HZ BOOKS
华章科技

内容系统而全面，HTML 5与CSS 3的新功能和新特性尽览无余
注重实战，包含200余设计精巧的示例，可操作性极强
资深专家亲自执笔，3大前端社区一致推荐，权威性毋庸置疑

實戰

陆凌牛 著

HTML 5 and CSS 3: The Definitive Guide

HTML 5与CSS 3权威指南

机械工业出版社
China Machine Press

實戰

HTML 51联合策划

作为一位前端工作者，如果你想在这场即将到来的Web技术革命中立于不败之地，建议你从现在开始学习和使用HTML 5和CSS 3。如果你想更好、更快、更系统地学习HTML 5和CSS 3，强烈推荐你阅读本书。无论从内容的权威性和全面性，还是案例的实战性和可操作性，这本书都是你当下的最佳选择。

——jQuery中文社区（jquery.org.cn）

HTML 5有两个使命：第一，弥补上一代HTML的不足；第二，实现富Web应用的本地化，使浏览器逃离Flash和Silverlight等富客户端插件的羁绊。它必定成为新一代的Web技术标准。本书基于HTML 5技术的最新发展现状，详尽地介绍了HTML 5的所有新功能和新特性，不仅内容全面，而且实战性强，强烈推荐！

——HTML51（http://www.html51.com/）

CSS 2.1发布至今已有12年的历史，但它已经无法满足当前Web应用日益增长的高性能、高用户体验的需求。CSS 3极大地简化了CSS的编程模型，它不仅对已有的功能进行了扩展和延伸，而且更多的是对Web UI的设计理念的和方法进行了革新。在未来，CSS 3配合HTML 5标准，将掀起一场新的Web应用变革，甚至是整个互联网产业的变革。本书全面而系统、极具针对性地对CSS 3进行了深入浅出地讲解，完整地将CSS 3的所有新功能和新特性呈现给了我们，是学习CSS 3的不二之选。

——CSS 3中文开发者社区

作者简介

陆凌牛 资深Web开发工程师、软件开发工程师和系统设计师。从事Web开发多年，对各种Web开发技术（包括前端和后端）都有非常深入的研究，经验极其丰富。HTML 5和CSS 3等新技术的先驱者和布道者，不仅对HTML 5与CSS 3的理论知识有比较深入的认识，而且已经大量在实践中付诸应用。此外，他还擅长微软与Java的相关技术，在C#、VB.NET、ASP.NET、Java、Struts、Spring、Hibernate、SQL Server、Oracle等方面积累了大量的实战经验。

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

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

封面设计：陈子平

上架指导：计算机/Web开发

ISBN 978-7-111-33624-2

9 787111 336242

定价：69.00元

實戰

HTML 5 and CSS 3: The Definitive Guide

HTML 5与CSS 3权威指南

陆凌牛 著

机械工业出版社
China Machine Press

如果你是一位有前瞻性的Web前端工作者，那么你一定会从本书中受益，因为它就是专门为你打造的。

本书内容系统而全面，详尽地讲解了HTML 5和CSS 3的所有新功能和新特性；技术新颖，所有知识点都紧跟HTML 5与CSS 3的最新发展动态（HTML 5和CSS 3仍在不断完善之中）；实战性强（包含246个示例页面），不仅每个知识点都配有精心设计的小案例（便于动手实践），而且还有两个综合性的案例（体现用HTML 5与CSS 3开发Web应用的思维和方法）。本书不仅能满足你全面而系统地学习理论知识的需求，还能满足你需要充分实践的需求。

全书共分为三大部分，第一部分详尽地讲解了HTML 5的相关知识，包括各主流浏览器对HTML 5的支持情况、HTML 5与HTML 4在语法上的区别、HTML 5的结构元素、表单与文件、图形绘制、多媒体播放、本地存储、离线应用、通信API、Web Workers、地理位置信息获取等内容；第二部分详细地阐述了CSS 3的相关知识，涵盖选择器、文字与字体的相关样式、颜色的相关样式、盒的相关样式、背景与边框的相关样式、布局的相关样式、UI的相关样式、Media Queries的相关样式、变形处理、多媒体和动画等内容。第三部分以迭代的方式逐步展现了两个完整的案例，旨在帮助读者将理论知识贯穿于实践中，迅速成为新一代Web开发技术中的弄潮儿。

无论你是未入门或刚入门的前端新人，还是有多年工作经验的资深前端工程师，这本书都会很适合你。

图书在版编目（CIP）数据

HTML 5与CSS 3权威指南/陆凌牛著 —北京：机械工业出版社，2011.4

ISBN 978-7-111-33624-2

Ⅰ.H… Ⅱ.陆… Ⅲ.①超文本标记语言 ②网页制作工具 Ⅳ.①TP312 ②TP393.092

中国版本图书馆CIP数据核字（2011）第032085号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：陈佳媛
北京市荣盛彩色印刷有限公司印刷
2011年4月第1版第1次印刷
186mm×240mm · 27印张（含彩插6印张）
标准书号：ISBN 978-7-111-33624-2
定价：69.00元

前　言

如果要盘点2010年IT届的十大热门技术，云计算、移动开发、物联网等无疑会在其中，HTML 5肯定也是少不了的。2010年，随着HTML 5的迅猛发展，各大浏览器开发公司如Google、微软、苹果、Mozilla和Opera的浏览器开发业务都变得异常繁忙。在整个2010年度，无论是Mozilla的Firefox、Google的Chrome、苹果的Safari，还是微软的Internet Explorer，以及Opera都处于不断地推陈出新的状态当中。

2010年3月，在微软的MIX2010大会上，微软的工程师在介绍Internet Explorer 9浏览器的同时，从前端技术的角度把互联网的发展分为了三个阶段：

第一阶段：Web 1.0的以内容为主的网络，前端主流技术是HTML和CSS；

第二阶段：Web 2.0的Ajax应用，热门技术是JavaScript/DOM/异步数据请求；

第三阶段：即将迎来的HTML 5时代，亮点是富图形和富媒体内容（Graphically-Rich and Media-Rich）。

前端技术将进入一个崭新的时代，至少已经开启了这扇门。

在这种局势下，学习HTML 5无疑成为Web开发者的一大重要任务，谁先学会HTML 5，谁就掌握了迈向未来Web平台的一把钥匙。因此，我希望能够借助此书帮助国内的Web开发者更好地学习HTML 5以及与之相伴随的CSS 3技术，能够早日运用这些技术开发出一个具有现代水平的、在未来的Web平台上能够正常运行的Web网站或Web应用程序。

本书面向的读者

本书主要适合如下人群阅读：

- 具有一定基础的Web前端开发工程师。
- 具有一定美术功底的Web前端设计师和UI设计师。
- Web项目的项目管理人员。

❑ 开设了Web开发等相关专业的高等院校和相关培训机构的师生。

如何阅读本书

本书共分为三个部分：

第一部分：对HTML 5中新增的语法与标记方法、新增元素、新增API以及这些元素与API目前为止受到了那些浏览器的支持等进行了详细的介绍。在对它们进行介绍的同时将其与HTML 4中的各种元素与功能进行了对比，以帮助读者更好地理解为什么需要使用HTML 5、使用HTML 5有什么好处、HTML 5中究竟增加了哪些目前HTML 4不具备而在第三代Web平台上将会起到重要作用的功能与API，以及这些功能与API的详细使用方法。

第二部分：详细介绍了CSS 3中的各种新增样式与属性，其中主要包括CSS 3中的各种选择器、文字与字体、背景与边框、各种盒模型、CSS 3中的布局方式、CSS 3中的变形与动画、CSS 3中与媒体类型相关的一些样式与属性等。在介绍的同时也详细讲述了这些样式与属性目前为止受到了那些浏览器的支持，以及针对各种浏览器应该怎样在样式代码中进行各种属性的正确书写。

第三部分：详细讲解了两个实例，第一个实例展示了如何在一个用HTML 5语言编写而成的页面中综合运用HTML 5中新增的各种结构元素，如何对这些结构元素综合使用CSS 3样式；第二个实例展示了如何使用HTML 5中新增的表单元素，以及操作本地数据库的功能来实现一个具有现代风格的Web应用程序，如何在这个由HTML 5语言及其功能编写而成的Web应用程序中综合使用CSS 3样式来完成页面的布局以及视觉效果的美化工作。

全书共有246个示例页面，每个页面都经过笔者上机实践，确保运行结果正确无误。每个页面的详细代码及其使用到的脚本文件、各种资源文件都可在华章公司的官方网站（www.hzbook.com）本书的页面上下载，因为是HTML 5编写而成的网页，所以可直接在各种浏览器中打开该文件并查看运行结果（少量页面需要首先建立网站，然后通过访问网站中该页面的方式来进行查看）。同时，对于HTML 5中的各种元素和各种API，以及CSS 3中的各种属性和样式受到了那些浏览器的支持在书中都进行了详细介绍，读者可以针对不同的页面选择正确的浏览器来查看其正确的运行结果。

致谢

在本书的写作过程中，策划编辑杨福川先生和曾姗女士给予了很大的帮助和支持，并提出了很多中肯的建议，在此表示感谢。同时，还要感谢机械工业出版社的所有编审人员为本书的出版所付出的辛勤劳动。本书的成功出版是大家共同努力的结果，谢谢你们。

另外，在本书的写作过程中，由于时间及水平上的原因，有可能存在一些对HTML 5及CSS 3上认识不全面或疏漏的地方，敬请读者批评更正，作者的QQ为240824399，联系邮箱为240824399@qq.com，谨以最真诚的心希望能与读者共同交流，共同成长。

陆凌牛

2011年2月

目　录

第1章
Web时代的变迁

自从2010年HTML 5正式推出以来，它立刻受到了世界各大浏览器的热烈欢迎与支持。根据世界上各大IT界知名媒体评论，新的Web时代，HTML 5的时代马上就要到来。本章重点介绍什么是HTML 5，HTML 5产生的时代背景，为什么HTML 5会如此深受业界欢迎，以及HTML能够解决什么问题。

学习内容：

- 初步了解什么是HTML 5，HTML 5与之前版本的HTML大致上有哪些区别。
- 了解世界各大知名浏览器目前的发展策略，为什么它们都不约而同地把支持HTML 5当成目前的工作重点，就连微软也把全面支持HTML 5作为新版Internet Explorer 9（IE 9）浏览器的开发重点与主要宣传手段。
- 了解为什么说开发者今后可以放心大胆地使用HTML 5进行Web网站与Web应用程序的开发，HTML 5被正式推广以后之前的Web网站与Web应用程序怎么办。
- 了解使用HTML 5到底可以解决哪些问题。

1.1 迎接新的Web时代

1.1.1 HTML 5时代即将来临

自从2010年HTML 5正式推出以来，它就以一种惊人的速度被迅速推广着，就连微软也因此为下一代IE 9做了标准上的改进，使其能够支持HTML 5。关于各主流浏览器对于HTML 5所表现出来的热烈欢迎、积极支持的详细情况，以及为什么HTML 5会如此受欢迎，我们将在后面几节中详细介绍，这里，笔者要告诉大家的是，目前业界全体都步调一致地朝着HTML 5的方向迈进着，HTML 5的时代马上就要到来了。

在全面介绍HTML 5的相关知识之前，我们先来认识一下HTML 5中的部分代码，对HTML 5有个初步的了解。

首先，我们来看一段HTML 4中常见的JavaScript代码，如代码清单1-1所示：

代码清单1-1 HTML 4中的JavaScript代码示例

```
<form>
<p><label>Username:<input name=search type="text" id="search"></label></p>
<script type="text/javascript">
        document.getElementById ('search').focus()
</script>
</form>
```

在HTML 5中，这段代码将会以怎样的形式出现呢？具体如代码清单1-2所示：

代码清单1-2 用HTML 5实现代码清单1-1中的JavaScript代码

```
<form>
<p><label>Search:<input name=search autofocus></label></p>
</form>
```

我们来看一下在HTML 4中常见的一种页面结构，代码如代码清单1-3所示：

代码清单1-3 div标签示例（用HTML 4实现）

```
<div id="header">...</div>
<div id="nav">...</div>
<div class="aritcle">
</div>
<div id="side-bar">...</div>
<div id="footer">...</div>
```

页面中有关该部分的结构示意图如图1-1所示：

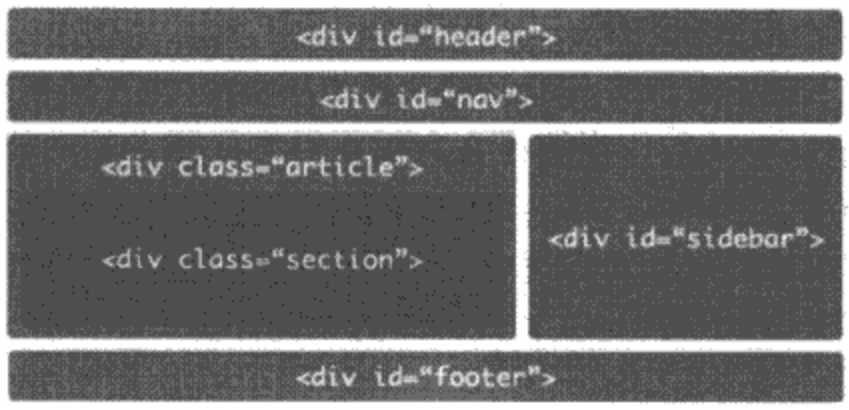

图1-1 HTML 4中的页面结构

那么，在HTML 5中，又会用怎样的页面代码来描述这种结构呢？具体如代码清单1-4所示：

代码清单1-4 HTML 5中的新型结构示例

```
<header>...</header>
<nav>...</nav>
<aritcle>
</aritcle>
<aside>...</aside>
<footer>...</footer>
```

页面中有关该部分的结构示意图如图1-2所示。

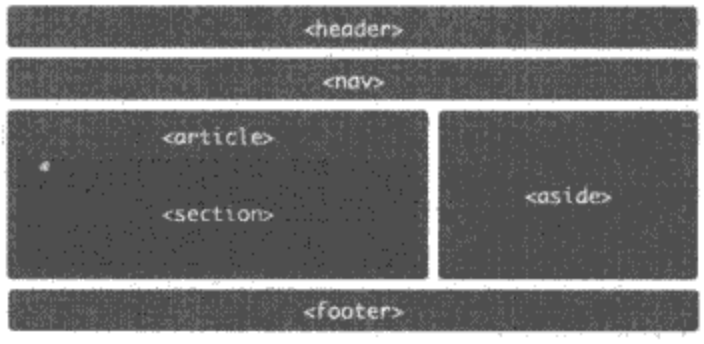

图1-2 HTML 5中的页面结构

怎么样？看出区别来了吗？在第一个示例中，我们可以看见，在HTML 4中的一段JavaScript代码，在HTML 5中消失了，取而代之的是一个在HTML 5中新出现的属性。在第二个示例中，我们可以看见，在HTML 4中常见的用div来划分页面结构的方法，到了

HTML 5中，也被一种HTML 5中新出现的标签给替代了。那么究竟为什么HTML 5要对HTML 4的脚本以及页面代码做这种修改呢？做这种修改的目的又是什么呢？

1.1.2 HTML 5的目标

HTML 5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。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，等等。总体来说，为下一代Web平台提供了许许多多新的功能。

那么让我们先来初步接触一下在HTML 5中究竟提供了哪些革命性的新功能。在第2章中，我们会针对这些功能做一个全面介绍。

首先，在HTML 5之前，有很多功能必须要使用JavaScript等脚本语言才能实现，譬如前面例子中提到，登录画面中经常使用的让文本框获得光标焦点的功能。如果使用HTML 5，同样的功能只要使用元素的属性标签就可以实现了。这样的话，整个页面就变得非常清楚直观，容易理解。因此，Web设计者可以非常放心大胆地使用这些HTML 5中新增的属性标签。由于HTML 5中提供了大量的这种可以替代脚本的属性标签，使得开发出来的界面语言也变得更加简洁易懂。

不但如此，HTML 5使页面结构也变得清楚明了。之前使用的div标签也不再使用了，而是使用前面HTML 5示例中所提到的更加语义化的结构标签。这样的话，书写出来的界面结构显得非常清晰，各部位要展示什么内容也让人一目了然。

虽然HTML 5宣称的立场是“非革命性的发展”，但是它所带来的功能是让人渴望的，使用它所进行的设计也是很简单的，因此，它深受Web设计者与Web开发者的欢迎。

1.2 HTML 5会深受欢迎的理由

1.2.1 世界知名浏览器厂商对HTML 5的支持

HTML 5被说成是划时代也好，具有革命性也好，如果不能被业界承认并且大面积地推广使用，这些都是没有意义的。事实上，今后HTML 5被正式地、大规模地投入应用的可能性是相当高的。

通过对Internet Explore、Google、Firefox、Safari、Opera等主要的Web浏览器的发展策略的调查，发现它们都在支持HTML 5上采取了措施。

- 微软：2010年3月16日，微软于拉斯维加斯市举行的MIX10技术大会上宣布已推出IE9浏览器开发者预览版。微软称，IE9完成开发后，将更多支持CSS 3、SVG和HTML 5等互联网浏览通用标准。
- Google：2010年2月19日，谷歌Gears项目经理伊安-费特通过博客宣布，谷歌将放弃对Gears浏览器插件项目的支持，以此重点开发HTML 5项目。据费特表示，目前，在谷歌看来，Gears面临的主要问题是，该应用与HTML 5的诸多创新非常相似，而且谷

歌一直积极发展HTML 5项目。因此，只要谷歌不断以加强新网络标准的应用功能为工作重点，那么为Gears增加新功能就无太大意义了。目前，多种浏览器将会越来越多地为GMail及其他服务提供更多脱机功能方面的支持，因此Gears面临的需求也在日益下降，这是谷歌做出上述调整的重要原因。

- 苹果：2010年6月7日，苹果在开发者大会的会后发布了Safari 5，这款浏览器支持10个以上的HTML 5新技术，包括全屏幕播放、HTML 5视频、HTML 5地理位置、HTML 5切片元素、HTML 5的可拖动属性、HTML 5的形式验证、HTML 5的Ruby、HTML 5的AJAX历史和WebSocket字幕。
- Opera：2010年5月5日，Opera软件公司首席技术官Hakon Wium Lie先生在访华之际，接受了中国软件资讯网等少数几家媒体的采访。号称“CSS之父”的Hakon Wium Lie认为，HTML 5与CSS 3将是全球互联网发展的未来趋势，目前包括Opera在内的诸多浏览器厂商，纷纷在研发HTML 5相关产品，Web的未来属于HTML 5。
- Mozilla：2010年7月，Mozilla基金会发布了即将推出的Firefox 4浏览器的第一个早期测试版。在该版本中的Firefox浏览器中进行了大幅改进，包括新的HTML 5语法分析器，以及支持更多HTML 5形式的控制等。从官方文档来看，Firefox 4对HTML 5是完全级别的支持。目前包括在线视频、在线音频等多种应用都已在该版中实现。

以上证据表明，目前这些浏览器都纷纷地朝着支持HTML 5、结合HTML 5的方向迈进着，因此HTML 5已经被广泛地推行开来了。为什么HTML 5会如此受欢迎，理由如1.2.2节和1.2.3节所示。

1.2.2 第一个理由：时代的要求

现在的时代已经迫切地要求有一个统一的互联网通用标准。HTML 5之前的情况是，由于各浏览器之间的不统一，光是修改Web浏览器之间的由于兼容性而引起的bug就浪费了大量时间。而HTML 5的目标就是将Web带入一个成熟的应用平台，在HTML 5平台上，视频、音频、图像、动画，以及同电脑的交互都被标准化。

关于Web浏览器，网页标准计划小组设计并推出了Acid3测试，它是针对网页浏览器及设计软件之标准相容性的一项测试。它针对Web应用程序中使用着的动态内容进行检查，测试焦点主要集中在ECMAScript、DOM Level 3、Media Queries和data: URL。

Acid3测试推出后，各大浏览器都认真接受了它的测试并希望能够获得比较高的分数。这个测试的设计者，正是在W3C开发及设计者，HTML 5的重要人物Ian Hickson。Ian Hickson是WHATWG（Web Hypertext Application Technology Working Group）开发团体的成员，担任Web标准规格的设计，现在是W3C的HTML 5工作组的负责人之一。

Ian Hickson设计Acid3测试的意图是给声称“让开发者能够什么都不必担心，可以放心大胆地进行开发”的各大Web浏览器提供一个机会，让他们能够以此来证明自己是优秀的。Acid3的宣传是很重要的，要想扩大Web浏览器的市场份额，宣称遵从它所依赖的标准是最有效的宣传方法。图1-3为Acid3的一个测试图。

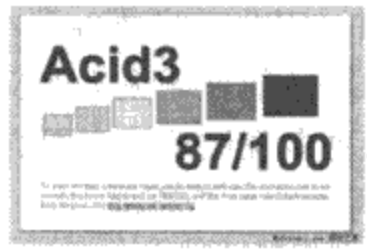

图1-3 Acid3测试图

1.2.3 第二个理由：Internet Explorer 8

Internet Explorer也积极地朝着支持HTML 5的方向迈进着。Internet Explorer对此十分重视。虽然它的使用者依然很多，但是由于最近被Firefox等其他Web浏览器抢去了很多市场份额，它很不甘心。于是继Internet Explorer 7（IE 7）的发表后不久，立刻推出了Internet Explorer 8（IE 8）的Release版。

新推出的IE 8宣称遵从互联网通用标准。虽然其他的浏览器由于标榜遵从该标准而获得了很多市场份额，但是Internet Explorer肯定是要对此采取强有力的对策的。因此Internet Explorer把宣称遵从互联网通用标准看成了很重要的一件事，并且开始在IE 8里支持HTML 5。

例如，HTML 5中代替Cookie的sessionStorage功能与globalStorage功能在IE 8里都获得了支持。使用Ajax时如果点击返回按钮也可以真正让操作返回了（在IE 7中点击返回按钮，画面跳转到其他画面）。很多Internet Explorer自己独特的处理方法与特性，今后也会有所改变。

因为现在市场份额最高的Internet Explorer也在针对HTML 5做出积极对应，微软也对新的互联网通用标准表示了赞同和支持，所以可以说HTML 5在市场上大面积推广的势头是非常强的。

1.3 可以放心使用HTML 5的三个理由

Web开发者最担心的是新技术推出时由于其不成熟所产生的问题。如果能够实现互联网通用标准，可以避免各浏览器之间的不统一，这一点已经被明确了，但是在朝着这方面前进的过程中会不会出现什么周折是令人担心的。

虽然Web开发者普遍认为有了HTML 5是比较好的，但是还是会很担心诸如“它在老版本的浏览器上也能正常运行吗？”，“会不会产生错误？”等各种问题。但是可以很高兴地告诉你，请放心，HTML 5就像以前CSS刚开始普及时一样不会存在什么问题。

有三个理由证明可以放心使用HTML 5：

- 兼容性：HTML 5在老版本的浏览器上也可以正常运行。
- 实用性：HTML 5内部并没有封装什么很复杂的、不切实际的功能，而只是封装了简单实用的功能。
- 非革命性的发展：HTML 5的内部功能不是革命性的，只是发展性的。

以上三点就是所谓的“HTML设计原则”，HTML 5也是以该设计原则为基本原则而开发

出来的，各主流浏览器使用HTML 5的前提也就是要求HTML 5能够符合这些原则，今后也将以其为前提来支持HTML 5。下面针对这些原则进行介绍。

首先是兼容性问题。虽然到了HTML 5时代，但并不代表现在用HTML 4创建出来的网站必须全部要重建，只会要求各Web浏览器今后能正常运行用HTML 5开发出来的功能。“非革命性的发展”这一点正是通过兼容性体现出来的。正是因为保障了兼容性才能让人毫不犹豫地用HTML 5来开发网站。

接着是实用性。实用性是指要求能够解决实际问题。HTML 5内只封装了切实有用的功能，不封装复杂而没有实际意义的功能。

通过以上列举的HTML设计原则，尤其是与HTML 4相兼容的部分，基本上可以让人放下心来，大胆地使用HTML 5。

1.4 HTML 5要解决的三个问题

HTML 5的出现，对于Web来说意义是非常重大的。因为它的意图是想要把目前Web上存在的各种问题一并解决掉，它是一个企图心比较强的HTML版本。

那么，到底Web上存在哪些问题，HTML 5又打算怎么解决呢？

❑ Web浏览器之间的兼容性很低。

首先要提到的就是，Web浏览器之间的兼容性是非常低的。在某个Web浏览器上可以正常运行的HTML/CSS/JavaScript等Web程序，在另一个Web浏览器上就不正常了的事情是非常多的。

如果用一句话来描述这个问题的原因，可以说是“规范不统一”。规范不统一，没有被标准化，是这个问题的主要原因。

在HTML 5中，这个问题将得到解决。HTML 5的使命是详细分析各Web浏览器所具有的功能，然后以此为基础，要求这些浏览器所有内部功能都要符合一个通用标准。

如果各浏览器都符合通用标准，然后以该标准为基础来书写程序，那么程序在各浏览器都能正常运行的可能性就大大提高了，这对于Web开发者和Web设计者都是一件令人可喜的事情。而且，今后开发者开发出来的Web功能只要符合通用标准，Web浏览器也都是很愿意封装该功能的。

❑ 文档结构不够明确。

第二个问题是，在之前的HTML版本中，文档的结构不够清晰、明确。例如，为了要表示“标题”，“正文”，之前一般都是用 元素。但是，严格说来， 不是一个能把文档结构表达得很清楚的元素，使用了过多的 要素的文章，阅读时不仔细研究，是很难看出文档结构的。而且，对于搜索引擎或屏幕阅读器等程序来说，过多使用了div元素，那么这些程序就连“从哪到哪算是重要的正文”，“这个<ul>要素是表示导航菜单，还是表示项目列表”等对于结构分析来说最基本的问题的答案也都不知道。

在HTML 5中，为了解决这个问题，追加了很多跟结构相关的元素。不仅如此，还结合

了包括微格式、无障碍应用在内的各种各样的周边技术。

❑ Web应用程序的功能受到了限制。

最后一个问题是，HTML与Web应用程序的关系十分薄弱。Web应用程序的特征是先从网络下载，然后忠实运行，因此应该对会威胁到用户安全的功能进行限制。

目前安全性的保障这方面已做到了，但对于Web应用程序来说，一直以来HTML真正所做出的贡献是很少的，譬如说就连上传文件时想同时选择一个以上的文件都做不到。

为了弥补这方面的不足，HTML 5已经开始提供各种各样Web应用上的新API，各浏览器也在快速地封装着这些API，HTML 5已经使富Web应用的实现变成了可能。

第2章 HTML 5与HTML 4的区别

HTML 5以HTML 4为基础，对HTML 4进行了大量的修改。本章从总体上概要介绍到底HTML 5对HTML 4进行了哪些修改，HTML 5与HTML 4之间比较大的区别是什么。

学习内容：

- 掌握HTML 5与HTML 4在基本语法上有什么区别，这个基本语法区别包括DOCTYPE声明、内容类型（ContentType）、字符编码的指定方法、元素标记的省略、具有布尔值的属性、引号的省略等几个方面。
- 了解在HTML 5中新增了哪些元素，删除了哪些HTML 4中的元素，为什么要删除这些元素，用什么元素或方法来取代这些被删除的元素。
- 了解在HTML 5中新增了哪些属性，删除了哪些HTML 4中的属性，在HTML 5中用什么方法来取代这些被删除的属性。
- 掌握什么是全局属性，掌握本章中介绍的几个常用全局属性，它们是contentEditable属性、designMode属性、hidden属性、spellcheck属性，以及tabindex属性。

2.1 语法的改变

2.1.1 HTML 5的语法变化

与HTML 4相比，HTML 5在语法上发生了很大的变化。可能有很多人会有疑问，“之前的HTML已经相当普及了！”，“如果改变基础语法，会产生什么影响？”等。

但是，HTML 5中的语法变化，与其他开发语言中的语法变化在根本意义上有所不同。它的变化，正是因为在HTML 5之前几乎没有符合标准规范的Web浏览器！

HTML的语法是在SGML（Standard Generalized Markup Language）语言的基础上建立起来的。但是SGML语法非常复杂，要开发能够解析SGML语法的程序也很不容易，所以很多浏览器都不包含SGML的分析器。因此，虽然HTML基本上遵从SGML的语法，但是对于HTML的执行在各浏览器之间并没有一个统一的标准。

在这种情况下，各浏览器之间的互兼容性和互操作性在很大程度上取决于网站或网络应用程序的开发者们在开发上所做的共同努力，而浏览器本身始终是存在缺陷的。

如上所述，在HTML 5中提高Web浏览器之间的兼容性是它的一个很大的目标，为了确保兼容性，就要有一个统一的标准。因此，在HTML 5中，就围绕着这个Web标准，重新定义了一套在现有的HTML的基础上修改而来的语法，使它运行在各浏览器时各浏览器都能够符合这个通用标准。

因为关于HTML 5语法解析的算法也都提供了详细的记载，所以各Web浏览器的供应商们可以把HTML 5分析器集中封装在自己的浏览器中。最新的Firefox（默认为4.0以后的版本）与WebKit浏览器引擎中都迅速地封装了供HTML 5使用的分析器，IE（Internet Explorer）与Opera也在努力加快对于HTML 5的支持——浏览器兼容性的提高指日可待。

接下来，让我们具体看一下在HTML 5中，到底对语法进行了哪些改变。

2.1.2 HTML 5中的标记方法

首先，让我们来看一下在HTML 5中的标记方法。

1. 内容类型（ContentType）

首先，HTML 5的文件扩展符与内容类型保持不变。也就是说，扩展符仍然为“.html”或“.htm”，内容类型（ContentType）仍然为“text/html”。

2. DOCTYPE声明

DOCTYPE声明是HTML文件中必不可少的，它位于文件第一行。在HTML 4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HTML 5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。HTML 5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HTML 5中像这样的DOCTYPE声明方式是允许的（不区分大小写,引号不区分是单引号还是双引号）。

3. 指定字符编码

在HTML 4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HTML 5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（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HTML 5中，这种字符编码方式将被认为是错误的，这一点请注意：

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

从HTML 5开始，对于文件的字符编码推荐使用UTF-8。

2.1.3 HTML 5确保了与之前HTML版本的兼容性

HTML 5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。例如，符合“没有<p>的结束标记”的HTML代码随处可见，HTML 5中并没有把这种情况作为错误来处理，而是允许存在这种情况，但明确地规定了这种情况应该怎么处理。

那么，针对这个问题，让我们从元素标记的省略、具有boolean值的属性、引号的省略这几方面来详细看一下在HTML 5中是如何确保与之前版本的HTML达到兼容的。

1. 可以省略标记的元素

在HTML 5中，元素的标记可以省略。具体来说，元素的标记分为“不允许写结束标记”、“可以省略结束标记”和“开始标记和结束标记全部可以省略”三种类型。让我们来针对这三类情况列举一个元素清单，其中包括HTML 5中的新元素（关于这些新元素，2.2节将进行介绍）。

- ❑ 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、wbr。
- ❑ 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、th。
- ❑ 可以省略全部标记的元素有：html、head、body、colgroup、tbody。

说明：“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，HTML 5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。请注意，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，但它在文档结构中还是存在的，可以使用document.body进行访问。

---

2. 具有boolean值的属性

对于具有boolean值的属性，例如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时，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。

属性值的设定方法可以参考下面的代码示例：

```
<!--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="">
```

3. 省略引号

大家已经知道，指定属性值的时候，属性值两边既可以用双引号，也可以用单引号。

HTML 5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。如下面的代码所示：

```
<!-- 请注意type的属性值两边的引号 -->
```

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

2.1.4　标记示例

现在，让我们通过前面学到的HTML 5的语法知识来看一个关于HTML 5标记的示例。

代码清单2-1完全是用HTML 5写成的，省略了<html>、<head>、<body>等元素。可以通过这个示例复习一下HTML 5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。

代码清单2-1　HTML 5标记示例

```
<!DOCTYPE html>
<meta charset="UTF-8">
<title>HTML 5标记示例</title>
<p>这段代码是根据HTML 5语法
<br/>编写出来的。
```

这段代码在Firefox 4浏览器中的运行结果如图2-1所示，另外，本书中如果没有特别说明使用什么浏览器的时候，默认使用的都是Firefox 4浏览器。

图2-1　HTML 5标记示例

2.2　新增的元素和废除的元素

本节将详细介绍HTML 5中新增和废除了哪些元素[⊖]。

2.2.1　新增的结构元素

在HTML 5中，新增了以下与结构相关的元素：

- section元素

⊖　其他资料介绍的新增元素可能会比下面要介绍的多，这是因为HTML 5在最新发布的版本中，又把这些本来想新增的元素给删除了。

section元素表示页面中的一个内容区块，比如章节、页眉、页脚或页面中的其他部分。它可以与h1、h2、h3、h4、h5、h6等元素结合起来使用，标示文档结构。

HTML 5中代码示例：

```
<section>...</section>
```

HTML 4中代码示例：

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

❑ article元素

article元素表示页面中的一块与上下文不相关的独立内容，譬如博客中的一篇文章或报纸中的一篇文章。

HTML 5中代码示例：

```
<article>...</article>
```

HTML 4中代码示例：

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

❑ aside元素

aside元素表示article元素的内容之外的、与article元素的内容相关的辅助信息。

HTML 5中代码示例：

```
<aside>…</aside>
```

HTML 4中代码示例：

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

❑ header元素

header元素表示页面中一个内容区块或整个页面的标题。

HTML 5中代码示例：

```
<header>...</header>
```

HTML 4中代码示例：

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

❑ hgroup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。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ footer元素

footer元素表示整个页面或页面中一个内容区块的脚注。一般来说，它会包含创作者的姓名、创作日期以及创作者联系信息。

HTML 5中代码示例：

```
<footer></footer>
```

HTML 4中代码示例：

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

❑ nav元素

nav元素表示页面中导航链接的部分。

HTML 5中代码示例：

```
<nav></nav>
```

HTML 4中代码示例：

```
<ul></ul>
```

❑ figure元素

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元素为figure元素组添加标题。

HTML 5中代码示例：

```
<figure>
<figcaption>PRC</figcaption>
<p>The People's Republic of China was born in 1949...</p>
</figure>
```

HTML 4中代码示例：

```
<dl>
<h1>PRC</h1>
<p>The People's Republic of China was born in 1949...</p>
</dl>
```

2.2.2 新增的其他元素

除了结构元素外，在HTML 5中，还新增了以下元素：

❑ video元素

video元素定义视频，比如电影片段或其他视频流。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ audio元素

audio元素定义音频，比如音乐或其他音频流。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ embed元素

embed元素用来插入各种多媒体，格式可以是 Midi、Wav、AIFF、AU、MP3等。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ mark元素

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。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ progress元素

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。

HTML 5中代码示例：

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

这是HTML 5中新增功能，故无法用HTML 4代码来实现。

❑ time元素

time元素表示日期或时间，也可以同时表示两者。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ ruby元素

ruby元素表示ruby注释（中文注音或字符）。

HTML 5中代码示例：

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

这也是HTML 5中新增的功能。

❑ rt元素

rt元素表示字符（中文注音或字符）的解释或发音。

HTML 5中代码示例：

```
<ruby>漢 <rt> 厂马'</rt></ruby>
```

这是HTML 5中新增的功能。

❑ rp元素

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。

HTML 5中代码示例：

```
<ruby>漢 <rt><rp>(</rp>厂马'<rp>)</rp></rt></ruby>
```

这是HTML 5中新增的功能。

❑ wbr元素

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。wbr元素好像对字符型的语言作用挺大，但是对于中文，貌似没多大用处。

HTML 5中代码示例：

```
<p> To learn AJAX, you must be fami<wbr>liar with the XMLHttp<wbr>
Request Object. </p>
```

这是HTML 5中新增的功能。

❑ canvas元素

canvas元素表示图形，比如图表和其他图像。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。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ command元素

command元素表示命令按钮，比如单选按钮、复选框或按钮。

HTML 5中代码示例：

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

这是HTML 5中新增的功能。

❑ details元素

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

HTML 5中代码示例：

```
<details>
   <summary>HTML 5</summary>
   This document teaches you everything you have to learn about HTML 5.
</details>
```

这是HTML 5中新增的功能。

❑ 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。

HTML 5中代码示例：

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

这是HTML 5中新增的功能。

❑ datagrid元素

datagrid元素表示可选数据的列表，它以树形列表的形式来显示。

HTML 5中代码示例：

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

这是HTML 5中新增的功能。

❑ keygen元素

keygen元素表示生成密钥。

HTML 5中代码示例：

```
<keygen>
```

这是HTML 5中新增的功能。

❑ output元素

output元素表示不同类型的输出，比如脚本的输出。

HTML 5中代码示例：

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

HTML 4中代码示例：

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

❑ source元素

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

HTML 5中代码示例：

```
<source>
```

HTML 4中代码示例：

```
<param>
```

❑ menu元素

menu元素表示菜单列表。当希望列出表单控件时使用该标签。

HTML 5中代码示例：

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

在HTML 4中，menu元素不被推荐使用。

2.2.3 新增的input元素的类型

HTML 5中新增了很多input元素的类型，现列举如下：

- email

email类型表示必须输入E-mail地址的文本输入框。

- url

url类型表示必须输入URL地址的文本输入框。

- number

number类型表示必须输入数值的文本输入框。

- range

range类型表示必须输入一定范围内数字值的文本输入框。

- Date Pickers

HTML 5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、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）

2.2.4 废除的元素

由于各种原因，在HTML 5中废除了很多元素，简单介绍如下。

1. 能使用CSS替代的元素

对于basefont、big、center、font、s、strike、tt、u这些元素，由于它们的功能都是纯粹为画面展示服务的，而HTML 5中提倡把画面展示性功能放在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。

2. 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HTML 5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，同时将以上这三个元素废除。

3. 只有部分浏览器支持的元素

对于applet、bgsound、blink、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Internet Explorer所支持，所以在HTML 5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编程的方式所替代。

4. 其他被废除的元素

其他被废除元素还有：

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

2.3 新增的属性和废除的属性

在HTML 5中，在增加和废除了很多元素的同时，也增加和废除了很多属性，本节对于这些增加和废除的属性进行简单介绍[⊖]。

2.3.1 新增的属性

1. 表单相关的属性

新增的与表单相关的元素如下：

- 可以对input（type=text）、select、textarea与button元素指定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。
- 可以对input元素（type=text）与textarea元素指定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。
- 可以对input、output、select、textarea、button与fieldset指定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。
- 可以对input元素（type=text）与textarea元素指定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元素增加了几个新的属性：autocomplete、min、max、multiple、pattern与step。

⊖ 其他资料介绍的新增属性可能会比下面要介绍的多，这是因为HTML 5在最新发布的版本中，又把这些本来想新增的属性给删除了。

同时还有一个新的list元素与datalist元素配合使用。datalist元素与autocomlete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，他们可以重载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、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。

2. 链接相关属性

新增的与链接相关的属性如下：

- ❑ 为a与area元素增加了media属性，该属性规定目标URL是为什么类型的媒介/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），该属性指定关联图标（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。

3. 其他属性

除了上面介绍的与表单和链接相关的属性外，HTML 5还增加了下面的属性：

- ❑ 为ol元素增加属性reversed，它指定列表倒序显示。
- ❑ 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、工具条与列表菜单的三种形式出现。
- ❑ 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，它定义脚本是否异步执行。
- ❑ 为html元素增加属性manifest，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元素增加三个属性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。

2.3.2 废除的属性

HTML 4中的一些属性在HTML 5中不再被使用，而是采用其他属性或其他方案进行替代，具体如表2-1所示。

表2-1 在HTML 5中被废除了的属性

在HTML 4中使用的属性	使用该属性的元素	在HTML 5中的替代方案
rev	link、a	rel
charset	link、a	在被链接的资源的中使用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	在被链接的资源的中使用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
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

（续）

在HTML 4中使用的属性	使用该属性的元素	在HTML 5中的替代方案
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	

2.4 全局属性

在HTML 5中，新增了一个“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，本节将详细介绍几种常用的全局属性。

2.4.1 contentEditable属性

contentEditable是由微软开发、被其他浏览器反编译并投入应用的一个全局属性。该属性的主要功能是允许用户编辑元素中的内容，所以该元素必须是可以获得鼠标焦点的元素，而且在点击鼠标后要向用户提供一个插入符号，提示用户该元素中的内容允许编辑。contentEditable属性是一个布尔值属性，可以被指定为true或false。

除此之外，该属性还有个隐藏的inherit（继承）状态，属性为true时，元素被指定为允许编辑；属性为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。

另外，除了contentEditable属性外，元素还具有一个isContentEditable属性，当元素可编辑时，该属性为true；当元素不可编辑时，该属性为false。

代码清单2-2中给出了一个使用contentEditable属性的示例，当列表元素被加上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对该示例进行试验。

代码清单2-2 contentEditable属性示例

```
<!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2-2所示。

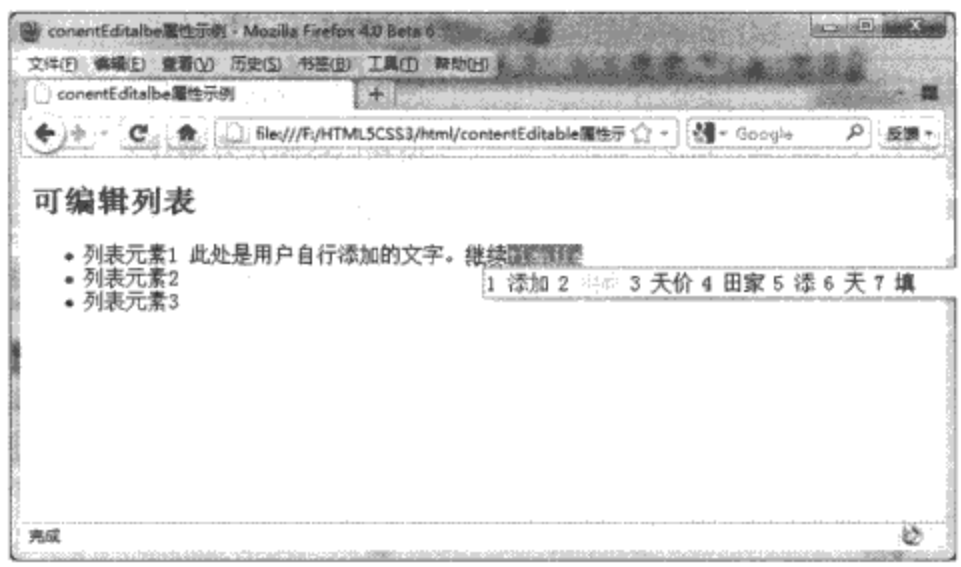

图2-2 可编辑列表示例

在编辑完元素中的内容后，如果想要保存其中内容，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。

最后，在这里列举一下支持contentEditable属性的元素：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、LABEL。

2.4.2 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属性的方法如下所示：

```
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。

2.4.3 hidden属性

在HTML 5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。

2.4.4 spellcheck属性

spellcheck属性是HTML 5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或false两种值。但是它在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，书写方法如下所示：

```
<!—以下两种书写方法正确-->
<textarea spellcheck="true" >
<input type=text spellcheck=false>
<!—以下书写方法为错误-->
<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、Opera等浏览器都对该属性提供了支持。图2-3为Opera浏览器中spellcheck属性的表现形式。

图2-3 Opera浏览器中spellcheck的属性示例

2.4.5 tabindex属性

tabindex是开发中的一个基本概念，当不断敲击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。

过去这个属性在编辑网页时是非常有用的，但如今控件的遍历顺序是由元素在页面上所处的位置决定的，所以就不再需要了。

但是tabindex还有另外一个作用，在默认情况下，只有链接元素与表单元素可以通过按键获得焦点。如果对其他元素使用tabindex属性后，也能让该元素获得焦点，那么当脚本中执行focus()语句的时候，就可以让该元素获得焦点了。但这样做会产生一个副作用：该元素也可以通过按Tab键获得焦点，而这有时可能不是开发者想要的结果。

把元素的tabindex值设为负数（通常为−1）后就解决这个问题了。tabindex的值为负数后，仍然可以通过编程的方式让元素获得焦点，但按下Tab键时该元素就不能获得焦点了，这在复杂的页面中或复杂的Web应用程序中是十分有用的。在HTML 4中，−1是一个无用的属性值，但到了HTML 5中，通过巧妙地运用让该属性值得到了极大的应用。

第3章
HTML 5的结构

在HTML 5对HTML 4所做的各种修改中，一个比较重大的修改就是为了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。本章将详细介绍这些新增的结构元素，包括它们的定义、使用方法以及使用示例，最后再介绍一下在HTML 5中，究竟怎样将这些新增的结构元素结合起来综合使用。

学习内容：

- 掌握HTML 5中新增的主体结构元素的定义、使用方法，以及使用场合。新增的主体结构元素包括article元素、section元素、nav元素及aside元素。
- 掌握HTML 5中新增的非主体结构元素的定义、使用方法，以及使用场合。新增的非主体结构元素包括header元素、hgroup元素、footer元素及address元素。
- 掌握HTML 5中应该怎样结合运用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，HTML 5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 新增的主体结构元素

在HTML 5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HTML 5中在页面的主体结构方面新增加的结构元素。

3.1.1 article元素

article元素代表文档、页面或应用程序中独立的、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，或其他任何独立的内容。

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。

现在，让我们以博客为例来看一段关于article元素的代码示例，如代码清单3-1所示。

代码清单3-1 article元素示例

```
<article>
    <header>
        <h1>苹果</h1>
        <p>发表日期：<time pubdate="pubdate">2010/10/09</time></p>
    </header>
    <p><b>苹果</b>，植物类水果，多次花果...（"苹果"文章正文）</p>
    <footer>
        <p><small>著作权归***公司所有。</small></p>
    </footer>
</article>
```

这个示例是一篇讲述苹果的博客文章，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“苹果”被镶嵌在h1元素中，文章的发表日期镶嵌在p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，一篇博客文章中，针对该文章的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。

接着，让我们来看一个关于article元素嵌套的代码示例，如代码清单3-2所示。

代码清单3-2 article元素嵌套示例

```
<article>
    <header>
        <h1>苹果</h1>
        <p>发表日期:
                <time pubdate datetime="2010/10/09">2010/10/09</time>
        </p>
    </header>
    <p><b>苹果</b> ，植物类水果，多次花果...（"苹果"文章正文）</p>
    <section>
        <h2>评论</h2>
        <article>
            <header>
                <h3>发表者：陆凌牛</h3>
                <p>
                    <time pubdate datetime="2010-10-10T19:10-08:00">
                        1小时前
                    </time>
                </p>
            </header>
            <p>我喜欢苹果，我最喜爱的品种是红富士。</p>
        </article>
        <article>
            <header>
                <h3>发表者：张玉</h3>
                <p>
                    <time pubdate datetime="2010-10-10T19:15-08:00">
                        1小时前
                    </time>
                </p>
            </header>
            <p>苹果？我不喜欢，我喜欢吃橘子。</p>
        </article>
    </section>
</article>
```

这个示例中的内容比代码清单3-1中的内容更加完整了，它添加了文章读者的评论内容，示例的整体内容还是比较独立、完整的，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（section元素马上就要讲到，是一个分块元素，用来把页面中的内容进行分块），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。

关于示例中提到的time元素与pubdate属性，请查看本节结尾处有关time元素与pubdate属性的说明。

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。代码清单3-3是它的一个示例。

代码清单3-3 用article元素表示插件

```
<article>
    <h1>My Fruit Spinner</h1>
    <object>
        <param name="allowFullScreen" value="true">
        <embed src="#" width="600" height="395"></embed>
    </object>
</article>
```

3.1.2 section元素

section元素用于对网站或应用程序中页面上的内容进行分块。一个section元素通常由内容及其标题组成。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。

我们可以这样理解：section元素中的内容可以单独存储到数据库中或输出到Word文档中。代码清单3-4是它的一个示例（注意：标题部分位于它的内部，而不是它的前面）。

代码清单3-4 section元素示例

```
<section>
    <h1>苹果</h1>
    <p><b>苹果</b> ，植物类水果，多次花果...（"苹果"文章正文）</p>
</section>
```

通常不推荐为那些没有标题的内容使用section元素，可以使用HTML 5轮廓工具来检查页面中是否有没标题的section，HTML 5轮廓工具的网址为“http://gsnedders.html5.org/outliner/”，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元素或aside元素没有标题是合理的）。

section元素的作用是对页面上的内容进行分块，或者说对文章进行分段，请不要与“有着自己的完整的、独立的内容”的article元素混淆。

下面，我们来看article元素与section元素结合使用的两个示例，希望能够帮助你更好地理解article元素与section元素的区别。

首先来看一个带有section元素的article元素示例，如代码清单3-5所示。

代码清单3-5 带有section元素的article元素示例

