彷徨在悠长、悠长<br />
        又寂寥的雨巷<br />
        我希望逢着<br />
        一个丁香一样地<br />
        结着愁怨的姑娘 </p>
</div>
<img src="images/bg2.jpg" />
</body>
</html>
```

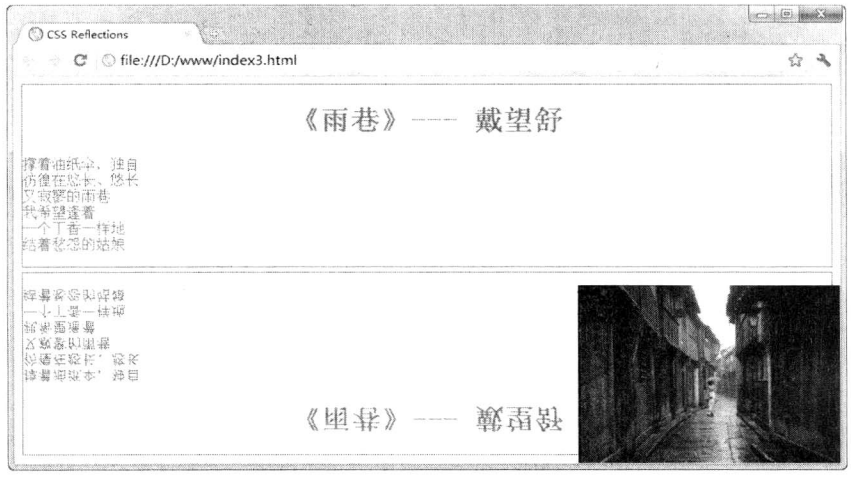


图 19.10　设置文本倒影效果

还可以为视频等多媒体界面设计倒影，这样能够设计出非常动感的视觉效果，这里不再一一举例进行演示。