```
<article>
    <h1>苹果</h1>
    <p><b>苹果</b> ，植物类水果，多次花果...</p>
    <section>
        <h2>红富士</h2>
        <p>红富士是从普通富士的芽(枝)变中选育出的着色系富的统称...</p>
    </section>
    <section>
        <h2>国光</h2>
        <p>国光苹果品，又名小国光、万寿。原产美国，1600年发现的偶然实生苗...</p>
    </section>
</article>
```

代码清单3-5中的内容首先是一段独立的、完整的内容，因此使用article元素。该内容是一篇关于苹果的文章，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。请记住，对文章分段的工作也是使用section元素完成的。可能有人会问，为什么没有对第一段使用section元素，这里其实是可以使用section元素的，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元素了。

接着，我们来看一个包含article元素的section元素示例，如代码清单3-6所示。

代码清单3-6 包含article元素的section元素示例

```
<section>
    <h1>水果</h1>
    <article>
        <h2>苹果</h2>
        <p>苹果，植物类水果，多次花果...</p>
    </article>
    <article>
        <h2>橘子</h2>
        <p>橘子，是芸香科柑橘属的一种水果...</p>
    </article>
    <article>
        <h2>香蕉</h2>
        <p>香蕉,属于芭蕉科芭蕉属植物，又指其果实，热带地区广泛栽培食用...</p>
    </article>
</section>
```

这个示例比前面的示例复杂了一些，首先，它是一篇文章中的一段，因此最初没有使用article元素。但是，在这一段中有几块独立的内容，因此，嵌入了几个独立的article元素。

看到这里，你可能又会糊涂了，这两个元素可以互换使用吗？它们的区别到底是什么呢？事实上，在HTML 5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，但是如果你想将一块内容分成几段的时候，应该使用section元素。另外，在HTML 5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。

另外再补充一点，在HTML 5中，你可以将所有页面的从属部分，譬如导航条、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。

最后，关于section元素的使用禁忌总结如下：

1）不要将section元素用作设置样式的页面容器，那是div元素的工作。

2）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。

3）不要为没有标题的内容区块使用section元素。

3.1.3 nav元素

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、版权声明等，这时使用footer元素是最恰当。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。

接着让我们来看一个nav元素的使用示例，在这个示例中，一个页面由几部分组成，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，如代码清单3-7所示。

代码清单3-7 nav元素示例

```
<body>
<h1>技术资料</h1>
<nav>
    <ul>
        <li><a href="/">主页</a></li>
        <li><a href="/events">开发文档</a></li>
        ...more...
    </ul>
</nav>
<article>
   <header>
        <h1>HTML 5与CSS 3的历史</h1>
        <nav>
            <ul>
                <li><a href="#HTML 5">HTML 5的历史</a></li>
                <li><a href="#CSS 3">CSS 3的历史</a></li>
                ...more...
            </ul>
        </nav>
    </header>
    <section id="HTML 5">
        <h1>HTML 5的历史</h1>
        <p>讲述HTML 5的历史的正文</p>
    </section>
    <section id="CSS 3">
        <h1>CSS 3的历史</h1>
        <p>讲述CSS 3的历史的正文</p>
    </section>
```

```
    ...more...
    <footer>
        <p>
            <a href="?edit">编辑</a> |
            <a href="?delete">删除</a> |
            <a href="?rename">重命名</a>
        </p>
    </footer>
</article>
<footer>
    <p><small>版权所有：陆凌牛</small></p>
</footer>
</body>
```

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（跳转到网站主页或开发文档目录页面）；第二个nav元素放置在article元素中，用作这篇文章中两个组成部分的页内导航。

具体来说，nav元素可以用于以下这些场合：

- 传统导航条。现在主流网站上都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。

除此之外，nav元素也可以用于其他所有你觉得是重要的、基本的导航链接组中。

请注意：在HTML 5中不要用menu元素代替nav元素。过去有很多Web应用程序的开发员喜欢用menu元素进行导航，我想有必要再次强调，menu元素是用在一系列发出命令的菜单上的，是一种交互性的元素，或者更确切地说是使用在Web应用程序中的。

3.1.4 aside元素

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。

1）被包含在article元素中作为主要内容的附属信息部分，其中的内容可以是与当前文章有关的参考资料、名词解释，等等。这部分代码请看代码清单3-8。

代码清单3-8 文章内部的aside元素示例

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>aside元素示例</title>
</head>
<body>
```

```
<header>
    <h1>F#入门</h1>
</header>
<article>
    <h1>第四节 词法闭包</h1>
    <p>lambda表达式可以创建词法闭包...(文章正文) </p>
    <aside>
        <!-- 因为这个aside元素被放置在一个article元素内部，
        所以分析器将这个aside元素的内容理解成是和article元素的内容相关联的。 -->
        <h1>名词解释</h1>
        <dl>
            <dt>F#</dt>
            <dd>F#为.Net2010中引入的新型函数型编程语言</dd>
        </dl>
        <dl>
            <dt>词法闭包</dt>
            <dd>词法闭包是指，将创建lambda表达式时的环境保存起来...(详细解释) </dd>
        </dl>
    </aside>
</article>
</body>
```

程序运行结果如图3-1所示。

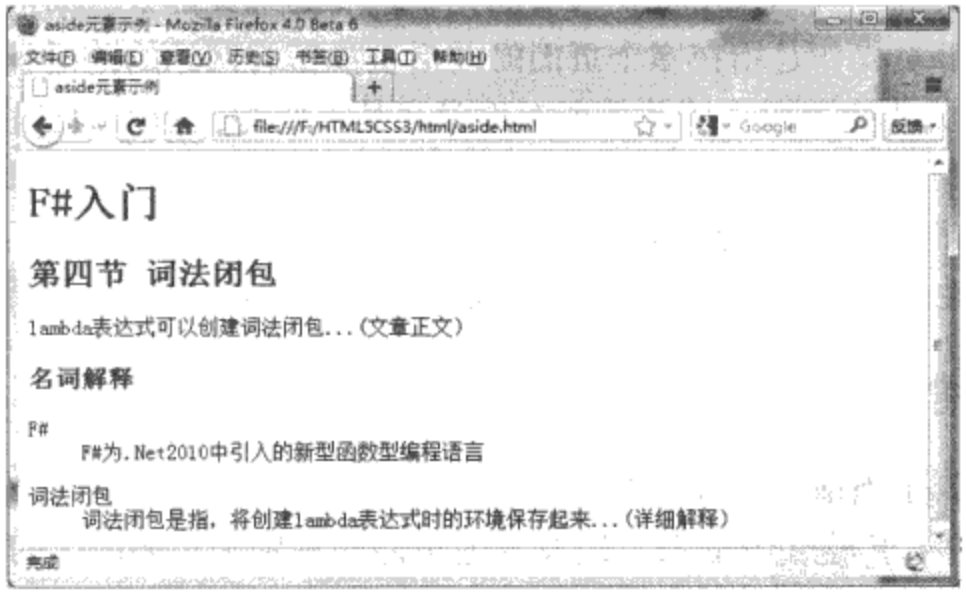

图3-1 aside元素示例

这是笔者博客网页中的一篇文章，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。

2）在article元素之外使用，作为页面或站点全局的附属信息部分。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。下面这个示例为标准博客网页中一个侧边栏的示例，示例中的“IT新技术”为博客的名称，如代码清单3-9所示。

代码清单3-9 侧边栏示例

```
<aside>
    <nav>
        <h2>评论</h2>
        <ul>
            <li>
            <a href="http://blog.sina.com.cn/1683">erway</a>     10-24 14:25
            </li>
            <li>
            <a href="http://blog.sina.com.cn/u/1345">太阳雨</a>10-22 23:48<br/>
            <a href="http://blog.sina.com.cn/s/blog_6a9kv8f.html#comment">
               顶，拜读一下老牛的文章
             </a>
            </li>
            <li>
            <a href="http://blog.sina.com.cn/u/1259295385">新浪官博</a>
            08-12 08:50<br/>
            <a href="#">恭喜！您已经成功开通了博客</a>
            </li>
        </ul>
    </nav>
</aside>
```

如果对这部分再加上CSS样式，在浏览器中的显示效果如图3-2所示。

该示例为一个典型的博客网站中的侧边栏部分，因此放在了aside元素中，但是该侧边栏又是具有导航作用的，因此放置在nav元素中，该侧边栏的标题是“评论”，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中。

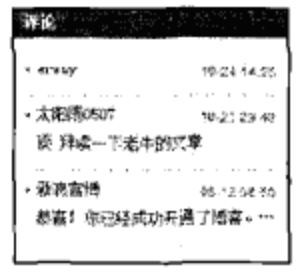

图3-2 用aside元素实现的侧边栏示例

3.1.5 time元素与微格式

首先来说一下微格式，它是一种利用HTML的class属性来对网页添加附加信息的方法，附加信息例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HTML 5之后才有的，在HTML 5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。HTML 5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="2010-11-13">2010年11月13日</time>
<time datetime="2010-11-13">11月13日</time>
<time datetime="2010-11-13">我的生日</time>
<time datetime="2010-11-13T20:00">我生日的晚上8点</time>
<time datetime="2010-11-13T20:00Z">我生日的晚上8点</time>
<time datetime="2010-11-13T20:00+09:00">我生日的晚上8点的美国时间</time>
```

编码时机器读到的部分在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”文字分隔，“T”表示时间。请注意倒数第二行，时间加上Z文字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。

3.1.6 pubdate属性

pubdate属性是一个可选的、boolean值的属性，它可以用到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，pubdate属性的具体使用方法如代码清单3-10所示。

代码清单3-10 pubdate与time结合使用

```
<article>
    <header>
        <h1>苹果</h1>
        <p>发布日期
            <time datetime="2010-10-29" pubdate>2010年10月29日</time>
        </p>
    </header>
   <p>苹果，植物类水果，多次花果...（"苹果"文章正文）</p>
  ...
</article>
```

你也许会疑惑为什么需要用到pubdate属性，为什么不能认为time元素就直接表示了文章或网页的发布日期呢？请看代码清单3-11。

代码清单3-11 pubdate与time结合使用

```
<article>
    <header>
        <h1>关于<time datetime=2010-10-29>10月29日</time>的舞会通知</h1>
        <p>发布日期：
            <time datetime=2010-10-11 pubdate>2010年10月11日</time>
        </p>
    </header>
    <p>大家好：我是法律系3年级学生代表，.....（关于舞会的通知）</p>
</article>
```

在这个例子中，有两个time元素，分别定义了两个日期——一个是舞会日期，另一个是通知发布日期。由于都使用了time元素，所以需要使用pubdate属性表明哪个time元素代表了通知的发布日期。

3.2 新增的非主体结构元素

除了以上几个主要的结构元素之外，HTML 5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。下面分别来介绍。

3.2.1 header元素

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，例如数据表格、搜索表单或相关的logo图片。

很明显，整个页面的标题应该放在页面的开头，我们可以用如下所示的形式书写页面的标题：

```
<header><h1>页面标题</h1></header>
```

需要强调的一点是：一个网页内并未限制header元素的个数，可以拥有多个，可以为每个内容区块加一个header元素，如代码清单3-12所示。

代码清单3-12 多个header元素示例

```
<header>
    <h1>网页标题</h1>
</header>
<article>
    <header>
        <h1>文章标题</h1>
    </header>
    <p>文章正文</p>
</article>
```

在HTML 5中，一个header元素通常包括至少一个heading元素（h1-h6），也可以包括我们后面将要讨论的hgroup元素，还可以包括其他元素（譬如table或form），根据最新的W3C HTML 5标准，还可以包括nav元素。最后，让我们看一下博客网页中header元素的一个应用示例。示例中header元素处于页面顶部。详见代码清单3-13。

代码清单3-13 博客网页中header元素的示例

```
<header>
<hgroup>
<h1>IT新技术</h1>
<a href="http://blog.sina.com.cn/itnewtech">
http://blog.sina.com.cn/itnewtech
</a>
<a href="#">[订阅]</a>
<a href="#">[手机订阅]</a>
</hgroup>
<nav>
<ul>
<li>首页</li>
<li><a href="http://blog.sina.com.cn/articlelist1.html">博文目录</a></li>
<li><a href="http://photo.blog.sina.com.cn/itnewtech1">图片</a></li>
<li><a href="http://photo.blog.sina.com.cn/itnewtech">关于我</a></li>
</nav>
</header>
```

如果对这段代码使用CSS样式，显示界面如图3-3所示。

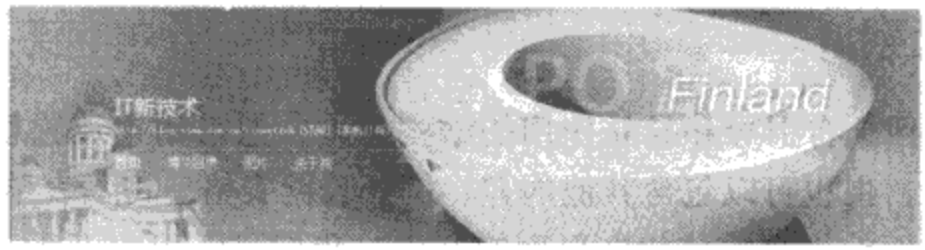

图3-3 博客网页中header元素示例

3.2.2 hgroup元素

hgroup元素是将标题及其子标题进行分组的元素。hgroup元素通常会将h1～h6元素进行分组，譬如一个内容区块的标题及其子标题算一组。

通常，如果文章只有一个主标题，是不需要hgroup元素的，如代码清单3-14所示。

代码清单3-14 只有header元素的示例

```
<article>
    <header>
        <h1>文章标题</h1>
        <p><time datetime="2010-03-20">2010年10月29日</time></p>
    </header>
    <p>文章正文</p>
</article>
```

但是，如果文章有主标题，主标题下有子标题，就需要使用hgroup元素了，如代码清单3-15所示。

代码清单3-15 hgroup元素示例

```
<article>
    <header>
        <hgroup>
            <h1>文章主标题</h1>
            <h2>文章子标题</h2>
        </hgroup>
        <p><time datetime="2010-03-20">2010年10月29日</time></p>
    </header>
    <p>文章正文</p>
</article>
```

3.2.3 footer元素

footer元素可以作为其上层父级内容区块或是一个根区块的脚注。footer通常包括其相关区块的脚注信息，如作者、相关阅读链接及版权信息等。

在HTML 5出现之前，我们使用下面的方式编写页脚，如代码清单3-16所示。

代码清单3-16 HTML 5之前的页脚示例

```
<div id="footer">
```

```
    <ul>
        <li>版权信息</li>
        <li>站点地图</li>
        <li>联系方式</li>
    </ul>
</div>
```

但是到了HTML 5之后，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，如代码清单3-17所示。

代码清单3-17 footer元素示例

```
<footer>
    <ul>
        <li>版权信息</li>
        <li>站点地图</li>
        <li>联系方式</li>
    </ul>
</footer>
```

与header元素一样，一个页面中也未限制footer元素的个数。同时，可以为article元素或section元素添加footer元素，请看下面两个示例。

代码清单3-18为一个在article元素中添加footer元素的示例。

代码清单3-18 在article元素中添加footer元素

```
<article>
    文章内容
    <footer>
        文章的脚注
    </footer>
</article>
```

代码清单3-19为一个在section元素中添加footer元素的示例。

代码清单3-19 在section元素中添加footer元素

```
<section>
    分段内容
    <footer>
        分段内容的脚注
    </footer>
</section>
```

3.2.4 address元素

address元素用来在文档中呈现联系信息，包括文档作者或文档维护者的名字、他们的网站链接、电子邮箱、真实地址、电话号码等。address应该不只是用来呈现电子邮箱或真实地址，还应用来展示跟文档相关的联系人的所有联系信息。譬如，在代码清单3-20中，展示了一些博客中某篇文章评论者的名字及其在博客中的网址链接。

代码清单3-20 address元素示例

```
<address>
    <a href=http://blog.sina.com.cn/itnewtech>陆凌牛</a>
    <a href=http://blog.sina.com.cn/zhangyu>张玉</a>
    <a href=http://blog.sina.com.cn/baiquanli>白权立</a>
</address>
```

下面我们通过代码清单3-21来看如何把footer元素、time元素与address元素结合起来使用。

代码清单3-21 footer、time与address结合使用的例子

```
<footer>
    <div>
        <address>
            <a title="文章作者：陆凌牛" href="http://blog.sina.com.cn/itnewtech">
            陆凌牛</a>
        </address>
        发表于<time datetime="2010-10-04">2010年10月4日</time>
    </div>
</footer>
```

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。

3.3 HTML 5结构

前面两节中详细介绍了在HTML 5中具体新增了哪些结构元素，以及这些元素的定义和使用方法。接下来，让我们看一下在HTML 5中进行总体页面布局的时候，具体应该怎样来综合运用这些结构元素。

3.3.1 大纲

通过使用新的结构元素，HTML 5的文档结构比大量使用div元素的HTML 4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或屏幕阅读程序来说，都很清晰易读的文档结构。

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”与“隐式编排”两种方式。

1. 显式编排内容区块

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），如代码清单3-22所示。

代码清单3-22 显式编排内容区块示例

```
<body>
    <h1>网页级内容区块标题</h1>
    <p>网页级内容区块的正文</p>
    <section>
        <h2>section级内容区块的标题</h2>
        <p>section级内容区块的正文</p>
    </section>
</body>
```

2. 隐式编排内容区块

所谓隐式编排，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HTML 5分析器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。代码清单3-23为一个隐式编排内容区块的示例。

代码清单3-23 隐式编排内容区块示例

```
<body>
    <h1>网页级内容区块标题</h1>
    <p>网页级内容区块的正文</p>
    <!--分析器根据h2等元素判断生成内容区块-->
    <h2>section级内容区块的标题</h2>
    <p>section级内容区块的正文</p>
</body>
```

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。

3. 标题分级

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。

第一条规则的示例与前面一样，现在我们来看关于第二条规则的示例，如代码清单3-24所示。

代码清单3-24 第二条规则示例

```
<body>
<section>
    <h2>section级别的内容区块的标题</h2>
    <p>section级别的内容区块的正文</p>
    <!—因为下面的标题级别比上一个标题级别高，所以自动创建新的内容区块 -->
    <h1>新的section级别的内容区块的标题</h1>
    <p>新的section级别的内容区块的正文</p>
</section>
</body>
```

如果把上一个示例改成显式编排，如代码清单3-25所示。

代码清单3-25 第二条规则的显式编排示例

```
<body>
<section>
```

```
    <h2>section级别的内容区块的标题</h2>
    <p>section级别的内容区块的正文</p>
</section>
<section>
    <h1>新的section级别的内容区块的标题</h1>
    <p>新的section级别的内容区块的正文</p>
</section>
</body>
```

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，所以请尽量使用显式编排。

4. 不同的内容区块可以使用相同级别的标题

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。这样做的好处是：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”（譬如书写文档时），这样做将会带来很大的便利性，如代码清单3-26所示。

代码清单3-26 不同的内容区块可以使用相同级别的标题

```
<body>
<h1>网页的标题</h1>
<article>
    <header>
        <hgroup>
            <h1>文章标题</h1>
            <h2>文章子标题</h2>
        </hgroup>
        <p>文章正文</p>
    </header>
</article>
</body>
```

5. 网页编排示例

基于以上讲解过的知识点，让我们来看应该怎样编排网页的内容。代码清单3-27为一个标准博客网页的示例，在这个示例中，具备了一个标准博客网页所需具备的基本要素，只缺少为了使用样式而补充添加的div元素。

代码清单3-27 网页编排示例

```
<!DOCTYPE html>
<head>
  <title>网页编排示例</title>
  <meta charset="UTF-8">
</head>
<body>
<!-- 网页标题 -->
<header>
    <h1>网页标题</h1>
    <!-- 网站导航链接 -->
    <nav>
```

```
        <ul>
            <li><a href="index.html">首页</a></li>
            <li><a href="help.html">帮助</a></li>
        </ul>
    </nav>
</header>
<!-- 文章正文 -->
<article>
    <hgroup>
        <h1>文章主标题</h1>
        <h2>文章子标题</h2>
    </hgroup>
    <p>文章正文</p>
    <!--文章评论 -->
    <section class="comments">
        <article>
            <h1>评论标题</h1>
            <p>评论正文</p>
        </article>
    </section>
</article>
<!-- 版权信息 -->
<footer>
    <small>版权所有：陆凌牛</small>
</footer>
</body>
```

在这个示例中，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HTML 5中，推荐使用这种方式。

3.3.2 对新的结构元素使用样式

因为很多浏览器尚未对HTML 5中新增的结构元素提供支持，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HTML 5中新增元素都是以块方式显示的，如下所示。

```
//追加block声明
article, aside, dialog, figure, footer, header, legend, nav, section { display: block; }
//正常使用样式
nav{float;left;width:20%;}
article{float:right;width:79%;}
```

另外，IE 8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Internet Explorer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：

```
//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;width:20%;}
article{float:right;width:79%;}
</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什么不良影响。另外，到了IE 9之后，这段脚本就不需要了。

3.3.3 article元素的样式

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：

```
header{display:block;color:red;text-align:right;}
aritcle header{color:blue;text-align:center;}
```

第4章

表单与文件

在开发Web应用程序的过程中，表单是页面上非常重要的一块内容，用户可以输入的大部分内容都是在表单的元素中完成的，它与后台的交互在大多数情况下也是通过点击表单中的按钮来完成的。在HTML 5中，大大加强了有关于表单这一部分的功能。本章将详细介绍在HTML 5中新增的表单元素、属性，以及对表单元素内容的有效性进行验证的功能，同时也会介绍在HTML 5的页面上，除了表单元素之外，其他的新增与改良的元素，最后会介绍在HTML 5中新增的与表单元素相关的两个API——文件API和拖放API。

学习内容：

- 掌握HTML 5中新增的表单中元素可以使用的属性及它们的使用方法。
- 掌握HTML 5中新增的表单元素及它们的使用方法。
- 掌握HTML 5中新增的关于表单内元素内容的有效性的验证方法，包括属性验证、显式验证、取消验证，以及自定义验证信息。
- 掌握HTML 5中除了表单以外，在页面上新增及改良的元素，以及它们的使用方法。
- 掌握文件API，包括掌握File对象与FileList对象的使用方法，掌握Blob对象的概念和使用方法，掌握FileReader对象以及它的方法、事件定义、事件触发条件，以及事件发生的先后顺序。
- 掌握怎样利用拖放API使页面中的元素可以互相拖放，掌握DataTransfer对象的属性和方法，掌握怎样设定拖放时的视觉效果，以及怎样自定义拖放图标。

4.1 新增元素与属性

在创建Web应用程序的时候，免不了会用到大量的表单元素。在HTML 5标准中，吸纳了Web Forms 2.0的标准，大幅度强化了针对表单元素的功能,使得关于表单的开发更快、更方便。

4.1.1 新增属性

首先，让我们看一下HTML 5中关于表单新增了哪些诱人的属性，以及现在有哪些浏览器支持了这些属性。

1. form属性

在HTML 4中，表单内的从属元素必须书写在表单内部，但是在HTML 5中，可以把它们书写在页面上任何地方，然后给该元素指定一个form属性，属性值为该表单的id，这样就可以声明该元素从属于指定表单了。form属性的使用示例如代码清单4-1所示。

代码清单4-1 form属性示例

```
<form id="testform">
<input type="text">
</form>
<textarea form="testform"></textarea>
```

input元素从属于foo表单，它被书写在表单内部，用不着再对它指定form属性。textarea

元素被书写在foo表单之外，但它从属于foo表单，所以将foo表单的id指定给textarea元素的form属性。

这样做的好处是当需要给页面中的元素添加样式时可以更方便地添加，因为它们不是被分散在各表单之内的了。

到目前为止只有Opera 10浏览器支持这一属性。

2. formaction属性

在HTML 4中，一个表单内的所有元素都只能通过表单的action属性统一提交到另一个页面，而在HTML 5中可以给所有的提交按钮，诸如<input type="submit">、<input type="image">、<button type="submit">都增加不同的formaction属性，使得点击不同的按钮，可以将表单提交到不同的页面，formaction属性的使用方法如代码清单4-2所示。

代码清单4-2 formaction属性示例

```
<form id="testform" action="serve.jsp">
   <input type="submit" name="s1" value="v1" formaction="s1.jsp">提交到S1
   <input type="submit" name="s2" value="v2" formaction="s2.jsp">提交到S2
   <input type="submit" name="s3" value="v3" formaction="s3.jsp">提交到S3
   <input type="submit">
</form>
```

目前尚没有浏览器支持这一属性。

3. formmethod属性

在HTML 4中，一个表单内只有一个action属性来对表单内所有元素统一指定提交页面，所以每个表单内也只有一个method属性来指统一指定提交方法。在HTML 5中，可以使用formaction属性来对每个表单元素分别指定不同的提交页面，同时也可以使用formmethod属性来对每个表单元素分别指定不同的提交方法，formmethod属性的使用方法如代码清单4-3所示。

代码清单4-3 formmethod属性示例

```
<form id="testform" action="serve.jsp">
   <input type="submit" name="s1" value="v1" formaction="s1.jsp" formmethod="post">提交到S1
   <input type="submit" name="s2" value="v2" formaction="s2.jsp" formmethod="get">提交到S2
   <input type="submit">
</form>
```

同formaction属性一样，目前尚没有任何浏览器支持该属性。

4. placeholder属性

placeholder是指当文本框（<input type="text">或<textarea>）处于未输入状态时文本框中显示的输入提示。如图4-1所示，当文本框处于未输入状态并且未获取光标焦点时，模糊显示输入提示文字。

图4-1 placeholder属性示例

实现方法非常简单，只要加上placeholder属性，然后指定提示文字就可以了。placeholder属性的使用方法如下所示。

```
<input type="text" placeholder="input me">
```

目前为止，支持该属性的浏览器有：Safari 4、Google Chrome 3、Firefox 4。

5. autofocus属性

给文本框、选择框或按钮控件加上该属性，当画面打开时，该控件自动获得光标焦点。目前为止要做到这一点需要使用JavaScript，譬如“control.focus()”。autofocus属性的使用方法如下所示。

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

一个页面上只能有一个控件具有该属性。从实用角度来说，请不要随便滥用该属性。强烈建议只有当一个页面是以使用某个控件为主要目的时，才对该控件使用autofocus属性，譬如搜索页面中的搜索文本框。

目前为止，支持该属性的浏览器有：Safari 4、Google Chrome 3、Firefox 4。

6. list属性

在HTML 5中，为单行文本框（<input type="text">）增加了一个list属性，该属性的值为某个datalist元素的id。datalist元素也是HTML 5中新增元素，该元素类似于选择框(select>)，但是当用户想要设定的值不在选择列表之内时，允许其自行输入。该元素本身并不显示，而是当文本框获得焦点时以提示输入的方式显示。为了避免在没有支持该元素的浏览器上出现显示错误，可以用CSS等将它设定为不显示。list属性的使用方法如代码清单4-4所示。

代码清单4-4 list属性示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>list属性示例</title>
</head>
text: <input type="text" name="greeting" list="greetings">
<!--使用style="display:none;"将detalist元素设定为不显示-->
<datalist id="greetings" style="display: none;">
    <option value="Good Morning">Good Morning</option>
    <option value="Hello">Hello</option>
    <option value="Good Afternoon">Good Afternoon</option>
</datalist>
```

这段代码运行结果如图4-2所示。

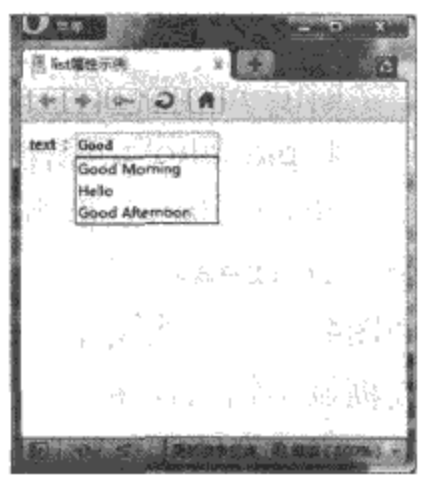

图4-2 list属性示例

到目前为止，只有Opera 10浏览器支持list属性。

为什么没有把input元素与datalist元素结合成一个元素，像其他语言中的可输入下拉框那样？这是基于兼容性的考虑——在不支持HTML 5的浏览器中，可以忽略datalist元素，以便正常输入及用脚本编程的方式对input元素执行其他操作。

7. autocomplete属性

辅助输入所用的自动完成功能，是一个节省输入时间，同时也十分方便的功能。在HTML 5之前，因为谁都可以看见输入的

值，所以存在安全隐患，但只要使用autocomplete属性，安全性就可以得到很好的控制。

对于autocomplete属性，可以指定“on”、“off”与“”（不指定）这三种值。不指定时，使用浏览器的默认值(取决于各浏览器的决定)。把该属性设为on时，可以显式指定候补输入的数据列表。使用detalist元素与list属性提供候补输入的数据列表，自动完成时，可以将该datalist元素中的数据作为候补输入的数据在文本框中自动显示。autocomplete属性的使用方法如下所示。

```
<input type="text" name="greeting" autocomplete="on" list="greetings">
```

到目前为止，只有Opera 10浏览器支持autocomplete属性。

4.1.2 大幅度地增加与改良了input元素的种类

在HTML 5中，大幅度地增加与改良了input元素的种类（见表4-1），可以简单地使用这些元素来实现HTML 5之前需要使用JavaScript才能实现的许多功能。

到目前为止，对于这些input的种类来说，支持得最多、最全面的是Opera 10浏览器。对于不支持新增input元素的浏览器来说，统一将这些input元素视为text类型。另外，HTML 5中也没有规定这些元素在各浏览器中的外观形式，所以同样的input元素在不同的浏览器中可能会有不同的外观。

表4-1 增加与改良的input元素

种类	说明	支持浏览器
search	与text文本框类似，但是它用于搜索，比如站点搜索或Google搜索	在Safari4浏览器中其外观与text不同，在其他浏览器中外观均与text相同
tel	与text文本框类似，但是专用于电话	在各浏览器中外观与text相同
url	与tex文本框t类似，但是要求用户必须在其中正确输入url格式的文字	Opera 10
email	与text文本框类似，但是要求用户必须在其中正确输入email格式的文字	Opera 10
datetime、date、month、week、time、datetime-local	各种日期与时间输入文本框	Opera 10
number	数值输入文本框。外观与text文本框相同，但不能输入数值以外的文字	Opera 10
range	只允许输入一段范围内数值的文本框。具有min属性与max属性，可以设定最小值与最大值（默认为0与100）。在Opera浏览器中，用滑动条的方式进行值的指定	Opera 10
color	颜色选择文本框。选择的值为“#000000”格式的文字	BlackBerry

（续）

种类	说明	支持浏览器
file	文件选择文本框。与HTML 4最大的不同是，可以通过指定multiple属性，一次选择多个文件。value属性的值为用逗号分割的一个或多个文件名。同时，通过把MIME类型指定给accept属性，可以限制选择文件的种类	所有最新版本的浏览器

下面，让我们针对这些input元素，来做一个比较全面的介绍。

1. url类型

url类型的input元素是一种专门用来输入url地址的文本框。提交时如果该文本框中内容不是url地址格式的文字，则不允许提交。url类型的input元素的使用方法如下所示。

```
<input name="url1" type="url" value="http://www.microsoft.com">
```

url类型的input元素在Opera 10浏览器中的外观如图4-3所示。

2. email类型

email类型的input元素是一种专门用来输入email地址的文本框。提交时如果该文本框中内容不是email地址格式的文字则不允许提交，但是它并不检查该email地址是否存在。提交时该文本框可以为空，除非加上了required属性。

email类型的文本框具有一个multiple属性——它允许在该文本框中输入一串以逗号分隔的email地址。当然，并不强制要求用户输入该email地址列表。在实际使用过程中，可以由开发者通过编程的方式将用户联系人地址列表中的邮件列表弹出，在每个联系人的邮件地址旁边带有复选框，供用户选择输入。email类型的input元素的使用方法如下所示。

```
<input name="email1" type="email" value="llingma2005@yahoo.com.cn">
```

email类型的input元素在Opera 10浏览器中的外观如图4-4所示。

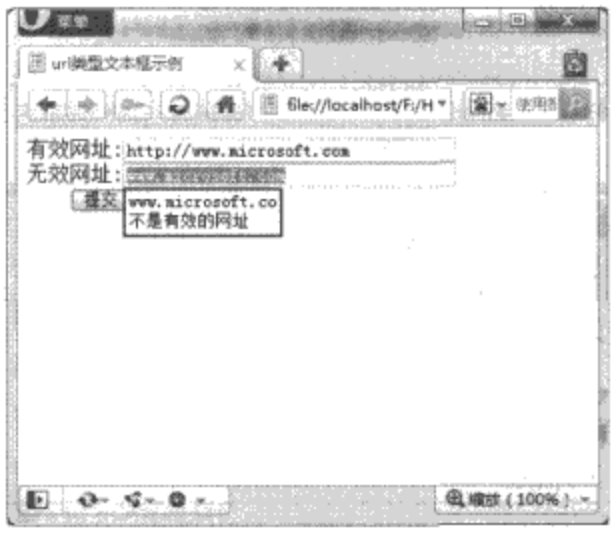

图4-3 url类型的input元素在Opera 10浏览器中的外观

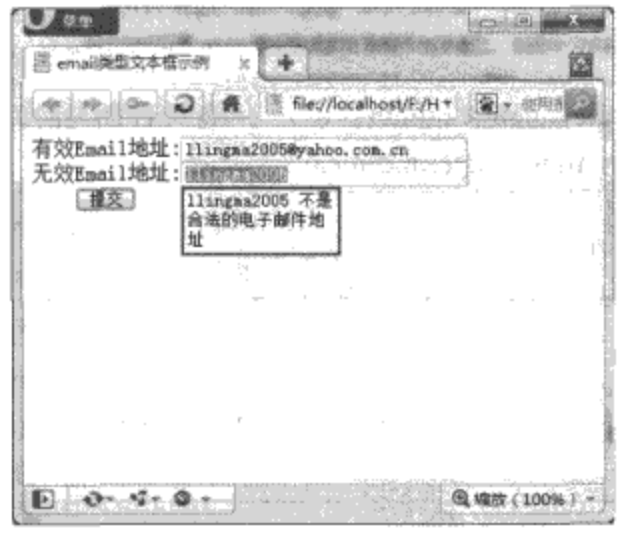

图4-4 email类型的input元素在Opera 10浏览器中的外观

3. date类型

date类型的input元素是深受开发者喜爱的一种元素，我们也经常看到网页中要求我们输入的各种各样的日期，例如生日、购买日期、订票日期等。date类型的input元素以日历的形式方便用户输入。在Opera浏览器中，当该文本框获得焦点时，显示日历，可以在日历中选择日期进行输入。date类型的input元素的使用方法如下所示。

```
<input name="date1" type="date" value="2010-10-02">
```

date类型的input元素在Opera 10浏览器中的外观如图4-5所示。

4. time类型

time类型的input元素是一种专门用来输入时间的文本框，并且在提交时会对输入时间的有效性进行检查。它的外观取决于浏览器，可能是简单的文本框，只在提交时检查是否在其中输入了有效的时间，也可能以时钟形式出现，还可以携带时区。time类型的input元素的使用方法如下所示。

```
<input name="time1" type="time" value="10:00">
```

time类型的input元素在Opera 10浏览器中的外观如图4-6所示。

图4-5　date类型的input元素在Opera 10浏览器中的外观

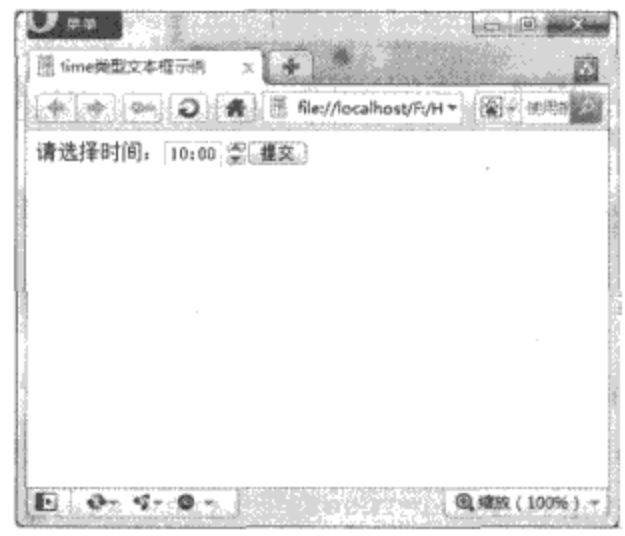

图4-6　time类型的input元素在Opera 10浏览器中的外观

5. datetime类型

datetime类型的input元素是一种专门用来输入UTC日期和时间的文本框，并且在提交时会对输入的日期和时间进行有效性检查。datetime类型的input元素的使用方法如下所示。

```
<input name="datetime1" type="datetime">
```

datetime类型的input元素在Opera 10浏览器中的外观如图4-7所示。

6. datetime-local类型

datetime-local类型的input元素是一种专门用来输入本地日期和时间的文本框，并且在提交时会对输入的日期和时间进行有效性检查。datetime-local类型的input元素的使用方法如下所示。

```
<input name="datetime-loacal1" type="datetime-local">
```

datetime-local类型的input元素在Opera 10浏览器中的外观如图4-8所示。

图4-7 datetime类型的input元素在Opera 10浏览器中的外观

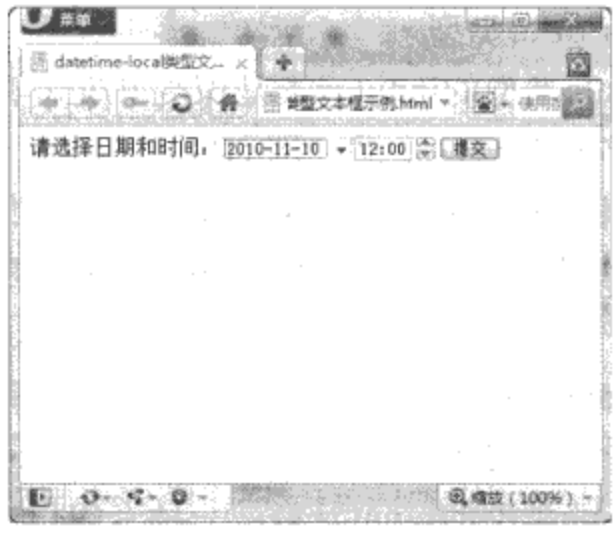

图4-8 datetime-local类型的input元素在Opera 10浏览器中的外观

7. month类型

month类型的input元素是一种专门用来输入月份的文本框，并且在提交时会对输入的月份的有效性进行检查。month类型的input元素的使用方法如下所示。

```
<input name="month1" type="month" value="2010-10">
```

month类型的input元素在Opera 10浏览器中的外观如图4-9所示。

8. week类型

week类型的input元素是一种专门用来输入周号的文本框，并且在提交时会对输入的周号之有效性进行检查。它可能是一个简单的输入文本框，允许用户输入一个数字；也可能更复杂、更精确。它的格式类似于“2010-W07”，代表2010年第7周。

Opera浏览器中提供了一个辅助输入的日历，可以在该日历中选取日期，选取完毕后文本框中自动显示周号。week类型的input元素的使用方法如下所示。

```
<input name="week1" type="week" value="2010-W40">
```

week类型的input元素在Opera 10浏览器中的外观如图4-10所示。

图4-9 month类型的input元素在Opera 10浏览器中的外观

图4-10 week类型的input元素在Opera 10浏览器中的外观

9. number类型

number类型的input元素是一种专门用来输入数字的文本框，并且在提交时会检查其中的内容是否为数字。它具有min、max与step属性。

在Opera浏览器中，它带有数值控制按钮，以控制其数值，使之不超过最大值与最小值，同时在点击该数值控制按钮时，其中的数值会按给定的步幅（step属性）进行增减，当然也可以直接在其中输入数字。number类型的input元素的使用方法如下所示。

```
<input name="number1" type="number" value="25" min="10" max="100" step="5">
```

number类型的input元素在Opera 10浏览器中的外观如图4-11所示。

10. range类型

range类型的input元素是一种只允许输入一段范围内数值的文本框，它具有min属性与max属性，可以设定最小值与最大值（默认为0与100），它还具有step属性，可以指定每次拖动的步幅。在Opera浏览器中，用滑动条的方式进行值的指定。range类型的input元素的使用方法如下所示。

```
<input name="range1" type="range" value="25" min="0" max="100" step="5">
```

range类型的input元素在Opera 10浏览器中的外观如图4-12所示。

图4-11　number类型的input元素在Opera 10浏览器中的外观

图4-12　range类型的input元素在Opera 10浏览器中的外观

11. search类型

search类型的input元素是一种专门用来输入搜索关键词的文本框。search类型与text类型仅仅在外观上有区别。在Safari 4浏览器中，它的外观为操作系统默认的圆角矩形文本框，但这个外观可以用CSS样式表进行改写。在其他浏览器中，它的外观暂与text类型的文本框外观相同，但可以用CSS样式表进行改写，如下所示。

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

12. tel类型

tel类型的input元素被设计为用来输入电话号码的专用文本框。它没有特殊的校验规则，

不强制输入数字（因为许多电话号码通常都带有其他文字），譬如86-0519-86670121。但是开发者可以通过pattern属性来指定对于输入的电话号码格式的验证。

13. color类型

color类型的input元素用来选取颜色，它提供了一个颜色选取器。现在，它只在BlackBerry浏览器中被支持。

14. 简单表单示例

在HTML 5中追加了这么多元素后，那么HTML 5最基本的表单是什么样子呢？这里拿一个网页上常用的简单用户注册页面来做例子。在该例子中，综合使用了HTML 5中新增的input元素，并对这些元素添加了必要的验证属性。详见代码清单4-5。

代码清单4-5 简单表单示例

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>简单表单示例</title>
<form name="form1">
<label for=username>姓名</label>
<input name=username id=username type=text required /><br/>
<label for=age>年龄</label>
<input name=age id=age type=number min=0 max=100/><br/>
<label for=birthday>出生日期</label>
<input name=birthday id=birthday type=date/><br/>
<label for=email>Email</label>
<input name=email id=email type=email required /><br/>
<label for=url>个人主页</label>
<input name=url id=url type=url /><br/>
<label for=memo>个人简介</label>
<textarea name=memo id=memo required ></textarea><br/>
<input type=submit>
</form>
```

此例中的代码在Opera 10浏览器中查看时显示效果最佳，此表单在网页中呈现最原始的表单外观，我们在开发时，通常会把各元素分配在table元素里，用以将元素对齐，同时对table元素使用样式。

4.1.3 对新的表单元素使用样式

HTML 5中新增加了上面提到的这么多表单元素，有人也许会问“那怎么对这些元素使用样式呢？”基本上与其他元素一样，字体、颜色等对于样式的编辑，基本上采取同样方法。但是如果你想把日历的背景改成浅蓝色，或者把number元素的增减调节按钮改大一点，或者修改错误信息的字体改成深蓝色之类的，这些修改是不可能的。到目前为止，还没有可以针对新元素的局部区域进行修改的样式。

4.1.4 output元素的追加

在HTML 5中，追加了新的元素output。output元素定义不同类型的输出，比如计算结果或脚本的输出。output元素必须从属于某个表单，也就是说，必须将它书写在表单内部，或者对它添加form属性。目前为止该元素只被Opera 10浏览器支持。

接下来，让我们用一个range元素来进行说明。见代码清单4-6。

代码清单4-6 output元素示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>output元素示例</title>
</head>
<form id="testform">
请选择一个数值：
<input name="range1" type=range min=0 max=100 step=5/>
<output onforminput="value=range1.value">50</output>
</form>
```

这段代码的运行结果如图4-13所示。

图4-13 output元素示例

在这个例子中，元素被绑定到一个range元素上，当拖动range元素的滑竿时，output元素的父表单会接收到消息，同时通知output元素，将它的被绑定元素range的值显示出来。可以对output元素使用样式（尽管现在只被Opera 10浏览器支持）。

4.2 表单验证

在HTML 5中，在增加了大量的表单元素与属性的同时，也增加了大量在提交时对表单与表单内新增元素进行内容有效性验证的功能，本节将针对这些验证进行详细介绍。

4.2.1 自动验证

在HTML 5中，通过对元素使用属性的方法，可以实现在表单提交时执行自动验证的功能。执行代码清单4-7后，将在表单提交时自动验证输入的内容是否为数字，如果验证通不

过，将显示错误信息文字。

代码清单4-7 表单验证示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>表单验证示例</title>
</head>
<body>
<form method="post">
    <input name="text" type="text" required pattern="^\w.*$">
    <input type="submit">
</form>
</body>
```

接下来，让我们详细看一下在HTML 5中追加的关于对元素内输入内容进行限制的属性的指定。

1. required属性

HTML 5中新增的required属性可以应用在大多数输入元素上（除了隐藏元素、图片元素按钮上）。在提交时，如果元素中内容为空白，则不允许提交，同时在浏览器中显示信息提示文字，提示用户这个元素中必须输入内容。如图4-14所示。

2. pattern属性

之前提到的一些新增的input元素，譬如email、number、url等，要求输入内容符合一定的格式，对input元素使用pattern属性，并且将属性值设为某个格式的正则表达式，在提交时会检查其内容是否符合给定格式。当输入的内容不符合给定格式时，则不允许提交，同时在浏览器中显示信息提示文字，提示输入的内容必须符合给定格式。譬如下面所示，要求输入内容为一个数字与三个大写字母。

```
<input pattern="[0-9][A-Z]{3}" name=part placeholder="输入内容:一个数字与三个大写字母。">
```

图4-15为在Opera浏览器中pattern属性的表现形式。

图4-14 Opera 10浏览器中的required属性检查示例

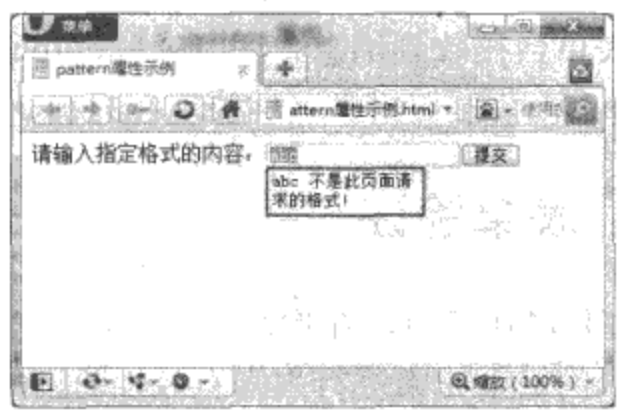

图4-15 Opera 10浏览器中的pattern属性检查示例

3. min属性与max属性

min与max这两个属性是数值类型或日期类型的input元素的专用属性，它们限制了在input元素中输入的数值与日期的范围。图4-16为在Opera浏览器中max属性的表现形式。

4. step属性

step属性控制input元素中的值增加或减少时的步幅。例如当你想让用户输入的值在0与100之间，但必须是5的倍数时，你可以指定step为5。图4-17为在Opera浏览器中step属性的表现形式。

图4-16 Opera 10浏览器中的max属性检查示例

图4-17 Opera 10浏览器中的step属性检查示例

4.2.2 显式验证

除了对inut元素添加属性进行元素内容有效性的自动验证外，在HTML 5中，form元素与input元素（包括select元素与textarea元素）都具有一个checkValidity方法。调用该方法，可以显式地对表单内所有元素内容或单个元素内容进行有效性验证。checkValidity方法以boolean的形式返回验证结果。请看代码清单4-8中的示例。

代码清单4-8 checkValidity方法调用

```
<!DOCTYPE html>
<meta charset=UTF-8 />
<title>checkValidity示例</title>
<script language="javascript">
function check()
{
    var email = document.getElementById("email");
    if(email.value=="")
    {
        alert("请输入Email地址");
        return false;
    }
    else if(!email.checkValidity())
        alert("请输入正确的Email地址");
    else
        alert("您输入的Email地址有效");
}
</script>
<form id=testform onsubmit="return check();">
<label for=email>Email</label>
<input name=email id=email type=email /><br/>
<input type=submit>
</form>
```

另外还要提到的是，在HTML 5中，form元素与input元素都还存在一个validity属性，该属性返回一个ValidityState对象。该对象具有很多属性，但最简单、最重要的属性为valid属性，它表示了表单内所有元素内容是否有效或单个input元素内容是否有效。

4.2.3 取消验证

有时我们可能想要把表单临时提交一下，但又不想让它进行表单中所有元素内容的有效性检查。譬如，一个非常大的表单需要分成两部分（或几部分），在第二部分中有个文本框中内容是必须要填的，如果填每一部分内容则会耗时较多，或填完第一部分之后，第二部分要过一段时间再填，在这种情况下应该允许用户先提交保存第一部分内容，但是同时需要临时取消第二部分的内容表单验证。

有两种方法取消表单验证，第一种方法是利用form元素的novalidate属性，它可以关闭整个表单验证。当整个表单的第二部分需要验证的内容比较多，但又想先提交表单的第一部分内容时，可以使用这种方法。先把该属性设为true，关闭表单验证，提交第一部分内容，然后在提交第二部分时再把它设为false，打开表单验证，提交第二部分内容。

第二种方法是利用input元素或submit元素的formnovalidate属性，利用input元素的formnovalidate属性可以让表单验证对单个input元素失效，在前面所举例子中，当表单的第二部分中需要验证的元素数量很少时，可以只利用这些元素的formnovalidate属性，让表单验证对这些元素失效。

而如果对submit按钮使用了formnovalidate属性，点击该按钮时，相当于利用了form元素的novalidate属性，整个表单验证都失效了。

利用这一点，可以实现”假提交”与”真提交”的效果，例如一个提交按钮，不带表单验证，提交时不进行数据有效性检查，提交时临时保存到文件或什么地方，另一个提交按钮为真提交，提交后保存到数据库中。

4.2.4 自定义错误信息

HTML 5中许多新的input元素都带有对于输入内容的有效性的检查，如果检查不通过，浏览器会针对该元素提供错误信息。但有时开发者不想使用这些默认的错误信息提示，而想使用自己定义的错误信息提示。或者有时，想给某个文本框增加一种错误信息提示，譬如密码与确认密码不一致时用浏览器错误信息提示方式提供关于密码不一致的错误信息。

在HTML 5中，可以使用JavaScript调用各input元素的setCustomValidity方法来自定义错误信息。请看代码清单4-9中的示例。

代码清单4-9 自定义错误信息示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>自定义错误信息示例</title>
```

```
<script language="javascript">
function check()
{
    var pass1=document.getElementById("pass1");
    var pass2=document.getElementById("pass2");
    if(pass1.value!=pass2.value)
        pass2.setCustomValidity("密码不一致。");
    else
        pass2.setCustomValidity("");
    var email=document.getElementById("email");
    if(!email.checkValidity())
        email.setCustomValidity("请输入正确的Email地址。");
}
</script>
<form id="testform" onSubmit="return check();">
密码：<input type=password name="pass1" id="pass1" /><br/>
确认密码：<input type=password name="pass2"  id="pass2"/><br/>
Email:<input type=email name="email1" id="email"/><br/>
<input type="submit" />
</form>
```

这段代码运行结果如图4-18所示。

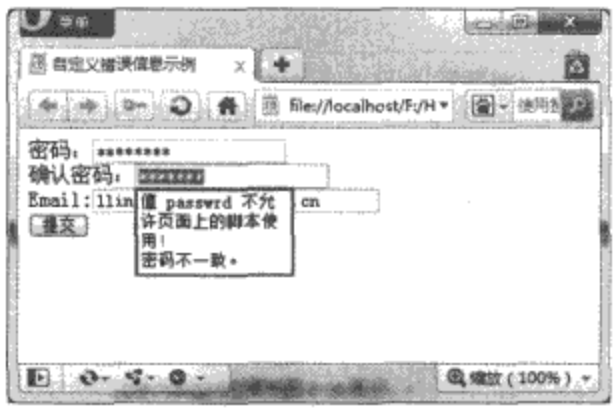

图4-18 Opera 10浏览器中自定义错误信息示例

在这个例子中，追加了两种错误信息提示。第一种情况为确认密码与密码不一致时，给确认密码文本框追加的自定义错误信息提示，浏览器提供的确认密码文本框本来没有这项检查内容。第二种情况为浏览器提供的Email文本框本来就有检查输入的Email是否符合Email格式的功能，但是开发者自行修改了浏览器默认的错误信息提示。

Opera是目前唯一支持自定义错误信息提示的浏览器。对于“值***不允许页面上的脚本使用！”这个前缀文字，也是该浏览器加在前面的，不能修改。所以，请小心使用自定义错误信息提示。

4.3 增强的页面元素

在HTML 5中，不仅增加了很多表单中的元素，同时也增加和改良了可以应用在整个页面中的元素，本节将针对这些元素进行介绍。

4.3.1 新增的figure元素与figcaption元素

figure元素也是一种元素的组合，带有可选标题。figure元素用来表示网页上一块独立的内容，将其从网页上移除后不会对网页上的其他内容产生任何影响。figure元素所表示的内容可以是图片、统计图或代码示例。

figcaption元素表示figure元素的标题，它从属于figure元素，必须书写在figure元素内部，可以书写在figure元素内的其他从属元素的前面或后面。一个figure元素内最多只允许放置一个figcaption元素,但允许放置多个其他元素。

代码清单4-10为一个不带标题的figure元素示例。

代码清单4-10 不带标题的figure元素示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>figure元素示例</title>
</head>
<figure>
  <img src="tyl.jpg" alt="谭咏麟">
</figure>
```

这段代码的运行结果如图4-19所示。

图4-19 不带标题的figure元素示例

代码清单4-11为将上面这个示例中的figure元素加上标题的示例。

代码清单4-11 带标题的figure元素示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>figure元素示例</title>
</head>
<figure>
    <img src="tyl.jpg" alt="谭咏麟">
    <figcaption>谭咏麟</figcaption>
</figure>
```

这段代码的运行结果如图4-20所示。

图4-20　带标题的figure元素示例

代码清单4-12为对多个图片使用同一个标题的示例。

代码清单4-12　多个图片使用同一个标题示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>figure元素示例</title>
</head>
<figure>
    <img src="tyll.jpg" alt="谭咏麟">
    <img src="zyf.jpg" alt="周润发">
    <img src="zxy.jpg" alt="张学友">
    <figcaption>我喜爱的明星</figcaption>
</figure>
```

这段代码的运行结果如图4-21所示。

图4-21　多个图片使用同一个标题示例

figure元素所表示的内容通常是图片、统计图或代码示例，但并不仅限于此，它同样可以用来表示音频插件、视频插件或统计表格等。

4.3.2　新增的details元素

details元素提供了一种替代Javascript的、将画面上局部区域进行展开或收缩的方法，不

过现在还没有任何浏览器对它提供支持。请看代码清单4-13中的这个示例。

代码清单4-13 details元素示例

```
<details>
   <summary>精武风云<summary>
   <p/>陈真（甄子丹 饰）当年为报杀师之仇，独闯虹口道场...(精武风云电影介绍）</p>
</details>
```

summary元素从属于details元素，用鼠标点击summary元素中的内容文字时，details元素中的其他所有从属元素将会展开或收缩。如果details元素内没有summary元素，浏览器会提供默认文字以供点击，譬如“details”或某些本地化文字（譬如“细节信息”），浏览器也提供一个诸如上下箭头之类的图标，标示该区域可以被展开或收缩。details元素具有一个属性“open”，使用“<details open>”语句对details添加该属性，添加该属性后在画面打开时details元素所表示的局部区域则会处于展开状态。details元素内并不限于放置文字，也可以在其内部放置表单、插件或对于一个统计图提供的详细数据表格。

4.3.3 新增的mark元素

mark元素表示页面中需要突出显示或高亮显示的，对于当前用户具有参考作用的一段文字。它通常使用于引用原文的时候，目的是引起读者的注意。mark元素是对原文内容具有补充作用的一个元素，它应该用于一段原文作者不认为重要，但为了与原文作者不相关的其他目的⊖而需要突出显示或高亮显示的文字上面。

mark元素最主要的目的是吸引当前用户的注意，因为标示出来的文字与用户的当前操作有关，通常该元素对于当前用户具有很好的帮助作用。

能够体现mark元素作用的最好例子是对网页全文检索某个关键词时显示的检索结果，现在许多搜索引擎用其他方法实现了mark元素所要达到的功能。

下面给出一个在浏览器中使用mark元素高亮显示对于“HTML 5”关键词搜索结果的示例，如代码清单4-14所示。

代码清单4-14 mark元素应用在网页检索时的示例

```
<!DOCTYPE html>
<meta charset="UTF-8" />
<title> mark元素应用在网页检索时的示例</title>
<h1>搜索"<mark>HTML 5<mark>",找到相关网页约10,200,000篇，用时0.041秒</h1>
<section id="search-results">
    <article>
        <h2>
            <a href="http://developer.51cto.com/art/200907/133407.htm">
                专题：<mark>HTML 5</mark> 下一代Web开发标准详解_51CTO.COM - 技术成就梦想 ...
            </a>
        </h2>
        <p><mark>HTML 5</mark>是近十年来Web开发标准最巨大的飞跃</p>
    </article>
```

⊖ “与原作者不相关的其他目的”通常都是和当前用户的操作有关，需要当前用户引起注意的一段文字。

```
    <article>
        <h2>
            <a href="http://paranimage.com/list-of-html-5/">
                <mark>HTML 5</mark>一览 | 帕兰映像
            </a>
        </h2>
        <p><mark>html 5</mark>最近被讨论的越来越多，越来越烈...</p>
    </article>
    <article>
        <h2>
            <a href="http://www.chinabyte.com/keyword/HTML+5/">
                <mark>html 5</mark>_比特网
            </a>
        </h2>
        <p><mark>HTML 5</mark>提供了一些新的元素和属性，反映典型...</p>
    </article>
    <article>
        <h2>
            <a href="http://www.slideshare.net/mienflying/HTML 5-4921810">
                <mark>HTML 5</mark>表单
            </a>
        </h2>
        <p>about <mark>HTML 5</mark> Form,the web form 2.0 tech</p>
    </article>
</section>
```

这段代码的运行结果如图4-22所示。

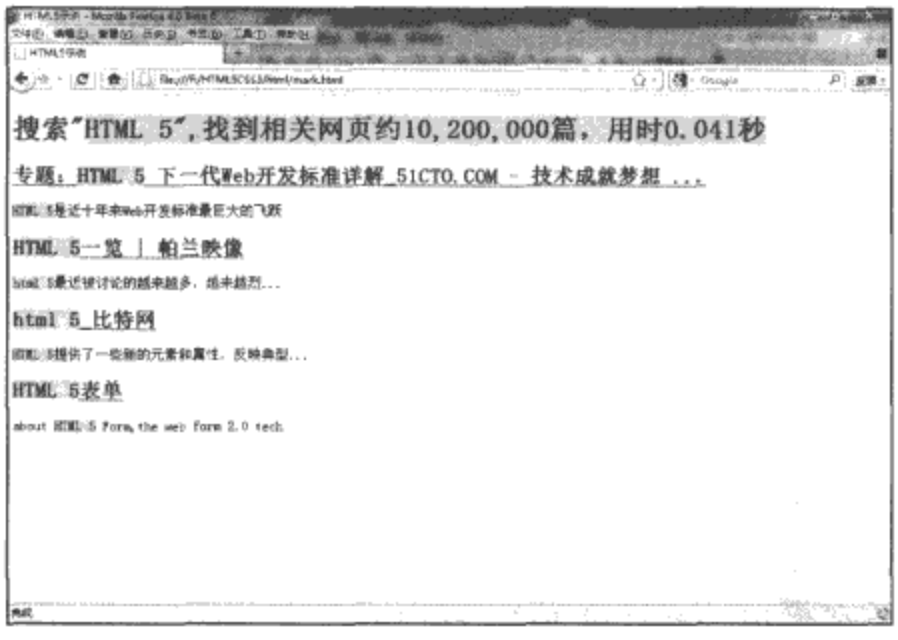

图4-22 mark元素应用在网页检索时的示例

除了在搜索结果中高亮显示关键词之外，mark元素的另一个主要作用是在引用原文的时候，为了某种特殊目的而把原文作者没有特别重点标示的内容给标示出来，请看代码清单4-15中的示例，该示例中引用了一段对刘德华的评论文章，原文中并没有把“汤姆克鲁斯”与“劳伦斯奥立佛”标示出来，但在网页中为了强调刘德华有些欧美人的韵味，特意把这两个词给高亮显示出来。

代码清单4-15 mark元素应用在文章引用时的示例

```
<!DOCTYPE html>
<meta charset=UTF-8 />
<title>mark元素应用在文章引用时的示例</title>
刘德华：光芒万丈的不老情人
<p>
有人说华仔是东方的<mark>汤姆·克鲁斯</mark>，是忧郁的王子<mark>劳伦斯·奥立佛</mark>。华仔那张以鹰钩鼻为核心的脸蛋的确有些欧美人的韵味，与一般的亚洲人的脸不太一样。
</p>
```

这段代码的运行结果如图4-23所示。

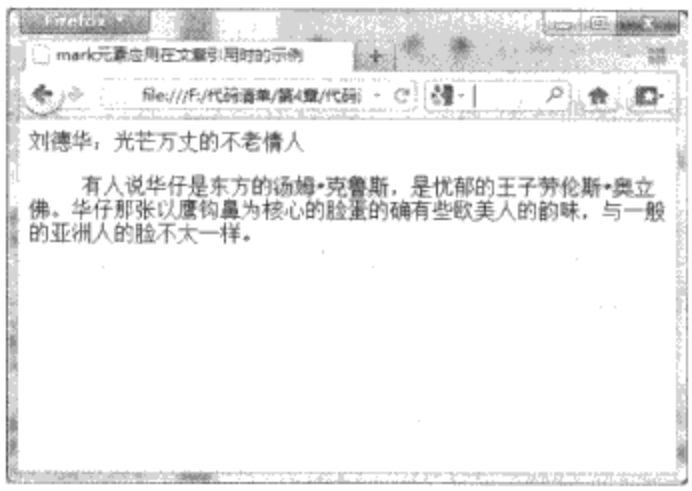

图4-23 mark元素应用在文章引用时的示例

最后需要强调的是：在HTML 4中，你可能已经习惯于用em元素或strong元素来突出显示文字，但请记住mark元素的作用与这两个元素的作用是有区别的，不能混同使用。mark元素的标示目的与原文作者无关，或者说它不是原文作者用来标示文字的，而是在后来引用时添加上去的，它的目的是吸引当前用户的注意力，提供给用户作参考，希望能够对用户有帮助。而strong是原文作者用来强调一段文字的重要性的，譬如警告信息、错误信息等，em元素是作者为了突出文章重点而使用的。

4.3.4 新增的progress元素

progress元素代表一个任务的完成进度，这个进度可以是不确定的，只是表示进度正在进行，但不清楚还有多少工作量没有完成，也可以用0到某个最大数字（譬如100）之间的数字来表示准确的进度完成情况（譬如进度百分比）。

该元素具有两个属性来表示当前任务完成情况，value属性表示已经完成了多少工作量，max属性表示总共有多少工作量。工作量的单位是随意的，不用指定。

在属性设定的时候，value属性和max属性只能指定为有效的浮点数，value属性的值必须大于0，且小于或等于max属性，max属性的值必须大于0。

下面在代码清单4-16中给出一个progress元素的使用示例。

代码清单4-16 progress元素的使用示例

```
<!DOCTYPE html>
<meta charset="UTF-8"/>
<title>progress元素的使用示例</title>
<script>
var progressBar = document.getElementById('p');
function button_onclick()
{
    var progressBar = document.getElementById('p');
    progressBar.getElementsByTagName('span')[0].textContent ="0";
    for(var i=0;i<=100;i++)
        updateProgress(i);
}
function updateProgress(newValue)
{
    var progressBar = document.getElementById('p');
    progressBar.value = newValue;
    progressBar.getElementsByTagName('span')[0].textContent = newValue;
}
</script>
<section>
    <h2>progress元素的使用示例</h2>
    <p>完成百分比: <progress id="p" max=100><span>0</span>%</progress></p>
    <input type="button" onclick="button_onclick()"  value="请点击"/>
</section>
```

这段代码的运行结果如图4-24所示。

图4-24 progress元素的使用示例

4.3.5 新增的meter元素

meter元素表示规定范围内的数量值。例如，磁盘使用量，对某个候选者的投票人数占总投票人数的比例等。

meter元素有如下六个属性：

- value：在元素中特地表示出来的实际值。该属性值默认为0，可以给该属性指定一个浮点小数值。
- min：指定规定的范围时允许使用的最小值，默认为0，设定该属性时设定的值不能小于0。

- max：指定规定的范围时允许使用的最大值，如果设定时该属性值小于min属性的值，那么把min属性的值视为最大值。max属性的默认值为1。
- low：规定范围的下限值。必须小于或等于high属性的值。同样的，如果low属性值小于min属性的值，那么把min属性的值视为low属性的值。
- high：规定范围的上限值。如果该属性值小于low属性的值，那么把low属性的值视为high属性的值，同样的，如果该属性值大于max属性的值，那么把max属性的值视为high属性的值。
- optimum：最佳值，属性值必须在min属性值与max属性值之间，可以大于high属性值。

meter属性的使用方法如下所示。

```
<p>磁盘使用量：<meter value="40" min="0" max="160">40/160</meter>GB</p>
<p>你的得分是：
<meter value="91" min="0" max="100" low="40" high="90" optimum="100">A+</meter>
</p>
<!--不是一定要使用属性-->
<meter>80%</meter>
<meter>3/4</meter>
<!—下面写法是不正确的，因为画面上什么都不显示-->
<meter min="0" max="100" value="75"></meter>
```

现在Safari 4、Firefox 4、Google Chrome 2、Opera 10浏览器都支持该属性。在IE 6/7/8中，需要在JavaScript脚本中添加如下代码，方可使用该元素。

```
<!--[if IE]>
<script src="http://HTML 5shiv.googlecode.com/svn/trunk/HTML 5.js"></script>
<![endif]-->
```

4.3.6 新增的menu元素与command元素

menu和command这两个元素是HTML 5中新增的用于Web应用程序的元素，它用于菜单、工具条及弹出菜单。其中menu元素相当于其他语言开发工具中的菜单，而command元素相当于其他开发语言工具中的菜单项。目前尚未有任何浏览器支持这两个元素，所以不做进一步介绍。

4.3.7 改良的ol列表

在HTML 5中，将ol列表进行了改良，为它添加了start属性与reversed属性。

如果你不想ol元素所代表的列表编号从1开始，那么可以使用star属性来自定义编号的初始值，如代码清单4-17中所示。

代码清单4-17 ol列表的star属性示例

```
<!DOCTYPE html>
<meta charset=UTF-8/>
<title>ol列表的star属性示例</title>
<h3>ol列表的star属性示例</h3>
<ol start=5>
```

```
<li>列表内容5</li>
<li>列表内容6</li>
<li>列表内容7</li>
<li>列表内容8</li>
<li>列表内容9</li>
<li>列表内容10</li>
</ol>
```

这段代码的运行结果如图4-25所示。

图4-25　ol列表的start属性示例

如果你想对列表进行反向排序，那么你可以使用ol列表的reversed属性，但是，现在还没有任何浏览器对该属性提供支持。该属性的使用方法如下所示。

```
<ol reversed>
```

4.3.8　改良的dl列表

在HTML 4中，dl元素是一个专门用来定义术语的列表，dl列表中包括几条术语定义，列表中的每一项包含一个术语及一个或多个对那条术语的定义，这些术语与名词解释是多对多的关系——一条术语可以有多个定义，但不同的术语又可能存在相同的定义，因此，从总体来说，这些术语的定义是模糊的、混乱的，经常被误解、错用或者根本就不被使用了。

在HTML 5中，将该元素进行重新定义，重新定义后的dl列表包含多个带名字的列表项。每一项包含一条或多条带名字的dt元素，用来表示术语，dt元素后面紧跟一个或多个dd元素，用来表示定义。在一个元素内，不允许有相同名字的dt元素，不允许有重复的术语。

dl列表可以用来定义文章或网页上的术语解释，如代码清单4-18中所示。

代码清单4-18　用dl列表来做术语解释

```
<!DOCTYPE html>
<meta charset=UTF-8/>
<title>用于术语解释的dl列表示例</title>
<h3>用于术语解释的dl列表示例</h3>
<article>
    <h1>aritcle元素</h1>
    <p>一块独立的内容。它可以用来表示RRS中一块独立的内容，也可以用来表示博客中独立的一篇文章。...</p>
    <aside>
        <h2>术语解释</h2>
```

```
            <dl>
                  <dt><dfn>RSS</dfn></dt>
                  <dd>RSS也叫聚合RSS是在线共享内容的一种简易方式(也叫聚合内容)...</dd>
                  <dt><dfn>博客</dfn></dt>
                  <dd>博客，又译为网络日志、部落格或部落阁等，是一种通常由个人管理...</dd>
            </dl>
      </aside>
</article>
```

这段代码的运行结果如图4-26中所示。

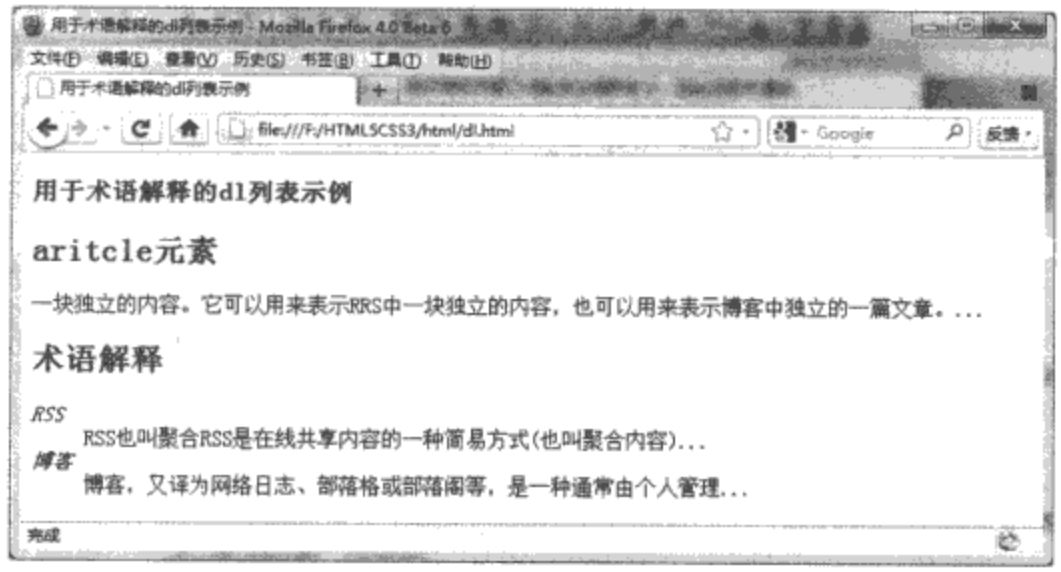

图4-26 用dl列表来做术语解释

dl列表也可以用来表示一些页面或article元素中内容的辅助信息，例如作者、类别等，如代码清单4-19中所示。

代码清单4-19 用dl列表来表示辅助信息

```
<dl>
<dt>作者</dt>
<dd>陆凌牛</dd>
<dt>出版社</dt>
<dd>机械工业出版社</dd>
<dt>类别</dt>
<dd>网络开发</dd>
</dl>
```

4.3.9 加以严格限制的cite元素

cite元素表示作品（例如一本书、一篇文章、一首歌曲等）的标题。该作品可以在页面中被详细引用，也可以只在页面中提一下。

在HTML 4中，cite元素可以用来表示作者，但在HTML 5中明确规定了不能用cite元素表示包括作者在内的任何人名，因为人的名字不是标题（当然除非标题就是一个人的名字），但是为了与HTML 4或之前版本的网页兼容，并没有把它当作错误，所以这只是一个规定而已。

代码清单4-20中的内容为一个使用cite元素的代码示例。

代码清单4-20 cite元素示例

```
<!DOCTYPE html>
<meta charset="UTF-8"/>
<title>cite元素示例</title>
<h3>cite元素示例</h3>
<p>我最喜欢的电影是由甄子丹主演的<cite>精武风云</cite>。</p>
```

这段代码的运行结果如图4-27所示。

图4-27 cite元素示例

4.3.10 重新定义的small元素

在HTML 5中，对small元素进行了重新定义，使其由原来的通用展示性元素变为更具体的、专门用来标识所谓“小字印刷体”的元素，通常用在诸如免责声明、注意事项、法律规定、与版权相关等的法律性声明文字中，同时不允许被应用在页面主内容中，只允许被当做辅助信息用inline方式内嵌在页面上使用。同时，small元素也不意味着元素中内容字体会变小，如果需要将字体变小，需要配合着CSS样式表来使用。

4.4 文件API

在HTML 5中，提供了一个关于文件操作的文件API（应用程序编程接口），通过使用这个API，对于从Web页面上访问本地文件系统的相关处理将会变得十分简单。本节将针对这个文件API做详细介绍。另外，关于文件API，到目前为止只有部分浏览器对它提供支持，譬如最新版的Firefox浏览器。

4.4.1 FileList对象与file对象

FileList对象表示用户选择的文件列表。在HTML 4中，file控件内只允许放置一个文件，但是到了HTML 5中，通过添加multiple属性，file控件内允许一次放置多个文件。控件内的每一个用户选择的文件都是一个file对象，而FileList对象则为这些file对象的列表，代表用户选择的所有文件。file对象有两个属性，name属性表示文件名，不包括路径，lastModifiedDate属性表示文件的最后修改日期。代码清单4-21为一个使用FileList对象与file对象的示例。

代码清单4-21 FileList与file使用示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>FileList与file示例</title>
</head>
<script language=javascript>
function ShowFileName()
{
    var file;
    //document.getElementById("file").files返回FileList文件列表对象
    for(var i=0;i<document.getElementById("file").files.length;i++)
    {
       //file对象为用户选择的单个文件
        file = document.getElementById("file").files[i];
        //此处您可以针对FileList文件列表中每个文件进行多种处理，本例中只弹出文件名
        alert(file.name);
    }
}
</script>
选择文件：
<input type="file" id="file" multiple size="80"/>
<input type="button" onclick="ShowFileName();" value="文件上传"/>
```

4.4.2 Blob对象

Blob表示二进制原始数据，它提供一个slice方法，可以通过该方法访问到字节内部的原始数据块。事实上，上面提到的file对象也继承了这个Blob对象。

Blob对象有两个属性，size属性表示一个Blob对象的字节长度，type属性表示Blob的MIME类型，如果是未知类型，则返回一个空字符串。

下面用代码清单4-22中的示例对Blob对象及它的两个属性做一些解释。

代码清单4-22 Blob对象使用示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>Blob对象使用示例</title>
<script language=javascript>
function ShowFileType()
{
    var file;
    //得到用户选择的第一个文件
    file = document.getElementById("file").files[0];
    var size=document.getElementById("size");
    //显示文件字节长度
    size.innerHTML=file.size;
    var type=document.getElementById("type");
    //显示文件类型
```

```
    type.innerHTML=file.type;
}
</script>
选择文件：
<input type="file" id="file" />
<input type="button" value="显示文件信息" onclick="ShowFileType();"/><br/>
文件字节长度:<span id="size"></span><br/>
文件类型：<span id="type"></span>
```

这段代码运行结果如图4-28所示。

图4-28　Blob对象使用示例

请注意：对于图像类型的文件，Blob对象的type属性都是以“image/”开头的，后跟图像类型，利用此特性我们可以在JavaScript中判断用户选择的文件是否为图像文件，如果在批量上传时，只允许上传图像文件，可以利用该属性，如果用户选择的多个文件中有不是图像的文件时，可以弹出错误提示信息，并停止后面的文件上传，或者跳过这个文件，不将该文件上传。请看代码清单4-23中的示例。

代码清单4-23　Blob对象的type属性利用

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>Blob对象的type属性利用示例</title>
<script language=javascript>
function FileUpload()
{
    var file;
    for(var i=0;i<document.getElementById("file").files.length;i++)
    {
        file = document.getElementById("file").files[i];
        if(!/image\/\w+/.test(file.type))
        {
            alert(file.name+"不是图像文件！");
            break;
        }
        else
        {
            //此处可加入文件上传的代码
            alert(file.name+"文件已上传");
        }
    }
}
</script>
选择文件：
<input type="file" id="file" multiple/>
<input type="button" value="文件上传" onclick="FileUpload();"/>
```

另外，HTML 5中已经对file控件添加了accept属性，企图让file控件只能接受某种类型的文件，但是目前各主流浏览器对其的支持都只限于在打开文件选择窗口时，默认选择图像文件而已，如果选择其他类型文件，file控件也能正常接受。

对file控件使用accept属性的方法如下所示。

```
<input type="file" id="file"  accept="image/*" />
```

图4-29为firefox浏览器对file控件的accept属性目前的支持情况，其他浏览器也与此类似。

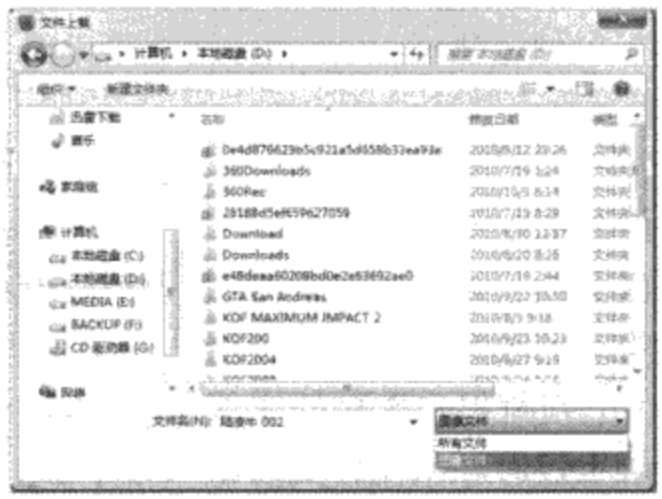

图4-29 firefox浏览器对file控件的accept属性目前的支持情况

4.4.3 FileReader接口

FileReader接口主要用来把文件读入内存，并且读取文件中的数据。FileReader接口提供了一个异步API，使用该API可以在浏览器主线程中异步访问文件系统，读取文件中的数据。到目前为止，只有Firefox 3.6+和Google Chrome 6.0+实现了FileReader接口。有一种方法可以检查您的浏览器是否对FileReader接口提供支持，如下所示。

```
if (typeof FileReader == 'undefined' )
{
    alert( " 您的浏览器未实现 FileReader 接口 " );
}
else
{
    var reader = new FileReader();
    //正常使用浏览器
}
```

1. FileReader接口的方法

FileReader接口拥有4个方法，其中3个用以读取文件，另一个用来将读取过程中断。表4-2列出了这些方法以及它们的参数和功能。需要注意的是：无论读取成功或失败，方法并不会返回读取结果，这一结果存储在 result属性中。

表4-2 FileReader接口的方法

方 法 名	参 数	描 述
readAsBinaryString	file	将文件读取为二进制码
readAsText	file, [encoding]	将文件读取为文本
readAsDataURL	file	将文件读取为DataURL
abort	(none)	中断读取操作

- readAsText：该方法有两个参数，其中第二个参数是文本的编码方式，默认值为UTF-8。这个方法非常容易理解，将文件以文本方式读取，读取的结果即是这个文本文件中的内容。
- readAsBinaryString：这个方法将文件读取为二进制字符串，通常我们将它传送到后端，后端可以通过这段字符串存储文件。
- readAsDataURL：该方法将文件读取为一串Data URL字符串，该方法事实上是将小文件以一种特殊格式的URL地址形式直接读入页面。这里的小文件通常是指图像与html等格式的文件。

2. FileReader接口的事件

除了上述方法之外，FileReader接口还包含了一套完整的事件模型，用于捕获读取文件时的状态，表4-3归纳了这些事件。

表4-3 FileReader接口的事件

事　件	描　述
onabort	数据读取中断时触发
onerror	数据读取出错时触发
onloadstart	数据读取开始时触发
onprogress	数据读取中
onload	数据读取成功完成时触发
onloadend	数据读取完成时触发，无论成功或失败

3. FileReader接口的使用示例

下面，我们通过代码清单4-24中的示例，来针对fileReader的方法进行讲解。

代码清单4-24 fileReader方法示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>fileReader方法示例</title>
</head>
<script language=javascript>
var result=document.getElementById("result");
var file=document.getElementById("file");
if (typeof FileReader == 'undefined' )
{
   result.innerHTML = "<p>抱歉，你的浏览器不支持 FileReader</p>";
   file.setAttribute( 'disabled','disabled' );
}
//将文件以Data URL形式读入页面
function readAsDataURL()
{
    //检查是否为图像文件
    var file = document.getElementById("file").files[0];
    if(!/image\/\w+/.test(file.type))
    {
```

```
            alert("请确保文件为图像类型");
            return false;
        }
        var reader = new FileReader();
        //将文件以Data URL形式读入页面
        reader.readAsDataURL(file);
        reader.onload = function(e)
        {
           var result=document.getElementById("result");
           //在页面上显示文件
           result.innerHTML = '<img src="'+this.result+'" alt=""/>'
        }
    }
    //将文件以二进制形式读入页面
    function readAsBinaryString()
    {
        var file = document.getElementById("file").files[0];
        var reader = new FileReader();
        //将文件以二进制形式读入页面
        reader.readAsBinaryString(file);
        reader.onload = function(f)
        {
            var result=document.getElementById("result");
            //在页面上显示二进制数据
            result.innerHTML=this.result;
        }
    }
    //将文件以文本形式读入页面
    function readAsText()
    {
        var file = document.getElementById("file").files[0];
        var reader = new FileReader();
        //将文件以文本形式读入页面
        reader.readAsText(file);
        reader.onload = function(f)
        {
            var result=document.getElementById("result");
            //在页面上显示读入文本
            result.innerHTML=this.result;
        }
    }
    </script>
    <p>
        <label>请选择一个文件：</label>
        <input type="file" id="file" />
        <input type="button" value="读取图像" onclick="readAsDataURL()"/>
        <input type="button" value="读取二进制数据" onclick="readAsBinaryString()"/>
        <input type="button" value="读取文本文件" onclick="readAsText()"/>
    </p>
    <div name="result" id="result">
          <!-- 这里用来显示读取结果 -->
    </div>
```

这段代码的运行的结果分别显示：

在file控件中选择一个图像文件，并且点击读取图像按钮，运行结果如图4-30所示。

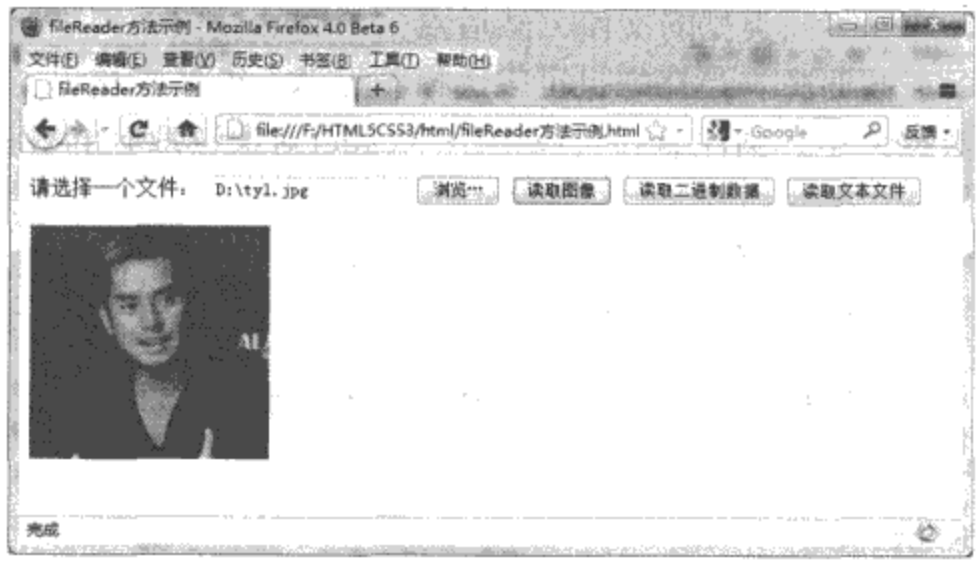

图4-30　读取图像文件并显示（点击读取图像按钮后）

在file控件中选择一个二进制文件，并且点击读取二进制数据按钮，运行结果，如图4-31所示。

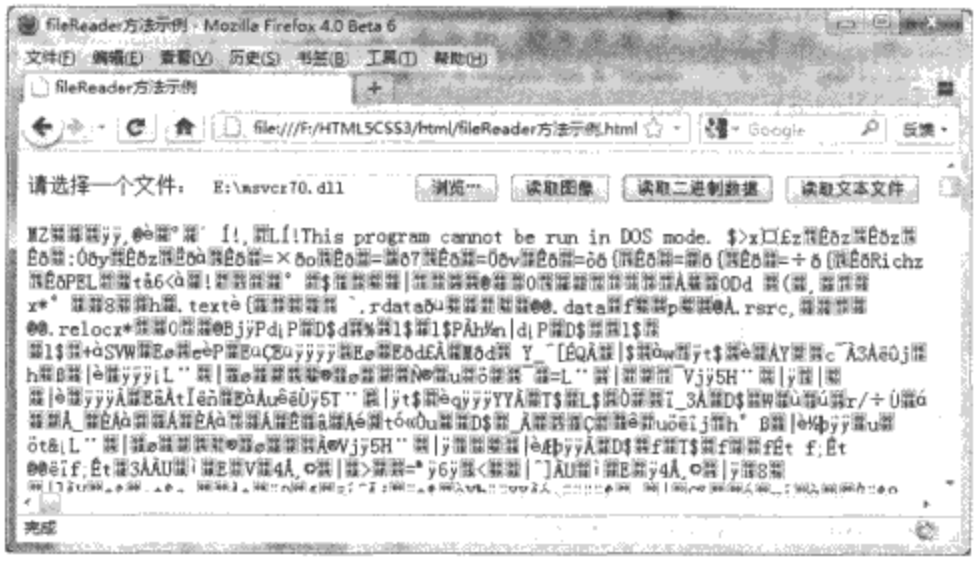

图4-31　读取二进制文件并显示（点击读取二进制数据按钮后）

在file控件中选择一个文本文件，并且点击读取文本文件按钮，运行结果如图4-32所示。

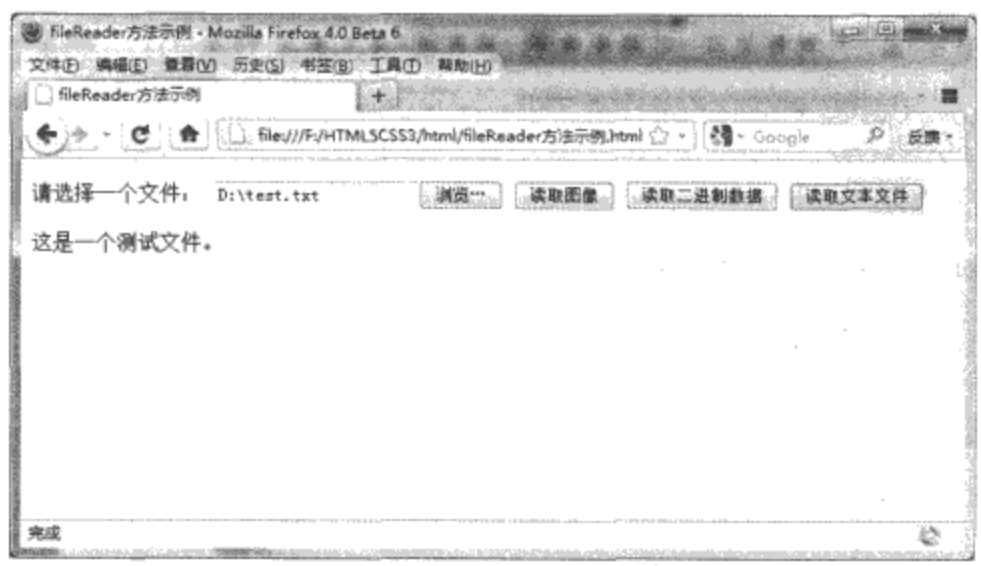

图4-32　读取文本文件并显示（点击读取文本文件按钮后）

在这个示例中，选取不同类型的文件，然后点击不同的按钮，浏览器会读取这些文件的各种数据，然后显示在画面中。当然您也可以选择不显示，而是直接提交到后端，然后保存到文件中或输送到数据库中。请注意：代码中fileReader对象读取到的数据都保存在了result属性中。

上面提到，当fileReader对象读取文件时，会伴随着一系列事件，它们表示读取文件时不同的读取状态，下面我们用代码清单4-25中的示例来看一下这些读取状态的先后顺序。

代码清单4-25 fileReader对象的事件先后顺序

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>fileReader对象的事件先后顺序</title>
</head>
<script language=javascript>
var result=document.getElementById("result");
var input=document.getElementById("input");
if(typeof FileReader=='undefined')
{
    result.innerHTML = "<p class='warn'>抱歉，你的浏览器不支持 FileReader</p>";
    input.setAttribute( 'disabled','disabled' );
}
function readFile()
{
    var file = document.getElementById("file").files[0];
    var reader = new FileReader();
reader.onload = function(e)
{
    result.innerHTML = '<img src="'+this.result+'" alt=""/>'
    alert("load");
}
reader.onprogress = function(e)
{
    alert("progress");
}
reader.onabort = function(e)
{
    alert("abort");
}
reader.onerror = function(e)
{
    alert("error");
}
reader.onloadstart = function(e)
{
    alert("loadstart");
}
reader.onloadend = function(e)
{
    alert("loadend");
}
reader.readAsDataURL(file);
```

```
}
</script>
<p>
<label>请选择一个图像文件：</label>
<input type="file" id="file" />
<input type="button" value="显示图像" onclick="readFile()" />
</p>
<div name="result" id="result">
<!-- 这里用来显示读取结果 -->
</div>
```

在这个示例中，我们通过点击显示图像按钮在画面中读入一个图像文件，通过这个过程我们可以了解按顺序触发了哪些事件，并用提示信息的形式报出这些事件的名字。我们需要编写的代码主要都是在onprogress事件中，譬如可以用HTML 5中的新增元素progress来显示大文件的读取完成百分比。

4.5 拖放API

在HTML 5中，提供了直接支持拖放操作的API。虽然HTML 5之前已经可以使用mousedown、mousemove、mouseup来实现拖放操作，但是这只支持在浏览器内部的拖放，而在HTML 5中，已经支持在浏览器与其他应用程序之间的数据互相拖动，同时也大大简化了有关于拖放方面的代码。

4.5.1 实现拖放的步骤

在HTML 5中要想实现拖放操作，至少要经过如下两个步骤：

1）将想要拖放的对象元素的draggable属性设为true(draggable="true")。这样才能将该元素进行拖放。另外，img元素与a元素（必须指定href）默认允许拖放。

2）编写与拖放有关的事件处理代码。关于拖放存在如表4-4所示的几个事件。

表4-4 拖放的相关事件

事　件	产生事件的元素	描　述
dragstart	被拖放的元素	开始拖放操作
drag	被拖放的元素	拖放过程中
dragenter	拖放过程中鼠标经过的元素	被拖放的元素开始进入本元素的范围内
dragover	拖放过程中鼠标经过的元素	被拖放的元素正在本元素范围内移动
dragleave	拖放过程中鼠标经过的元素	被拖放的元素离开本元素的范围
drop	拖放的目标元素	有其他元素被拖放到了本元素中
dragend	拖放的对象元素	拖放操作结束

下面在代码清单4-26中给出了一个按照上述步骤实现的拖放示例。该示例中，有一个显示“请拖放”文字的div元素，可以把它拖放到位于它下部的div元素中，每次被拖放时，在下部的div元素中会追加一次“你好”文字。

代码清单4-26 简单拖放示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>拖放示例</title>
<script type="text/javascript">
function init()
{
    var source = document.getElementById("dragme");
    var dest = document.getElementById("text");
    // (1) 拖放开始
    source.addEventListener("dragstart", function(ev)
    {
        // 向dataTransfer对象追加数据
        var dt = ev.dataTransfer;
        dt.effectAllowed = 'all';
       //(2) 拖动元素为dt.setData("text/plain", this.id);
        dt.setData("text/plain", "你好");
    }, false);
    // (3) dragend: 拖放结束
    dest.addEventListener("dragend", function(ev)
    {
        //不执行默认处理（拒绝被拖放）
        ev.preventDefault();
    }, false);
    // (4) drop:被拖放
    dest.addEventListener("drop", function(ev)
    {
        // 从DataTransfer对象那里取得数据
        var dt = ev.dataTransfer;
        var text = dt.getData("text/plain");
        dest.textContent += text;
        //(5) 不执行默认处理（拒绝被拖放）
        ev.preventDefault();
        //停止事件传播
        ev.stopPropagation();
    }, false);
}
//(6) 设置页面属性，不执行默认处理（拒绝被拖放）
document.ondragover = function(e){e.preventDefault();};
document.ondrop = function(e){e.preventDefault();};
</script>
</head>
<body onload="init()">
<h1>简单拖放示例</h1>
<!-- (7) 把draggable属性设为true -->
<div id="dragme" draggable="true" style="width: 200px; border: 1px solid gray;">
  请拖放
</div>
<div id="text" style="width: 200px; height: 200px; border: 1px solid gray;"></div>
</body>
```

这段代码的运行结果如图4-33所示。

下面我们把上例代码中的几个知识要点解释一下：

- 开始拖动（dragstart事件发生）时，把要拖动的数据存入DataTransfer对象（setData()方法）。DataTransfer对象专门用来存放拖放时要携带的数据，它可以被设置为拖动事件对象的dataTransfer属性。setData方法中的第一个参数为携带数据的数据种类的字符串，第二个参数为要携带的数据。第一个参数中表示数据种类的字符串里只能填入类似“text/plain”或“text/html”的表示MIME类型的文字，不能填入其他文字。
- 如果把“dt.setData("text/plain", "你好");”改为“dt.setData("text/plain", this.id);”，因为把被拖动元素的id当成了参数，所以携带的数据就是被拖动元素中的数据了，因为浏览器在使用getData()方法读取数据时会自动读取该元素中的数据。
- 针对拖放的目标元素，必须在dragend或dragover事件内调用“事件对象.preventDefault()”方法。因为默认情况下，拖放的目标元素是不允许接受元素的，为了把元素拖放到其中，必须把默认处理给关闭掉。
- 目标元素接受到被拖放的元素后，执行getData()方法从DataTransfer那里获得数据。getData()方法的参数为setData()方法中指定的数据种类，本例中为“text/plain”。
- 要实现拖放过程，还必须在目标元素的drop事件中关闭默认处理（拒绝被拖放），否则目标元素不能接受被拖放的元素。
- 要实现拖放过程，还必须设定整个页面为不执行默认处理（拒绝被拖放），否则拖放处理也不能被实现。因为页面是先于其他元素接受拖放的，如果页面上拒绝拖放，那么页面上其他元素就都不能接受拖放了。
- 要使元素可以被拖放，首先必须把该元素的draggable属性设为true。另外，为了让这个示例在所有支持拖放API的浏览器中都能正常运行，需要指定“-webkit-user-drag:element”这种Webkit特有的CSS属性。
- 因为这个示例中的数据种类使用了“text/plain”这个MIME类型，也可以从其他使用同样MIME类型的应用程序中把该类型的数据拖动到目标元素中。

图4-33 拖放示例

现在支持拖动处理的MIME的类型为以下几种：

text/plain：文本文字；

text/html：HTML文字；

text/xml：XML文字；

text/uri-list：URL列表，每个URL为一行。

4.5.2 DataTransfer对象的属性与方法

如果DataTransfer对象的属性和方法使用得好，可以实现定制拖放图标，让它只支持特定拖放（譬如拷贝/移动/）等，甚至可以实现更复杂的拖放操作。表4-5简单列举了DataTransfer对象的属性与方法。

表4-5 DataTransfer对象的属性与方法

属性/方法	描　述
dropEffect属性	表示拖放操作的视觉效果，允许对其进行值的设定 该效果必须在用effectAllowed属性指定的允许的视觉效果的范围内，允许指定的值为none、copy、link、move
effectAllowed属性	用来指定当元素被拖放时所允许的视觉效果 可以指定的值为none、copy、copyLink、copyMove、link、linkMove、move、all、unintialize
types属	存入数据的种类，字符串的伪数组
void clearData(DOMString format)方法	清除DataTransfer对象中存放的数据 如果省略参数format,则清除全部数据
void setData(DOMString format、DOMString data)	向DataTransfer对象内存入数据
DOMString getData(DOMString format)	从DataTransfer对象中读数据
void setDragImage(Element image, long x, long y)	用img元素来设置拖放图标 （部分浏览器中可以用canvas等其他元素来设置）

对于getData、setData这两个方法，我们前面已解释过——setData方法在拖放开始时向dataTransfer对象中存入数据，它用types属性来指定数据的MIME类型，而getData方法在拖动结束时读取dataTransfer对象中的数据。

clearData方法可以用来清除DataTransfer对象内的数据，譬如上例中在getData()方法前先加上“dt.clearData();”语句，则目标元素内就不会放入任何数据了。

4.5.3 设定拖放时的视觉效果

dropEffect属性与effectAllowed属性结合起来可以设定拖放时的视觉效果。effectAllowed属性表示当一个元素被拖动时所允许的视觉效果，一般在ondragstart事件中设定，允许设定

的值为none、copy、copyLink、copyMove、link、linkMove、move、all、unintialize。dropEffect属性表示实际拖放时的视觉效果，一般在ondragover事件中指定，允许设定的值为none、copy、link、move。dropEffect属性所表示的实际视觉效果必须在effectAllowed属性所表示的允许的视觉效果范围内。规则如下所示。

1）如果effectAllowed属性设定为none，则不允许拖放元素。

2）如果dropEffect属性设定为none，则不允许被拖放到目标元素中。

3）如果effectAllowed属性设定为all或不设定，则dropEffect属性允许被设定为任何值，并且按指定的视觉效果进行显示。

4）如果effectAllowed属性设定为具体效果（不为none、all），dropEffect属性也设定了具体视觉效果，则两个具体效果值必须完全相等，否则不允许将被拖放元素拖放到目标元素中。

以下是对effectAllowed属性及dropEffect属性进行设定的代码片段，完整代码参考上例。

```
source.addEventListener("dragstart", function(ev)
{
    var dt = ev.dataTransfer;
    //设定effectAllowed属性
    dt.effectAllowed = 'copy';
    dt.setData("text/plain", "你好");
}, false);
dest.addEventListener("dragover", function(ev)
{
    var dt = ev.dataTransfer;
    //设定dropEffect属性
    dt.dropEffect = 'copy';
    ev.preventDefault();
}, false);
```

4.5.4 自定义拖放图标

除了上面所说的使用effectAllowed属性与dropEffect属性外，HTML 5中还允许自定义拖放图标——指的是在用鼠标拖动元素的过程中，位于鼠标指针下部的小图标。

上面提到DataTransfer对象有一个setDragImage方法，该方法有三个参数，第一个参数image设定为拖放图标的图标元素，第二个参数x为拖放图标离鼠标指针的x轴方向的位移量，第三个参数y为拖放图标离鼠标指针的y轴方向的位移量。

以下是调用setDragImage方法的代码片段，其余代码参考前面示例。

```
//创建图标元素
var dragIcon = document.createElement('img');
//设定图标来源
dragIcon.src='http://twivatar.org/twitter/mini';
source.addEventListener("dragstart", function(ev) {
    var dt = ev.dataTransfer;
    //设定自定义图标
    dt.setDragImage(dragIcon, -10, -10);
    dt.setData("text/plain","aaa");
}, false);
```

第5章
绘 制 图 形

本章将介绍HTML 5中的一个新增元素——canvas元素以及伴随这个元素而来的一套编程接口——Canvas API。使用Canvas API，你可以在页面上绘制出任何你想要的、非常漂亮的图形与图像，创造出更加丰富多彩、赏心悦目的下一代Web页面。

学习内容：

- ❑ 掌握canvas元素的基本概念，学会如何在页面上放置一个canvas元素，如何使用canvas元素绘制出一个简单矩形。
- ❑ 掌握使用路径的方法，能够利用路径绘制出圆形与多边形。
- ❑ 掌握渐变图形的绘制方法，学会图形变形、图形缩放、图形组合，以及给图形绘制阴影的方法。
- ❑ 掌握在Canvas画布中使用图像的方法，能够在画布中绘制图像、复制图像、平铺图像、裁剪图像，以及使用图像中的像素来进行图像处理的方法。
- ❑ 掌握如何在画布中绘制文字，给文字加上边框的方法。
- ❑ 掌握如何保存及恢复绘图状态，如何保存绘制出来的图形、图像，掌握在画布中制作简单动画的方法。

5.1 canvas元素的基础知识

canvas元素是HTML 5中新增的一个重要元素，专门用来绘制图形。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。

但是，在canvas元素里进行绘画，并不是指拿鼠标来作画。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，您可以把它理解为类似于其他开发语言中的canvas画布。

那么，让我们来看一下在HTML 5中有关canvas元素的知识吧。

5.1.1 在页面中放置canvas元素

首先，我们来看一下在页面上的HTML代码中，应该怎样来放置一个canvas元素。在代码清单5-1中，给出了一个canvas元素在页面上放置时的代码示例。在学习代码之前，首先对其中的知识要点做一下介绍。

首先，应该要指定的是ID、width、height三个属性。虽然canvas元素是HTML 5中新增元素，但它在页面放置的时候跟其他元素没有太大区别。在该示例中，放置了一个400 × 300的canvas元素。

在脚本导入之处，使用了<script type="text/javascript" src="script.js" charset="gb2312"></script>语句，通过该语句，在本页面上导入了script.js这个脚本文件。然后，使用该脚本文件，进行图形描绘。

在body的属性中，使用了onload="draw('canvas');"语句。调用脚本文件中的draw函数进行图形描画。

代码清单5-1 canvas元素页面代码示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>canvas元素示例</title>
<script type="text/javascript" src="script.js" charset="gb2312"></script>
</head>
<body onload="draw('canvas');">
<h1>canvas元素示例</h1>
<canvas id="canvas" width="400" height="300" />
</body>
</html>
```

这段代码的运行结果如图5-1所示。

图5-1 canvas元素示例

5.1.2 绘制矩形

接下来，让我们来看一下script.js这个脚本文件中的内容。在这个文件里，只有draw这一个函数。这个函数的功能是把canvas画布的背景用浅蓝色涂满，然后画出一个红色正方形，边框为蓝色。

用canvas元素绘制图形时，需要经过几道步骤：

1）取得canvas元素。

首先，用document.getElementById等方法取得canvas对象。因为需要调用这个对象提供的方法来进行图形绘制。

2）取得上下文（context）。

进行图形绘制时，需要用到图形上下文（graphics context），图形上下文中是一个封装了很多绘图功能的对象。需要使用canvas对象的getContext方法来获得图形上下文。在draw函数中，将参数设为“2d”。您可能会想，“那可不可以设为3d或4d呢？”，答案是，不能。

3）填充与绘制边框。

用canvas元素绘制图形的时候，有两种方式——填充（fill）与绘制边框（stroke）。填充是指填满图形内部；绘制边框是指不填满图形内部，只绘制图形的外框。canvas元素结合使用这两种方式来绘制图形。

4）设定绘图样式（style）。

在进行图形绘制的时候，首先要设定好绘图的样式（style），然后调用有关方法进行图形的绘制。所谓绘图的样式，主要是针对图形的颜色而言的，但是并不限于图形的颜色，在后面几节中我们将讲到如何设定颜色以外的样式，例如：

❑ 设定填充图形的样式

fillStyle属性——填充的样式，在该属性中填入填充的颜色值。

❑ 设定图形边框的样式

strokeStyle——图形边框的样式。在该属性中填入边框的颜色值。

5）指定线宽。

使用图形上下文对象的lineWidth属性设置图形边框的宽度。在绘制图形的时候，任何直线都可以通过lineWidth属性来指定直线的宽度。

6）指定颜色值。

绘图时填充的颜色或边框的颜色分别通过fillStyle属性与strokeStyle属性来指定。颜色值使用的是普通样式表中使用的颜色值。例如“red”与“blue”这种颜色名，或“#FF0000”这种十六进制的颜色值。

另外，也可以通过rgb（红色值、绿色值、蓝色值）或rgba（红色值、绿色值、蓝色值、透明度）这种rgb函数或rgba函数来指定颜色的值。

7）绘制矩形。

分别使用fillRect方法与strokeRect方法来填充矩形和绘制矩形边框。这两个方法的定义如下所示。

```
context.fillRect(x,y,width,height)
context.strokeRect(x,y,width,height)
```

这里的context指的是图形上下文对象，这两个方法使用同样的参数，x是指矩形起点的横坐标，y是指矩形起点的纵坐标，坐标原点为canvas画布的最左上角,width是指矩形的长度，height是指矩形的高度——通过这四个参数，矩形的大小同时也就被决定了。

通过上述步骤：通过getContext来取得图形上下文，通过fillStyle属性与strokeStyle属性来指定颜色，通过fillRect方法与strokeRect方法来绘制图形，就可以绘制出简单的图形了。

但是，这样只能绘制最简单的矩形。如果要绘制其他图形，还需要用到后文讲述的一些知识。

script.js脚本文件中的内容如代码清单5-2所示。

代码清单5-2 绘制矩形的脚本文件

```
function draw(id) {
    var canvas = document.getElementById(id);
```

```
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    context.fillStyle = "red";
    context.strokeStyle = "blue";
    context.lineWidth=1;
    context.fillRect(50,50,100,100);
    context.strokeRect(50,50,100,100);
}
```

这段代码的运行结果如图5-2所示。

图5-2 绘制矩形示例

另外，关于矩形，除了示例中所讲到的两个方法之外，还有一个clearRect方法，该方法将擦除指定的矩形区域中的图形，使得矩形区域中的颜色全部变为透明。该方法定义如下所示。

```
context.clearRect(x,y, width,height)
```

与绘制矩形的方法一样，该方法也是使用四个参数：x是指矩形起点的横坐标；y是指矩形起点的纵坐标，坐标原点为canvas画布的最左上角；width是指矩形的长度；height是指矩形的高度。

5.2 使用路径

5.2.1 绘制圆形

接下来，我们来看一看应该怎样绘制除了长方形或正方形以外的其他图形。

要想绘制其他图形，需要使用路径。同样的，绘制开始时还是要取得图形上下文，然后需要执行如下步骤。

❑ 开始创建路径。

❑ 创建图形的路径。

❑ 路径创建完成后，关闭路径。

❑ 设定绘制样式，调用绘制方法，绘制路径。

也就是说，首先使用路径来勾勒图形轮廓，然后设置颜色，进行绘制。

接下来，我们用一个示例来对路径的使用方法进行介绍。该示例的html代码部分仍然使用代码清单5-1中的html代码，也采用了代码清单5-2中所述的script.js脚本文件中的draw函数，但是对其进行了修改。在该函数中使用了循环，先绘制一个最小的圆，然后不断地把半径扩大，绘制新的半透明圆形。

在查看代码之前，首先还是把示例中的有关知识点说明一下。

❑ 开始创建路径。

首先，使用图形上下文对象的beginPath方法，该方法的定义如下所示。

```
context.beginPath();
```

该方法不使用参数。通过调用该方法，开始路径的创建。在几次循环地创建路径的过程中，每次开始创建时都要调用beginPath函数。

❑ 创建圆形路径。

创建圆形路径时，需要使用图形上下文对象的arc方法。该方法的定义如下所示。

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

该方法使用六个参数，x为绘制圆形的起点横坐标，y为绘制圆形的起点纵坐标，radius为圆形半径，startAngle为开始角度，endAngle为结束角度，anticlockwise为是否按顺时针方向进行绘制。

arc方法不仅可以用来绘制圆形，也可以用来绘制圆弧。因此，使用时必须要指定开始角度与结束角度。因为这两个角度决定了弧度。anticlockwise参数为一个布尔值的参数，参数值为true时，按顺时针绘制；参数值为false时，按逆时针方向绘制。

❑ 关闭路径。

路径创建完成后，使用图形上下文对象的closePath方法将路径关闭。该方法定义如下所示。

```
context.closePath();
```

将路径关闭后，路径的创建工作就完成了，但是请注意，这时只是路径创建完毕而已，还没有真正绘制任何图形。

❑ 设定绘制样式，进行图形绘制。这部分代码如下所示。

```
context.fillStyle = 'rgba(255, 0, 0, 0.25)';
context.fill();
```

使用创建好的路径绘制图形。在指定绘制样式时，与上例中所述矩形的绘制方法一样，使用fillStyle方法与strokeStyle方法。

绘制图形的时候，还使用了fill方法（也可以使用stroke方法）。这两个方法的功能分别为“填充图形”与“绘制图形边框”。因为路径已经决定了图形的大小，所以就不需要在该

方法中使用参数来指定图形的大小了。

绘制圆形的具体代码如代码清单5-3所示。

代码清单5-3 绘制圆形

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    var n = 0;
    for(var i = 0; i < 10; i++)
    {
        context.beginPath();
        context.arc(i * 25, i * 25, i * 10, 0, Math.PI * 2, true);
        context.closePath();
        context.fillStyle = 'rgba(255, 0, 0, 0.25)';
        context.fill();
    }
}
```

这部分代码的运行结果如图5-3所示。

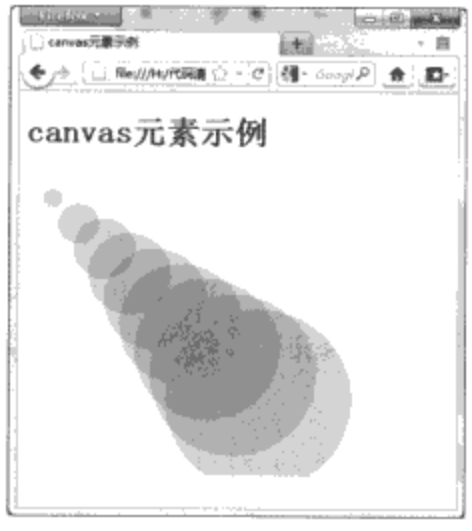

图5-3 使用路径绘制圆形

5.2.2 如果没有关闭路径会怎么样

关于路径，有一个有趣的实验。如果把上例中开始创建路径语句与关闭路径这一语句删除，会绘制出怎样的图形？

试验后的结果表明，在画布中先是绘制一个深红色的半径最小的圆，然后每次半径变大的同时，圆的颜色仿佛也在逐渐变淡。究竟为什么会这样？

这里，我们来介绍一下在循环时的具体绘制过程。

1）创建并且绘制第一个圆。

2）创建第二个圆。这时，因为没有把第一个绘制圆的路径给关闭掉，所以第一个圆的路径也保留着。绘制第二个圆的时候，第一个圆会根据该路径重复绘制，第二个圆只绘制一次，而第一个圆绘制了两次。

3）创建第三个圆。绘制时，第三个圆绘制一次，第二个圆绘制两次，第一个圆绘制三次。

4）同上……

明白了吧？如果不关闭路径，已经创建的路径会永远保留着。就算用fill方法与stroke方法在页面上将图形已经绘制完毕，路径都不会消失。因此，像上例中那样，如果把“使用路经进行绘制”这个方法进行循环，创建的图形会一次又一次地进行重叠。

所以，如果不仔细对路径进行管理的话，会绘制出意想不到的图形。当然，也可以利用这一特点绘制出有趣的图形，更加漂亮的图形。所以，只创建一次，而重叠绘制也会得到广泛的应用。因此，在进行绘制的时候，还是要仔细计算好路经从哪里开始，在哪里关闭。

这部分代码如代码清单5-4所示。

代码清单5-4 重叠绘制

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    var n = 0;
    for(var i = 0; i < 10; i++)
    {
        context.arc(i * 25, i * 25, i * 10, 0, Math.PI * 2, true);
        context.fillStyle = 'rgba(255, 0, 0, 0.25)';
        context.fill();
    }
}
```

这部分代码的运行结果如图5-4所示。

图5-4 图形重叠绘制

5.2.3 moveTo与lineTo

接下来，我们来看一下除了arc方法以外，其他使用路径绘制图形时会使用到的方法，首先从绘制直线开始。

绘制直线时，一般会用到moveTo与lineTo两种方法。它们的功能如下所示。

- moveTo

moveTo方法的作用是将光标移动到指定坐标点，绘制直线的时候以这个坐标点为起点。该方法的定义如下所示。

```
moveTo(x,y)
```

该方法使用两个参数——x表示指定坐标点的横坐标，y表示指定坐标点的纵坐标。

- lineTo

lineTo方法在moveTo方法中指定的直线起点与参数中指定的直线终点之间绘制一条直线。该方法定义如下所示。

```
lineTo(x,y)
```

与moveTo方法一样，它使用两个参数，x表示直线终点的横坐标，y表示直线终点的纵坐标。使用该方法绘制完直线后，光标自动移动到lineTo方法的参数所指定的直线终点。

因此，在创建路径时，需要使用moveTo方法将光标移动到指定的直线起点，然后使用lineTo方法在直线起点与直线终点之间创建路径，然后将光标移动到直线终点，在下一次使用lineTo方法的时候，会以当前光标所在坐标点为直线起点，并在下一个用lineTo方法指定的直线终点之间创建路径，它会不断重复这个过程，来完成复杂图形的路径绘制。

在代码清单5-5中，给出一个复杂图形的绘制示例。该示例中，使用三角函数计算顶点，循环调用lineTo语句来绘制图形。第一个lineTo语句中指定的坐标点即为直线起点，然后不断将直线绘制到下一个lineTo语句指定的直线终点，循环结束后关闭路径，最后一个坐标点与第一个坐标点自动闭合，使用fill语句填充图形。

代码清单5-5 绘制复杂图形

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    var n = 0;
    var dx = 150;
    var dy = 150;
    var s = 100;
    context.beginPath();
    context.fillStyle = 'rgb(100,255,100)';
    context.strokeStyle = 'rgb(0,0,100)';
    var x = Math.sin(0);
```

```
    var y = Math.cos(0);
    var dig = Math.PI / 15 * 11;
    for(var i = 0; i < 30; i++)
    {
        var x = Math.sin(i * dig);
        var y = Math.cos(i * dig);
        context.lineTo( dx + x * s,dy + y * s);
    }
    context.closePath();
    context.fill();
    context.stroke();
}
```

这段代码的运行结果如图5-5所示。

图5-5 绘制复杂图形

5.2.4 使用bezierCurveTo绘制贝济埃曲线

接下来，让我们来看一下贝济埃曲线的绘制。绘制贝济埃曲线时，需要使用“bezierCurveTo”方法。该方法可以说是lineTo的曲线版，将从当前坐标点到指定坐标点中间的贝济埃曲线追加到路径中。该方法的定义如下所示。

```
context.bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y);
```

该方法使用六个参数。绘制贝济埃曲线的时候，需要两个控制点，cp1x为第一个控制点的横坐标，cp1y为第一个控制点的纵坐标；cp2x为第二个控制点的横坐标，cp2y为第二个控制点的纵坐标；x为贝济埃曲线的终点横坐标，y为贝济埃曲线的终点纵坐标。

只要对上一个示例稍加修改，就可以使用路径绘制出贝济埃曲线。在代码清单5-6中，对上一个示例进行修改，将上例中的复杂图形拉伸，绘制出贝济埃曲线。

代码清单5-6 绘制贝济埃曲线

```
function draw(id)
{
    var canvas = document.getElementById(id);
```

```
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    var n = 0;
    var dx = 150;
    var dy = 150;
    var s = 100;
    context.beginPath();
    context.globalCompositeOperation = 'and';
    context.fillStyle = 'rgb(100,255,100)';
    var x = Math.sin(0);
    var y = Math.cos(0);
    var dig = Math.PI / 15 * 11;
    context.moveTo(dx,dy);
    for(var i = 0; i < 30; i++)
    {
        var x = Math.sin(i * dig);
        var y = Math.cos(i * dig);
        context.bezierCurveTo(dx + x * s,dy + y * s - 100,dx + x * s + 100,
        dy + y * s,dx + x * s,dy + y * s);
    }
    context.closePath();
    context.fill();
    context.stroke();
}
```

这部分代码的运行结果如图5-6所示。

图5-6 绘制贝济埃曲线

另外，除了可以在canvas画布中绘制贝济埃曲线之外，还可以使用quadraticCurveTo方法绘制二次样条曲线，该方法定义如下所示。

```
context.quadraticCurveTo(in float cpx,in float cpy,in float x,in float y)
```

相对来说，二次样条曲线的绘制比贝济埃曲线的绘制容易一些，因为绘制贝济埃曲线时需要两个控制点，而绘制二次样条曲线时只需要一个控制点。因此quadraticCurveTo方法只

需要四个参数就可以了，分别是控制点的横坐标、控制点的纵坐标、二次样条曲线终点的横坐标、二次样条曲线终点的纵坐标。

5.3 绘制渐变图形

5.3.1 绘制线性渐变

到现在为止，使用canvas API绘制图形的基本知识已介绍完毕，从本节开始，我们将介绍其他比较高级的绘制图形知识，首先，我们来看一下填充。

前面讲过，可以使用fillStyle方法在填充时指定填充的颜色。其实，使用该方法，除了指定颜色之外，还可以用来指定填充的对象。

这里，我们来看一下渐变。渐变是指在填充时从一种颜色慢慢过渡到另外一种颜色。渐变分为几种，首先我们介绍一下最简单的两点之间的线性渐变。

绘制线性渐变时，需要使用到LinearGradient对象。使用图形上下文对象的createLinearGradient方法创建该对象。该方法的定义如下所示。

```
context.createLinearGradient(xStart, yStart, xEnd, yEnd);
```

该方法使用四个参数，xStart为渐变起始地点的横坐标，yStart为渐变起始地点的纵坐标，xEnd为渐变结束地点的横坐标，yEnd为渐变结束地点的纵坐标。

通过使用该方法，创建了一个使用两个坐标点的LinearGradient对象。那么，渐变的颜色该怎么设定呢？在LinearGradient对象后，使用addColorStop方法进行设定，该方法的定义如下所示。

```
context.addColorStop(offset, color);
```

使用这个方法可以追加渐变的颜色。该方法使用两个参数——offset和color。offset为所设定的颜色离开渐变起始点的偏移量。该参数的值是一个范围在0到1之间的浮点值，渐变起始点的偏移量为0，渐变结束点的偏移量为1。在图5-7中用图形的方式来描述这个偏移量的含义。color为绘制时使用的颜色。

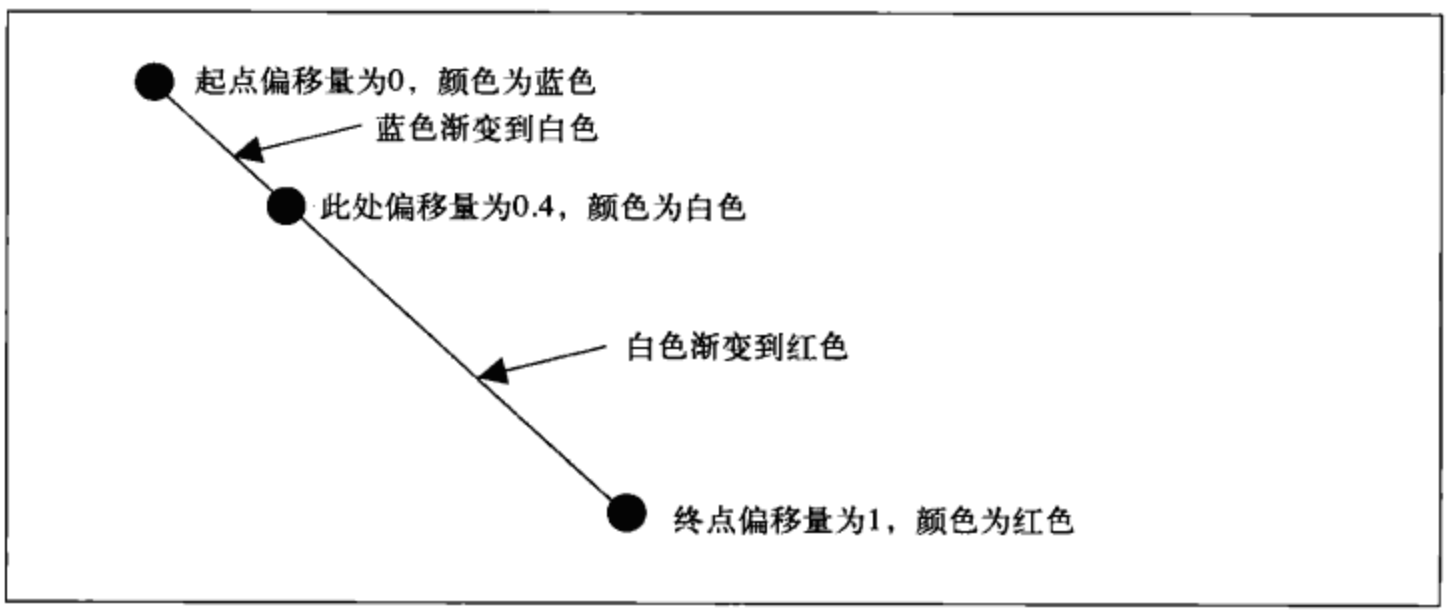

图5-7 addColorStop方法中offset参数的图示

因为是渐变，所以至少需要使用两次addColorStop方法以追加两个颜色（开始颜色与结束颜色），可以追加多个颜色。例如“从蓝色渐变到白色，然后渐变到绿色。”。蓝色起始点坐标到白色终点坐标之间的距离与白色起始点坐标到绿色终点坐标之间的距离相等，这时蓝色的位移量为0，白色的位移量为0.5，绿色的位移量为1。

接着把fillStyle设定为LinearGradient对象，然后执行填充的方法，就可以绘制出渐变图形了。

代码清单5-7给出了一个简单的绘制渐变的示例。在示例中，用从黄色到青色的渐变对画面进行填充，然后从左向右绘制一连串圆形，圆形的颜色也从蓝色渐变到红色。

代码清单5-7　绘制线性渐变

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    var g1 = context.createLinearGradient(0,0,0,300);
    g1.addColorStop(0,'rgb(255,255,0)');
    g1.addColorStop(1,'rgb(0,255,255)');
    context.fillStyle = g1;
    context.fillRect(0, 0, 400, 300);
    var n = 0;
    var g2 = context.createLinearGradient(0,0,300,0);
    g2.addColorStop(0,'rgba(0,0,255,0.5)');
    g2.addColorStop(1,'rgba(255,0,0,0.5)');
    for(var i = 0; i < 10; i++)
    {
        context.beginPath();
        context.fillStyle = g2;
        context.arc(i * 25, i * 25, i * 10, 0, Math.PI * 2, true);
        context.closePath();
        context.fill();
    }
}
```

这段代码的运行结果如图5-8所示。

图5-8　绘制线性渐变

5.3.2 绘制径向渐变

使用Canvas API，除了可以绘制线性渐变之外，还可以绘制径向渐变。径向渐变是指沿着圆形的半径方向向外进行扩散的渐变方式。譬如在描绘太阳时，沿着太阳的半径方向向外扩散出去的光晕，就是一种径向渐变。

使用图形上下文对象的createRadialGradient方法绘制径向渐变，该方法的定义如下所示。

```
context.createRadialGradient (xStart,yStart,radiusStart,xEnd,yEnd, radiusEnd);
```

该方法使用六个参数，xStart为渐变开始圆的圆心横坐标，yStart为渐变开始圆的圆心纵坐标，radiusStart为开始圆的半径，xEnd为渐变结束圆的圆心横坐标，yEnd为渐变结束圆的圆心纵坐标，radiusEnd为结束圆的半径。

在这个方法中，分别指定了两个圆的大小与位置。从第一个圆的圆心处向外进行扩散渐变，一直扩散到第二个圆的外轮廓处。

在设定颜色时，与线性渐变相同，使用addColorStop方法进行设定即可。同样也需要设定0到1之间的浮点数来作为渐变转折点的偏移量。

在代码清单5-8中，修改了线性渐变的示例，把使用createLinearGradient方法修改成了使用createRadialGradient方法，然后使用addColorStop方法，绘制出非常漂亮的径向渐变。

代码清单5-8 绘制径向渐变

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    var g1 = context.createRadialGradient(400,0,0,400,0,400);
    g1.addColorStop(0.1,'rgb(255,255,0)');
    g1.addColorStop(0.3,'rgb(255,0,255)');
    g1.addColorStop(1,'rgb(0,255,255)');
    context.fillStyle = g1;
    context.fillRect(0, 0, 400, 300);
    var n = 0;
    var g2 = context.createRadialGradient(250,250,0,250,250,300);
    g2.addColorStop(0.1,'rgba(255,0,0,0.5)');
    g2.addColorStop(0.7,'rgba(255,255,0,0.5)');
    g2.addColorStop(1,'rgba(0,0,255,0.5)');
    for(var i = 0; i < 10; i++)
    {
        context.beginPath();
        context.fillStyle = g2;
        context.arc(i * 25, i * 25, i * 10, 0, Math.PI * 2, true);
        context.closePath();
        context.fill();
    }
}
```

这段代码的运行结果如图5-9所示。

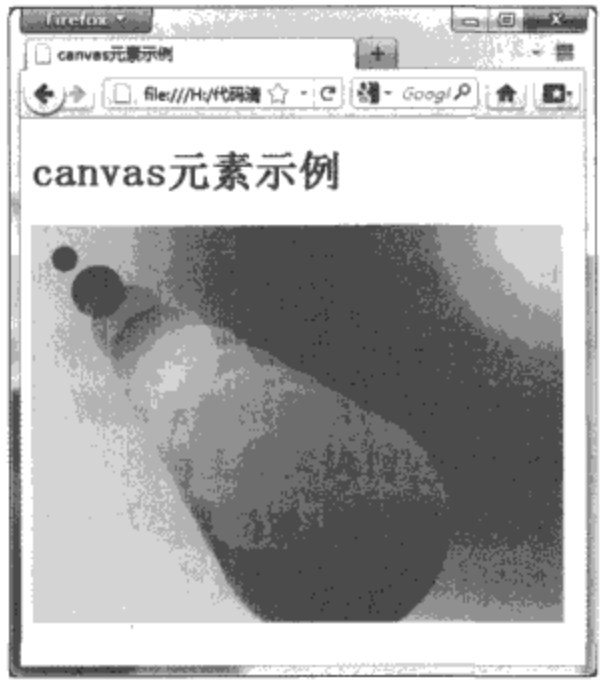

图5-9 绘制径向渐变

5.4 绘制变形图形

5.4.1 坐标变换

绘制图形的时候，我们可能经常会想要旋转图形，或者对图形使用变形处理，使用Canvas API的坐标轴变换处理功能，可以实现这种效果。

在计算机上绘制图形的时候，是以坐标单位为基准来进行图形绘制的。默认情况下，Canvas画布的最左上角对应于坐标轴原点（0，0）。前面我们所讲的所有利用Canvas API绘制出来的图形都是以画布的最左上角为坐标轴原点，并以一个像素为一个坐标单位来进行绘制的。

如果对这个坐标使用变换处理，就可以实现图形的变形处理了。对坐标的变换处理，有以下三种方式：

❑ 平移

使用图形上下文对象的translate方法移动坐标轴原点。该方法的定义如下所示。

```
context.translate(x,y);
```

translate方法使用两个参数——x表示将坐标轴原点向左移动多少个单位，默认情况下为像素；y表示将坐标轴原点向下移动多少个单位。

❑ 扩大

使用图形上下文对象的scale方法将图形放大。该方法的定义如下所示。

```
context.scale(x,y);
```

scale方法使用两个参数，x是水平方向的放大倍数，y是垂直方向的放大倍数。将图形缩

小的时候，将这两个参数设为0到1之间的小数就可以了，譬如0.5是指将图形缩小一半。

❑ 旋转

使用图形上下文对象的rotate方法将图形进行旋转。该方法的定义如下所示。

```
context.rotate(angle);
```

rotate方法接受一个参数angle，angle是指旋转的角度，旋转的中心点是坐标轴的原点。旋转是以顺时针方向进行的，要想逆时针旋转时，将angle设定为负数就可以了。

下面在代码清单5-9中给出一个使用坐标变换的方法绘制变形图形的简单示例。示例中首先绘制了一个正方形，然后在一个循环中反复使用平移坐标轴、图形缩小、图形旋转这三种技巧，最后绘制出来一个非常漂亮的变形图形。

代码清单5-9 坐标变换示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    // 图形绘制
    context.translate(200,50);
    context.fillStyle = 'rgba(255,0,0,0.25)';
    for(var i = 0;i < 50;i++)
    {
        context.translate(25,25);
        context.scale(0.95,0.95);
        context.rotate(Math.PI / 10);
        context.fillRect(0,0,100,50);
    }
}
```

这段代码的运行结果如图5-10所示。

图5-10 坐标变换示例

5.4.2 坐标变换与路径的结合使用

如果要对矩形进行变形，使用坐标变换就足够了。但是对使用路径绘制出来的图形进行变换的时候，要考虑的事情就比较多了。因为使用了坐标变换之后，已经创建好的路径就不能用了。必须要重新创建路径。重新创建好路径之后，坐标变换方法又失效了。

那怎么办呢？必须先另外写一个创建路径的函数，然后在坐标变换的同时调用该函数，这样才能解决这个问题。在代码清单5-10中，我们给出一个将坐标变换与路径结合使用的示例。执行该示例中代码，可以绘制一个将五角星一边旋转一边缩小的图形。

在示例中，首先单独书写了一个create5Star的函数，在该函数中创建了一个五角星的路径。然后在draw函数中的for循环中，首先依次执行translate、scale、rotate方法，然后执行create5Star函数创建路径，最后执行fill填充。

在create5Star函数中，只创建了一个五角星，因坐标轴变换，在Canvas画布中，此五角星会一边缩小一边旋转，之后产生一个新的五角星，新的五角星又采用同样的方法进行绘制，最终绘制出来一连串具有变形效果的五角星的图形。

代码清单5-10 坐标变换与路经结合使用示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    // 图形绘制
    context.translate(200,50);
    for(var i = 0;i < 50;i++)
    {
        context.translate(25,25);
        context.scale(0.95,0.95);
        context.rotate(Math.PI / 10);
        create5Star(context);
        context.fill();
    }
}
function create5Star(context)
{
    var n = 0;
    var dx = 100;
    var dy = 0;
    var s = 50;
    // 创建路径
    context.beginPath();
    context.fillStyle = 'rgba(255,0,0,0.5)';
    var x = Math.sin(0);
    var y = Math.cos(0);
    var dig = Math.PI / 5 * 4;
    for(var i = 0; i < 5; i++)
```

```
    {
        var x = Math.sin(i * dig);
        var y = Math.cos(i * dig);
        context.lineTo( dx + x * s,dy + y * s);
    }
    context.closePath();
}
```

这段代码的运行结果如图5-11所示。

图5-11 将坐标变换与路径结合使用的示例

5.4.3 矩阵变换

在上一节里，我们介绍了Canvas API中利用坐标变换实现的图形变形技术，当利用坐标变换不能满足我们的需要时，我们可以利用矩阵变换的技术。接下来，我们将介绍另一种更为复杂的利用矩阵变换实现的变形技术。

在介绍矩阵变换之前，首先要介绍一下变换矩阵，这个矩阵是专门用来实现图形变形的，它与坐标一起配合使用，以达到变形的目的。当图形上下文被创建完毕时，事实上也创建了一个默认的变换矩阵，如果不对这个变换矩阵进行修改，那么接下来绘制的图形将以画布的最左上角为坐标原点绘制图形，绘制出来的图形也不经过缩放、变形的处理，但是如果对这个变换矩阵进行修改，那么情况就完全不一样了。

使用图形上下文对象的transforms方法修改变换矩阵，该方法的定义如下所示。

```
context.transform(m11, m12, m21, m22, dx, dy);
```

该方法使用一个新的变换矩阵与当前变换矩阵进行乘法运算，该变换矩阵的形式如下所示。

m11	m21	dx
m12	m22	dy
0	0	1

其中m11，m21，m12，m22四个参数用来修改使用这个方法之后，绘制图形时的计算方法，以达到变形目的，dx与dy参数移动坐标原点，dx表示将坐标原点在x轴上向右移动x个单位，默认情况下以像素为单位，dy表示将坐标原点在y轴上向下移动y个单位。

要想了解m11，m21，m12，m22四个参数是如何修改变形矩阵以达到变形目的的，需要掌握矩阵乘法的有关知识，这里不具体详述，只通过几个例子来大略看一下它的工作原理。

首先，上一节使用坐标变换进行图形变形的技术中所提到的三个方法，实际上都是隐式地修改了变换矩阵，都可以使用transform方法来进行代替：

- translate(x,y)

可以使用context.transform(1,0,0,1,x,y)或context.transform(0,1,1,0,x,y)方法进行代替，前面四个参数1,0,0,1或0,1,1,0表示不对图形进行缩放操作，变形操作，将dx设为x表示将坐标原点向右移动x个单位，dy设为y表示将坐标原点向下移动y个单位。

- scale(x,y)

可以使用context.transform(x,0,0,y,0,0)或context.transform(0,y,x,0,0,0)方法代替，前面四个参数x,0,0,y或0,y,x,0表示将图形横向扩大x倍，纵向扩大y倍。dx,dy为0表示不移动坐标原点。

- rotate(angle)

替换方法如下所示。

```
context.transform(Math.cos(angle * Math.PI / 180),
   Math.sin(angle * Math.PI / 180),
   -Math.sin(angle * Math.PI / 180),
   Math.cos(angle * Math.PI / 180), 0, 0 );
```

或

```
context.transform(-Math.sin(angle * Math.PI / 180),
   Math.cos(angle * Math.PI / 180),
   Math.cos(angle * Math.PI / 180),
   Math.sin(angle * Math.PI / 180), 0,0) ;
```

其中前面四个参数以三角函数的形式结合起来，共同完成图形按angle角度的顺时针旋转处理，dx,dy为0表示不移动坐标原点。

使用transform方法进行图形的变形处理是相对复杂的，需要读者多多练习和揣摩，下面我们在代码清单5-11中看一个代码示例。在该示例中，用循环的方法绘制了几个圆弧，圆弧的大小与位置均不变，只是使用了transform方法让坐标原点每次向下移动10个像素，使得绘制出来的圆弧相互重叠，然后对圆弧设置七彩颜色，使这些圆弧的外观达到彩虹的效果。

代码清单5-11　用transform方法实现变形的示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
```

```
    /* 定义颜色 */
    var colors = ["red","orange","yellow","green", "blue", "navy", "purple"];
    /* 定义线宽*/
    context.lineWidth = 10;
    context.transform(1, 0, 0, 1, 100,0)
    /*循环绘制圆弧*/
    for( var i=0; i<colors.length; i++ )
    {
        /* 定义每次向下移动10个像素的变换矩阵 */
        context.transform(1, 0, 0, 1, 0, 10);
        /* 设定颜色 */
        context.strokeStyle = colors[i];
        /* 绘制圆弧 */
        context.beginPath();
        context.arc(50, 100, 100, 0, Math.PI, true);
        context.stroke();
    }
}
```

这段代码的运行结果如图5-12所示。

图5-12 用transform方法实现变形示例

使用了transform方法后，接下来要绘制的图形都会按照移动后的坐标原点与新的变换矩阵相结合的方法进行绘制，必要时可以使用setTransform方法将变换矩阵进行重置，setTransform方法的定义如下所示。

```
context.setTransform(m11, m12, m21, m22, dx, dy);
```

setTransform方法的参数及参数的用法与transform相同，事实上，该方法的作用为将画布上的最左上角重置为坐标原点，当图形上下文创建完毕时将所创建的初始变换矩阵设置为当前变换矩阵，然后使用transform方法。

下面我们在代码清单5-12中看一个使用setTransform方法的代码示例，该示例中首先创建一个红色边框的长方形，然后将该长方形顺时针旋转45度，绘制出一个新的长方形，并且绘制其边框为蓝色，然后将红色长方形扩大2.5倍绘制新的长方形，边框为绿色，最后在红色长方形右下方绘制同样大小的长方形，边框为灰色。

代码清单5-12 使用setTransform方法绘制变形图形的示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    /* -----------------------------------------
    * 绘制红色长方形
    * ----------------------------------------- */
```

```
    context.strokeStyle = "red";
    context.strokeRect(30, 10, 60, 20);
    /* ------------------------------------------
    * 绘制顺时针旋转45° 后的蓝色长方形
    * ------------------------------------------ */
    /*绘制45° 圆弧 */
    var rad = 45 * Math.PI / 180;
    /*定义顺时针旋转45° 的变换矩阵*/
    context.setTransform(Math.cos(rad), Math.sin(rad), -Math.sin(rad),
    Math.cos(rad), 0, 0 );
    /* 绘制图形 */
    context.strokeStyle = "blue";
    context.strokeRect(30, 10, 60, 20);
    /* ------------------------------------------
    * 绘制放大2.5倍后的绿色长方形
    * ------------------------------------------ */
    /* 定义放大2.5倍的变换矩阵 */
    context.setTransform(2.5, 0, 0, 2.5, 0, 0);
    /* 绘制图形 */
    context.strokeStyle = "green";
    context.strokeRect(30, 10, 60, 20);
    /* ------------------------------------------
    * 将坐标原点向右移动40像素，向下移动80像素后绘制灰色长方形
    * ------------------------------------------ */
    /* 定义将坐标原点向右移动40像素，向下移动80像素的矩阵 */
    context.setTransform(1, 0, 0, 1, 40, 80);
    /* 绘制图形 */
    context.strokeStyle = "gray";
    context.strokeRect(30, 10, 60, 20);
}
```

这段代码的运行结果如图5-13所示。

图5-13　使用setTransform方法绘制变形图形示例

5.5 图形组合

在代码清单5-12的示例中，我们可以看到使用Canvas API可以将一个图形重叠绘制在另一个图形上面，但图形中能够被看到的部分完全取决于图形的绘制顺序，有时，我们想将两块图形进行组合，并且自己决定以哪种方式进行组合，这时，我们需要使用到Canvas API的图形组合技术。

在HTML 5中，只要用图形上下文对象的globalCompositeOperation属性就能自己决定图形的组合方式了，使用方法如下所示：

```
context.globalCompositeOperation=type
```

type的值必须是下面几种字符串之一：

❑ source-over（默认值）

source-over为globalCompositeOperation属性的默认值，表示新图形覆盖在原有图形之上。

❑ destination-over

表示在原有图形之下绘制新图形。

❑ source-in

新图形与原有图形作in运算，只显示新图形中与原有图形相重叠的部分，新图形与原有图形的其他部分均变成透明。

❑ destination-in

原有图形与新图形作in运算，只显示原有图形中与新图形相重叠的部分，新图形与原有图形的其他部分均变成透明。

❑ source-out

新图形与原有图形作out运算，只显示新图形中与原有图形不重叠的部分，新图形与原有图形的其他部分均变成透明。

❑ destination-out

原有图形与新图形作out运算，只显示原有图形中与新图形不重叠的部分，新图形与原有图形的其他部分均变成透明。

❑ source-atop

只绘制新图形中与原有图形重叠的部分与未被重叠覆盖的原有图形，新图形的其他部分变成透明。

❑ destination-atop

只绘制原有图形中被新图形重叠覆盖的部分与新图形的其他部分，原有图形中的其他部分变成透明，不绘制新图形中与原有图形相重叠的部分。

❑ lighter

原有图形与新图形均绘制，重叠部分做加色处理。

❑ xor

只绘制新图形中与原有图形不重叠的部分，重叠部分变成透明。

❑ copy

只绘制新图形，原有图形中未与新图形重叠的部分变成透明。

如果指定的type不在这几个字符串当中，则按默认方式组合图形。

使用globalCompositeOperation属性指定图形组合方式的示例代码如代码清单5-13所示。在该示例中将所有的组合方式放在一个数组中，然后通过变量i来指定挑选哪种组合方式进行显示。

代码清单5-13 图形组合示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    var oprtns = new Array(
    "source-atop",
    "source-in",
    "source-out",
    "source-over",
    "destination-atop",
    "destination-in",
    "destination-out",
    "destination-over",
    "lighter",
    "copy",
    "xor"
    );
    i=8;     //读者可自行修改该参数来显示想要查看的组合效果
    //绘制原有图形（蓝色长方形）
    context.fillStyle = "blue";
    context.fillRect(10, 10, 60, 60);
    /*设置组合方式，从组合的参数数组中挑选组合方式，此处因为i是8，
    所以选择oprtns数组中第9(数组从0开始计算）个组合方式lighter*/
    context.globalCompositeOperation = oprtns[i];
    //设置新图形（红色圆形）
    context.beginPath();
    context.fillStyle = "red";
    context.arc(60, 60, 30, 0, Math.PI*2, false);
    context.fill();
}
```

该部分代码的运行结果如图5-14所示。

另外，这个示例中对于图形组合的显示方式是通过一个参数从参数数组中挑选一个组合方式来进行显示，也可以通过简单动画的方式对所有的组合方式作一个循环显示，本章最后一节将介绍这个使用动画的方式来循环显示的示例。

图5-14 图形组合示例

5.6 给图形绘制阴影

在HTML 5中，使用Canvas元素可以给图形添加阴影效果。添加阴影效果时，只需利用图形上下文对象的几个关于阴影绘制的属性就可以了，如下所示：

shadowOffsetX——阴影的横向位移量。

shadowOffsetY——阴影的纵向位移量。

shadowColor——阴影的颜色。

shadowBlur——阴影的模糊范围。

这里shadowOffsetX与shadowOffsetY所代表的阴影横向位移量与纵向位移量是指图形向横向方向移动，或纵向方向移动以产生阴影效果时的移动距离。这两个属性值在默认情况下均为0。

shadowBlur属性是可选的，它表示图形阴影边缘的模糊范围。如果不希望阴影的边缘太清晰，需要将阴影的边缘模糊化时可以使用该属性。设定该属性值时必须要设定为比0大的数字，否则将被忽略。一般设定在0至10之间，开发时可自行调整这个数值，以达到满意效果。

接下来，我们在代码清单5-14中看一个阴影绘制的简单示例。在这个示例中，我们使用前文讲到的translate方法，绘制几个呈移动状态的五角星。同时给每个五角星都加上阴影效果。

绘制阴影的时候，使用了图形上下文对象的绘制阴影属性，这几个属性与路径是无关的，只要设定一次之后，全部五角星就都具有阴影效果了。

但是，如果不想让全部五角星都具有阴影效果该怎么办呢？实际上，这里起关键作用的是shadowColor属性。能够看见阴影效果，是因为使用该属性设置了颜色，如果想让后面的图形不再具有阴影效果，只要把shadowColor设定为rgba(0,0,0,0)就可以了。

代码清单5-14 给图形绘制阴影

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
```

```
            return false;
        var context = canvas.getContext('2d');
        context.fillStyle = "#EEEEFF";
        context.fillRect(0, 0, 400, 300);
        context.shadowOffsetX = 10;
        context.shadowOffsetY = 10;
        context.shadowColor = 'rgba(100,100,100,0.5)';
        context.shadowBlur = 7.5;
        // 图形绘制
        context.translate(0,50);
        for(var i = 0;i < 3;i++)
        {
            context.translate(50,50);
            create5Star(context);
            context.fill();
        }
    }
    function create5Star(context)
    {
        var n = 0;
        var dx = 100;
        var dy = 0;
        var s = 50;
        //创建路径
        context.beginPath();
        context.fillStyle = 'rgba(255,0,0,0.5)';
        var x = Math.sin(0);
        var y = Math.cos(0);
        var dig = Math.PI / 5 * 4;
        for(var i = 0; i < 5; i++)
        {
            var x = Math.sin(i * dig);
            var y = Math.cos(i * dig);
            context.lineTo( dx + x * s,dy + y * s);
        }
        context.closePath();
    }
```

该段代码的运行结果如图5-15所示。

图5-15　给图形绘制阴影

5.7 使用图像

5.7.1 绘制图像

在HTML 5中，不仅可以使用Canvas API来绘制图形，还可以读取磁盘或网络中的图像文件，然后使用Canvas API将该图像绘制在画布中。

绘制图像时，需要使用drawImage方法，该方法的定义如下所示。

```
context.drawImage(image,x,y);
context.drawImage(image,x,y,w,h);
context.drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh);
```

第一种方法只使用三个参数，image是一个Image对象，用该对象来装载图像文件。x与y为绘制时该图像在画布中的起始坐标。

第二种方法中前三个参数的使用方法与第一种方法中的使用方法一样，w、h是指绘制时的图像的宽度与高度。第一种方法中省略了这两个参数，所以绘制出来的图像与原图大小相同，而第二种方法可以用来进行图像缩放。

第三种方法可以用来将画布中已绘制好的图像的全部或者局部区域复制到画布中的另一个位置上。该方法使用九个参数，image仍然代表被复制的图像文件，sx与sy分别表示源图像的被复制区域在画布中的起始横坐标与起始纵坐标，sw与sh表示被复制区域的宽度与高度，dx与dy表示复制后的目标图像在画布中的起始横坐标与起始纵坐标，dw与dh表示复制后的目标图像的宽度与高度。该方法可以只复制图像的局部，只要将sx与sy设为局部区域的起始点坐标，将sw与sh设为局部区域的宽度与高度就可以了。该方法也可以用来将源图像进行缩放，只要将dw与dh设为缩放后的宽度与高度就可以了。

绘制图像时首先使用不带参数的new方法创建Image对象，然后设定该Image对象的src属性为需要绘制的图像文件的路径,具体代码如下所示。

```
image = new Image();
image.src = "image1.jpg"; //设置图像路径
```

然后就可以使用drawImage方法绘制该图像文件了。

事实上，即使设定好Image对象的src属性后，也不一定立刻就能把图像绘制完毕，譬如有时该图像文件是一个来源于网络的比较大的图像文件，这时用户就得耐心等待图像全部装载完毕才能看见该图像了。

这种情况下，只要使用如下所示的方法，就可以解决这个问题了。

```
image.onload = function(){ 绘制图像的函数 }
```

在Image对象的onload事件中同步执行绘制图像的函数，就可以一边装载一边绘制了。

在代码清单5-15中，给出了一个简单的代码示例，它使用与页面同一个目录中的图像文件image1.jpg进行装载，在一个循环中将同一个图像文件绘制在画布中的不同位置上。在这段代码中，首先使用new Image创建Image对象，然后指定该Image对象的图像文件路径，然后使用onload方法装载图像，在装载的同时进行绘制。

代码清单5-15 绘制图像示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    image = new Image();
    image.src = "tyl.jpg";
    image.onload = function()
    {
        drawImg(context,image);
    };
}
function drawImg(context,image)
{
    for(var i = 0;i < 7;i++)
       context.drawImage(image,0 + i * 50,0 + i * 25,100,100);
}
```

这段代码的运行结果如图5-16所示。

图5-16 绘制图像示例

下面，我们在代码清单5-16中看一个使用了八个参数的drawImage方法将图像的局部放大，并复制到画布中另一个地方的示例，该示例通常用来做图像局部的特写放大处理，本例中将明星的头像放大显示在明星半身照片的旁边。

代码清单5-16 复制放大局部图像示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
```

```
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0, 400, 300);
    image = new Image();
    image.src = "tyl.jpg";
    image.onload = function()
    {
        drawImg(context,image);
    };
}
function drawImg(context,image)
{
    var i=0;
    //首先调用该方法绘制原始图像
    context.drawImage(image,0,0,100,100);
    //绘制将局部区域进行放大后的图像
    context.drawImage(image,23,5,57,80,110,0,100,100);
}
```

这段代码的运行结果如图5-17所示。

图5-17 复制放大局部图像示例

5.7.2 图像平铺

谈到绘制图像的时候，有一个非常重要的功能，就是图像平铺技术。所谓图像平铺就是用按一定比例缩小后的图像将画布填满，有两种方法可以实现该技术，一种是使用前面所介绍的drawImage方法，请看代码清单5-17中的示例。

代码清单5-17 图像平铺示例

```
function draw(id)
{
    var image = new Image();
```

```
        var canvas = document.getElementById(id);
        if (canvas == null)
            return false;
        var context = canvas.getContext('2d');
        image.src = "ty12.jpg";
        image.onload = function()
        {
            drawImg(canvas,context,image);
        };
    }
    function drawImg(canvas,context,image)
    {
        //平铺比例
        var scale=5
        //缩小后图像宽度
        var n1=image.width/scale;
       //缩小后图像高度
        var n2=image.height/scale;
       //平铺横向个数
        var n3=canvas.width/n1;
       //平铺纵向个数
        var n4=canvas.height/n2;
        for(var i=0;i<n3;i++)
            for(var j=0;j<n4;j++)
             context.drawImage(image,i*n1,j*n2,n1,n2);
    }
```

这段代码的运行结果如图5-18所示。

图5-18 图像平铺示例

但是该方法还需要使用到几个变量以及循环处理，处理方法相对来说复杂一些，在HTML 5中，要达到平铺效果，我们还可以使用更简便的图形上下文对象的createPattern方法，该方法只使用几个参数就达到了上面所述的平铺效果。createPattern方法的定义如下所示。

```
context.createPattern(image,type);
```

该方法使用两个参数，image参数为要平铺的图像，type参数的值必须是下面的字符串值之一：

- no-repeat：不平铺
- repeat-x：横方向平铺
- repeat-y：纵方向平铺
- repeat：全方向平铺

创建了image对象并指定图像文件后，使用createPattern方法创建填充样式，然后将该样式指定给图形上下文对象的fillStyle属性，最后再填充画布，就可以看到重复填充的效果了。

这部分代码如代码清单5-18所示。

代码清单5-18 使用createPattern方法平铺对象

```
function draw(id)
{
    var image = new Image();
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    image.src = "tyl3.jpg";
    image.onload = function()
    {
        //创建填充样式，全方向平铺
        var ptrn = context.createPattern(image,'repeat');
        //指定填充样式
        context.fillStyle = ptrn;
        //填充画布
        context.fillRect(0,0,400,300);
    };
}
```

这部分代码的运行结果与代码清单5-17中的运行结果相同。

5.7.3 图像裁剪

使用Canvas绘制图像的时候，我们经常会想要只保留图像的一部分，这时我们可以使用Canvas API自带的图像裁剪功能来实现这一想法。

Canvas API的图像裁剪功能是指，在画布内使用路径，只绘制该路径所包括区域内的图像，不绘制路径外部的图像。

使用图形上下文对象的不带参数的clip方法来实现Canvas元素的图像裁剪功能。该方法使用路径来对Canvas画布设置一个裁剪区域。因此，必须先创建好路径。路径创建完成后，调用clip方法设置裁剪区域。

下面，我们在代码清单5-19中看一个示例。在该示例中，把画布背景绘制完成后，调用create5StarClip函数。在函数中，创建一个五角星的路径，然后使用clip方法设置裁剪区域。示例中具体的执行流程为先装载图像，然后调用drawImg函数，在该函数中调用create5StarClip创

建路径，设置裁剪区域，然后绘制经过裁剪后的图像——最终可以绘制出一个五角星范围内的图像。

裁剪区域一旦设置好之后，后面绘制的所有图形就都可以使用这个裁剪区域，但是如果要取消掉这个已经设置好的裁剪区域，该怎么做呢？需要使用到本章最后一节中介绍的绘制状态的保存与恢复功能。这两个功能保存与恢复图形上下文的临时状态。在设置图像裁剪区域时，首先调用save方法保存图形上下文的当前状态，在绘制完经过裁剪的图像后，再调用restore恢复之前保存的图形上下文的状态，通过这种方法，对之后绘制的图像取消裁剪区域。

代码清单5-19 图像裁剪示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    var gr = context.createLinearGradient(0,400,300,0);
    gr.addColorStop(0,'rgb(255,255,0)');
    gr.addColorStop(1,'rgb(0,255,255)');
    context.fillStyle = gr;
    context.fillRect(0, 0, 400, 300);
    image = new Image();
    image.onload = function()
    {
        drawImg(context,image);
    };
    image.src = "tyl.jpg";
}

function drawImg(context,image)
{
    create5StarClip(context);
    context.drawImage(image,-50,-150,300,300);
}

function create5StarClip(context)
{
    var n = 0;
    var dx = 100;
    var dy = 0;
    var s = 150;
    context.beginPath();
    context.translate(100,150);
    var x = Math.sin(0);
    var y = Math.cos(0);
    var dig = Math.PI / 5 * 4;
    for(var i = 0; i < 5; i++)
    {
        var x = Math.sin(i * dig);
        var y = Math.cos(i * dig);
```

```
            context.lineTo( dx + x * s,dy + y * s);
        }
        context.clip();
    }
```

这段代码的运行结果如图5-19所示。

图5-19 图像裁剪示例

5.7.4 像素处理

在HTML 5中使用Canvas API所能够做到的图像处理技术中，还有一个令人赞叹的技术是像素处理技术。使用Canvas API能够获取图像中的每一个像素，然后得到该像素颜色的rgb值或rgba值。使用图形上下文对象的getImageData方法来获取图像中的像素，该方法的定义如下所示。

```
var imagedata= context.getImageData(sx, sy, sw, sh);
```

该方法使用四个参数，sx、sy分别表示所获取区域的起点横坐标、起点纵坐标，sw、 sh分别表示所获取区域的宽度和高度。

imagedata变量是一个CanvasPixelArray对象，具有height,width,data等属性。data属性是一个保存像素数据的数组，内容类似“[r1, g1, b1, a1, r2, g2, b2, a2, r3, g3, b3, a3, ...]”，其中，r1,g1,b1,a1为第一个像素的红色值，绿色值，蓝色值，透明度值；r2,g2,b2,a2分别为第二个像素的红色值，绿色值，蓝色值，透明度值，依此类推。data.length为所取得像素的数量。

使用Canvas API获取图像中所有像素的方法如代码清单5-20所示。

代码清单5-20 获取图像中所有像素

```
var image = new Image();
var context = canvas.getContext('2d');
image.onload = function ()
{
```

```
        var imagedata;
        context.drawImage(image, 0, 0);
        imagedata = context.getImageData(0, 0, image.width,image.height);
};
```

取得了这些像素后，就可以对这些像素进行处理了，接下来，你可以进行譬如蒙版处理、面部识别等较复杂的图像处理操作。

下面在代码清单5-21中给出一个用Canvas API将图像进行反显操作的示例，在该示例中，在得到像素数组后，将该数组中每个像素的颜色进行了反显操作，然后保存回像素数组，最后使用图形上下文对象的putImageData方法将反显操作后的图像重新绘制在画布上。该方法的定义如下所示。

```
context.putImageData(imagedata,dx,dy[,dirtyX,dirtyY,dirtyWidth,dirtyHeight ]);
```

该方法使用七个参数，imagedata为前面所述的像素数组，dx、dy分别表示重绘图像的起点横坐标、起点纵坐标。后面dirtyX、dirtyY、dirtyWidth、dirtyHeight这四个参数为可选参数，给出一个矩形的起点横坐标、起点纵坐标、宽度与高度，如果加上这四个参数，则只绘制像素数组中这个矩形范围内的图像。

另外，对于Canvas API的像素操作也只有部分浏览器支持，所以笔者选择Opera浏览器进行代码测试。

代码清单5-21 将图像进行反显操作

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    var image = new Image();
    image.src = "tyl.jpg";
    image.onload = function ()
    {
       context.drawImage(image, 0, 0);
       var imagedata = context.getImageData(0,0,image.width,image.height);
       for (var i = 0, n = imagedata.data.length; i < n; i += 4)
      {
          imagedata.data[i+0] = 255 - imagedata.data[i+0]; // red
          imagedata.data[i+1] = 255 - imagedata.data[i+2]; // green
          imagedata.data[i+2] = 255 - imagedata.data[i+1]; // blue
      }
      context.putImageData(imagedata, 0, 0);
    };
}
```

这部分代码在Opera浏览器中的运行后结果如图5-20所示。

图5-20 在Opera浏览器中的图像反显示例

5.8 绘制文字

在HTML 5中，可以在Canvas画布中进行文字的绘制，同时也可以指定绘制文字的字体、大小、对齐方式等，还可以进行文字的纹理填充等。

绘制文字时可以使用fillText方法或strokeText方法。

fillText方法用填充方式绘制字符串，该方法的定义如下所示。

```
void fillText(text, x, y, [maxWidth]);
```

该方法接受四个参数，第一个参数text表示要绘制的文字，第二个参数x表示绘制文字的起点横坐标，第三个参数y表示绘制文字的起点纵坐标，第四个参数maxWidth为可选参数，表示显示文字时的最大宽度，可以防止文字溢出。

strokeText方法用轮廓方式绘制字符串，该方法的定义如下所示。

```
void stroke Text(text, x, y, [maxWidth]);
```

该方法参数部分的解释与fillText方法相同。

在使用Canvas API来进行文字的绘制之前，可以先对该对象的有关文字绘制的属性进行设置，它们是：

- font属性：设置文字字体。
- textAlign属性：设置文字水平对齐方式，属性值可以为start、end、left、right、center。默认值为start。
- textBaseline属性：设置文字垂直对齐方式，属性值可以为top、hanging、middle、alphabetic、ideographic、bottom。默认值为alphabetic。

下面在代码清单5-22中具体给出绘制文字的一个示例。

代码清单5-22 绘制文字示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context=canvas.getContext('2d');
    context.fillStyle= '#00f';
    context.font= 'italic 30px sans-serif';
    context.textBaseline = 'top';
    //填充字符串
    context.fillText  ('示例文字', 0, 0);
    context.font='bold  30px sans-serif';
    //轮廓字符串
    context.strokeText('示例文字', 0, 50);
}
```

运行结果如图5-21所示。

图5-21 文字绘制示例

在使用CSS样式的时候，有时我们会希望能在文字周围制作一个漂亮的边框，在定义边框宽度的时候，我们需要首先计算出在这个边框里最长一行的文字的宽度。这时，我们可以使用图形上下文对象的measureText方法来得到文字的宽度，该方法的定义如下所示。

```
metrics=context.measureText(text);
```

measureText方法接受一个参数text,该参数为需要绘制的文字，该方法返回一个TextMetrics对象，TextMetrics对象的width属性表示使用当前指定的字体后text参数中指定的文字的总文字宽度。

measureText的使用方法如代码清单5-23所示。

代码清单5-23 测量文字宽度的示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
```

```
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.font = 'italic 20px sans-serif';
    /* 定义绘制文字*/
    var txt = "字符串的宽度为";
    /* 获取文字宽度 */
    var tm1 = context.measureText(txt);
    /* 绘制文字 */
    context.fillText(txt, 10, 30);
    context.fillText(tm1.width, tm1.width+10, 30);
    /* 改变字体 */
    context.font = "bold  30px sans-serif";
    /* 重新获取文字宽度 */
    var tm2 = context.measureText(txt);
    /* 重新绘制文字*/
    context.fillText(txt, 10, 70);
    context.fillText(tm2.width,tm2.width+10, 70);
}
```

运行结果如图5-22所示。

图5-22 测量文字宽度示例

5.9 补充知识

到目前为止，本章已详细介绍了HTML 5中的Canvas API中的各个知识要点。作为补充，我们还将介绍几个相对来说比较独立的知识点：绘图状态的保存、文件的保存与简单动画的制作。

5.9.1 保存与恢复状态

在介绍图像裁剪的时候，有一个遗留问题，如果接下来要想继续绘制别的图像，但是要取消掉裁剪范围，该怎么做呢？需要使用到Canvas API中的save与restore两个方法，这两个

方法均不带参数，分别保存与恢复图形上下文的当前绘画状态。这里的绘画状态指前面所讲的坐标原点、变形时的变换矩阵，以及图形上下文对象的当前属性值等很多内容。在需要保存与恢复当前状态时，首先调用save方法将当前状态保存到栈中，在做完想做的工作后，再调用restore从栈中取出之前保存的图形上下文的状态进行恢复，通过这种方法，对之后绘制的图像取消裁剪区域。

具体来说，恢复与保存的步骤类似如下所示。

```
var x,y;
for (var j=1;j<50;j++)
{
    ctx.save();
    //改变绘画状态，进行想要的操作
    ctx.fillStyle = '#fff';
    x=75-Math.floor(Math.random()*150);
    y=75-Math.floor(Math.random()*150);
    ctx.translate(x,y);
    drawStar(ctx,Math.floor(Math.random()*4)+2);
    ctx.restore();
}
```

图形上下文对象的当前状态的保存与恢复是一个独立的话题，它与Canvas API中的其他技术都不相关，但可以应用在以下场合：

- ❑ 图像或图形变形。
- ❑ 图像裁剪。
- ❑ 改变图形上下文的以下属性的时候：fillStyle、font、globalAlpha、globalCompositeOperation、lineCap、lineJoin、lineWidth、miterLimit、shadowBlur、shadowColor、shadowOffsetX、shadowOffsetY、strokeStyle、textAlign、textBaseline。

5.9.2 保存文件

在画布中绘制完成一幅图形或图像后，很多时候我们需要将该图形或图像保存到文件中，使用Canvas API当然可以完成这步最后的工作。

Canvas API保存文件的原理实际上是把当前的绘画状态输出到一个data URL地址所指向的数据中的过程，所谓data URL，是指目前大多数浏览器能够识别的一种base64位编码的URL，主要用于小型的、可以在网页中直接嵌入，而不需要从外部文件嵌入的数据，譬如img元素中的图像文件等。data URL的格式类似于“...etc”，它目前得到了大多数浏览器的支持。

Canvas API使用toDataURL方法把绘画状态输出到一个data URL中，然后重新装载，客户可直接把装载后的文件进行保存。

toDataURL的使用方法如下所示。

```
canvas.toDataURL (type) ;
```

该方法使用一个参数type,表示要输出数据的MIME类型。

下面在代码清单5-24中给出一个使用Canvas API将图像输出到data URL的示例。

代码清单5-24 使用Canvas API将图像输出到data URL示例

```
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    context.fillStyle = "rgb(0, 0, 255)";
    context.fillRect(0, 0, canvas.width, canvas.height);
    context.fillStyle = "rgb(255, 255, 0)";
    context.fillRect(10, 20, 50, 50);
    window.location =canvas.toDataURL("image/jpeg");
}
```

这段代码的运行结果如图5-23所示。

图5-23 使用Canvas API将图像输出到data URL的示例

5.9.3 简单动画的制作

在Canvas画布中制作动画相对来说比较简单，实际上就是一个不断擦除、重绘、擦除、重绘的过程，具体步骤如下：

1）预先编写好用来绘图的函数，在该函数中先用clearRect方法将画布整体或局部擦除。

2）使用setInterval方法设置动画的间隔时间。

setInterval方法为HTML中固有方法，该方法接受两个参数，第一个参数表示执行动画的函数，第二个参数为时间间隔，单位为毫秒。

在比较复杂的情况下，我们也可以在清除与绘制动画的当中插入当前绘制状态的保存与恢复，变成擦除、保存绘制状态、进行绘制、恢复状态的过程。

下面在代码清单5-25中给出一个使用Canvas API绘制简单动画的示例，该示例中将绘制一个红色小方块，使其在画布中从左向右缓慢移动，读者可在这个基础上，使用JavaScript脚本文件编写出许多更复杂的动画。

代码清单5-25 使用Canvas API绘制简单动画

```
var context;
var width,height;
```

```
var i;
function draw(id)
{
    var canvas = document.getElementById(id);
    if (canvas == null)
        return false;
    context = canvas.getContext('2d');
    width=canvas.width;
    height=canvas.height;
    i=0;
    setInterval(rotate,100);          //十分之一秒
}
function rotate()
{
    context.clearRect(0,0,width,height);
    context.fillStyle = "red";
    context.fillRect(i, 0, 20, 20);
    i=i+20;
}
```

该代码的运行结果如图5-24所示。

最后，在前面图形组合一节中介绍的图形组合方式的代码示例中，使用了一个变量从图形上下文的globalCompositeOperation属性的所有参数构成的数组中挑选一个参数，来显示对应的图形组合效果，下面在代码清单5-26中给出一个通过动画来循环显示所有参数组合效果的代码示例。

图5-24 Canvas动画示例

代码清单5-26 循环显示所有参数的组合效果

```
var globalId;
var i=0;
function draw(id)
{
    globalId=id;
    setInterval(Composite,1000);
}
```

```
function Composite()
{
    var canvas = document.getElementById(globalId);
    if (canvas == null)
        return false;
    var context = canvas.getContext('2d');
    var oprtns = new Array(
    "source-atop",
    "source-in",
    "source-out",
    "source-over",
    "destination-atop",
    "destination-in",
    "destination-out",
    "destination-over",
    "lighter",
    "copy",
    "xor"
    );
    if(i>10) i=0;
    context.clearRect(0,0,canvas.width,canvas.height);
    context.save();
    //绘制原有图形（蓝色长方形）
    context.fillStyle = "blue";
    context.fillRect(10, 10, 60, 60);
    //设置组合方式
    context.globalCompositeOperation = oprtns[i];
    //设置新图形（红色圆形）
    context.beginPath();
    context.fillStyle = "red";
    context.arc(60, 60, 30, 0, Math.PI*2, false);
    context.fill();
    context.restore();
    i=i+1;
}
```

这段代码的运行结果如图5-25所示。

图5-25 循环显示各种图形组合效果

第6章
多媒体播放

在HTML 5问世之前，要在网络上展示视频、音频、动画，除了使用第三方自主开发的播放器之外，使用得最多的工具应该算是FLASH了，但是它们都需要在浏览器中安装各种插件才能使用，而且有时速度很慢。HTML 5的出现使这一局面得到了改观。在HTML 5中，提供了音频视频的标准接口，通过HTML 5中的相关技术，视频、动画、音频等多媒体播放再也不需要安装插件了，只需要一个支持HTML 5的浏览器就可以了。

本章介绍HTML 5中新增的两个元素——video元素与audio元素，它们分别用来处理视频数据与音频数据。需要说明的是，本章中的“媒体”一词是对音频和视频的总称。

学习内容：

- 知道什么是video元素与audio元素，为什么HTML 5中要增加这两个元素，怎样在页面中放置一个video元素或audio元素，至少需要指定它的什么属性。
- 掌握video元素与audio元素有哪些属性，如何对这些属性的属性值进行设定。
- 掌握video元素与audio元素有哪些方法，怎样使用这些方法。
- 掌握video元素与audio元素有哪些事件，什么时候会触发这些事件，如何捕捉这些事件。

6.1 video元素与audio元素的基础知识

6.1.1 HTML 4页面中播放视频或音频的方法

在介绍HTML 5的video元素与audio元素之前，让我们先来回忆一下在HTML 4页面中是如何播放视频或音频数据的，如代码清单6-1所示。

代码清单6-1 在HTML 4页面中播放视频

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 width="425" height="344" codebase="swflash.cab#version=6,0,40,0">
    <param name="allowFullScreen" value="true" />
    <param name="allowscriptaccess" value="always" />
    <param name="allowfullscreen" value="true" />
    <embed type="application/x-shockwave-flash"
           width="425"
           height="344"
           src="p.swf"
           allowscriptaccess="always"
           allowfullscreen="true">
</embed>
</object>
```

很明显，这段代码具有三个缺点：

- 冗长并且丑陋。
- 需要使用Flash插件。如果用户没有安装该插件，这段视频就不能看了，画面上会出现一片空白。
- 需要结合使用比较复杂的object元素与embed元素，并且为object元素添加许多属性和

参数，代码的书写比较烦琐。

6.1.2 HTML 5页面中播放视频或音频的方法

因此，在HTML 5中，新增了两个元素——video元素与audio元素，其中video元素专门用来播放网络上的视频或电影，而audio元素专门用来播放网络上的音频数据。使用这两个元素，就不再需要使用其他任何插件了，只要使用支持HTML 5的浏览器就可以了，同时在开发的时候也不再需要书写复杂的object元素和embed元素了。

现在，Safari 3以上、Firefox 4以上、Opera 10以上，以及Google chrome 3.0以上的浏览器都实现了对于video元素与audio元素的支持。

这两个元素的使用方法都很简单，首先以audio元素为例，只要把播放视频的URL地址指定给该元素的src属性就可以了，如下所示。

```
<audio src=" http://Lulingniu/demo/test.mp3" >
    您的浏览器不支持audio元素
</audio>
```

通过这种方法，可以把指定的音频数据直接嵌入在网页上，其中“您的浏览器不支持audio元素”为在不支持audio元素的浏览器中所显示的替代文字。

video元素的使用方法也很简单，只要设定好元素的长、宽等属性，并且把播放视频的URL地址指定给该元素的src属性就可以了，如下所示。

```
<video width="640" height="360" src="http://Lulingniu/test.mp4" >
    您的浏览器不支持video元素
</video>
```

另外，你还可以通过source元素来为同一个媒体数据指定多个播放格式与编码方式，以确保浏览器可以从中选择一种自己支持的播放格式进行播放，浏览器的选择顺序为代码中的书写顺序，它会从上往下判断自己对该播放格式是否支持，直到选择到自己支持的播放格式为止。source元素的使用方法如下所示。

```
<video>
  <!—在Ogg theora格式、Quicktime格式与mp4格式之间，挑选自己支持的
播放格式-->
  <source src='sample.ogv' type='video/ogg; codecs="theora, vorbis"'>
  <source src='sample.mov' type='video/quicktime'>
</video>
```

source元素具有几个属性：src属性是指播放媒体的URL地址；type表示媒体类型，其属性值为播放文件的MIME类型，该属性中的codes参数表示所使用的媒体的编码格式。type属性是可选属性，但是最好不要省略type属性，否则浏览器会在从上往下选择时无法判断自己能不能播放而先行下载一小段视频（或音频）数据，这样就有可能浪费带宽和时间了。

因为各浏览器对于各种媒体的媒体类型及其编码格式的支持情况都各不相同，所以使用source元素来指定多种媒体类型是非常有必要的。

接下来，我们来具体看一下各浏览器对于编码格式的支持情况。

❑ IE9

支持H.264视频编码格式和vp8视频编码格式。

支持MP3音频编码格式和WAV音频编码格式。

❑ Firefox 4以上

支持Ogg Theora视频编码格式和vp8视频编码格式。

支持Ogg vorbis音频编码格式和WAV音频编码格式。

❑ Opera 10以上

支持Ogg Theora视频编码格式和vp8视频编码格式。

支持Ogg vorbis音频编码格式和WAV音频编码格式。

❑ Chrome 6以上

支持H.264视频编码格式、Ogg Theora视频编码格式、vp8视频编码格式。

支持Ogg vorbis音频编码格式和MP3音频编码格式。

6.2 属性

audio元素与video元素所具有的属性大致相同，所以我们接下来看一下这两个元素都具有哪些属性。

❑ src

在该属性中指定媒体数据的URL地址。

❑ autoplay

在该属性中指定媒体是否在页面加载后自动播放。该属性的使用方法如下所示。

```
<video src="sample.mov" autoplay></video>
```

❑ preload

在该属性中指定视频或音频数据是否预加载。如果使用预加载的话，浏览器会预先将视频或音频数据进行缓冲，这样可以加快播放的速度，因为播放时数据已经预先缓冲完毕。

该属性有三个可选择的值：none、metadata与auto，默认值为auto。

none表示不进行预加载。

metadata表示只预加载媒体的元数据（媒体字节数、第一帧、播放列表、持续时间等）。

auto表示预加载全部视频或音频。

该属性的使用方法如下所示。

```
<video src="sample.mov" preload="auto"></video>
```

❑ poster（video元素独有属性）

当视频不可用时，可以使用该元素向用户展示一幅替代用的图片。当视频不可用时，最好使用该属性，以免展示视频的区域中出现一片空白。该属性的使用方法如下所示。

```
<video src="sample.mov" poster="CannotUse.jpg"></video>
```

❑ loop

在该属性中指定是否循环播放视频或音频。该属性的使用方法如下所示。

```
<video src="sample.mov"  autoplay loop></video>
```

❑ controls

在该属性中指定是否为视频或音频添加浏览器自带的播放用的控制条。控制条中具有播放、暂停等按钮。该属性的使用方法如下所示。

```
<video src="sample.mov"  controls></video>
```

图6-1中所示为Opera 10浏览器自带的播放视频时用的控制条的外观。

图6-1 Opera 10浏览器自带的播放视频时用的控制条

开发者也可以在脚本中自定义控制条，而不使用浏览器默认的控制条。

❑ width与height（video元素独有属性）

在该属性中指定视频的宽度与高度（以像素为单位）。这两个属性的使用方法如下所示。

```
<video src="sample.mov" width="500"  height="500"></video>
```

❑ error属性

在读取、使用媒体数据的过程中，在正常情况下，video元素或audio元素的error属性为null，但是任何时候只要出现错误，error属性将返回一个MediaError对象，该对象的code返回对应的错误状态，错误状态共有4个可能值，如下所示。

MEDIA_ERR_ABORTED（数字值为1）：媒体数据的下载过程由于用户的操作原因而被中止。

MEDIA_ERR_NETWORK（数字值为2）：确认媒体资源可用，但是在下载时出现网络错误，媒体数据的下载过程被中止。

MEDIA_ERR_DECODE（数字值为3）：确认媒体资源可用，但是解码时发生错误。

MEDIA_ERR_SRC_NOT_SUPPORTED（数字值为4）：媒体格式不被支持。

error属性为只读属性。

读取错误状态的代码如代码清单6-2所示。

代码清单6-2 读取错误状态

```
<video id="videoElement" src="sample.mov">
<script>
var video = document.getElementById("veido");
video.addEventListener("error", function()
{
    var error = video.error;
    switch(error.code)
    {
        case 1:
            alert("视频的下载过程被中止。");
            break;
        case 2:
            alert("网络发生故障，视频的下载过程被中止。");
```

```
            break;
        case 3:
            alert("解码失败。");
            break;
        case 4:
            alert("不支持播放的视频格式。");
            break;
    }
}, false);
...(后略)
</script>
```

❑ networkState属性

在媒体数据加载过程中可以使用video元素或audio元素的networkState属性读取当前网络状态，共有如下所示的4个可能值：

NETWORK_EMPTY（数字值为0）：元素处于初始状态。

NETWORK_IDLE（数字值为1）：浏览器已选择好用什么编码格式来播放媒体，但尚未建立网络连接。

NETWORK_LOADING（数字值为2）：媒体数据加载中。

NETWORK_NO_SOURCE（数字值为3）：没有支持的编码格式，不执行加载。

networkState属性为只读属性。

读取网络状态的代码如代码清单6-3所示。

代码清单6-3 读取网络状态

```
<script>
...(前略)
var video = document.getElementById("video");
video.addEventListener("progress", function(e)
{
    var networkStateDisplay =
    document.getElementById("networkState");
    //根据networkState属性的值来执行处理。
    if(video.networkState == 2)
    {
        //计算已加载的字结数与总字节数。
        networkStateDisplay.innerHTML ="加载中...["
        +e.loaded+ " / " +e.total+ " byte]";
    }
    else if(video.networkState == 3)
    {
        networkStateDisplay.innerHTML ="加载失败";
    }
}, false);
</script>
```

❑ currentSrc属性

可以使用video元素或audio元素的currentSrc属性来读取播放中的媒体数据的URL地址。

currentSrc属性为只读属性。

❑ buffered属性

可以使用video元素或audio元素的buffered属性来返回一个对象，该对象实现TimeRanges接口，以确认浏览器是否已缓存媒体数据。TimeRanges对象表示一段时间范围，在大多数情况下，TimeRanges对象表示的时间范围是一个单一的以0开始的范围，但是如果浏览器发出Range Requests请求，这时TimeRanges对象表示的时间范围是多个时间范围。

TimeRanges对象具有一个length属性，表示有多少个时间范围，大多数情况下存在时间范围时，该值为1；不存在时间范围时，该值为0。TimeRanges对象还具有两个方法，TimeRanges.start(index)与TimeRanges.end(index)[⊖]，大多数情况下将index值设为0就可以了。当用videoElement.buffered语句来实现TimeRanges接口时，TimeRanges.start(0)表示当前缓存区内从媒体数据的什么时间开始进行缓存，TimeRanges.end(0)表示当前缓存区内的结束时间。

buffered属性为只读属性。

❑ readyState属性

可以使用video元素或audio元素的readyState属性返回媒体当前播放位置的就绪状态，共有5个可能值。

HAVE_NOTHING（数字值为0）：没有获取到媒体的任何信息，当前播放位置没有可播放数据。

HAVE_METADATA（数字值为1）：已经获取到了足够的媒体数据，但是当前播放位置没有有效的媒体数据（也就是说，获取到的媒体数据无效，不能播放）。

HAVE_CURRENT_DATA（数字值为2）：当前播放位置已经有数据可以播放，但没有获取到可以让播放器前进的数据。当媒体为视频时，意思是当前帧的数据已获得，但还没有获取到下一帧的数据，或者当前帧已经是播放的最后一帧。

HAVE_FUTURE_DATA（数字值为3）：当前播放位置已经有数据可以播放，而且也获取到了可以让播放器前进的数据。当媒体为视频时，意思是当前帧的数据已获得，而且也获取到了下一帧的数据，当前帧是播放的最后一帧时，readyState属性不可能为HAVE_FUTURE_DATA。

HAVE_ENOUGH_DATA（数字值为4）：当前播放位置已经有数据可以播放，同时也获取到了可以让播放器前进的数据，而且浏览器确认媒体数据以某一种速度进行加载，可以保证有足够的后续数据进行播放。

readyState属性为只读属性。

❑ seeking属性与seekable属性

可以使用video元素或audio元素的seeking属性返回一个布尔值，表示浏览器是否正在请求某一特定播放位置的数据，true表示浏览器正在请求数据，false表示浏览器已停止请求。

可以使用video元素或audio元素的seekable属性来返回一个TimeRanges对象，该对象表示请求到的数据的时间范围。当媒体为视频时，开始时间为请求到视频数据第一帧的时间，结束时间为请求到视频数据最后一帧的时间。

seeking属性与seekable属性均为只读属性。

⊖ index表示第几个时间范围。

❑ currentTime属性、startTime属性与duration属性

可以使用video元素或audio元素的currentTime属性来读取媒体的当前播放位置，也可以通过修改currentTime属性来修改当前播放位置。如果修改的位置上没有可用的媒体数据时，将抛出INVALID_STATE_ERR异常；如果修改的位置超出了浏览器在一次请求中可以请求的数据范围，将抛出INDEX_SIZE_ERR异常。

可以使用video元素或audio元素的startTime属性来读取媒体播放的开始时间，通常为0。

可以使用video元素或audio元素的duration属性来读取媒体文件总的播放时间。

三者的值均为时间，单位为秒，currentTime为可读写属性，其余两个均为只读属性。

❑ played属性、paused属性、ended属性

可以使用video元素或audio元素的played属性来返回一个TimeRanges对象，从该对象中可以读取媒体文件的已播放部分的时间段。开始时间为已播放部分的开始时间，结束时间为已播放部分的结束时间。

可以使用video元素或audio元素的paused属性来返回一个布尔值，表示是否处于暂停播放中，true表示媒体暂停播放，false表示媒体正在播放。

可以使用video元素或audio元素的end属性来返回一个布尔值，表示是否播放完毕，true表示媒体播放完毕，false表示还没有播放完毕。

三者均为只读属性。

❑ defaultPlaybackRate属性与playbackRate属性

可以使用video元素或audio元素的defaultPlaybackRate属性读取或修改媒体默认的播放速率。

可以使用video元素或audio元素的playbackRate属性读取或修改媒体当前的播放速率。

❑ volume属性与muted属性

可以使用video元素或audio元素的volume属性读取或修改媒体的播放音量，范围为0到1，0为静音，1为最大音量。

可以使用video元素或audio元素的muted属性读取或修改媒体的静音状态，该值为布尔值，true表示处于静音状态，false表示处于非静音状态。

6.3 方法

video元素与audio元素都具有以下四种方法。

❑ play方法

使用play方法来播放媒体，自动将元素的paused属性的值变为false。

❑ pause方法

使用pause方法来暂停播放，自动将元素的paused属性的值变为true。

❑ load方法

使用load方法来重新载入媒体进行播放，自动将元素的playbackRate属性值变为

defaultPlaybackRate属性的值，自动将元素的error的值变为null。

下面首先来看一个媒体播放的示例，如代码清单6-4所示。

代码清单6-4 媒体播放示例

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"></meta>
<script>
var video;
function init()
{
    video = document.getElementById("video1");
    //监听视频播放结束事件
    video.addEventListener("ended", function()
    {
      alert("播放结束。");
    }, true);
// 发生错误
    video.addEventListener("error",function(){
        switch (video.error.code)
        {
            case MediaError.MEDIA_ERROR_ABORTED:
                alert("视频的下载过程被中止。");
                break;
            case MediaError.MEDIA_ERROR_NETWORK:
                alert("网络发生故障，视频的下载过程被中止。");
                break;
            case MediaError.MEDIA_ERROR_DECODE:
                alert("解码失败。");
                break;
            case MediaError.MEDIA_ERROR_SRC_NOT_SUPPORTED:
                alert("不支持播放的视频格式。");
                break;
            default:
                alert("发生未知错误。");
        }
    },false);
}
function play()
{
    // 播放视频
    video.play();
}
function pause()
{
    //暂停播放
    video.pause();
}
</script>
</head>
<body onload="init()">
  <!-可以添加controls属性来显示浏览器自带的播放用的控制条。 -->
```

```
    <video id="video1" src="test.ogv">
    </video><br/>
    <button onclick="play()">播放</button>
    <button onclick="pause()">暂停</button>
</body>
</html>
```

这段代码的运行结果如图6-2所示。

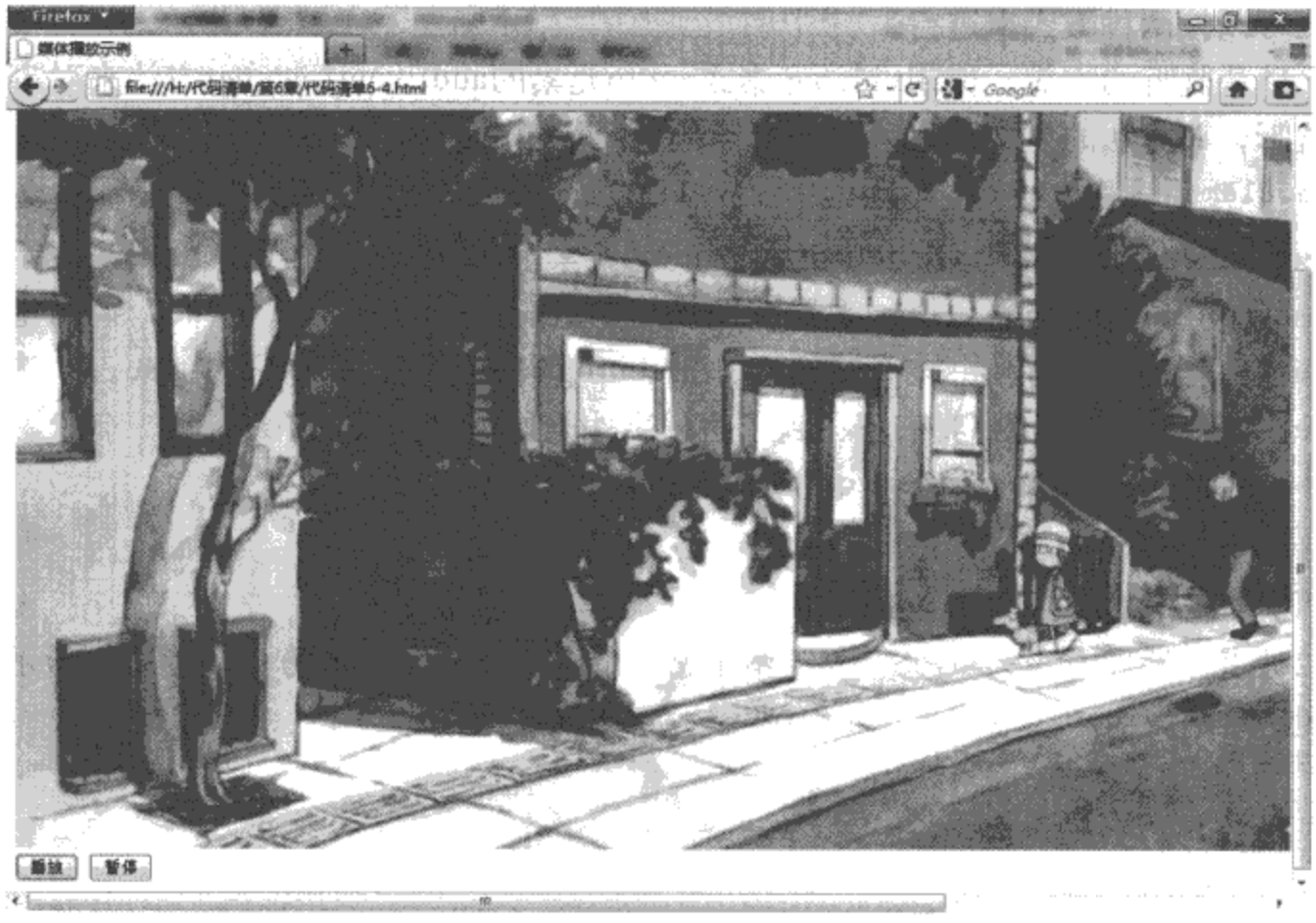

图6-2　媒体播放示例

❑ canPlayType方法

使用canPlayType方法来测试浏览器是否支持指定的媒体类型，该方法的定义如下所示。

```
var support=videoElement.canPlayType(type);
```

videoElement表示页面上的video元素或audio元素。该方法使用一个参数type,该参数的指定方法与source元素的type参数的指定方法相同，都用播放文件的MIME类型来指定，可以在指定的字符串中加上表示媒体编码格式的codes参数。

该方法返回3个可能值：

空字符串：表示浏览器不支持此种媒体类型。

maybe：表示浏览器可能支持此种媒体类型。

probably：表示浏览器确定支持此种媒体类型。

6.4 事件

6.4.1 事件处理方式

在利用video元素或audio元素读取或播放媒体数据的时候，会触发一系列的事件，如果用JavaScript脚本来捕捉这些事件，就可以对这些事件进行处理了。对于这些事件的捕捉及其处理，可以按两种方式来进行。

第一种是监听的方式，使用video元素或audio元素的addEventListener方法来对事件的发生进行监听，该方法的定义如下所示。

```
videoElement.addEventListener (type,listener,useCapture);
```

videoElement表示页面上的video元素或audio元素。type为事件名称，listener表示绑定的函数，useCapture是一个布尔值，表示该事件的响应顺序，该值如果为true，则浏览器采用Capture响应方式，如果为false，浏览器采用bubbing响应方式，一般采用false，默认情况下也为false。在代码清单9-5中我们已见到了这种事件的处理方式，如下所示。

```
video.addEventListener("error", function()
{
(中略)
},false) ;
```

第二种事件处理方式为JavaScript脚本中常见的获取事件句柄的方式，如下例所示。

```
<video id="video1" src="sample.mov" onplay=" begin_playing();" ></video>
function  begin_playing()
{
    (中略)
};
```

6.4.2 事件介绍

接下来，我们在表6-1中详细介绍一下浏览器在请求媒体数据、下载媒体数据、播放媒体数据一直到播放结束这一系列过程中，到底会触发哪些事件。

表6-1 video元素与audio元素相关事件

事　件	描　述
loadstart	浏览器开始在网上寻找媒体数据
progress	浏览器正在获取媒体数据
suspend	浏览器暂停获取媒体数据，但是下载过程并没有正常结束
abort	浏览器在下载完全部媒体数据之前中止获取媒体数据，但是并不是由错误引起的
error	获取媒体数据过程中出错
emptied	video元素或audio元素所在网络突然变为未初始化状态 （可能引起的原因有两个： 1. 载入媒体过程中突然发生一个致命错误 2. 在浏览器正在选择支持的播放格式时，又调用了load方法重新载入媒体）

（续）

事　件	描　述
stalled	浏览器尝试获取媒体数据失败
play	即将开始播放，当执行了play方法时触发，或数据下载后元素被设为autoplay(自动播放)属性
pause	播放暂停，当执行了pause方法时触发
loadedmetadata	浏览器获取完毕媒体的时间长和字节数
loadeddata	浏览器已加载完毕当前播放位置的媒体数据，准备播放
waiting	播放过程由于得不到下一帧而暂停播放（例如下一帧尚未加载完毕），但很快就能够得到下一帧
playing	正在播放
canplay	浏览器能够播放媒体，但估计以当前播放速率不能直接将媒体播放完毕，播放期间需要缓冲
canplaythrough	浏览器能够播放媒体，而且以当前播放速率能够将媒体播放完毕，不再需要进行缓冲
seeking	seeking属性变为true,浏览器正在请求数据
seeked	seeking属性变为false,浏览器停止请求数据
timeupdate	当前播放位置被改变，可能是播放过程中的自然改变，也可能是被人为地改变，或由于播放不能连续而发生的跳变
ended	播放结束后停止播放
ratechange	defaultplaybackRate属性（默认播放速率）或playbackRate属性（当前播放速率）被改变
durationchange	播放时长被改变
volumechange	volume属性（音量）被改变或muted属性（静音状态）被改变

6.4.3 事件捕捉示例

最后，我们在代码清单6-5中看一个如何捕捉事件的示例，在播放过程中会经常触发timeupdate事件来通知当前播放位置的改变，在该示例中，我们捕捉这个timeupdate事件来显示当前的播放进度。

代码清单6-5 事件捕捉示例

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title>视频播放器</title>
<script type="text/javascript">
function playOrPauseVideo()
{
    var videoUrl = document.getElementById("videoUrl").value;
    var video = document.getElementById("video");
    //使用事件监听方式捕捉事件
    video.addEventListener("timeupdate", function()
    {
```

```
        var timeDisplay = document.getElementById("time");
        //用秒数来显示当前播放进度
        timeDisplay.innerHTML = Math.floor(video.currentTime) +"/"+
Math.floor(video.duration) +"(秒)";
    }, false);

    if(video.paused)
    {
        if(videoUrl != video.src)
        {
            video.src = videoUrl;
            video.load();
        }
        else
        {
            video.play();
        }
        document.getElementById("playButton").value = "暂停";
    }
    else
    {
        video.pause();
        document.getElementById("playButton").value = "播放";
    }
}
</script>
</head>
<body>
<video id="video" width="400" height="300" autoplay loop="loop">
</video>
<br />
视频地址：<input type="text" id="videoUrl"/>
<input id="playButton" type="button"
onclick="playOrPauseVideo()" value="播放"/>
<span id="time"></span>
</body>
</html>
```

这段代码的运行结果如图6-3所示。

图6-3 事件捕捉示例

第7章

本地存储

7.1 Web Storage

7.2 本地数据库

本章介绍HTML 5中与本地存储相关的两个重要内容——Web Storage与本地数据库。其中，Web Storage存储机制是对HTML 4中cookies存储机制的一个改善。由于cookies存储机制有很多缺点，HTML 5中不再使用它，转而使用改良后的Web Storage存储机制。本地数据库是HTML 5中新增的一个功能，使用它可以在客户端本地建立一个数据库——原本必须要保存在服务器端数据库中的内容现在可以直接保存在客户端本地了，这大大减轻了服务器端的负担，同时也加快了访问数据的速度。到目前为止本地数据库的功能受到了opera浏览器、Safari浏览器以及Chrome浏览器的支持。

学习内容：

- 掌握Web Storage的基本概念，了解sessionStorage和localStorage,以及两者之间的区别，掌握localStorage与sessionStorage的使用方法。能够使用这两者进行复杂数据的存储，能够使用这两者进行JavaScript对象的存储。
- 掌握本地数据库的基本概念，能够使用openDatabase方法创建与打开数据库，能够使用transaction方法进行事务的处理，能够结合使用transaction方法与executeSql方法来实现数据在本地数据库中的增、删、查、改。

7.1 Web Storage

7.1.1 Web Storage是什么

在HTML 5中，除了Canvas元素之外，另一个新增的非常重要的功能是可以在客户端本地保存数据的Web Storage功能，我们知道，在HTML 4中可以使用cookies在客户端保存诸如用户名等简单的用户信息，但是，通过长期的实际使用下来，人们发现用cookies储存永久数据存在以下几个问题：

- 大小：cookies的大小被限制在4KB。
- 带宽：cookies是随HTTP事务一起被发送的,因此会浪费一部分发送cookies时使用的带宽。
- 复杂性：要正确地操纵cookies是很困难的。

在这种情况下，在HTML 5中重新提供了一种在客户端本地保存数据的功能，它就是Web Storage功能。

Web Storage功能，顾名思义，就是在Web上储存数据的功能，而这里的储存，是针对客户端本地而言的。具体来说，Web Storage又分为两种。

- sessionStorage

将数据保存在session对象中。所谓session，是指用户在浏览某个网站时，从进入网站到浏览器关闭所经过的这段时间，也就是用户浏览这个网站所花费的时间。session对象可以用来保存在这段时间内所要求保存的任何数据。

- localStorage

将数据保存在客户端本地的硬件设备（通常指硬盘，但也可以是其他硬件设备）中，即

使浏览器被关闭了，该数据仍然存在，下次打开浏览器访问网站时仍然可以继续使用。

这两者的区别在于，sessionStorage为临时保存，而localStorage为永久保存。接下来，让我们结合示例来具体看一下。

首先，需要准备一个用来保存数据的网页。在示例网页中，我们在页面上放置的控件如表7-1所示。

表7-1 Web Storage示例的页面中元素

元 素	id	用 途
input type=text	input	输入数据
p	msg	显示数据
button		保存数据
button		保存数据

该示例的HTML页面代码如代码清单7-1所示。

代码清单7-1 Web Storage示例的HTML页面代码

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>Web Storage示例</title>
<script type="text/javascript" src="script.js"></script>
</head>
<body>
<h1>Web Storage 示例</h1>
<p id="msg"></p>
<input type="text" id="input">
<input type="button" value="保存数据" onclick="saveStorage('input');">
<input type="button" value="读取数据"  onclick="loadStorage('msg');">
</body>
</html>
```

点击保存数据按钮时，调用saveStorage方法来保存数据，点击读取数据按钮时，调用loadStorage方法调用数据，这两个方法均在脚本文件script.js中。

接下来，让我们打开这个脚本文件script.js来看一下。在这个脚本文件里，分别使用了sessionStorage与localStorage两种方法。这两种方法都是当用户在input文本框中输入内容并点击保存数据按钮时保存数据，点击读取数据按钮时读取保存后的数据。

但是两种方法对数据的处理方式不一样，使用sessionStorage方法时，如果关闭了浏览器，这个数据就丢失了，下一次打开浏览器，点击读取数据按钮时，读取不到任何数据。

使用localStorage方法时，即使浏览器关闭了，下次打开浏览器时仍然能够读取被保存的数据。但是，数据保存是按不同的浏览器分别进行的，也就是说，如果打开别的浏览器，是读取不到在这个浏览器中保存的数据的。

那么，我们来看一下读取数据部分。下面是读写数据时使用的基本方法。

- sessionStorage

保存数据：sessionStorage.setItem（key,value）；

读取数据：变量=sessionStorage.getItem(key)；

❑ localStorage

保存数据：localStorage.setItem（key,value）；

读取数据：变量= localStorage.getItem(key)；

在保存数据时，如果使用sessionStorage读取或保存数据，则需要使用sessionStorage对象并调用该对象的读写方法；如果使用localStorage读取或保存数据，则需要使用localStorage对象并调用该对象的读写方法。

进行读写时，不管是哪个对象，都会使用getItem方法来读取数据，使用setItem方法来保存数据。保存数据时按“键名/键值”的形式进行保存。使用getItem方法读取数据时，将参数指定为键名，返回键值。使用setItem方法保存数据时，将第一个参数指定为键名，将第二个参数指定为键值。

保存时不允许重复保存相同的键名。保存后可以修改键值，但不允许修改键名（只能重新取键名，然后再保存键值）。

script.js脚本文件中的内容如代码清单7-2所示。

代码清单7-2 Web Storage示例的JavaScript脚本代码

```
//sessionStorage示例
function saveStorage(id)
{
    var target = document.getElementById(id);
    var str = target.value;
    sessionStorage.setItem("message",str);
}
function loadStorage(id)
{
    var target = document.getElementById(id);
    var msg = sessionStorage.getItem("message");
    target.innerHTML,= msg;
}
//localStorage示例
function saveStorage(id)
{
    var target = document.getElementById(id);
    var str = target.value;
    localStorage.setItem("message",str);
}
function loadStorage(id)
{
    var target = document.getElementById(id);
    var msg = localStorage.getItem("message");
    target.innerHTML = msg;
}
```

这段程序在Opera 10浏览器中的运行结果如图7-1所示。

图7-1 Opera 10浏览器中的WebStorage示例

7.1.2 简单Web留言本

前面一节讲到了使用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用处并不是很大，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。因此，让我们来看一下如何利用Web Storage来保存和读取大量数据。

这里，让我们来看一个简单Web留言本的示例。使用一个多行文本框来输入数据，点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（如：1970年1月1日凌晨12点后经过的秒数）的形式进行管理的，所以保存时不可能存在重复的键名。

下面，在代码清单7-3中看一下该示例显示页面用的HTML代码部分。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，点击追加按钮来保存数据，点击初始化按钮来消除全部数据。

代码清单7-3 简单Web留言本的HTML代码

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>简单Web留言本</title>
<script type="text/javascript" src="script.js"></script>
</head>
<body>
<h1>简单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>
```

接下来，让我们再打开script.js这个脚本文件看一下。现在，这个脚本文件里含有三个供点击按钮时调用的函数，分别是saveStorage、loadStorage、clearStorage。

❑ 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戳，然后调用localStorage.setItem方法，将得到的时间戳作为键值，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——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。譬如想得到第6条数据，传入的index为5（index是从0开始计算的）。

先用loadStorage.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句语句“localStorage.clear()”，调用localStorage的clear方法时，所有保存在localStorage中的数据会全部被清除。

script.js脚本中的全部代码如代码清单7-4所示。

代码清单7-4　简单Web留言本的JavaScript脚本代码

```
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');
}
```

该示例在Opera 10浏览器中的运行结果如图7-2所示。

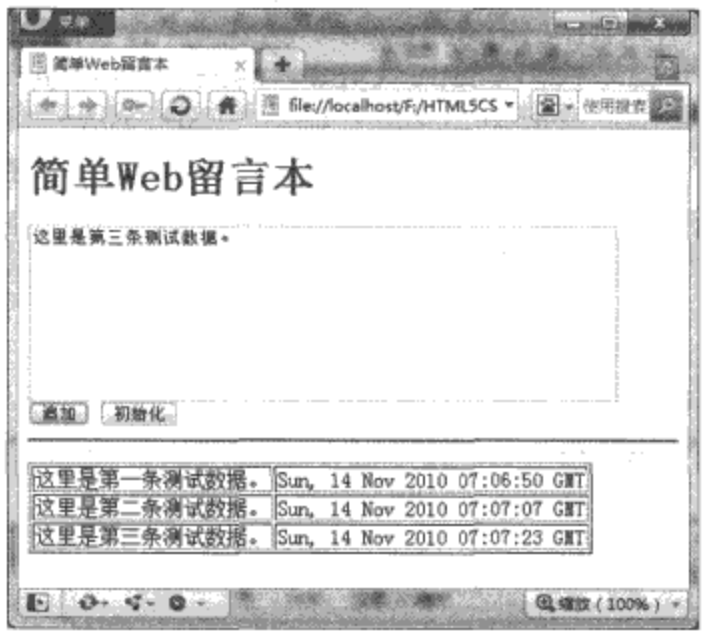

图7-2 Opera 10浏览器中的简单Web留言本示例

7.1.3 作为简易数据库来利用

接下来，让我们来看一个更复杂的问题，能不能将Web Storage作为简易数据库来利用呢？

如果想要将Web Storage作为数据库来利用的话，有几个必须要考虑的问题。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？然后，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。

这一次，我们来看一个非常简单的“客户联系信息管理网页”这个示例。客户的联系信息分为姓名、E-mail地址、电话号码、备注这几列，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，怎样将客户联系信息分几列来进行保存呢？要做到这一点，需要使用JSON格式[⊖]。

⊖ JSON格式是JavaScript Object Notation的缩写，是将JavaScript中的对象作为文本形式来保存时使用的一种格式。

用这种JSON的格式作为文本保存来保存对象，获取该对象时再通过JSON格式来获取，应该就可以在Web Storage中保存和读取具有复杂结构的数据了。

首先，我们在代码清单7-5中看一下这个示例的HTML 5页面代码。

代码清单7-5 简易数据库的HTML页面代码

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>简易数据库示例</title>
<script type="text/javascript" src="script.js"></script>
</head>
<body>
<h1>使用Web Storage来做简易数据库示例</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>
```

接下来，让我们仍然打开script.js这个JavaScript脚本文件来看一下，这一次，在这个脚本文件中存放了两个函数，分别是保存数据用的saveStorage函数与检索数据用的findStorage函数。让我们来分别看一下这两个函数。

❑ saveStorage函数中的流程。

1）从各输入文本框中获取数据。

2）创建对象，将获取的数据作为对象的属性进行保存。

3）将对象转换成JSON格式的文本数据。

4）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.stringify(data);
```

该方法接受一个参数data，该参数表示要转换成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。

❑ findStorage函数中的流程。

1）从localStorage中，将检索用的姓名作为键值，获取对应的数据。

2）将获取的数据转换成JSON对象。

3）取得JSON对象的各个属性值，创建要输出的内容。

4）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。该方法的使用方法如下所示。

```
var data =  JSON.parse(str);
```

该方法接受一个参数str，此参数表示从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。

本次示例的关键是利用了JSON对象的stringify方法与parse方法。但是，这个JSON对象只是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，请注意这一点。现在支持JSON对象的浏览器有IE8以上，Firefox 3.6以上，Google Chrome 5以上，Safari 5以上，Opera 10以上的浏览器。

script.js脚本文件中的代码如代码清单7-6所示。

代码清单7-6 简易数据库的JavaScript脚本代码

```
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;
}
```

这段代码在Opera 10浏览器中的运行结果如图7-3所示。

图7-3 使用Web Storage来做简易数据库的示例

7.2 本地数据库

7.2.1 本地数据库的基本概念

在HTML 5中，大大丰富了客户端本地可以存储的内容，添加了很多功能来将原本必须要保存在服务器上的数据转为保存在客户端本地，从而大大提高了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。

在这其中，一项非常重要的功能就是数据库的本地存储功能。在HTML 5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HTML 5中，可以就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。

现在，像这种不需要存储在服务器上的，被称为“SQLLite”的文件型SQL数据库已经得到了很广泛的利用，所以HTML 5中也采用了这种数据库来作为本地数据库。因此，如果先掌握了SQLLite数据库的基本知识的话，接着再学如何使用HTML 5的数据库也就不困难了。

那么，要使用SQLLite数据库，应该怎样编写JavaScript脚本呢？总的说来，有两个必要的步骤。

- 创建访问数据库的对象。
- 使用事务处理。

首先，必须要使用openDatabase方法来创建一个访问数据库的对象。该方法的使用方法如下所示。

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

该方法使用四个参数，第一个参数为数据库名，第二个参数为版本号，第三个参数为数据库的描述，第四个参数为数据库的大小。该方法返回创建后的数据库访问对象，如果该数

据库不存在，则创建该数据库。

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打扰。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给修改掉的话，会引起很多意想不到的后果。因此，可以使用事务来达到在操作完了之前，阻止别的用户访问数据库的目的。

transaction方法的使用方法如下所示。

```
db.transaction(function (tx) {
   tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, Log)');
});
```

transaction方法使用一个回调函数为参数。在这个函数中，执行访问数据库的语句。

7.2.2 用executeSql来执行查询

接下来，我们来看一下在transaction的回调函数内，到底是怎样访问数据库的。这里，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);
```

该方法使用四个参数，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 dataHandler(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,errmsg) {//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。

那么，我们来看一下，当执行查询操作时，如何从查询到的结果数据集中，依次把数据取出到页面上来,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google Chrome 5浏览器中，不支持rows.Item（[index]）的形式。

7.2.3 使用数据库实现Web留言本

接下来，让我们仍然用Web留言本作为示例，来看一下具体应该怎样对数据库进行一些简单的操作。虽然在第7.1节中，我们已经介绍过了使用Web Storage，可以实现Web留言本的功能，但是在这里，我们来看一下怎样利用数据库来实现同样的功能。

首先，我们来看一下这个示例的界面。界面中，存在一个输入姓名用的文本框，一个输入留言用的文本框，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。

点击按钮时，调用saveData函数（onclick="saveData();"），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（<body onload="init()">），将数据库中全部已保存的留言信息显示在表格中。

界面代码显示如代码清单7-7所示。

代码清单7-7 Web留言本界面

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>使用数据库实现Web留言本</title>
<script type="text/javascript" src="script.js"></script>
</head>
<body onload="init();">
<h1>使用数据库实现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>
```

接下来，让我们打开script.js这个脚本文件来看一下。

❑ 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);
```

在这个脚本文件的一开始，使用了一个变量datatable。用这个变量来代表页面中的表格（table元素），之后会使用到这个变量。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这个数据库并对其进行访问。

❑ 初始化

接下来，编写了一个init函数。这个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。

❑ 擦除表格中当前显示的数据

脚本文件中init函数之后是removeAllData函数，这个函数是在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

接下来是showData函数，该函数使用一个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在表格中添加的这一行对应的各列中。

❑ 显示全部数据

接下来是showAllData函数，在该函数中使用transaction方法，在该方法的回调函数中执行executeSql方法获取全部数据。获取到数据之后，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

接下来是addData函数，该函数在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，将作为参数传入进来的数据保存在数据库中。

❑ 保存数据

最后是saveData函数，在该函数中首先调用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。

这个脚本文件的整个流程就介绍到这里，在下一节中将详细介绍该脚本中的有关细节部分，在这里我们首先来看一下这个脚本文件的完整代码以及运行结果。代码清单7-8所示为这个脚本文件的全部代码。

代码清单7-8 Web留言本界面的脚本文件

```
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)
                        {
                            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();
}
```

这段代码在Chrome浏览器中的运行结果如图7-4所示。

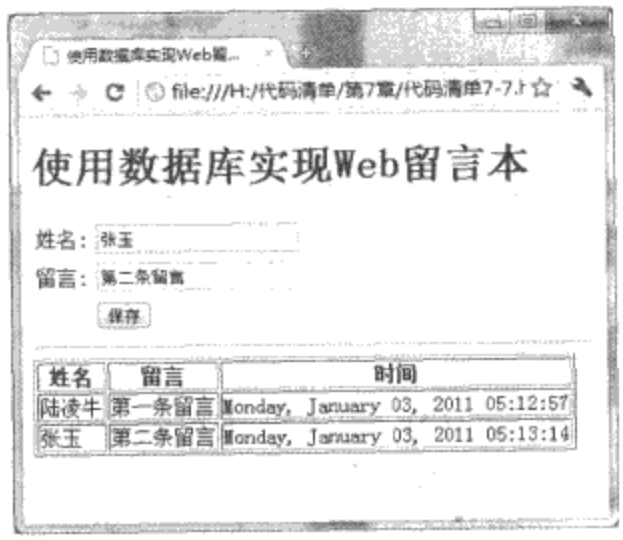

图7-4 Web留言本示例

7.2.4 transaction方法中的处理

在7.2.3节中讲到了script.js这个脚本文件中的两个函数——addData函数与showAllData函数，其中使用到了访问数据库用的transaction方法。现在，我们针对transaction方法做一个详细介绍。

❑ 追加数据

在addData函数中的transaction方法中，使用了回调函数，如下所示。

```
tx.executeSql(
                'INSERT INTO MsgData VALUES(?, ?, ?)',
                [name, message, time],
                function(tx, rs) {……追加数据成功时执行的处理……},
                function(tx, error) {……追加数据失败时执行的处理……}
               );
```

在这里，使用了“INSERT INTO MsgData VALUES(?, ?, ?)”这句SQL语句来追加数据，同时使用了[name, message, time]数组来传入SQL语句中所需的参数。真正对数据库执行时的

SQL语句如下所示。

```
INSERT INTO MsgData VALUES(name, message, time);
```

这条语句的作用是在数据表中插入一条数据。

❑ 创建数据表

获取全部数据并显示的代码被书写在showAllData函数中。在这个函数中，使用了两次transaction方法，第一次使用时的代码如下所示。

```
tx.executeSql('CREATE TABLE IF NOT EXISTS MsgData(name TEXT,
message TEXT,time INTEGER)',[]);
```

真正对数据库执行时的SQL语句如下所示。

```
CREATE TABLE IF NOT EXISTS MsgData(name TEXT, message TEXT, time INTEGER);
```

这条语句的作用是在数据库中创建一张数据表。

在本例中，在数据库里创建了一个带有三个字段的数据表MsgData：第一个字段为TEXT类型的name字段，第二个字段为TEXT类型的message字段，第三个字段为INTEGER类型的time字段。

请注意，如果已经存在了数据表，重复创建该数据表时会引发错误，所以前面必须要加上“IF NOT EXISTS”条件判断语句。这样，当想创建的表在数据库中已经存在时，就不会重复创建了。

❑ 获取全部数据

在showAllData函数中，除了上面讲的为了创建数据表而使用了transaction方法之外，为了获取全部数据，又使用了一次transaction方法，如下所示。

```
tx.executeSql('SELECT * FROM MsgData', [], ……略…… );
```

这里，使用了“SELECT * FROM MsgData”这个SQL语句，该语句的作用是从MsgData这张数据表中获取全部数据。如果获取成功时，执行回调函数，如下所示。

```
function(tx, rs)
{
        removeAllData();
        for(var i = 0; i < rs.rows.length; i++)
        {
            showData(rs.rows.item(i));
        }
}
```

调用removeAllData将页面上表格中当前显示的数据全部擦除后，执行循环，将获取到的所有数据都以rs.rows.item(i)的形式作为参数传入showData函数中进行显示。rs.rows代表了获取到的数据的所有行，而rs.rows.item(i)则代表了第i行中的数据，这些数据都以属性和属性值的形式存放在rs.rows.item(i)对象中，并通过访问属性的方法来获取每个字段的内容。本例中通过访问rs.rows.item(i).name、rs.rows.item(i).length、rs.rows.item(i).time这三个属性来获取每行数据的name字段、length字段、time字段中的内容。

第8章

离线应用程序

8.1 离线Web应用程序详解

8.2 manifest文件

8.3 浏览器与服务器的交互过程

8.4 applicationCache对象

在HTML 5中，提供了一个供本地缓存使用的API。使用这个API，我们可以实现离线Web应用程序[⊖]的开发，本章将对这个API做一个详细介绍。

学习内容：

- 掌握离线Web应用程序的基本概念，什么是离线Web应用程序，为什么要开发离线的Web应用程序。掌握什么是本地缓存，本地缓存与网页缓存有什么区别。掌握什么是manifest文件，怎样在manifest文件中指定哪些内容需要进行本地缓存，哪些内容不需要。
- 掌握首次访问网站后，再次访问网站而manifest文件没有更新，再次访问网站而manifest文件已经被更新时浏览器与服务器的交互过程，在这些过程中如何进行本地缓存的更新。
- 掌握进行本地缓存时所使用到的applicationCache对象，怎样利用这个对象的swapCache方法来手工更新本地缓存，控制什么时候进行本地缓存的更新，以及在浏览器与服务器进行交互的过程中依次触发的applicationCache对象的事件。

8.1 离线Web应用程序详解

8.1.1 新增的本地缓存

今天，Web应用程序已经变得越来越复杂，越来越成熟了，很多领域都在利用着Web应用程序。但是，它有一个致命的缺点：如果用户没有和Internet建立连接，他就不能利用这个Web应用程序了。

因此，在HTML 5中，新增了一个API，它使用一个本地缓存机制很好地解决了这个问题，为离线Web应用程序的开发提供了可能性。

为了让Web应用程序在离线状态时候也能正常工作，就必须要把所有构成Web应用程序的资源文件，诸如HTML文件、CSS文件、JavaScript脚本文件等放在本地缓存中，当服务器没有和Internet建立连接的时候，也可以利用本地缓存中的资源文件来正常运行Web应用程序。

8.1.2 本地缓存与浏览器网页缓存的区别

Web应用程序的本地缓存与浏览器的网页缓存在许多方面都存在着明显的区别。

首先，本地缓存是为整个Web应用程序服务的，而浏览器的网页缓存只服务于单个网页。任何网页都具有网页缓存，而本地缓存只缓存那些你指定缓存的网页。其次，网页缓存也是不安全、不可靠的，因为我们不知道在网站中到底缓存了哪些网页，以及缓存了网页上的哪些资源。而本地缓存是可靠的，我们可以控制对哪些内容进行缓存，不对哪些内容进行缓存，开发人员还可以用编程的手段来控制缓存的更新，利用缓存对象的各种属性、状态和事件来

⊖ 离线Web应用程序是指：当客户端本地与Web应用程序的服务器没有建立连接时，也能正常在客户端本地使用该Web应用程序进行有关操作。

开发出更为强大的离线应用程序。

8.2　manifest文件

Web应用程序的本地缓存是通过每个页面的manifest文件来管理的。manifest文件是一个简单文本文件，在该文件中以清单的形式列举了需要被缓存或不需要被缓存的资源文件的文件名称，以及这些资源文件的访问路径。你可以为每一个页面单独指定一个manifest文件，也可以对整个Web应用程序指定一个总的manifest文件。代码清单8-1为manifest文件的一个示例，该文件为hello.html网页的manifest文件，我们用这个示例来对manifest文件做一个详细介绍。

代码清单8-1　manifest文件示例

```
CACHE MANIFEST
#文件的开头必须要书写CACHE MANIFEST
#这个manifest文件的版本号
#version 7
CACHE:
other.html
hello.js
images/myphoto.jpg
NETWORK:
http://Lulingniu/NotOffline
NotOffline.asp
*
FALLBACK:
online.js locale.js
CACHE:
newhello.html
newhello.js
```

在manifest文件中，第一行必须是“CACHE MANIFEST”文字，以把本文件的作用告知给浏览器，即对本地缓存中的资源文件进行具体设置。同时，真正运行或测试离线Web应用程序的时候，需要对服务器进行配置，让服务器支持text/cache-manifest这个MIME类型（在HTML 5中规定manifest文件的MIME类型是 text/cache-manifest）。例如对Apache服务器进行配置的时候，需要找到{apache_home}/conf/mime.types这个文件，并在文件最后添加如下所示的一行代码。

```
text/cache-manifest    manifest
```

在微软的IIS服务器中的步骤如下所示。

1）右键选择默认网站或需要添加类型的网站，弹出属性对话框。

2）选择“HTTP头”标签。

3）在MIME映射下，单击文件类型按钮。

4）在打开的MIME类型对话框中单击新建按钮。

5）在关联扩展名文本框中输入“manifest”，在内容类型文本框中输入“text/cache-manifest”，然后单击确定按钮。

在manifest文件中，可以加上注释来进行一些必要的说明或解释，注释行以“#”文字开头。

在manifest文件中可以（而且最好）加上一个版本号，以表示这个manifest文件的版本。版本号可以是任何形式，譬如上面的“version 201011211108”，更新manifest文件的时候一般也会对这个版本号进行更新。

接下来，指定资源文件，文件路径可以是相对路径，也可以是绝对路径。指定时每个资源文件为一行。

在指定资源文件的时候，可以把资源文件分为三类，分别是CACHE、NETWORK、FALLBACK。

- 在CACHE类别中指定需要被缓存在本地的资源文件。为某个页面指定需要本地缓存的资源文件时，不需要把这个页面本身指定在CACHE类别中，因为如果一个页面具有manifest文件，浏览器会自动对这个页面进行本地缓存。
- NETWORK类别为显式指定不进行本地缓存的资源文件，这些资源文件只有当客户端与服务器端建立连接的时候才能访问。本示例该类别中的“*”为通配符，表示没有在本manifest文件中指定的资源文件都不进行本地缓存。
- FALLBACK类别中的每行中指定两个资源文件，第一个资源文件为能够在线访问时使用的资源文件，第二个资源文件为不能在线访问时使用的备用资源文件。

每个类别都是可选的。但是如果文件开头没有指定类别而直接书写资源文件的时候，浏览器把这些资源文件视为CACHE类别，直到看见文件中第一个被书写出来的类别为止。例如，代码清单8-2中，浏览器会把NETWORK类别之前的文件都视为CACHE类别。

代码清单8-2 省略CACHE类别示例

```
CACHE MANIFEST
#此处没有写明CACHE类别
other.html
hello.js
images/myphoto.jpg
NETWORK:
http://Lulingniu/NotOffline
NotOffline.asp
```

允许在同一个manifest文件中重复书写同一类别，如代码清单8-3所示。

代码清单8-3 允许重复书写同一类别

```
CACHE MANIFEST
CACHE:
other.html
hello.js
```

```
NETWORK:
http://Lulingniu/NotOffline
NotOffline.asp
//追加CACHE类别中的内容
CACHE:
images/myphoto.jpg
```

为了让浏览器能够正常阅读该文本文件，需要在Web应用程序页面上的html标签的manifest属性中指定manifest文件的URL地址。指定方法如下所示。

```
<!—你可以为每个页面单独指定一个manifest文件—>
<html manifest=" hello.manifest" >
...
</html>
<!—也可以为整个Web应用程序指定一个总的manifest文件—>
<html manifest=" global.manifest" >
...
</html>
```

通过这些步骤，将资源文件保存到本地缓存区的基本操作就完成了。当要对本地缓存区的内容进行修改时，只要修改manifest文件就可以了。文件被修改后，浏览器可以自动检查manifest文件，并自动更新本地缓存区中的内容。

8.3 浏览器与服务器的交互过程

当使用离线Web应用程序进行工作的时候，有必要理解一下浏览器与服务器之间的交互过程。譬如一个http://Lulingniu网站，以index.html为主页，该主页使用index.manifest文件为manifest文件，在该文件中请求本地缓存index.html、hello.js、hello1.jpg、hello2.jpg这几个资源文件。首次访问http://Lulingniu网站时，它们的交互过程如下所示。

1）浏览器请求访问http://Lulingniu。

2）服务器返回index.html网页。

3）浏览器解析index.html网页，请求页面上所有资源文件，包括HTML文件、图像文件、CSS文件、JavaScript脚本文件，以及manifest文件。

4）服务器返回所有资源文件。

5）浏览器处理manifest文件，请求manifest中所有要求本地缓存的文件，包括index.html页面本身，即使刚才已经请求过这些文件。如果你要求本地缓存所有文件，这将是一个比较大的重复的请求过程。

6）服务器返回所有要求本地缓存的文件。

7）浏览器对本地缓存进行更新，存入包括页面本身在内的所有要求本地缓存的资源文件，并且触发一个事件，通知本地缓存被更新。

现在浏览器已经把本地缓存更新完毕。如果再次打开浏览器访问http://Lulingniu网站，

而且manifest文件没有被修改过，它们的交互过程会如下所示。

1）浏览器再次请求访问http://Lulingniu。

2）浏览器发现这个页面被本地缓存，于是使用本地缓存中index.html页面。

3）浏览器解析index.html页面，使用所有本地缓存中的资源文件。

4）浏览器向服务器请求manifest文件。

5）服务器返回一个304代码，通知浏览器manifest没有发生变化。

只要页面上的资源文件被本地缓存过，下次浏览器打开这个页面时，总是先使用本地缓存中的资源，然后请求manifest文件。

如果再次打开浏览器时manifest文件已经被更新过了，那么浏览器与服务器之间的交互过程如下所示。

1）浏览器再次请求访问http://Lulingniu。

2）浏览器发现这个页面被本地缓存，于是使用本地缓存中index.html页面。

3）浏览器解析index.html页面，使用所有本地缓存中的资源文件。

4）浏览器向服务器请求manifest文件。

5）服务器返回更新过的manifest文件。

6）浏览器处理manifest文件，发现该文件已被更新，于是请求所有要求进行本地缓存的资源文件，包括index.html页面本身。

7）浏览器返回要求进行本地缓存的资源文件。

8）浏览器对本地缓存进行更新，存入所有新的资源文件。并且触发一个事件，通知本地缓存被更新。

需要注意的是，即使资源文件被修改过了，在上面的第3中已经装入的资源文件是不会发生变化的，譬如图片不会突然变成新的图片，脚本文件也不会突然使用新的脚本文件，也就是说，这时更新过后的本地缓存中的内容还不能被使用，只有重新打开这个页面的时候才会使用更新过后的资源文件。另外，如果你不想修改manifest文件中对于资源文件的设置，但是你对服务器上请求缓存的资源文件进行了修改，那么你可以通过修改版本号的方式来让浏览器认为manifest文件已经被更新过了，以便重新下载修改过的资源文件。

在下一节中，让我们来看一看怎样利用applicationCache对象手工进行本地缓存的更新。

8.4 applicationCache对象

applicationCache对象代表了本地缓存，可以用它来通知用户本地缓存中已经被更新，也允许用户手工更新本地缓存。

在前面讲到的浏览器与服务器的交互过程中，当浏览器对本地缓存进行更新，装入新的资源文件时，会触发applicationCache对象的updateready事件，通知本地缓存已被更新。你可以利用这个事件告诉用户本地缓存已经被更新，用户需要手工刷新页面来得到最新版本的应用程序。这部分代码如下所示。

```
applicationCache.onUpdateReady = function () {
//本地缓存已被更新，通知用户。
alert( "本地缓存已被更新,您可以刷新页面来得到本程序的最新版本。" );
};
```

另外，你可以通过applicationCache的swapCache方法来控制如何进行本地缓存的更新及更新的时机。

8.4.1 swapCache方法

swapCache方法用来手工执行本地缓存的更新，它只能在applicationCache对象的updateReady事件被触发时调用，updateReady事件只有在服务器上的manifest文件被更新，并且把manifest文件中所要求的资源文件下载到本地后触发。顾名思义，这个事件的含义是“本地缓存准备被更新”。当这个事件被触发后，我们可以用swapCache方法来手工进行本地缓存的更新。下面，我们来看一下哪些场合需要使用到这个方法。

首先，如果本地缓存的容量非常大（譬如超过100MB），本地缓存的更新工作将需要相对较长的时间，而且还会把浏览器给锁住。这时最好有一个提示，告诉用户正在进行本地缓存的更新，该部分代码如下所示。

```
applicationCache.onUpdateReady = function () {
//本地缓存已被更新，通知用户。
alert( "正在更新本地缓存……" );
applicationCache. swapCache();
alert( "本地缓存已被更新,您可以刷新页面来得到本程序的最新版本。" );
};
```

这时，让我们来考虑一个问题，在上面的代码中，如果不调用swapCache方法会怎么样？本地缓存就不会被更新了吗？回答是否定的，但是，更新的时间不一样。如果不调用swapCache方法，本地缓存将在下一次打开本页面时被更新；如果调用swapCache方法的话，本地缓存将会被立刻更新。因此，你可以使用confirm方法让用户自己选择更新的时机——是立刻更新，还是在下次打开画面时再更新，特别是当他们有可能正在页面上执行一个较大的操作的时候。

另外，尽管使用swapCache方法立刻更新了本地缓存，但是并不意味着我们页面上的图像和脚本文件也会被立刻更新，它们都是在重新打开本页面时才会生效。

接下来，我们看一个完整的使用swapCache方法的示例。在该示例中，使用到了applicationCache对象的另一个方法applicationCache.update，该方法的作用是检查服务器上的manifest文件是否有更新。在打开画面时设定每5秒钟执行一次该方法，检查服务器上的manifest文件是否有更新。如果有更新，浏览器会自动下载manifest文件中所有请求本地缓存的资源文件，当这些资源文件下载完毕时，会触发updateReady事件，询问用户是否立刻刷新页面以使用最新版本的应用程序，如果用户选择立刻刷新，则调用swapCache方法手工更新本地缓存，更新完毕后刷新页面。

画面的HTML代码部分如代码清单8-4所示。

代码清单8-4 swapCache方法示例的HTML代码

```
<!DOCTYPE HTML>
<html manifest=" swapCache.manifest" >
<head>
<meta charset=" UTF-8" >
<title> swapCache方法示例</title>
<script src=" script.js" ></script>
</head>
<body onload=" init()" >
<p> swapCache方法示例</p>
</body>
</html>
```

该HTML中嵌入了一个script.js脚本文件，该脚本文件中的代码如代码清单8-5所示。

代码清单8-5 swapCache方法示例的JavaScript代码

```
function init() {
     setInterval(function() {
         // 手工检查是否有更新
         applicationCache.update();
    }, 5000);
    applicationCache.addEventListener( "updateready" , function() {
         if (confirm("本地缓存已被更新,需要刷新画面来获取应用程序最新版本，是否刷新？")) {
         // (3) 手工更新本地缓存
                applicationCache.swapCache();
                // 重载画面
                location.reload();
          }
    }, true);
}
```

该示例中使用的swapCache.manifest文件内容比较简单，如代码清单8-6所示。

代码清单8-6 swapCache方法示例的manifest文件

```
CACHE MANIFEST
#version 1.20
CACHE:
script.js
```

这部分代码在Opera 10浏览器中的运行结果如图8-1所示。

8.4.2 applicationCache对象的事件

applicationCache对象除了具有update方法与swapCache方法之外，还具有一系列的事件，现在我们对前面讲过的浏览器与服务器的交互过程的内容进行扩充，看看在这个过程里这些事件是如何触发的。

首次访问http://Lulingniu网站：

1）浏览器请求访问http://Lulingniu。

图8-1　swapCache方法示例

2）服务器返回index.html网页。

3）浏览器发现该网页具有manifest属性，触发checking事件，检查manifest文件是否存在。不存在时，触发error事件，表示manifest文件未找到，不执行步骤6开始的交互过程。

4）浏览器解析index.html网页，请求页面上所有资源文件。

5）服务器返回所有资源文件。

6）浏览器处理manifest文件，请求manifest中所有要求本地缓存的文件，包括index.html页面本身，即使刚才已经请求过该文件。如果你要求本地缓存所有文件，这将是一个比较大的重复的请求过程。

7）服务器返回所有要求本地缓存的文件。

8）浏览器触发downloading事件，然后开始下载这些资源。在下载的同时，周期性地触发progress事件，开发人员可以用编程的手段获取多少文件已被下载，多少文件仍然处于下载队列等信息。

9）下载结束后触发cached事件，表示首次缓存成功，存入所有要求本地缓存的资源文件。

再次访问http://Lulingniu网站，步骤1～5同上，在步骤5执行完之后，浏览器将核对manifest文件是否被更新，若没有被更新，触发noupdate事件，步骤6开始的交互过程不会被执行。如果被更新了，将继续执行后面的步骤，在步骤9中不触发cached事件，而是触发updateready事件，这表示下载结束，可以通过刷新页面来使用更新后的本地缓存，或调用swapCache方法来立刻使用更新后的本地缓存。

另外，在访问缓存名单时如果返回一个HTTP404错误（页面未找到），或者410错误（永

久消失），则触发obsolete事件。

在整个过程中，如果任何与本地缓存有关的处理中发生错误的话，都会触发error事件。可能会触发error事件的情况分为以下几种。

- 缓存名单返回一个HTTP404错误（页面未找到），或者410错误（永久消失）。
- 缓存名单被找到且没有更改，但引用缓存名单的HTML页面不能正确下载。
- 缓存名单被找到且被更改，但浏览器不能下载某个缓存名单中列出的资源。
- 开始更新本地缓存时，缓存名单再次被更改。

为了说明这个事件流程，我们在代码清单8-7中，将浏览器与服务器在交互过程中所触发的一系列事件用文字的形式显示在页面上，从这个页面中我们可以看出这些事件发生的先后顺序。

代码清单8-7 applicationCache事件流程示例

```
<!DOCTYPE HTML>
<html manifest=" applicationCacheEvent.manifest" >
<head>
<meta charset=" UTF-8" >
<title>applicationCache事件流程示例</title>
<script>
function init()
{
    var msg=document.getElementById( "msg" );
    applicationCache.addEventListener( "checking" , function() {
        msg.innerHTML+=" checking<br/>" ;
    }, true);
    applicationCache.addEventListener( "noupdate" , function() {
        msg.innerHTML+=" noupdate<br/>" ;
    }, true);
    applicationCache.addEventListener( "downloading" , function() {
        msg.innerHTML+=" downloading<br/>" ;
    }, true);
    applicationCache.addEventListener( "progress" , function() {
        msg.innerHTML+=" progress<br/>" ;
    }, true);
    applicationCache.addEventListener( "updateready" , function() {
        msg.innerHTML+=" updateready<br/>" ;
    }, true);
    applicationCache.addEventListener( "cached" , function() {
        msg.innerHTML+=" cached<br/>" ;
    }, true);
    applicationCache.addEventListener( "error" , function() {
    msg.innerHTML+=" error<br/>" ;
    }, true);
}
</script>
</head>
<body onload=" init()" >
<h1>applicationCache事件流程示例</h1>
<p id=" msg" ></p>
</body>
</html>
```

这段代码运行结果分为以下三种情况。

❑ 在Opera 10浏览器中首次打开网页时的页面如图8-2所示。

图8-2 applicationCache事件流程（首次打开页面时）

❑ 在Opera 10浏览器中再次打开网页（且manifest文件没有更新时）的页面如图8-3所示。

图8-3 applicationCache事件流程（再次打开网页且manifest文件没有更新时）

❑ 在Opera 10浏览器中再次打开网页（且manifest文件已被更新时）的页面如图8-4所示。

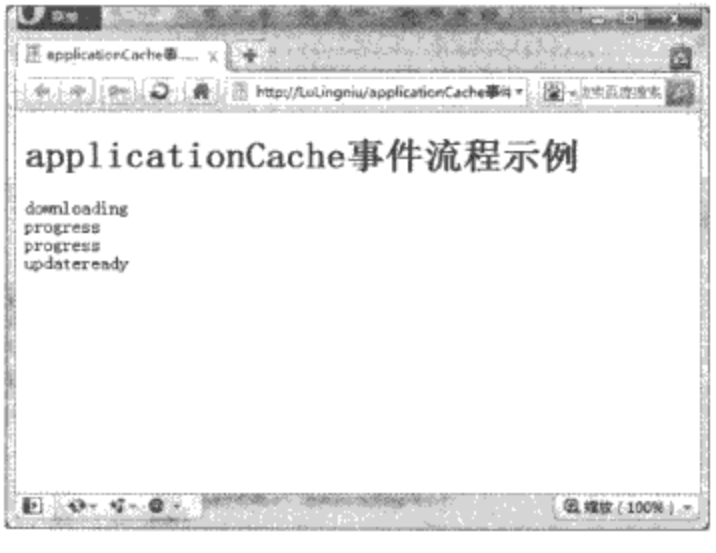

图8-4 applicationCache事件流程（再次打开网页且manifest文件已被更新时）

第9章

通信API

本章介绍HTML 5中新增的与通信相关的两个功能——跨文档消息传输功能与使用Web Sockets API来通过socket端口传递数据的功能。

使用跨文档消息传输功能，你可以在不同网页文档、不同端口、不同域之间进行消息的传递。

使用Web Sockets API，你可以让客户端与服务器端通过socket端口来传递数据，这样做的好处是可以实现数据推送技术——服务器端不再是被动地等待客户端发出的请求，只要客户端与服务器端建立了一次连接之后，服务器端就可以在需要的时候，主动地将数据推送到客户端，直到客户端显示关闭这个连接。

学习内容：

- ❑ 掌握跨文档消息传输的基本概念，掌握怎样实现不同页面、不同端口、不同域之间的消息传递。
- ❑ 掌握Web Sockets通信技术的基本知识，能够在客户端与服务器端之间建立socket连接，并且通过这个连接进行消息的传递，能够实现所有JavaScript对象的传递，能够让客户端显式关闭这个连接。

9.1 跨文档消息传输

9.1.1 跨文档消息传输的基本知识

HTML 5提供了在网页文档之间互相接收与发送信息的功能。使用这个功能，只要获取到网页所在窗口对象的实例，不仅同源（域+端口号）的Web网页之间可以互相通信，甚至可以实现跨域通信。

首先，要想接受从其他的窗口那里发过来的消息，就必须对窗口对象的message事件进行监视，代码如下所示。

```
window.addEventListener("message", function() {...}, false);
```

使用window对象的postMessage方法向其他窗口发送消息，该方法的定义如下所示。

```
otherWindow.postMessage(message, targetOrigin);
```

该方法使用两个参数：第一个参数为所发送的消息文本，但也可以是任何JavaScript对象（通过JSON转换对象为文本）；第二个参数为接收消息的对象窗口的URL地址（例如http://localhost:8080/）。可以在URL地址字符串中使用通配符“*”指定全部地址，不过，建议使用准确的URL地址。otherWindow为要发送窗口对象的引用，可以通过window.open返回该对象，或通过对window.frames数组指定序号（index）或名字的方式来返回单个frame所属的窗口对象。

9.1.2 跨文档消息传输示例

接下来，我们来看一个跨文档消息传输的示例。在该示例中，实现了主页面与主页面中

的iframe子页面之间的互相通信。首先，主页面向iframe子页面发送消息，iframe子页面接受消息，显示在本页面中，然后向主页面返回消息。最后，主页面接受消息，然后将该消息用alert方法弹出。

另外，主页面与iframe子页面被配置在不同域中，以实现跨域通信。

首先，在代码清单9-1中，我们来看一下示例中主页面的代码。

代码清单9-1 跨域通信示例的主页面

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>跨域通信示例</title>
<script type="text/javascript">
// (1) 监听message事件
window.addEventListener("message", function(ev) {
  // (2) 忽略指定URL地址之外的页面传过来的消息
  if (ev.origin != "http://www.blue-butterfly.net") {
    return;
  }
  // (3) 显示消息
  alert("从"+ev.origin + "那里传过来的消息:\n\"" + ev.data + "\"");
}, false);
function hello()
{
  var iframe = window.frames[0];
  // (4) 传递消息
  iframe.postMessage("你好", "http://www.blue-butterfly.net/test/");
}
</script>
</head>
<body>
<h1>跨域通信示例</h1>
<iframe width="400" src=http://www.blue-butterfly.net/test/
onload="hello()">
</iframe>
</body>
</html>
```

该示例的iframe子页面中的代码如代码清单9-2所示。

代码清单9-2 跨域通信示例的iframe页面

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
window.addEventListener("message", function(ev)
```

```
{
    if (ev.origin != "http://Lulingniu")
    {
        return;
    }
    document.body.innerHTML = "从"+ev.origin + "那里传来的消息。<br>\""
    + ev.data + "\"";
    // (5) 向主页面发送消息
    ev.source.postMessage("你好。这里是" + this.location, ev.origin);
}, false);
</script>
</head>
<body></body>
</html>
```

下面对本示例中几个关键之处进行补充说明。

- 通过对window对象的message事件进行监听，可以接收消息。
- 通过访问message事件的origin属性，可以获取消息的发送源（在本例中，主页面的发送源为http://Lulingniu，iframe，子页面的发送源为http://www.blue-butterfly.net）。请注意，发送源与网站的URL地址不是同一概念，发送源只包括域名与端口号，为了不接收从其他源恶意发送过来的消息，最好对发送源做个检查。
- 通过访问message事件的data属性，可以取得消息内容（可以是任何JavaScript对象）。
- 使用postMessage方法发送消息。
- 通过访问message事件的source属性，可以获取消息发送源的窗口对象（准确地说，应该是窗口的代理对象）。

这段代码的运行结果如图9-1所示。

图9-1 跨域通信示例

请记住，本节中所说的网页文档与网页文档之间的消息传输，针对的不仅仅是文本消息。如果使用JSON对象的stringify方法将JavaScript对象转成文本，使用JSON对象的parse方法将文本还原为JavaScript对象，则任何JavaScript对象都可以通过这种方式在网页文档与网页文档之间、端口与端口之间、域与域之间互相传递。

现在，Firefox 4、Safari 4、Chrome 8、Opera 10都支持这种跨文档的消息传输方式。

9.2 Web Sockets通信

9.2.1 Web Sockets通信的基本知识

Web Sockets是HTML 5提供的在Web应用程序中客户端与服务器端之间进行的非HTTP的通信机制。它实现了用HTTP不容易实现的服务器端的数据推送等智能通信技术，因此受到了高度关注。

使用Web Sockets API可以在服务器与客户端之间建立一个非HTTP的双向连接。这个连接是实时的，也是永久的，除非被显式关闭。这意味着当服务器想向客户端发送数据时，可以立即将数据推送到客户端的浏览器中，无须重新建立连接。只要客户端有一个被打开的socket（套接字）并且与服务器建立了连接，服务器就可以把数据推送到这个socket上，服务器不再需要轮询客户端的请求，从被动转为了主动。

9.2.2 使用Web Sockets API

Web Sockets的API本身非常简单。将URL字符串作为参数，然后调用WebSocket对象的构造器来建立与服务器之间的通信连接，如下所示。

```
var webSocket = new WebSocket("ws://localhost:8005/socket");
```

URL字符串必须以“ws”或“wss”（加密通信时）文字作为开头。这个URL字符串被设定好之后，在JavaScript脚本中可以通过访问WebSocket对象的url属性来重新获取。

通信连接建立好之后，就可以进行客户端与服务器端的双向通信了。使用WebSocket对象的send方法对服务器发送数据，只能发送文本数据，但是可以使用JSON对象把任何JavaScript对象转换成文本数据后进行发送。使用send方法的代码如下所示。

```
webSocket.send("data");
```

通过获取onmessage事件句柄来接收服务器传过来的数据，如下所示。

```
webSocket.onmessage=function(event)
{
    var data=event.data;
    ...
}
```

通过获取onopen事件句柄来监听socket的打开事件，如下所示。

```
webSocket.onopen=function(event)
{
    //开始通信时的处理
};
```

通过获取onclose事件句柄来监听socket的关闭事件，如下所示。

```
webSocket.onclose=function(event)
{
    //通信结束时的处理
};
```

通过close方法来关闭socket，切断通信连接，如下所示。

```
webSocket.close();
```

另外，可以通过读取readyState的属性值来获取WebSocket对象的状态。readyState属性存在以下几种属性值：

- CONNECTING（数字值为0），表示正在连接。
- OPEN（数字值为1），表示已建立连接。
- CLOSING（数字值为2），表示正在关闭连接。
- CLOSED（数字值为2），表示已关闭连接。

9.2.3 Web Sockets API使用示例

接下来，我们在代码清单9-3中看一个完整的使用Web Sockets的示例，在运行该示例之前，首先要在服务器端指定好它用的socket（套接字）应用程序，并且在服务器的配置文件中指定好运行该socket应用程序的主机与端口，然后再运行配置好的服务器，本示例中的服务器端就准备好了。

这里，我们主要看一下怎样在客户端使用Web Sockets技术与服务器端进行连接并且收发信息。

代码清单9-3 Web Sockets使用示例

```
<html>
<meta charset="UTF-8"></meta>
<head>
<title>WebSockets客户端示例</title>
</head>
<script>
var webSocket;
function connect()
{
    try
    {
        var readyState=new Array("正在连接","已建立连接","正在关闭连接",
        "已关闭连接");
        var host="ws://localhost:8005/socket";
        webSocket=new WebSocket(host);
        var message=document.getElementById("message");
        message.innerHTML+="<p>Socket状态："
        +readyState[webSocket.readyState]+"</p>";
        webSocket.onopen=function()
        {
            message.innerHTML+="<p>Socket状态："
            +readyState[webSocket.readyState]+"</p>";
        }
        webSocket.onmessage=function(msg)
        {
            message.innerHTML+="<p>接收信息："+msg.data+"</p>";
        }
        webSocket.onclose=function()
```

```
            {
                message.innerHTML+="<p>Socket状态："
                +readyState[webSocket.readyState]+"</p>";
            }
        }
        catch(exception)
        {
        message.innerHTML+="<p>有错误发生</p>";
        }
    }
    function send()
    {
        var text=document.getElementById("text").value;
        var message=document.getElementById("message");
        if(text=="")
        {
            message.innerHTML+="<p>请输入一些文字</p>";
            return;
        }
        try
        {
            webSocket.send(text);
            message.innerHTML+="<p>发送数据："+text+"</p>";
        }
        catch(exception)
        {
            message.innerHTML+="<p>发送数据出错</p>";
        }
        document.getElementById("text").value="";
    }
    function disconnect()
    {
        webSocket.close();
    }
    </script>
    <body>
    <h1>WebSockets客户端示例</h1>
    <div id="message"></div>
    <p>请输入一些文字</p>
    <input id="text" type="text"></input>
    <button id="connect" onclick="connect();">建立连接</button>
    <button id="send" onclick="send();">发送数据</button>
    <button id="disconnect" onclick="disconnect();">断开连接</button>
    </body>
    </html>
```

这段代码的运行结果如图9-2所示。

9.2.4 发送对象

在前面我们介绍过，使用WebSockets API，不仅可以发送文本数据，而且可以使用JSON对象来发送一切JavaScript中的对象。使用JSON对象的关键是使用它的两个方法——JSON.parse方法与JSON.stringify方法，其中JSON.stringify方法把JavaScript对象转换成为文

本数据，JSON.parse方法将文本数据转回为JavaScript对象。接下来我们在代码清单9-4中看一下具体怎么使用JSON对象来发送和接收JavaScript对象，在该示例中，假定接收的对象为一个操纵数据库的对象，根据数据库对象的类型来选择数据库，然后在数据库中插入接收的数据，最后把插入结果与时间作为对象重新使用WebSocket对象进行返回。

图9-2 WebSockets客户端示例

代码清单9-4 使用WebSocket API发送JavaScript对象的示例

```
<script>
var host="ws://localhost:8005/socket";
var webSocket=new WebSocket(host);
var userName;
var userAge;
var successFlag;
var currentTime;
webSocket.onmessage=function(event)
{
    var DataBase=JSON.parse(event.data);
    userName= DataBase.userName;
    userAge= DataBase.userAge;
    successFlag=false;
    if(DataBase.DataType=="SQLServer")
    {
        //在SQL Server数据库中插入数据
        successFlag=InsertSQLData();
    }
    else if(DataBase.DataType=="ORACLE")
    {
        //在ORACLE数据库中插入数据
        successFlag=InsertORACLEData();
    }
    currentTime= new Date();
    webSocket.send(JSON.stringify({
        result:successFlag,
        time: currentTime}
    ));
}
</script>
```

第10章
使用Web Workers处理线程

本章介绍HTML 5中新增的与线程相关的一个功能——使用Web Workers来实现Web平台上的多线程处理功能。通过Web Workers，你将可以创建一个不会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。通过Web Workers，你可以将耗时较长的处理交给后台线程去运行，从而解决了HTML 5之前因为某个处理耗时过长而跳出一个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。

学习内容：

- 掌握Web Workers的基本知识，能够使用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。
- 掌握在主线程之间嵌套子线程的方法，能够利用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、方法与事件。

10.1　基础知识

Web Workers是在HTML 5中新增的，用来在Web应用程序中实现后台处理的一项技术。

在使用HTML 4与JavaScript创建出来的Web程序中，因为所有的处理都是在单线程内执行的，所以如果花费的时间比较长的话，程序界面会处于长时间没有响应的状态。最恶劣的是，当时间长到一定程度的话，浏览器还会跳出一个提示脚本运行时间过长的提示框，使用户不得不中断正在执行的处理。

为了解决这个问题，HTML 5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（在HTML 5中被称为worker），如果将可能耗费较长时间的处理交给后台去执行的话，对用户在前台页面中执行的操作就完全没有影响了。

创建后台线程的步骤十分简单。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对象就可以了，如下例所示。

```
var worker=new Worker("worker.js");
```

但是，要注意在后台线程中是不能访问页面或窗口对象的。如果在后台线程的脚本文件中使用到window对象或document对象，则会引起错误的发生。

另外，可以通过发送和接收消息来与后台线程互相传递数据。通过对Worker对象的onmessage事件句柄的获取可以在后台线程之中接收消息，如下例所示。

```
worker.onmessage=function(event)
{
    //处理收到的消息
}
```

使用Worker对象的postMessage方法来对后台线程发送消息，如下例所示。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(message);
```

另外，同样可以通过获取Worker对象的onmessage事件句柄及Worker对象的postMessage方法在后台线程内部进行消息的接收和发送。

接下来，让我们看一个使用后台线程的示例。在该示例中，放置了一个文本框，用户在该文本框中输入数字，然后点击旁边的计算按钮，在后台计算从1到给定数值的合计值。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，采取了循环计算的方法。

首先，在代码清单10-1中，给出在HTML 4中的关于这个求和运算的示例代码。

代码清单10-1 HTML 4中的求和运算示例

```
<!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;
    //循环计算求和
    for (var i = 0; i <= num; i++)
    {
      result += i;
    }
    alert("合计值为" + result + "。");
}
</script>
</head>
<body>
<h1>从1到给定数值的求和示例</h1>
输入数值:<input type="text" id="num">
<button onclick="calculate()">计算</button>
</body>
</html>
```

执行这段代码的时候，在数值文本框中输入数值，在点击计算按钮之后，并在弹出合计值消息框之前，用户是不能在该页面上进行操作的。另外，虽然用户在文本框中输入比较小的值时，不会有什么问题，但是当用户在该文本框中输入100亿以上的值（这个值因不同浏览器而异）时，浏览器跳出一个如图10-1所示的提示脚本运行时间过长的

图10-1 Firefox浏览器中提示脚本运行时间过长的对话框

对话框，导致用户不得不停止当前计算。

在HTML 5中，可以对以上示例重新书写，使用WebWorkers API让耗时较长的运算在后台运行，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。代码清单10-2是对上例进行修改后的HTML 5中的代码。

代码清单10-2　HTML 5中的求和运算示例

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
// 创建执行运算的线程
var worker = new Worker("SumCalculate.js");
//接收从线程中传出的计算结果
worker.onmessage = function(event)
{
    //消息文本放置在data属性中,可以是任何JavaScript对象.
    alert("合计值为" + event.data + "。");
};
function calculate()
{
    var num = parseInt(document.getElementById("num").value, 10);
    //将数值传给线程
    worker.postMessage(num);
}
</script>
</head>
<body>
<h1>从1到给定数值的求和示例</h1>
输入数值:<input type="text" id="num">
<button onclick="calculate()">计算</button>
</body>
```

在这个示例中，把对于给定值的求和运算的处理放到了线程中单独执行，并且把线程代码单独书写在SumCalculate.js这个脚本文件中，代码清单10-3为这个脚本文件中的代码。

代码清单10-3　求和运算的线程脚本文件

```
onmessage = function(event)
{
    var num = event.data;
    var result = 0;
    for (var i = 0; i <= num; i++)
        result += i;
    //向线程创建源送回消息
    postMessage(result);
}
```

这个求和运算的示例在Firefox 4、Safari 4、Google Chrome 3、Opera 10浏览器中均能正常运行，在Firefox4中的运行结果如图10-2所示。

图10-2 WebWorkers API使用示例

10.2 与线程进行数据的交互

在上一节中我们介绍过，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。接下来我们来看一个后台线程与前台页面进行数据交互的示例。在该示例中页面上随机生成了一个整数的数组，然后将该整数数组传入线程，挑选出该数组中可以被3整除的数字，然后显示在画面的表格中，如果能够把数组显示在画面的表格中，那么就能够把字符串、数组、列表中的数据都采取同样的方法显示在画面的表格、表单控件甚至于统计图中了。

首先，我们在代码清单10-4中看一下该示例的前台页面中的代码，该页面的HTML代码部分有一个空白表格，在前台脚本中随机生成整数数组，然后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。

代码清单10-4 与线程进行数据交互示例的前台页面代码

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>与线程进行数据交互示例</title>
<script type="text/javascript">
var intArray=new Array(100);//随机数组
var intStr="";                    //将随机数组用字符串进行连接
//生成100个随机数
for(var i=0;i<100;i++)
{
    intArray[i]=parseInt(Math.random()*100);
    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;                  //行号
        var k;                  //列号
        var tr;
        var 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;
            //该行不存在
            if(k==0)
            {
                //添加行
                tr=document.createElement("tr");
                tr.id="tr"+j;
                table.appendChild(tr);
            }
            //该行已存在
            else
            {
                //获取该行
                tr=document.getElementById("tr"+j);
            }
            //添加列
            td=document.createElement("td");
            tr.appendChild(td);
            //设置该列内容
            td.innerHTML=intArray[j*10+k];
            //设置该列背景色
            td.style.backgroundColor="blue";
            //设置该列字体颜色
            td.style.color="white";
            //设置列宽
            td.width="30";
        }
    }
};
</script>
</head>
<body>
<h1>从随机生成的数字中抽取3的倍数并显示示例</h1>
<table id="table">
</table>
</body>
```

该示例使用了一个脚本文件script.js，将后台线程中的代码存放在该文件中，代码清单10-5为该脚本文件中的代码。

代码清单10-5　与线程进行数据交互示例的后台脚本代码

```
onmessage = function(event) {
    var data = event.data;
```

```
    var returnStr; //将3的倍数拼接成字符串并返回
    var intArray=data.split(";"); //返回字符串中数字分隔符为;
    returnStr="";
    for(var i=0;i<intArray.length;i++)
    {
        if(parseInt(intArray[i])%3==0)  //能否被3整除
        {
            if(returnStr!="")
                returnStr+=";";
            returnStr+=intArray[i];
        }
    }
    postMessage(returnStr); //返回3的倍数拼接成的字符串
}
```

这段代码的运行结果如图10-3所示。

图10-3 与线程进行数据交互示例

10.3 线程嵌套

线程中可以嵌套子线程，这样的话我们可以把一个较大的后台线程切分成几个子线程，在每个子线程中各自完成相对独立的一部分工作。

10.3.1 单层嵌套

接下来，我们来看一个单层嵌套的示例，该示例中修改了前面所述与线程进行数据交互的示例，并把生成随机数组的工作也放到后台线程中，然后使用一个子线程在随机数组中挑选可以被3整除的数字。同时本示例中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。

代码清单10-6中为这个单层嵌套示例的HTML 5页面代码部分。

代码清单10-6 单层嵌套示例的HTML 5页面代码

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
var worker = new Worker("script.js");
```

```
worker.postMessage("");
// 从线程中取得计算结果
worker.onmessage = function(event) {
    if(event.data!="")
    {
        var j;    //行号
        var k;  //列号
        var tr;
        var 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)    //该行不存在
            {
                //添加行
                tr=document.createElement("tr");
                tr.id="tr"+j;
                table.appendChild(tr);
            }
            else  //该行已存在
            {
                //获取该行
                tr=document.getElementById("tr"+j);
            }
            //添加列
            td=document.createElement("td");
            tr.appendChild(td);
            //设置该列内容
            td.innerHTML=intArray[j*10+k];
            //设置该列背景色
            td.style.backgroundColor="blue";
            //设置该列字体颜色
            td.style.color="white";
            //设置列宽
            td.width="30";
        }
    }
};
</script>
</head>
<body>
<h1>从随机生成的数字中抽取3的倍数并显示示例</h1>
<table id="table">
</table>
</body>
```

接下来，我们在代码清单10-7中看一下该示例的后台线程的主线程代码部分，该线程中随机生成100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。

代码清单10-7 单层嵌套示例的主线程代码

```
onmessage=function(event){
    var intArray=new Array(100);     //随机数组
    //生成100个随机数
    for(var i=0;i<100;i++)
        intArray[i]=parseInt(Math.random()*100);
    var worker;
    //创建子线程
    worker=new Worker("worker2.js");
    //把随机数组提交给子线程进行挑选工作
    worker.postMessage(JSON.stringify(intArray));
    worker.onmessage = function(event) {
        //把挑选结果返回主页面
        postMessage(event.data);
    }
}
```

注意：本示例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。

接下来，我们在代码清单10-8中看一下该示例中子线程部分的代码，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。

代码清单10-8 单层嵌套示例的子线程代码

```
onmessage = function(event) {
    //还原整数数组
    var intArray= JSON.parse(event.data);
    var returnStr;
    returnStr="";
    for(var i=0;i<intArray.length;i++)
    {
        //能否被3整除
        if(parseInt(intArray[i])%3==0)
        {
            if(returnStr!="")
                returnStr+=";";
            //将能被3整除的数字拼接成字符串
            returnStr+=intArray[i];
        }
    }
    //返回拼接字符串
    postMessage(returnStr);
    //关闭子线程
    close();
}
```

注意：在子线程中向发送源发送回消息后，最好使用close语句关闭子线程，如果该子线程不再使用的话。

10.3.2 在多个子线程中进行数据的交互

接下来，我们来看一下当主线程使用到多个子线程时，多个子线程之间如何实现数据的

交互。

要实现子线程与子线程之间的数据交互，大致需要如下几个步骤：

- 先创建发送数据的子线程。
- 执行子线程中的任务，然后把要传递的数据发送给主线程。
- 在主线程接受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。
- 执行接收数据子线程中的代码。

接下来我们看一个在多个子线程中进行数据交互的示例，该示例修改了代码清单10-7中的示例，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，该示例的HTML页面代码仍使用代码清单10-6中的HTML页面代码。我们看一下该示例主线程中的代码，如代码清单10-9所示。

代码清单10-9　在多个子线程中进行数据交互示例的主线程代码

```
onmessage=function(event){
    var worker;
    //创建发送数据的子线程
    worker=new Worker("worker1.js");
    worker.postMessage("");
    worker.onmessage = function(event) {
        //接收子线程中数据，本示例中为创建好的随机数组
        var data=event.data;
        //创建接收数据子线程
        worker=new Worker("worker2.js");
        //把从发送数据的子线程中发回的消息传递给接收数据的子线程
        worker.postMessage(data);
        worker.onmessage = function(event) {
            //获取接收数据的子线程中传回的数据，本示例中为挑选结果
            var data=event.data;
            //把挑选结果发送回主页面
            postMessage(data);
    }
  }
}
```

接下来我们在代码清单10-10中看一下该示例中发送数据的子线程中的代码，该子线程中创建了一个100个整数构成的随机数组。

代码清单10-10　在多个子线程中进行数据交互示例的发送数据子线程

```
onmessage = function(event) {
    var intArray=new Array(100);     //随机数组
    for(var i=0;i<100;i++)
        intArray[i]=parseInt(Math.random()*100);
    //发送回随机数组
    postMessage(JSON.stringify(intArray));
    //关闭子线程
    close();
}
```

该示例中接收数据子线程中的代码与代码清单10-7中所示的代码相同。

10.4 线程中可用的变量、函数与类

最后，我们再来总体看一下在线程用的JavaScript脚本文件中所有可用的变量、函数与类，如下所示。

❑ self

self关键词用来表示本线程范围内的作用域。

❑ postMessage(message)

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，如下所示。

```
importScripts('script1.js','scirpts\script2.js',' scirpts\script3.js');
```

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。

❑ navigator对象

与window.navigator对象类似，具有appName、platform、userAgent、appVersion这些属性。

❑ sessionStorage/localStorage

可以在线程中使用Web Storage。

❑ XMLHttpRequest

可以在线程中处理Ajax请求。

❑ Web Workers

可以在线程中嵌套线程。

❑ setTimeout()/setInterval()

可以在线程中实现定时处理。

❑ close

可以结束本线程。

❑ 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WebSockets API来向服务器发送和接收信息。

第11章
获取地理位置信息

关于HTML 5部分，最后要介绍的是如何使用Geolocation API来获得用户的地理位置信息。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。

学习内容：

- 掌握Geolocation API的基本知识，掌握geolocation属性的三个方法。
- 掌握position对象存在哪些属性，能够使用getCurrentPosition方法来取得存放在position对象内的当前用户的地理位置信息。
- 掌握在页面上使用Google地图的基本方法，能够在页面上正确显示google地图，并且把用户当前所在的地理位置在地图上正确标注出来。

11.1 Geolocation API的基本知识

在HTML 5中，为window.navigator对象新增了一个geolocation属性，可以使用Geolocation API来对该属性进行访问。window.navigator对象的geolocation属性存在以下三个方法。

11.1.1 取得当前地理位置

可以使用getCurrentPosition方法来取得用户当前的地理位置信息，该方法的定义如下所示。

```
void getCurrentPosition(onSuccess, onError, options);
```

其中第一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，第二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，第三个参数为一些可选属性的列表。其中，第二、三个参数为可选属性。

getCurrentPosition方法中的第一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，我们将在第11.2节中对这个对象进行具体介绍。

getCurrentPosition方法中的第二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，你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，如图11-1所示。

如果你在该画面中拒绝共享的话，也会引起错误的发生。

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属性为以下三个值其中之一：

用户拒绝了位置服务（属性值为1）；

获取不到位置信息（属性值为2）；

获取信息超时错误（属性值为3）。

❑ 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。有些浏览器中不支持message属性，譬如Firefox 3.6以上。

图11-1 浏览器询问用户是否共享位置信息（在Opera 10浏览器中）

在getCurrentPosition方法中使用第二个参数来捕获错误信息的具体使用方法如下所示。

```
navigator.geolocation.getCurrentPosition(
    function(position) {
       var coords = 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] + ":, 不能确定你的当前地理位置");
    }
);
```

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没用，因为使用在设备上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（单位为毫秒）。例如maximumAge：120000（1分钟是60000）。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处理
  },
  function(error) {
    //获取地理位置信息失败时所做处理
  },
  // 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
  }
);
```

11.1.2 持续监视当前地理位置的信息

使用watchPosition方法来持续获取用户的当前地理位置信息，它会定期地自动获取，该方法定义如下所示。

```
int watchCurrentPosition(onSuccess, onError, options);
```

该方法三个参数的说明与使用方法与getCurrentPosition方法的参数说明与使用方法相同。该方法返回一个数字，这个数字的使用方法与JavaScript脚本中setInterval方法的返回参数的使用方法类似，可以被clearWatch方法使用，停止对当前地理位置信息的监视。

11.1.3 停止获取当前用户的地理位置信息

使用该方法可以停止对当前用户的地理位置信息的监视。该方法定义如下所示。

```
void clearWatch(watchId);
```

该方法的参数为调用watchCurrentPosition方法监视地理位置信息时的返回参数。

11.2 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对象具有如下这些属性。

- latitude

当前地理位置的纬度。

- longitude

当前地理位置的精度。

- 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。

接下来我们在代码清单11-1中看一个示例，该示例使用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。

代码清单11-1 获取当前位置的地理信息示例

```
<!DOCTYPE html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<title>获取当前位置的地理信息示例</title>
<script type="text/javascript"src=http://maps.google.com/maps/api/js?sensor=false />
<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HTML 5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>
```

这段代码在Opera 10浏览器中的运行结果如图11-2所示。另外，这个运行结果在不同设备的浏览器上也各不相同，具体运行结果取决于运行浏览器的设备。

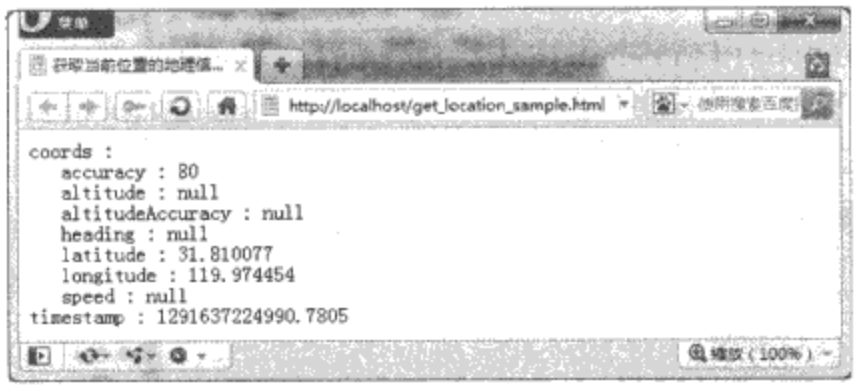

图11-2 Opera 10浏览器中的获取地理位置信息的示例

11.3 在页面上使用google地图

接下来，我们通过一个示例来看一下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。

首先，我们对该示例中的有关知识要点做一个必要的说明。

要在页面中使用google地图，需要使用到Google Map API。使用时按如下步骤进行。

1）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/>
```

2）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,//将地图中心点设定为指定的坐标点
    mapTypeId: google.maps.MapTypeId.ROADMAP //指定地图类型
};
```

在本例中，将用户当前位置的纬度、精度设定为页面打开时google地图的中心点。

3）创建地图，并在页面中显示，如下所示。

```
var map1 = new google.maps.Map(document.getElementById("map"),myOptions);
```

本例中将地图显示在id为“map”的div元素中。

4）在地图上创建标记，方法如下所示。

```
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
});
```

5）设置标注窗口并指定标注窗口中注释文字，如下所示。

```
var infowindow = new google.maps.InfoWindow({
    content: "当前位置!"     //指定标注窗口中注释文字
});
```

6）打开标注窗口，如下所示。

```
infowindow.open(map1, marker);
```

该示例的全部代码如代码清单11-2所示。

代码清单11-2　将用户当前位置在google地图中标注出来

```
<!DOCTYPE html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<title>在页面上使用google地图示例</title>
<script type="text/javascript"src=http://maps.google.com/maps/api/js?sensor=false />
<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
          };
          //创建地图并在"map"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>
```

该程序的运行结果如图11-3所示。

图11-3 在页面上使用google地图的示例

第12章

CSS 3概述

12.1 概要介绍

12.2 使用CSS 3能做什么

从2010年开始，HTML 5与CSS 3就一直是互联网技术中最受关注的两个话题。2010年MIX10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三个阶段：第一阶段是Web 1.0的以内容为主的网络，前端主流技术是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HTML 5+CSS 3的时代，这两者相辅相成，使互联网又进入了一个崭新的时代。

本章将对CSS 3进行一个全面的、概要的介绍，使大家对CSS 3有一个初步的、总体上的认识。

学习内容：

- 掌握CSS 3的基础知识，知道什么是CSS 3，了解CSS 3的发展历史。
- 掌握CSS 3的模块化结构，了解CSS 3中包含了哪些结构。
- 了解CSS 3与CSS 2有什么主要区别，了解CSS 3将对下一代Web平台上的界面设计做出哪些重大贡献。

12.1 概要介绍

12.1.1 CSS 3是什么

首先，我们对CSS 3做一个概要的介绍。什么是CSS 3？CSS 3是CSS技术的一个升级版本，是由Adobe Systems、Apple、Google、HP、IBM、Microsoft、Mozilla、Opera、Sun Microsystems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。虽然目前很多细节还在讨论之中，但它还是不断地朝前发展着。2010年在HTML 5成为IT界人士关注的焦点的同时，它也开始慢慢地普及开来。

12.1.2 CSS 3的历史

接下来，我们从总体上看一下CSS的发展历史。

- CSS 1。

1996年12月，CSS 1(Cascading Style Sheets,level 1)正式推出。在这个版本中，已经包含了font的相关属性、颜色与背景的相关属性、文字的相关属性、box的相关属性等。

- CSS 2。

1998年5月，CSS 2(Cascading Style Sheets,level 2)正式推出。在这个版本中开始使用样式表结构。

- CSS 2.1。

2004年2月，CSS 2.1(Cascading Style Sheets,level 2 revision 1)正式推出。它在CSS 2的基础上略微做了改动，删除了许多诸如text-shadow等不被浏览器所支持的属性。

现在所使用的CSS基本上是在1998年推出的CSS 2的基础上发展而来的。10年前在Internet刚开始普及的时候，就能够使用样式表来对网页进行视觉效果的统一编辑，确实是一件可喜的事情。但是在这10年间CSS可以说是基本上没有什么很大的变化，一直到2010年终于推出了一个全新的版本——CSS 3。

12.2 使用CSS 3能做什么

12.2.1 模块与模块化结构

在CSS 3中，并没有采用总体结构，而是采用了分工协作的模块化结构，这些模块如表12-1所示。

表12-1 CSS 3中的模块

模块名称	功能描述
basic box model	定义各种与盒相关的样式
Line	定义各种与直线相关的样式
Lists	定义各种与列表相关的样式
Hyperlink Presentation	定义各种与超链接相关的样式。譬如锚的显示方式、激活时的视觉效果等
Presentation Levels	定义页面中元素的不同的样式级别
Speech	定义各种与语音相关的样式。譬如音量、音速、说话间歇时间等属性
Background and border	定义各种与背景和边框相关的样式
Text	定义各种与文字相关的样式
Color	定义各种与颜色相关的样式
Font	定义各种与字体相关的样式
Paged Media	定义各种页眉、页脚、页数等页面元数据的样式
Cascading and inheritance	定义怎样对属性进行赋值
Value and Units	将页面上各种各样的值与单位进行统一定义，以供其他模块使用
Image Values	定义对image元素的赋值方式
2D Transforms	在页面中实现2维空间上的变形效果
3D Transforms	在页面中实现3维空间上的变形效果
Transitions	在页面中实现平滑过渡的视觉效果
Animations	在页面中实现动画
CSSOM View	查看管理页面或页面的视觉效果，处理元素的位置信息
Syntax	定义CSS样式表的基本结构、样式表中的一些语法细节、浏览器对于样式表的分析规则
Generated and Replaced Content	定义怎样在元素中插入内容
Marquee	定义当一些元素的内容太大，超出了指定的元素尺寸时，是否以及怎样显示溢出部分
Ruby	定义页面中ruby元素（用于显示拼音文字）的样式
Writing Modes	定义页面中文本数据的布局方式
Basic User Interface	定义在屏幕、纸张上进行输出时页面的渲染方式
Namespaces	定义使用命名空间时的语法
Media Queries	根据媒体类型来实现不同的样式
‘Reader’ Media Type	定义用于屏幕阅读器之类的阅读程序时的样式
Multi-column Layout	在页面中使用多栏布局方式
Template Layout	在页面中使用特殊布局方式
Flexible Box Layout	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
Grid Position	在页面中使用网格布局方式
Generated Content for Paged Media	在页面中使用印刷时使用的布局方式

那么，为什么需要分成这么多模块来进行管理呢？

这是为了避免产生浏览器对于某个模块支持不完全的情况。如果只有一个总体结构，这个总体结构会过于庞大，在对其支持的时候很容易造成支持不完全的情况。如果把总体结构分成几个模块，各浏览器可以选择对于哪个模块进行支持、对哪个模块不进行支持，支持的时候也可以集中把某一个模块全部支持完了再支持另一个模块，以减少支持不完全的可能性。

例如，台式计算机、笔记本和手机上用的浏览器应该针对不同的模块进行支持。如果采用模块分工协作的话，不仅是台式计算机，各种设备上所用的浏览器都可以选用不同模块进行支持。

12.2.2 一个简单的CSS 3示例

现在，我们已经对CSS 3的模块和模块化结构有了一个初步的认识，那么，究竟我们能够用CSS 3来做些什么呢？

这里，我们通过一个示例，来将CSS 2与CSS 3做一个对比，借此对CSS 3有一个初步的印象。

在这个示例中，我们给页面上的某个div区域添加一个彩色图像边框，这样可以使这个区域看上去漂亮很多，生动很多。

在CSS 2中，当然可以实现这个效果，如代码清单12-1所示。

代码清单12-1 使用CSS 2给div区域添加图像边框

```
<!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">
#image-boarder{
margin:3px;
width:450px;
height:104px;
padding-left:14px;
padding-top:20px;
background:url(test.png);
background-repeat:no-repeat;
}
</style>
</head>
<body>
<div id="image-boarder">
• 示例文字1<br/>
• 示例文字2<br/>
• 示例文字3<br/>
• 示例文字4<br/>
</div>
</body>
</html>
```

这段代码在Firefox浏览器中的运行结果如图12-1所示。

接下来，我们看一下在CSS 3中如何实现这个功能。

在CSS 3中，添加了很多新的样式，譬如可以创建圆角边框，可以在边框中使用图像，可以修改背景图像的大小，可以对背景指定多个图像文件，可以修改颜色的透明度，可以给文字添加阴影，可以在CSS中重新指定表单的尺寸等。

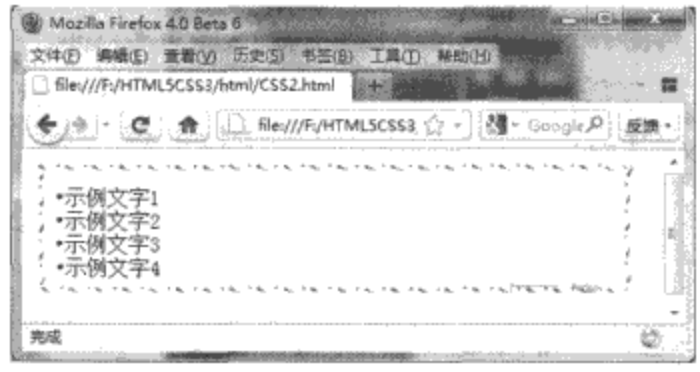

图12-1 使用CSS 2样式添加图像边框

在代码清单12-2中，我们使用CSS 3来实现与代码清单12-1相同的功能。具体操作的时候，只要给页面中的div元素增加一个border-image属性，然后在该属性中指定图像文件、边框宽度与图像拉伸方式就可以了。

代码清单12-2 使用CSS 3给div区域添加图像边框

```
<!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">
#image-boarder{
width:450px;
padding-top:20px;
padding-left:14px;
-moz-border-image:url(test.png) 3  stretch stretch;     //指定边框图像
}
</style>
</head>
<body>
<div id="image-boarder">
• 示例文字1<br/>
• 示例文字2<br/>
• 示例文字3<br/>
• 示例文字4<br/>
</div>
</body>
</html>
```

这段代码的运行后结果与图12-1所示结果相同。

虽然目前看来两种方法都达到了同样的效果，只是实现方法不同而已。但是如果再在div中增加一行文字，我们看一下使用CSS 2中的样式表后会是什么情况，如图12-2所示。

同样的，来看一下使用CSS 3中的样式表后会是什么情况，如图12-3所示。

为什么在CSS 3中文字没有超出边框图像之外？这是因为在CSS 3样式表中，在指定边框图像的同时，也指定了图像允许拉伸来自动适应div区域的高度，而不是采取CSS 2中将div区域高度设为边框图像高度的方式。那么，也许有人会问，如果在CSS 2的div元素的样式代码中不指定div区域的高度是否可以呢？这样的话就会出现如图12-4所示的情况。

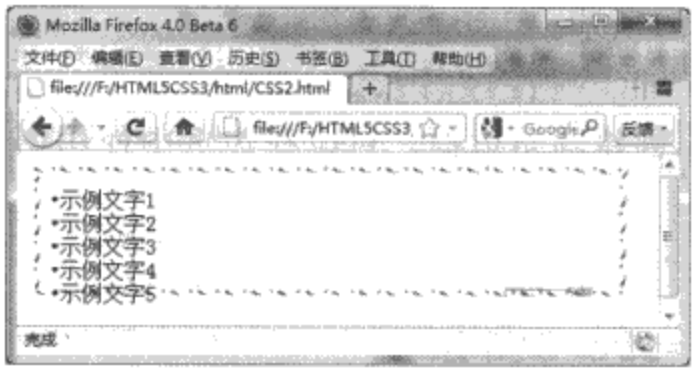

图12-2 使用CSS 2样式表，当文字超过图像高度时的页面外观

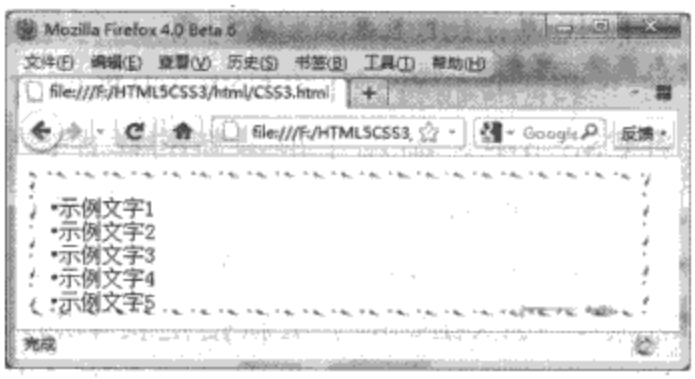

图12-3 使用CSS 3样式表，当文字超过图像高度时的页面外观

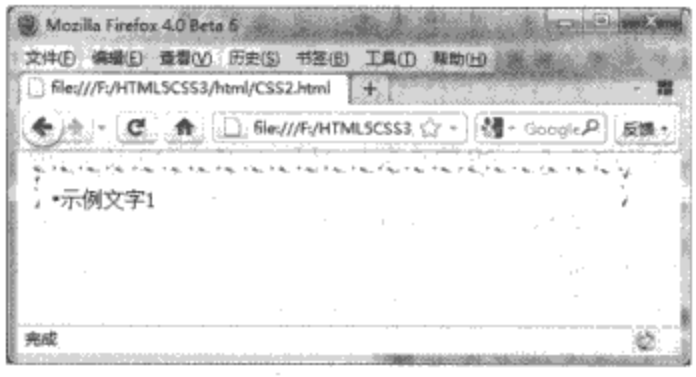

图12-4 在CSS 2的样式代码中不指定div区域高度的效果

从图中可以看出，当只有一行文字的时候，该边框图像又不能完全显示了。因此，当div区域中的文字高度处于不断变化的状态时，使用CSS 2样式表添加边框图像的操作相对来说就比较麻烦。在CSS 3中考虑到了这种情况，添加了允许边框图像自动拉伸的属性，从而解决了这个问题。

关于如何使用border-image这个属性，我们将在后面进行详细介绍。在这里，我们通过这个示例，向大家表明了目前在CSS 2中一些比较难以处理的情况，在CSS 3中通过使用新增属性很容易就能够解决。

这对界面设计来说，无疑是一件非常可喜的事情。在界面设计中，最重要的就是创造性，如果能够使用CSS 3中新增的各种各样的属性，就能够在页面中增加许多CSS 2中没有办法解决的样式，摆脱现在界面设计中存在的许多束缚，从而使整个网站或Web应用程序的界面设计进入一个新的台阶。

第13章

选 择 器

本章针对CSS 3中使用的各种选择器进行详细介绍。通过选择器的使用，你不再需要在编辑样式时使用多余的以及没有任何语义的class属性，而可以直接将样式与元素绑定起来，从而节省在网站或Web应用程序完成之后又要修改样式所需花费的大量时间。

学习内容：

- ❑ 掌握CSS 3中使用的选择器的基本概念。知道什么是选择器以及为什么需要使用选择器，使用选择器有什么好处。
- ❑ 掌握CSS 3中各种属性选择器的概念以及使用方法，其中包括：

1）[att=val] 选择器
2）[att*=val] 选择器
3）[att^=val] 选择器
4）[att$=val] 选择器

- ❑ 掌握CSS 3中各种结构性伪类选择器的概念以及使用方法，其中包括：

1）root选择器
2）not选择器
3）empty选择器
4）target选择器
5）first-child选择器
6）last-child选择器
7）nth-child选择器
8）nth-last-child选择器
9）nth-of-type选择器
10）nth-last-of-type选择器
11）only-child选择器

- ❑ 掌握CSS 3中各种UI元素状态伪类选择器的概念以及使用方法，其中包括：

1）E:hover选择器
2）E:active选择器
3）E:focus选择器
4）E:enabled选择器
5）E:disabled选择器
6）E:read-only选择器
7）E:read-write选择器
8）E:checked选择器
9）E:default选择器
10）E: indeterminate选择器
11）E:selection选择器

- ❑ 掌握CSS 3中的通用兄弟元素选择器的概念以及使用方法。

13.1 选择器概述

选择器是CSS 3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属性的话，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，这样其实是非常混乱的，修改样式的时候也很不方便。

所以，在CSS 3中，提倡使用选择器来将样式与元素直接绑定起来，这样的话，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，我们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。

具体来说，使用选择器进行样式指定的时候，采用类似E[foo$="val"]这种正则表达式的形式。在样式中，声明该样式应用于什么元素，该元素的某个属性的属性值必须是什么。例如，我们可以指定将页面中id为“div_Big”的div元素的背景色设定为红色，代码如下所示。

```
div[id="div_Big"] {background: red;}
```

这样，符合这个条件（id为“div_Big”）的div元素的背景色会被设为红色，不符合这个条件的div元素则不使用这个样式。

另外，我们还可以在指定样式的时候使用“^”通配符（开头字符匹配）、“？”通配符（结尾字符匹配）与“*”通配符（包含字符匹配）。譬如指定id末尾字母为“t”的div元素的背景色为蓝色，代码如下所示。

```
div[id$="t"] {background: red;}
```

通过通配符的使用，更加提高了样式表的书写效率。

13.2 属性选择器

13.2.1 属性选择器是什么

在HTML中，通过各种各样的属性，我们可以给元素增加很多附加信息。例如，通过width属性，我们可以指定div元素的宽度；通过id属性，我们可以将不同的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。

接下来，我们在代码清单13-1中看一个HTML页面，该页面中具有一些div，每个div之间用id属性进行区分。

代码清单13-1 一个具有很多div元素的页面

```
<!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" />
</head>
<div id="section1">示例文本1</div>
<div id="subsection1-1">示例文本1-1</div>
<div id="subsection1-2">示例文本1-2</div>
<div id="section2">示例文本2</div>
<div id="subsection2-1">示例文本2-1</div>
<div id="subsection2-2">示例文本2-2</div>
```

接下来，我们回顾一下CSS 2中对div元素使用样式的方法，如果要将id为“section1”的这个div元素的背景色设定为黄色，我们首先追加样式，如下所示。

```
<style type="text/css">
.divYellow{background:yellow}
</style>
```

然后指定id为“section1”的这个div元素的class属性，如下所示。

```
<div id="section1" class="divYellow">示例文本1</div>
```

接下来，我们看一下在CSS 2 中如何使用属性选择器来实现同样的处理。

使用属性选择器时，需要声明属性与属性值，声明方法如下所示。

```
[att=val]
```

其中att代表属性，val代表属性值。例如，要将id为“section1”的这个div元素的背景色设定为黄色，我们只要在代码清单13-1中加入如下所示的样式代码就可以了。

```
<style type="text/css">
[id=section1]{
    background-color: yellow;
}
</style>
```

最后，我们在代码清单13-2中完整地看一下使用CSS 2的属性选择器的示例代码，在本节接下来的部分都只会针对这个示例中的样式代码进行修改，其他部分不会再被修改。

代码清单13-2 使用CSS 2的属性选择器的示例

```
<!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">
[id=section1]{
    background-color: yellow;
}
</style>
</head>
<div id="section1" class="divYellow">示例文本1</div>
```

```
<div id="subsection1-1">示例文本1-1</div>
<div id="subsection1-2">示例文本1-2</div>
<div id="section2">示例文本2</div>
<div id="subsection2-1">示例文本2-1</div>
<div id="subsection2-2">示例文本2-2</div>
```

追加了这个属性选择器后的运行结果如图13-1所示。

图13-1 使用CSS 2的属性选择器的示例

13.2.2 CSS 3中的属性选择器

在CSS 3中，增加了如下所示的三个属性选择器，使得属性选择器有了通配符的概念。

- [att*=val]
- [att^=val]
- [att$=val]

1. [att*=val] 属性选择器

[att*=val]属性选择器的含义是：如果元素用att表示的属性之属性值中包含用val指定的字符的话，则该元素使用这个样式。针对上面所述“[id=section1]”这个属性选择器，可以将其修改成“[id*=section1]”，其中“id”相当于[att*=val]属性选择器中的“att”，“section1”相当于[att*=val]属性选择器中的“val”。

在代码清单13-1所述示例的样式代码中，如果使用如下代码中所示的[att*=val]属性选择器，则页面中id为“section1”、“subsection1-1”、“subsection1-2”的div元素的背景色都变为黄色，因为这些元素的id属性中都包含“section1”字符。

```
[id*=section1]{
    background-color: yellow;
}
```

代码清单13-1所述示例的样式代码中使用这个[att*=val]属性选择器的运行结果如图13-2所示。

2. [att^=val]属性选择器

[att^=val]属性选择器的含义是：如果元素用att表示的属性之属性值的开头字符为用val指定的字符的话，则该元素使用这个样式。针对上面所述“[id=section1]”这个属性选择器，可以将其修改成“[id^=section1]”。

在代码清单13-1所述示例的样式代码中，如果将使用的[att=val]属性选择器改为使用如下所示的[att^=val]属性选择器，并且将val指定为“section”，则页面中id为“section1”、“section2”的div元素的背景色都变为黄色，因为这些元素的id属性的开头字符都为“section”字符。

```
[id^=section]{
    background-color: yellow;
}
```

代码清单13-1所述示例的样式代码中使用这个[att^=val]属性选择器的运行结果如图13-3所示。

图13-2 使用[att*=val]属性选择器的示例

图13-3 使用[att^=val]属性选择器的示例

3. [att$=val] 属性选择器

[att$=val]属性选择器的含义是：如果元素用att表示的属性之属性值的结尾字符为用val指定的字符的话，则该元素使用这个样式。针对上面所述“[id=section1]”这个属性选择器，可以将其修改成“[id$=section1]”。

在代码清单13-1所述示例的样式代码中，如果采用如下所示的[att$=val]属性选择器，并且将val指定为“-1”，则页面中id为“subsection1-1”、“subsection2-1”的div元素的背景色都变为黄色，因为这些元素的id属性的结尾字符都为“-1”字符。另外请注意该属性选择器中必须在指定匹配字符前加上“\”这个转义字符。

```
[id$=\-1]{
    background-color: yellow;
}
```

代码清单13-1所述示例的样式代码中使用这个[att$=val]属性选择器的运行结果如图13-4所示。

图13-4 使用[att$=val]属性选择器的示例

13.2.3 灵活运用属性选择器

如果能够灵活运用属性选择器的话，目前为止需要依靠id或class名才能实现的样式完全可以使用属性选择器来实现。

例如，利用[att$=val]属性选择器，可以根据超链接中不同的文件扩展符使用不同的样式。在代码清单13-3所示的示例中，在超链接地址的末尾为“/”、“htm”、“html”的时候显示

“Web网页”文字，在超链接地址的末尾为“jpg”、“jpeg”的时候显示“JPEG图像文件”文字。

代码清单13-3 灵活运用属性选择器的示例

```
<!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">
a[href$=\/]:after, a[href$=htm]:after, a[href$=html]:after{
    content:"Web网页";
    color: red;
}
a[href$=jpg]:after{
    content:"JPEG图像文件";
    color: green;
}
</style>
</head>
<ul>
<li><a href="http://Lulingniu/">HTML 5+CSS 3权威指南</a></li>
<li><a href="http://Lulingniu/CSS 3.htm">CSS 3的新特性</a></li>
<li><a href="photo.jpg">图像素材</a></li>
</ul>
```

这段代码的运行结果如图13-5所示。

图13-5 灵活运用属性选择器的示例

另外，如果使用Internet Explorer浏览器来运行本示例的时候，因为在Internet Explorer 8之前尚未支持after伪元素选择器，所以该示例只在Internet Explorer 8之后的浏览器中能够正确显示，在13.3.1节中将针对after伪元素选择器做详细说明。

13.3 结构性伪类选择器

本节将介绍CSS 3中的结构性伪类选择器，在介绍结构性伪类选择器之前，首先有必要针对CSS中的伪类选择器及伪元素进行介绍。

13.3.1 CSS中的伪类选择器及伪元素

1. 什么是伪类选择器

我们知道，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，譬如针对一

个p元素，我们可以做如下所示的定义。

```
p.right{text-align:right}
p.center{text-align:right}
```

然后在页面上对p元素使用class属性，来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，类选择器可以随便起名，譬如上面的“p.right”与“p.center”，你也可以命名为“p.class1”与“p.class2”，然后在页面上使用“class=‘class1’”与“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中我们最常用的伪类选择器是使用在a(锚)元素上的几种选择器，它们的使用方法如下所示。

```
a:link {color:#FF0000;text-decoration:none}
a:visited {color:#00FF00;text-decoration:none}
a:hover {color:#FF00FF;text-decoration:underline}
a:active {color:#0000FF;text-decoration:underline}
```

2. 什么是伪元素选择器

所谓伪元素选择器，是指并不是针对真正的元素使用的选择器，而是针对CSS中已经定义好的伪元素使用的选择器，它的使用方法如下所示。

```
选择器：伪元素{属性：值}
```

伪元素选择器也可以与类配合使用，使用方法如下所示。

```
选择器 类名：伪元素{属性：值}
```

在CSS中，主要有如下四个伪元素选择器。

- first-line伪元素选择器。

first-line伪元素选择器用于为某个元素中的第一行文字使用样式。

代码清单13-4是它的一个使用示例，在该示例中，有一个p元素，在该元素内存在两行文字，使用first-line伪元素选择器将第一行文字设为蓝色。

代码清单13-4 first-line伪元素选择器使用示例

```
<!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>first-line伪元素使用示例</title>
<style type="text/css">
p:first-line{color:#0000FF}
</style>
</head>
<body>
<p>段落中的第一行。<br>段落中的第二行</p>
```

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

这段代码的运行结果如图13-6所示。

图13-6 first-line伪元素选择器使用示例

❑ first-letter伪元素选择器。

first-letter伪元素选择器用于为某个元素中的文字的首字母（欧美文字）或第一个字（中文或日文等汉字）使用样式。

代码清单13-5是first-letter伪元素选择器的一个使用示例，在该示例中，有两段文字——一段是英文，另一段是中文，使用first-letter伪元素选择器来设置这两段文字的开头字母或文字的颜色为蓝色。

代码清单13-5 first-letter伪元素选择器使用示例

```
<!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>first-letter伪元素使用示例</title>
<style type="text/css">
p:first-letter{color:#0000FF}
</style>
</head>
<body>
<p>This is an english text。</p>
<p>这是一段中文文字。</p>
</body>
</html>
```

这段代码的运行结果如图13-7所示。

❑ before伪元素选择器。

before伪元素选择器用于在某个元素之前插入一些内容，使用方法如下所示。

```
//可以插入一段文字
<元素>：before
{
    content：插入文字
}
//也可以插入其他内容
<元素>：before
{
    content：url(test.wav)
}
```

代码清单13-6是before伪元素选择器的一个使用示例，在该示例中有一个ul列表，该列表中有几个li列表项目，使用before伪元素选择器在每个列表项目的文字的开头插入“•”字符。

代码清单13-6 before伪元素选择器的使用示例

```
<!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>before伪元素选择器使用示例</title>
<style type="text/css">
li:before{content: •}
</style>
</head>
<body>
<ul>
<li>列表项目1</li>
<li>列表项目2</li>
<li>列表项目3</li>
<li>列表项目4</li>
<li>列表项目5</li>
</li>
</ul>
</body>
</html>
```

这段代码的运行结果如图13-8所示。

图13-7 first-letter伪元素选择器使用示例

图13-8 before伪元素选择器使用示例

❑ after伪元素选择器。

after伪元素选择器用于在某个元素之后插入一些内容，使用方法如下所示。

```
<元素>：after
{
    content：插入文字
}
//也可以插入其他内容
<元素>：after
{
    content：url(test.wav)
}
```

代码清单13-7是after伪元素选择器的一个使用示例，在该示例中有一个ul列表，这个ul列表的内容为某个网站上播放电影的节目清单。该列表中有几个列表项目，每个列表项目中存放了对于某部电影的超链接，使用after伪元素选择器在每个超链接的后面加入“(仅用于测试，请勿用于商业用途。)”的文字，并且将文字颜色设为红色。

代码清单13-7 after伪元素选择器的使用示例

```
<!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>after伪元素选择器使用示例</title>
<style type="text/css">
li:after{
    content: "(仅用于测试，请勿用于商业用途。)";
    font-size:12px;
    color:red;
}
</style>
</head>
<body>
<h1>电影清单</h1>
<ul>
<li><a href="movie1.mp4">狄仁杰之通天帝国</a></li>
<li><a href="movie2.mp4">精武风云</a></li>
<li><a href="movie3.mp4">大笑江湖</a></li>
</ul>
</body>
</html>
```

这段代码的运行结果如图13-9所示。

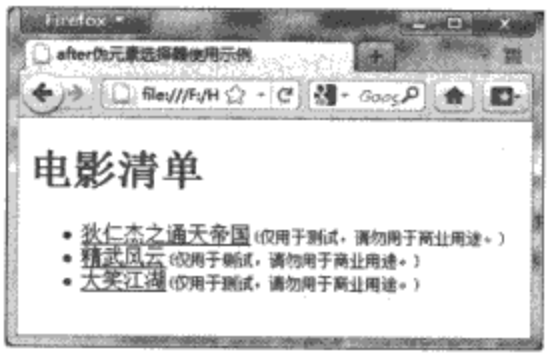

图13-9 after伪元素选择器使用示例

在介绍完了CSS中的伪类选择器与伪元素选择器之后，让我们来看一下CSS 3中的结构性伪类选择器。结构性伪类选择器的共同特征是允许开发者根据文档树中的结构来指定元素的样式。

首先，我们来看4个最基本的结构性伪类选择器——root选择器、not选择器、empty选择器与target选择器。

13.3.2 选择器root、not、empty和target

1. root选择器

root选择器将样式绑定到页面的根元素中。所谓根元素，是指位于文档树中最顶层结构的元素，在HTML页面中就是指包含着整个页面的“<html>”部分。

下面我们在代码清单13-8中看一个HTML页面，在该页面中，有一段文章，并且有一个文章的标题。

代码清单13-8 root选择器示例的HTML界面

```
<!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>root选择器</title>
</head>
<body>
<h1>选择器概述</h1>
<p>
    选择器是CSS 3中一个重要的内容。首先需要说明的是，使用选择器的目的是为了提高开发人员书写或修改样式表时的工作效率，因为在样式表中，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。
</p>
</body>
</html>
```

针对这个网页，使用如下所示的root选择器来指定整个网页的背景色为黄色，将网页中body元素的背景色设为绿色。

```
<style type="text/css">
:root{
    background-color: yellow;
}
body{
    background-color: limegreen;
}
</style>
```

使用了root选择器后的运行结果如图13-10所示。

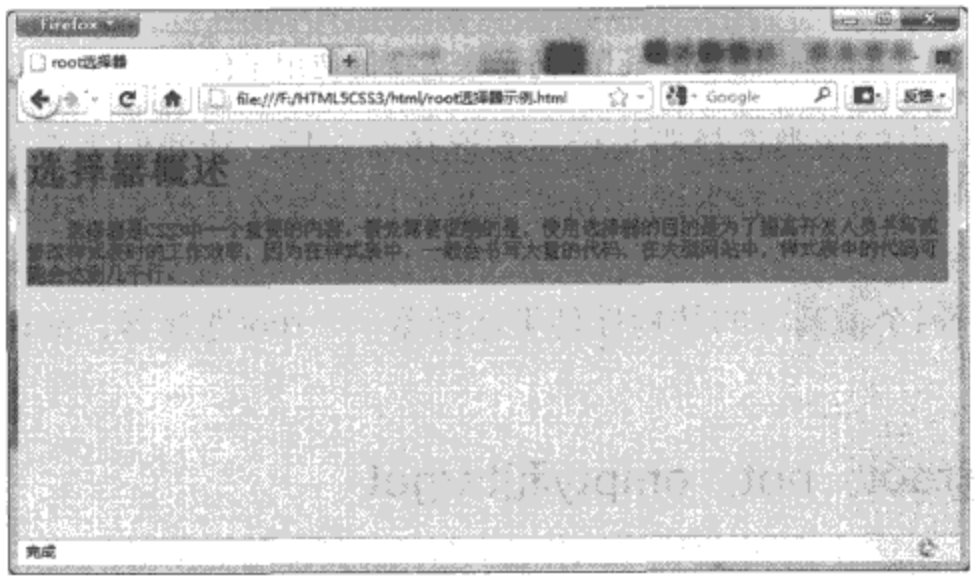

图13-10 root选择器使用示例

另外，在使用样式指定root元素与body元素的背景时，根据不同的指定条件，背景色的显示范围会有所变化，这一点请注意。譬如同样是上面这个示例，如果采取如下所示的样式，不使用root选择器来指定root元素的背景色，只指定body元素的背景色，则整个页面就全部变成绿色的了。

```
<style type="text/css">
body{
    background-color: limegreen;
}
</style>
```

删除root选择器后的页面如图13-11所示。

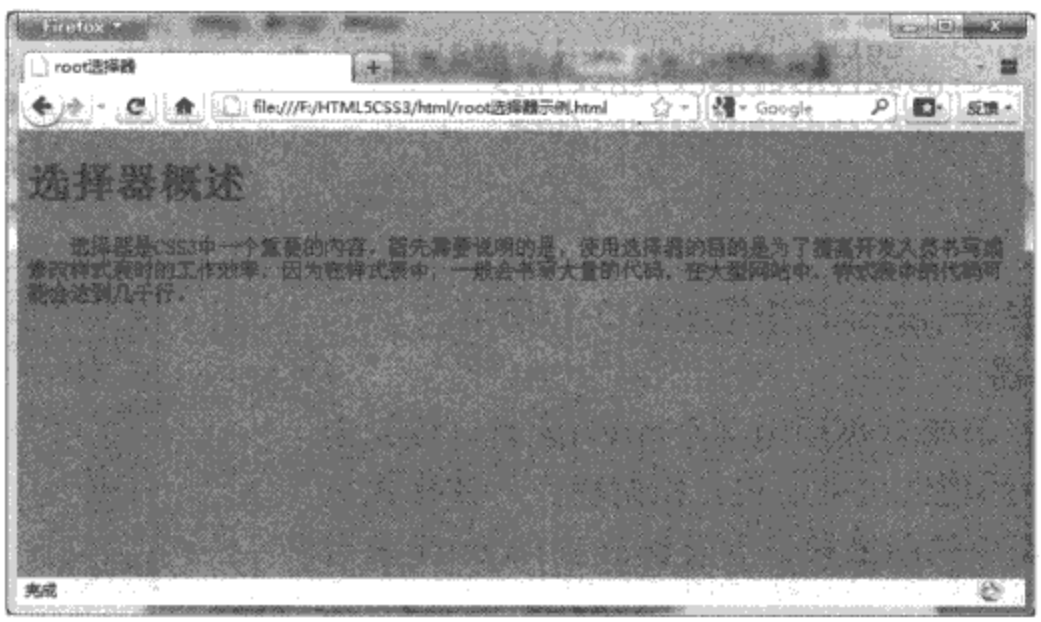

图13-11 删除root选择器后的显示效果

2. not选择器

如果想对某个结构元素使用样式，但是想排除这个结构元素下面的子结构元素，让它不使用这个样式时，可以使用not选择器。

譬如针对代码清单13-8所示的HTML页面，我们可以使用“body *”语句来指定body元素的背景色为黄色，但是在“:not(h1)”语句中使用not选择器排除h1元素，代码如下所示。

```
<style type="text/css">
body *:not(h1){
    background-color: yellow;
}
</style>
```

使用not选择器后的运行结果如图13-12所示。

图13-12 使用not选择器示例

3. empty选择器

使用empty选择器来指定当元素内容为空白时使用的样式。例如，在代码清单13-9所示的HTML页面中有一个表格，可以使用empty选择器来指定当表格中某个单元格内容为空白时，该单元格背景为黄色。

代码清单13-9 empty选择器使用示例

```
<!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>empty选择器</title>
<style type="text/css">
:empty{
    background-color: yellow;
}
</style>
</head>
<body>
<table border="1" cellpading="0" cellspacing="0">
<tr><td>A</td><td>B</td><td>C</td></tr>
<tr><td>D</td><td>E</td><td></td></tr>
</table>
</body>
</html>
```

使用empty选择器后的运行结果如图13-13所示。

图13-13 使用empty选择器的示例

4. target选择器

使用target选择器来对页面中某个target元素（该元素的id被当做页面中的超链接来使用）指定样式，该样式只在用户点击了页面中的超链接，并且跳转到target元素后起作用。

接下来我们来看一个target选择器的使用示例。页面中具有几个div元素，每个div元素都存在一个书签，当用户点击了页面中的超链接跳转到该div元素时，该div元素使用target选择器中指定的样式。在target选择器中，指定该div元素的背景色变为黄色。其中指定target选择器时的代码如下所示。

```
target{
    background-color: yellow;
}
```

该示例的详细代码如代码清单13-10所示。

代码清单13-10 target选择器使用示例

```
<!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>target选择器</title>
<style type="text/css">
:target{
    background-color: yellow;
}
</style>
</head>
<body>
<p id="menu">
<a href="#text1">示例文字1</a> |
<a href="#text2">示例文字2</a> |
<a href="#text3">示例文字3</a> |
```

```
<a href="#text4">示例文字4</a> |
<a href="#text5">示例文字5</a>
</p>
<div id="text1">
<h2>示例文字1</h2>
<p>...此处略去</p>
</div>
<div id="text2">
<h2>示例文字2</h2>
<p>...此处略去</p>
</div>
<div id="text3">
<h2>示例文字3</h2>
<p>...此处略去</p>
</div>
<div id="text4">
<h2>示例文字4</h2>
<p>...此处略去</p>
</div>
<div id="text5">
<h2>示例文字5</h2>
<p>...此处略去</p>
</body>
</html>
```

使用target选择器后的运行结果如图13-14所示。

图13-14 使用target选择器示例

13.3.3 选择器：first-child、last-child、nth-child和nth-last-child

本节将介绍first-child选择器、last-child选择器、nth-child选择器与nth-last-child选择器。利用这几个选择器，能够特殊针对一个父元素中的第一个子元素、最后一个子元素、指定序号的子元素、甚至第偶数个、第奇数个子元素进行样式的指定。

1. 单独指定第一个子元素、最后一个子元素的样式

我们来看一个示例。该示例对ul列表中的li列表项目进行样式的指定，在样式中对第一个列表项目与最后一个列表项目分别指定不同的背景色。

如果要对第一个列表项目与最后一个列表项目分别指定不同的背景色，到目前为止采取的做法都是：分别给这两个列表项目加上class属性，然后给这两个class使用不同的样式，在两个样式中分别指定不同的背景色。但是，如果使用first-child选择器与last-child选择器，这个多余的class属性就不需要了。

接下来，我们在代码清单13-11中看一下如何使用first-child选择器与last-child选择器将第一个列表项目的背景色指定为黄色，将最后一个列表项目的背景色设定为浅蓝色。

代码清单13-11 first-child选择器与last-child选择器的使用示例

```
<!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>first-child选择器与last-child选择器使用示例</title>
<style type="text/css">
li:first-child{
    background-color: yellow;
}
li:last-child{
    background-color: skyblue;
}
</style>
</head>
<body>
<h2>列表A</h2>
<ul>
<li>列表项目1</li>
<li>列表项目2</li>
<li>列表项目3</li>
<li>列表项目4</li>
<li>列表项目5</li>
</ul>
</body>
</html>
```

这段代码的运行结果如图13-15所示。

另外，如果页面中具有多个ul列表，则该first-child选择器与last-child选择器对所有ul列表都适用，如代码清单13-12所示。

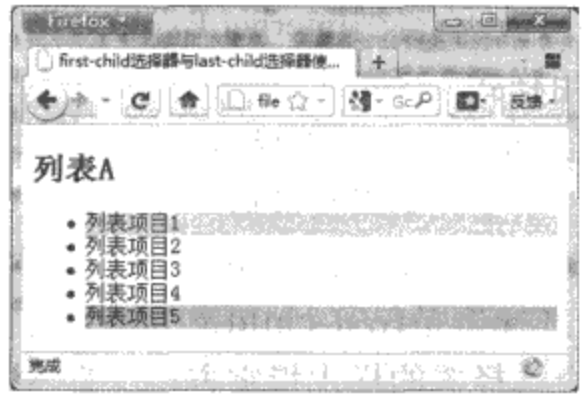

图13-15 first-child选择器与last-child选择器的使用示例

代码清单13-12 具有多个列表时first-child选择器与last-child选择器的使用示例

```
<!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>first-child选择器与last-child选择器使用示例</title>
<style type="text/css">
li:first-child{
    background-color: yellow;
}
li:last-child{
    background-color: skyblue;
}
</style>
</head>
<body>
<h2>列表A</h2>
<ul>
<li>列表项目1</li>
<li>列表项目2</li>
<li>列表项目3</li>
<li>列表项目4</li>
<li>列表项目5</li>
</ul>
<h2>列表B</h2>
<ul>
<li>列表项目1</li>
<li>列表项目2</li>
<li>列表项目3</li>
<li>列表项目4</li>
<li>列表项目5</li>
</ul>
</body>
</html>
```

这段代码的运行结果如图13-16所示。

另外，first-child选择器在CSS 2中就已经提供了，到目前为止被Firefox、Safari、Google Chrome、Opera浏览器所支持，从IE7开始被Internet Explorer浏览器所支持。

last-child选择器从CSS 3开始提供，到目前为止被Firefox、Safari、Google Chrome、

Opera浏览器所支持，到IE8为止还没有得到Internet Explorer浏览器的支持。

2. 对指定序号的子元素使用样式

如果使用nth-child选择器与nth-last-child选择器，不仅可以指定某个父元素中第一个子元素以及最后一个子元素的样式，还可以针对父元素中某个指定序号的子元素来指定样式。这两个选择器是first-child及last-child的扩展选择器。这两个选择器的样式指定方法如下所示。

```
nth-child(n){
//指定样式
}
<子元素>：nth-last-child(n){
//指定样式
}
```

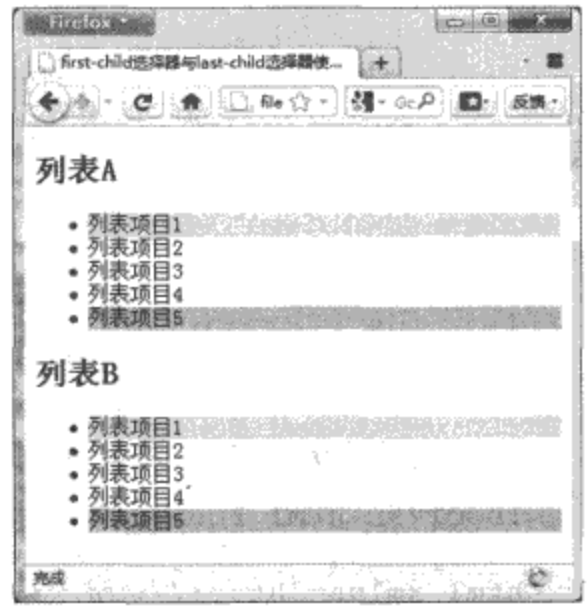

图13-16 具有多个列表时first-child选择器与last-child选择器的使用示例

将指定序号书写在“nth-child”或“nth-last-child”后面的括号中，譬如“nth-child(3)”表示第三个子元素，“nth-last-child(3)”表示倒数第三个子元素。

在代码清单13-13中，我们给出一个使用这两个选择器的示例，在该示例中，指定ul列表中第二个li列表项目的背景色为黄色，倒数第二个列表项目的背景色为浅蓝色。

代码清单13-13 nth-child选择器与nth-last-child选择器的使用示例

```
<!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>nth-child选择器与nth-last-child选择器使用示例</title>
<style type="text/css">
li:nth-child(2){
    background-color: yellow;
}
li:nth-last-child(2){
    background-color: skyblue;
}
</style>
</head>
<body>
<h2>列表A</h2>
<ul>
<li>列表项目1</li>
<li>列表项目2</li>
<li>列表项目3</li>
<li>列表项目4</li>
<li>列表项目5</li>
</ul>
</body>
</html>
```

这段代码的运行结果如图13-17所示。

这两个选择器都是从CSS 3开始提供的，到目前为止被Firefox、Safari、Google Chrome、Opera浏览器所支持，到IE8为止还没有得到Internet Explorer浏览器的支持。

图13-17 nth-child选择器与nth-last-child选择器的使用示例

3. 对所有第奇数个子元素或第偶数个子元素使用样式

除了对指定序号的子元素使用样式以外，nth-child选择器与nth-last-child选择器还可以用来对某个父元素中所有第奇数个子元素或第偶数个子元素使用样式。使用方法如下所示。

```
nth-child(odd){
//指定样式
}
//所有正数下来的第偶数个子元素
<子元素>:nth-child(even){
//指定样式
}
//所有倒数上去的第奇数个子元素
<子元素>:nth-last-child(odd){
//指定样式
}
//所有倒数上去的第偶数个子元素
<子元素>:nth-last-child(even){
//指定样式
}
```

接下来，我们在代码清单13-14中看一个使用nth-child选择器来分别针对ul列表的第奇数个列表项目与第偶数个列表项目指定不同背景色的示例。在该示例中将所有第奇数个列表项目的背景色设为黄色，将所有第偶数个列表项目的背景色设为浅蓝色。

代码清单13-14 使用nth-child对第奇数个、第偶数个子元素使用不同样式的示例

```
<!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>使用nth-child对第奇数个、第偶数个子元素使用不同样式示例</title>
<style type="text/css">
li:nth-child(odd){
    background-color: yellow;
}
li:nth-child(even){
```

```
        background-color: skyblue;
    }
    </style>
    </head>
    <body>
    <h2>列表A</h2>
    <ul>
    <li>列表项目1</li>
    <li>列表项目2</li>
    <li>列表项目3</li>
    <li>列表项目4</li>
    <li>列表项目5</li>
    </ul>
    </body>
    </html>
```

这段代码的运行结果如图13-18所示。

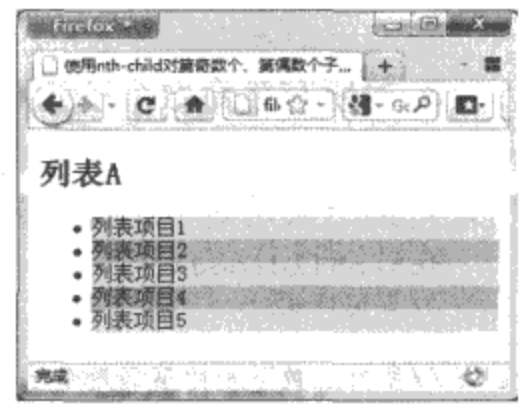

图13-18 使用nth-child对第奇数个、第偶数个子元素使用不同样式的示例

另外，虽然在对列表项目使用nth-child及nth-last-child选择器时没有问题，但是当把这两个选择器用于其他元素时，还是会有些问题，在13.3.4节中，我们将阐述会产生哪些问题，以及怎么解决这些问题。

13.3.4 选择器：nth-of-type和nth-last-of-type

1. 使用选择器nth-child和nth-last-child时会产生的问题

在13.3.3节中，我们介绍过将nth-child选择器与nth-last-child选择器用于某些元素时，会产生一些问题，这里我们首先来看一下究竟会产生什么问题。

在代码清单13-15中，我们给出一个HTML页面的代码，在该页面中，存在一个div元素，在该div元素中，又给出了几篇文章的标题与每篇文章的正文。

代码清单13-15 nth-of-type选择器与nth-last-of-type选择器使用示例的HTML页面

```
<!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>nth-of-type选择器与nth-last-of-type选择器使用示例</title>
</head>
<body>
<div>
<h2>文章标题A</h2>
<p>文章正文。</p>
<h2>文章标题B</h2>
<p>文章正文。</p>
<h2>文章标题C</h2>
<p>文章正文。</p>
<h2>文章标题D</h2>
```

```
<p>文章正文。</p>
</div>
</body>
</html>
```

为了让第奇数篇文章的标题与第偶数篇文章的标题的背景色不一样，我们首先使用nth-child选择器来进行指定，指定第奇数篇文章的标题背景色为黄色，第偶数篇文章的标题背景色为浅蓝色，书写方法如下所示。

```
<style type="text/css">
h2:nth-child(odd){
    background-color: yellow;
}
h2:nth-child(even){
    background-color: skyblue;
}
</style>
```

将上面这段指定样式的代码添加在如代码清单13-15所示的HTML页面中，然后在浏览器中查看该页面的运行结果，如图13-19所示。

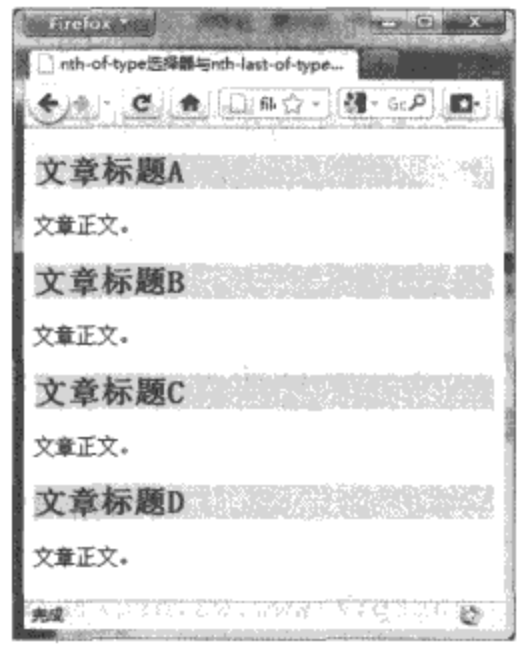

图13-19 在代码清单13-15所示的HTML页面中使用nth-child选择器

运行结果并没有如预期的那样，让第奇数篇文章的标题背景色为黄色，第偶数篇文章的标题背景色为浅蓝色，而是所有文章的标题都变成了黄色。

这个问题的产生原因是，nth-child选择器在计算子元素是第奇数个元素还是第偶数个元素的时候，是连同父元素中的所有子元素一起计算的。

换句话说，“h2:nth-child(odd)”这句话的含义，并不是指“针对div元素中第奇数个h2子元素来使用”，而是指“当div元素中的第奇数个子元素如果是h2子元素的时候使用”。

所以在上面这个示例中，因为h2元素与p元素相互交错，所有h2元素都处于奇数位置，所以所有h2元素的背景色都变成了黄色，而处于偶数位置的p元素，因为没有指定第偶数个位置的子元素的背景色，所以没有发生变化。

当父元素是列表的时候，因为列表中只可能有列表项目一种子元素，所以不会有问题，而当父元素是div的时候，因为div元素中有了不止一种子元素，所以引起了问题的产生。

2. 使用选择器nth-of-type和nth-last-of-type

在CSS 3中，使用nth-of-type选择器与nth-last-of-type选择器可以避免这类问题的发生。使用这两个选择器的时候，CSS 3在计算子元素是第奇数个子元素还是第偶数个子元素的时候，就只针对同类型的子元素进行计算了。这两个选择器的使用方法如下所示。

```
<style type="text/css">
h2:nth-of-type(odd){
    background-color: yellow;
```

```
}
h2:nth-of-type(even){
    background-color: skyblue;
}
</style>
```

把以上这段代码添加到代码清单13-15所示页面中，然后运行该页面，运行的结果如图13-20所示。

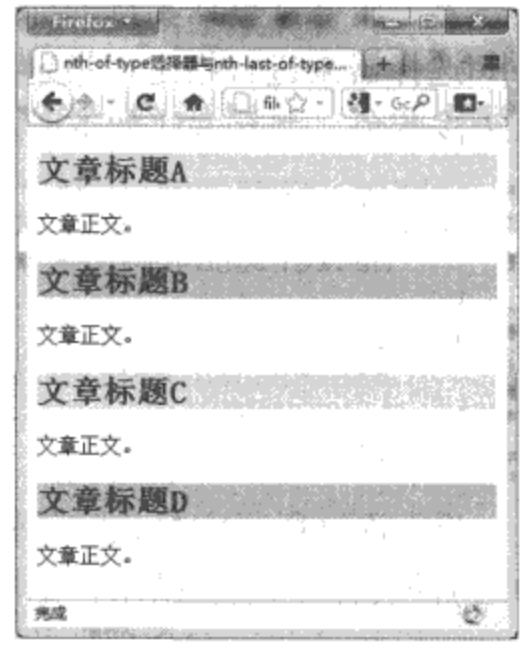

图13-20 nth-of-type选择器使用示例

另外，如果计算奇数还是偶数的时候需要从下往上倒过来计算的话，可以使用nth-last-of-type选择器来代替nth-last-child选择器，进行倒序计算。

nth-of-type选择器与nth-last-of-type选择器都是从CSS 3开始提供，到目前为止被Firefox、Safari、Google Chrome、Opera浏览器所支持，到IE8为止，还没有得到Internet Explorer浏览器的支持。

13.3.5 循环使用样式

通过前几节的介绍我们已经知道，使用nth-child选择器、nth-last-child选择器、nth-of-type选择器与nth-last-of-type选择器，可以对父元素中指定序号的子元素、第奇数个子元素、第偶数个子元素来单独进行样式的指定。这里我们再通过代码清单13-16中的示例，来复习一下nth-child选择器的用法。在该示例中，有一个ul列表，通过nth-child选择器来指定该列表中第一个列表项目、第二个列表项目、第三个列表项目及第四个列表项目的背景色。

代码清单13-16 使用nth-child选择器指定项目背景色

```
<!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>使用nth-child选择器指定项目背景色</title>
<style type="text/css">
li:nth-child(1) {
    background-color: yellow;
}
li:nth-child(2) {
    background-color: limegreen;
}
li:nth-child(3) {
    background-color: red;
}
li:nth-child(4) {
    background-color: white;
}
</style>
```

```
</head>
<body>
<ul>
<li>列表项目1</li>
<li>列表项目2</li>
<li>列表项目3</li>
<li>列表项目4</li>
<li>列表项目5</li>
<li>列表项目6</li>
<li>列表项目7</li>
<li>列表项目8</li>
<li>列表项目9</li>
<li>列表项目10</li>
<li>列表项目11</li>
<li>列表项目12</li>
</ul>
</body>
</html>
```

这段代码的运行结果如图13-21所示。

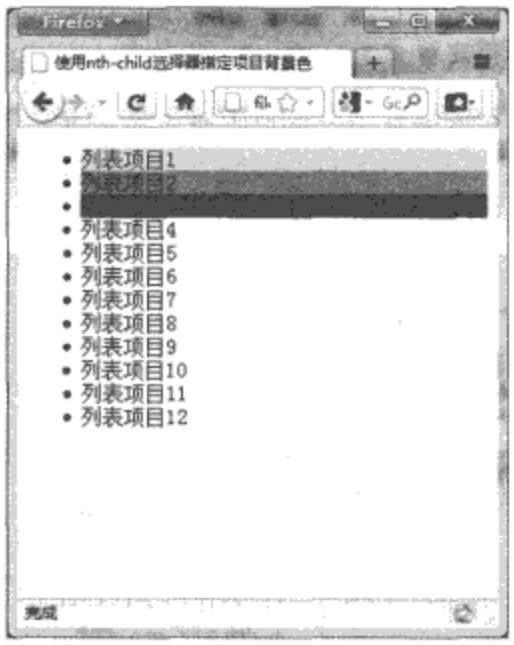

图13-21 使用nth-child选择器指定项目背景色

在图中，我们可以看见该列表中前4个列表项目的背景色已设定好，其他列表项目的背景色均未设定。现在，要讨论一个问题，如果开发者想对所有的列表项目都设定背景色，但是不采用这种一个个列表项目分别指定的方式（如果有100个列表项目的话，工作量就太大了），而是采用循环指定的方式，让剩下来的列表项目循环采用一开始已经指定好的背景色，应该怎么处理呢？

这时，仍然可以采用nth-child选择器，只要在“nth-child（n）”语句处，把参数n改成可循环的αn+β的形式就可以。α表示每次循环中共包括几种样式，β表示指定的样式在循环中所片的位置。譬如此处是4种背景色作为一组循环，则将前面代码清单13-16中样式指定的地方修改成如下所示的指定方法。

```
<style type="text/css">
li:nth-child(4n+1) {
    background-color: yellow;
}
li:nth-child(4n+2) {
    background-color: limegreen;
}
li:nth-child(4n+3) {
    background-color: red;
}
li:nth-child(4n+4) {
    background-color: white;
```

```
}
</style>
```

将这段代码替代到代码清单13-16中样式指定的地方，然后运行代码清单13-16中的代码，运行结果如图13-22所示。

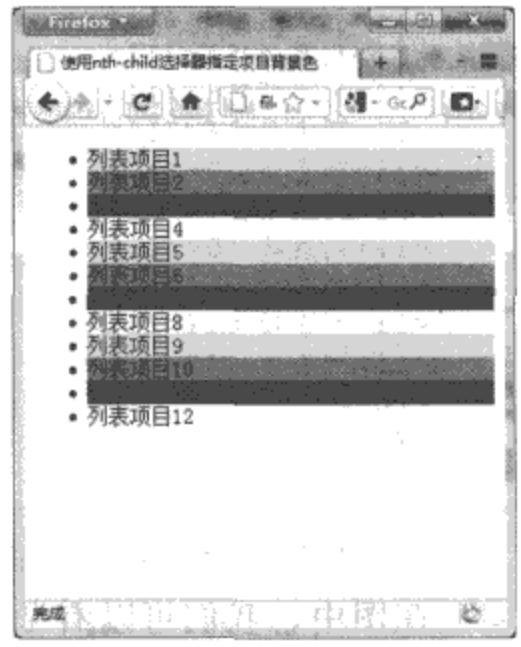

图13-22 循环使用样式示例

在运行结果中，我们可以清楚地看到，所有列表项目均循环使用了开头4个列表项目中的背景色。

另外，“4n+4”的写法可略写成“4n”的形式。

因此，前面我们所说的nth-child(odd)选择器、nth-child(even)选择器实际上可以采用如下形式进行代替。

```
//所有正数下来的第奇数个子元素
<子元素>:nth-child(2n+1){
//指定样式
}
//所有正数下来的第偶数个子元素
<子元素>:nth-child(2n+2){
//指定样式
}
//所有倒数上去的第奇数个子元素
<子元素>:nth-last-child(2n+1){
//指定样式
}
//所有倒数上去的第偶数个子元素
<子元素>:nth-last-child(2n+2){
//指定样式
}
```

13.3.6 only-child选择器

如果采用如下所示的方法结合运用nth-child选择器与nth-last-child选择器的话，可以指定当某个父元素中只有一个子元素时才使用的样式。

```
<子元素>:nth-child(1). nth-last-child(1){
    //指定样式
}
```

接下来，我们在代码清单13-17中看一个示例，该示例中有两个ul列表，一个ul列表里有几个列表项目，另一个ul列表里只有一个列表项目。在样式中指定li列表的背景色为黄色，但是由于采用了结合运用nth-child选择器与nth-last-child选择器并且将序号都设定为1的方式，所以显示出来的页面中只有拥有唯一列表项目的那个ul列表中的列表项目背景色为黄色。

代码清单13-17 只对唯一列表项目使用样式示例

```
<!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>只对唯一列表项目使用样式示例</title>
```

```
<style type="text/css">
li:nth-child(1):nth-last-child(1){
    background-color: yellow;
}
</style>
</head>
<body>
<h2>ul列表A</h2>
<ul>
<li>列表项目A01</li>
</ul>
<h2>ul列表B</h2>
<ul>
<li>列表项目B01</li>
<li>列表项目B02</li>
<li>列表项目B03</li>
</ul>
</body>
</html>
```

这段代码的运行结果如图13-23所示。

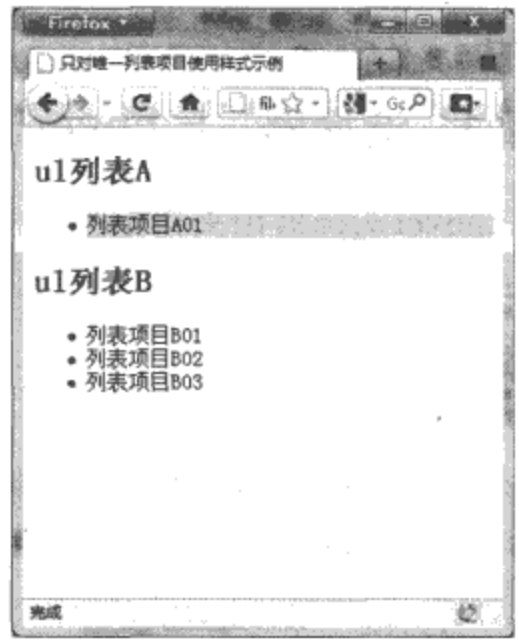

图13-23 只对唯一列表项目使用样式示例

另外，可以使用only-child选择器来代替使用“nth-child(1)：nth-last-child(1)”的实现方法。譬如在上面这个示例中，可以将样式指定中的代码改成如下所示的指定方法。

```
<style type="text/css">
li:only-child{
    background-color: yellow;
}
</style>
```

读者可自行将上面示例中的样式指定代码用这段代码进行替代，然后在浏览器中重新查看运行结果。另外，也可使用only-of-type选择器替代“nth-of-type(1):nth-last-of-type(1)”这种结合nth-of-type选择器与nth-last-of-type选择器来让样式只对唯一子元素起作用的实现方法。nth-of-type选择器与nth-last-of-type选择器的作用与使用方法在前文已经介绍，此处不再赘述。

13.4 UI元素状态伪类选择器

在CSS 3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。

在CSS 3中，共有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及E::selection。

到目前为止，这11种选择器被浏览器的支持情况如表13-1所示。

表13-1 各UI元素状态伪类选择器受浏览器的支持情况

选择器	Firefox	Safari	Opera	Internet Explorer 8	Chrome
E:hover	✓	✓	✓	✓	✓
E:active	✓	✓	✓	×	✓
E:focus	✓	✓	✓	✓	✓
E:enabled	✓	✓	✓	×	✓
E:disabled	✓	✓	✓	×	✓
E:read-only	✓	×	✓	×	×
E:read-write	✓	×	✓	×	×
E:checked	✓	✓	✓	×	✓
E::selection	✓	✓	✓	×	✓
E:default	✓	×	×	×	×
E: indeterminate	×	×	✓	×	×

13.4.1 选择器：E:hover、E:active和E:focus

E:hover选择器用来指定当鼠标指针移动到元素上面时元素所使用的样式，使用方法如下所示：

```
<元素>:hover{
//指定样式
}
```

可以在“<元素>”中添加元素的type属性，使用方法如下：

```
input[type="text"]: hover{
    //指定的样式
}
```

所有UI元素状态伪类选择器的使用方法均与此类似，后面不再赘述。

- E:active选择器用来指定元素被激活（鼠标在元素上按下还没有松开）时使用的样式。
- E:focus选择器用来指定元素获得光标焦点时使用的样式，主要是在文本框控件获得焦点并进行文字输入的时候使用。

代码清单13-18是使用了这3个选择器的综合示例，该示例中有两个文本框控件，使用这3个选择器来指定当鼠标指针移动到文本框控件上面时、文本框控件被激活时以及光标焦点落在文本框之内时的样式。

代码清单13-18 选择器E:hover、E:active和E:focus的使用示例

```
<!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>E:hover选择器、E:active选择器与E:focus选择器使用示例</title>
</head>
<style type="text/css">
input[type="text"]:hover{
        background-color: greenyellow;
}
input[type="text"]:focus{
        background-color: skyblue;
}
input[type="text"]:active{
        background-color: yellow;
}
</style>
<body>
<form>
<p>姓名：<input type="text" name="name" /></p>
<p>地址：<input type="text" name="address" /></p>
</form>
</body>
</html>
```

对于示例中的任意一个文本框控件来说，这段代码的运行结果都可能有如下4种情况。

- 没有对文本框控件进行任何操作时的页面显示如图13-24所示（文本框背景色为白色）。
- 鼠标指针移动到某一个文本框控件上面时的页面显示如图13-25所示（文本框背景色为绿色）。

图13-24 代码清单13-18的运行结果
（没有对文本框控件进行任何操作时）

图13-25 代码清单13-18的运行结果
（鼠标指针移动到姓名文本框控件上面时）

- 文本框控件被激活时的页面显示如图13-26所示（文本框背景色为黄色）。
- 文本框控件获得光标焦点后的页面显示如图13-27所示（文本框背景色为浅蓝色）。

图13-26 代码清单13-18的运行结果
（姓名文本框控件被激活时）

图13-27 代码清单13-18的运行结果
（姓名文本框控件获得光标焦点时）

13.4.2 E:enabled伪类选择器与E:disabled伪类选择器

- E:enabled伪类选择器用来指定当元素处于可用状态时的样式。
- E:disabled伪类选择器用来指定当元素处于不可用状态时的样式。

当一个表单中的元素经常在可用状态与不可用状态之间进行切换的时候，通常会将E:disabled伪类选择器与E:enabled伪类选择器结合使用，用E:disabled伪类选择器来设置该元素处于不可用状态时的样式，用E:enabled伪类选择器来设置该元素处于可用状态时的样式。

代码清单13-19中给出了一个将E:disabled伪类选择器与E:enabled伪类选择器结合使用的示例，在该示例中有两个radio单选框与一个文本框，在JavaScript脚本中编写代码，当用户选中其中一个radio单选框时，文本框变为可用状态，选中另一个radio单选框时，文本框变为不可用状态。通过结合使用E:disabled伪类选择器与E:enabled伪类选择器，让文本框处于不同的状态时分别使用不同的样式。

代码清单13-19 E:disabled伪类选择器与E:enabled伪类选择器结合使用的示例

```
<!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>E: disabled伪类选择器与E:enabled伪类选择器结合使用示例</title>
<script>
function radio_onchange()
{
    var radio=document.getElementById("radio1");
    var text=document.getElementById("text1");
    if(radio.checked)
        text.disabled="";
    else
    {
        text.value="";
        text.disabled="disabled";
    }
}
</script>
<style>
input[type="text"]:enabled{
    background-color:yellow;
}
input[type="text"]:disabled{
    background-color:purple;
}
</style>
</head>
<body>
<form>
<input type="radio" id="radio1" name="radio"
 onchange="radio_onchange();">可用</radio>
<input type="radio" id="radio2" name="radio"
onchange="radio_onchange();">不可用</radio><br/>
```

```
<input type=text id="text1" disabled />
</form>
</body>
</html>
```

这段代码的运行结果可分为如下两种情况：

- 文本框处于可用状态时的页面显示如图13-28所示（背景色为黄色）。
- 文本框处于不可用状态时的页面显示如图13-29所示。

图13-28 代码清单13-19的运行结果
（文本框处于可用状态时）

图13-29 代码清单13-19的运行结果
（文本框处于不可用状态时）

13.4.3 E: read-only伪类选择器与E:read-write伪类选择器

- E:read-only伪类选择器用来指定当元素处于只读状态时的样式。
- E:read-write伪类选择器用来指定当元素处于非只读状态时的样式。

在Firefox浏览器中，需要写成“-moz-read-only”或“-moz-read-write”的形式。

代码清单13-20为E:read-only选择器与E:read-write选择器结合使用的一个示例，在该示例中有一个姓名文本框控件和一个地址文本框控件。其中姓名文本框控件不是只读控件，使用E:read-write选择器定义样式；地址文本框控件是只读控件，使用E:read-only选择器定义样式。

代码清单13-20 E:read-only伪类选择器与E:read-write伪类选择器结合使用的示例

```
<!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> E: read-only伪类选择器与E:read-write伪类选择器结合使用示例
</title>
<style type="text/css">
input[type="text"]:read-only{
        background-color: gray;
}
input[type="text"]:read-write{
        background-color: greenyellow;
}
input[type="text"]:-moz-read-only{
        background-color: gray;
}
```

```
input[type="text"]:-moz-read-write{
        background-color: greenyellow;
}
</style>
</head>
<body>
<form>
<p>名前：<input type="text" name="name" />
<p>地址：<input type="text" name="address" value="江苏省常州市"
 readonly="readonly" />
</p>
</form>
</body>
</html>
```

这段代码的运行结果如图13-30所示。

图13-30 E:read-only伪类选择器与E:read-write伪类选择器结合使用的示例

13.4.4 伪类选择器：E:checked、E:default和E: indeterminate

E:checked伪类选择器用来指定当表单中的radio单选框或checkbox复选框处于选取状态时的样式。在Firefox浏览器中，需要把它写成“-moz-checked”的形式。

代码清单13-21为一个E:checked伪类选择器的使用示例，在该示例中使用了几个checkbox复选框，复选框在非选取状态时边框默认为黑色，当复选框处于选取状态时通过E:checked伪类选择器让选取框的边框成为蓝色。

代码清单13-21 E:checked伪类选择器的使用示例

```
<!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>E:checked伪类选择器使用示例</title>
<style type="text/css">
input[type="checkbox"]:checked {
    outline:2px solid blue;
}
input[type="checkbox"]:-moz-checked {
    outline:2px solid blue;
}
</style>
</head>
```

```
<body>
<form>
兴趣:<input type="checkbox">阅读</input>
<input type="checkbox">旅游</input>
<input type="checkbox">看电影</input>
<input type="checkbox">上网</input>
</form>
</body>
</html>
```

这段代码的运行结果如图13-31所示。

图13-31 E:checked伪类选择器的使用示例

E:default选择器用来指定当页面打开时默认处于选取状态的单选框或复选框控件的样式。需要注意的是，即使用户将该单选框或复选框控件的选取状态设定为非选取状态，E:default选择器中指定的样式仍然有效。

代码清单13-22为一个E:default选择器的使用示例，该示例中有几个复选框，第一个复选框被设定为默认打开时为选取状态，使用E:default选择器设定该复选框的边框为蓝色。

代码清单13-22 E:default选择器的使用示例

```
<!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>E:default选择器的使用示例</title>
<style type="text/css">
input[type="checkbox"]:default {
    outline:2px solid  blue;
}
</style>
</head>
<body>
<form>
兴趣:<input type="checkbox" checked>阅读</input>
<input type="checkbox">旅游</input>
<input type="checkbox">看电影</input>
<input type="checkbox">上网</input>
</form>
</body>
</html>
```

这段代码的运行结果如图13-32所示。

需要注意的是，即使用户将默认设定为选取状态的单选框或复选框修改为非选取状态，使用default选择器设定的样式依然有效，如图13-33所示。

E:indeterminate伪类选择器用来指定当页面打开时，如果一组单选框中任何一个单选框都没有被设定为选取状态时整组单选框的样式，如果用户选取了其中任何一个单选框，则该样式被取消指定。到目前为止，只有Opera浏览器对这个选择器提供支持。

图13-32 E:default选择器的使用示例

图13-33 复选框被修改为非选取状态后使用default选择器设定的样式依然有效

代码清单13-23为一个E:indeterminate选择器的使用示例，该示例中有一组单选框，其中任何一个单选框都没有被设定为默认选取状态，使用E:indeterminate选择器来设定页面打开时该组单选框的边框为蓝色。

代码清单13-23 E:indeterminate选择器的使用示例

```
<!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> E:indeterminate选择器的使用示例</title>
<style type="text/css">
input[type="radio"]:indeterminate{
        outline: solid 3px blue;
}
</style>
</head>
<body>
<form>
年龄:
<input type="radio" name="radio" value="male" />男
<input type="radio" name="radio" value="female" />女
</form>
</body>
</html>
```

这段代码所示的示例在页面打开时的页面显示如图13-34所示。

用户只要选取其中任何一个单选框则使用E:indeterminate选择器指定的样式就会被取消，如图13-35所示。

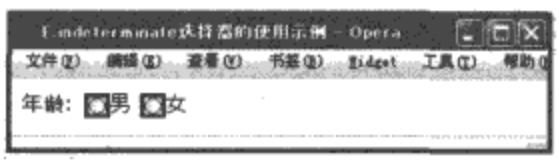

图13-34 E:indeterminate选择器的使用示例

图13-35 用户选取任何一个单选框后，使用E:indeterminate选择器指定的样式就会被取消

13.4.5 E::selection伪类选择器

E::selection伪类选择器用来指定当元素处于选中状态时的样式。

代码清单13-24为一个E::selection伪类选择器的使用示例，在该示例中分别给出了一个p元素、一个文本框控件以及一个表格。当p元素处于选中状态时，被选中文字变为红色；当文本框控件处于选中状态时，被选中文字变为灰色；当表格处于选中状态时，被选中文字变为绿色。

代码清单13-24 E::selection伪类选择器的使用示例

```
<!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>E::selection伪类选择器使用示例</title>
<style type="text/css">
p::selection{
    background:red;
    color:#FFF;
}
p::-moz-selection{
    background:red;
    color:#FFF;
}
input[type="text"]::selection{
    background:gray;
    color:#FFF;
}
input[type="text"]::-moz-selection{
    background:gray;
    color:#FFF;
}
td::selection{
    background:green;
    color:#FFF;
}
td::-moz-selection{
    background:green;
    color:#FFF;
}
</style>
</head>
<body>
<p>这是一段测试文字。</p>
<input type="text" value="这是一段测试文字。"/><p/>
<table border="1" cellspacing="0" cellpadding="0">
<tr>
<td>测试文字</td>
<td>测试文字</td>
</tr>
<tr>
<td>测试文字</td>
<td>测试文字</td>
</tr>
</body>
</html>
```

这段代码的运行结果如图13-36所示。

图13-36 E::selection伪类选择器的使用示例

13.5 通用兄弟元素选择器

关于选择器部分，最后要介绍的一个选择器是通用兄弟元素选择器，它用来指定位于同一个父元素之中的某个元素之后的所有其他某个种类的兄弟元素所使用的样式，它的使用方法如下所示。

```
<子元素> ~<子元素之后的同级兄弟元素> {
//指定样式
}
```

这里的同级是指子元素和兄弟元素的父元素是同一个元素。

下面我们在代码清单13-25中来看一个通用兄弟元素选择器的使用示例，该示例中对所有div元素之后的、与div元素同级的p元素指定其背景色为绿色，但是对div元素内部的p元素的背景色不做指定。

代码清单13-25 通用兄弟元素选择器的使用示例

```
<!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">
div ~ p {background-color:#00FF00;}
</style>
<title>通用兄弟元素选择器 E ~ F</title>
</head>
<body>
<div style="width:733px; border: 1px solid #666; padding:5px;">
<div>
    <p>p元素为div元素的子元素</p>
    <p>p元素为div元素的子元素</p>
</div>
<hr />
<p>p元素为div元素的兄弟元素</p>
<p>p元素为div元素的兄弟元素</p>
<hr />
```

```
<p>p元素为div元素的兄弟元素</p>
<hr />
<div>p元素为div元素的子元素</div>
<hr />
<p>p元素为div元素的兄弟元素</p>
</div>
</body>
</html>
```

这段代码的运行结果如图13-37所示。

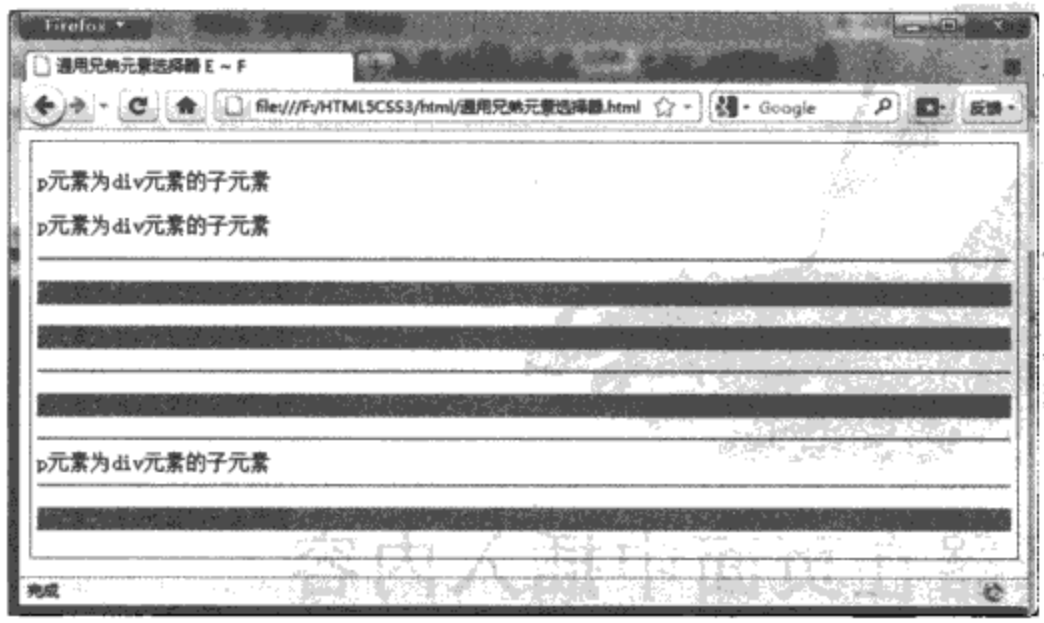

图13-37 通用兄弟元素选择器的使用示例

第14章
使用选择器在页面中插入内容

在13.3.1节中介绍CSS中的伪元素时，我们曾经介绍过，在CSS中可以使用before伪元素选择器与after伪元素选择器在页面中的元素的前面或后面插入内容，而插入的内容是用content属性来定义的。确切地说，before伪元素选择器与after伪元素选择器是在CSS 2.0中添加的，但是从CSS 2.1开始，一直到CSS 3中，都不断地在针对这两个选择器进行改良和扩展，这使得before伪元素选择器与after伪元素选择器的作用越来越强大，因此本章将特别针对这两个选择器做详细的介绍。

学习内容：

- 掌握CSS 3中使用选择器在页面中插入文字的方法，能够使用before选择器与after选择器在页面中元素的前面或后面插入文字。
- 掌握CSS 3中使用选择器在页面中插入图像的方法，能够使用before选择器与after选择器在页面中元素的前面或后面插入图像文件。
- 掌握CSS 3中使用选择器在页面中插入项目编号的方法，能够使用before选择器与after选择器在页面中各种项目的前面或后面插入各种级别、各种样式的项目编号。

14.1　使用选择器来插入文字

14.1.1　使用选择器来插入内容

首先，让我们来回顾一下，在CSS 2中是如何使用样式在元素的前面或后面插入内容的。

在CSS 2中，使用before选择器在元素前面插入内容，使用after选择器在元素后面插入内容，在选择器的content属性中定义要插入的内容。例如，在如下所示的代码中，对h2元素使用before选择器，并且用content属性来定义在h2元素前面插入的内容为“COLUMN”文字。另外，当插入内容为文字的时候，必须要在插入文字的两旁加上单引号或者双引号。

```
<style type="text/css">
h2:before{
    content: 'COLUMN'
}
</style>
<h2>标题</h2>
```

为了让插入的文字具有美观效果，我们可以在选择器中加入文字的颜色、背景色、文字的字体等各种样式。代码清单14-1是一个before选择器的使用示例，在该示例中，在“标题”文字前加入“COLUMN”文字，在before选择器中，指定文字颜色为白色，背景色为橘色，并且用padding属性与margin属性对文字周围的余白进行适当的设定，同时，指定字体为“Comic Sans MS”。

代码清单14-1　before选择器的使用示例

```
<!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> before选择器的使用示例</title>
</head>
<style type="text/css">
h2:before{
    content: 'COLUMN';
    color: white;
    background-color: orange;
    font-family: 'Comic Sans MS', Helvetica, sans-serif;
    padding: 1px 5px;
    margin-right: 10px;
}
</style>
<body>
<h2>标题文字</h2>
</body>
</html>
```

这段代码的运行结果如图14-1所示。

图14-1 before选择器的使用示例

另外，如果将before选择器改为after选择器，则将“COLUMN”文字插入到标题文字的后面。

14.1.2 指定个别元素不进行插入

在代码清单14-1的示例中，因为对页面上的h2元素使用了before选择器，所以该页面上如果有多个h2元素，则所有的h2元素前面都会被插入内容。如果想让其中一个或几个h2元素的前面不要插入内容时，应该怎么指定呢？

在CSS 2.1中，针对这个问题在content属性中追加了一个none属性值，使用方法如下代码所示。

```
<style type="text/css">
h2.sample:before{
    content: none
}
</style>
<h2>标题1</h2>
<h2 class="sample">标题2</h2>
```

通过这种方法，替h2元素增加一个类，然后替这个类起个名字，在这个类的样式指定中将content属性值设定为“none”，然后在不需要插入内容的元素中将class属性的属性值设定为这个给定的类名就可以了。

代码清单14-2为将代码清单14-1修改后使用none属性值的示例，该页面中有三个h2元素，其中第二个h2元素前面没有被插入内容。

代码清单14-2　content属性的none属性值使用示例

```
<!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> content属性的none属性值使用示例</title>
</head>
<style type="text/css">
h2:before{
    content: "COLUMN";
    color: white;
    background-color: orange;
    font-family: 'Comic Sans MS', Helvetica, sans-serif;
    padding: 1px 5px;
    margin-right: 10px;
}
h2.sample:before{
    content: none
}
</style>
<body>
<h2>标题文字1</h2>
<h2 class="sample">标题文字2</h2>
<h2>标题文字3</h2>
</body>
</html>
```

这段代码的运行结果如图14-2所示。

另外，在CSS 2.1中，除了none属性值外，还为content属性添加了一个“normal”属性值，其作用与使用方法none属性值的作用相同，并且使用方法也相同，读者可自行在代码清单14-2中，将none属性值修改为normal属性值，然后在浏览器中重新运行该示例，观察运行结果。

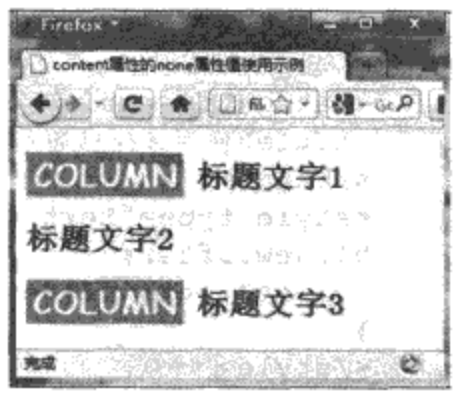

图14-2　content属性的none属性值使用示例

那么，既然normal属性值的作用与none属性值的作用相同，为什么CSS 3中还要追加这个normal属性值呢？它们的区别又是什么呢？这里要补充说明的是，从CSS 2.1开始，只有当使用before选择器与after选择器的时候，normal属性值的作用才与none属性值的作用相同，都是不让选择器在个别元素的前面或后面插入内容。但是none属性值只能应用在这两个选择器中，而normal属性值还可以应用在其他用来插入内容的选择器中，而在CSS 2中，只有before选择器与after选择器能够用来在元素的前面或后面插入内容，所以这两者的作用完全相同。在CSS 3草案中，已经追加了其他一些可以用来插入内容的选择器的提案，针对这一类选择器，就只

能使用normal属性值了，而且normal属性值的作用也会根据选择器的不同而发生变化。

14.2 插入图像文件

14.2.1 在标题前插入图像文件

使用before选择器或after选择器，除了可以在元素的前面或后面插入文字之外，还可以插入图像文件。插入图像时，需要使用url属性值来指定图像文件的路径。在如下所示的代码中，在h2标题元素前插入了mark.png图像文件。

```
h2:before{
    content:url(mark.png);
}
<h2>你好</h2>
```

目前Firefox、Safari、Opera浏览器都支持这种插入图像文件的功能，在IE8中只支持插入文字的功能，不支持插入图像文件的功能。

另外，在CSS 3的定义中还可以通过url属性来插入音频文件、视频文件等其他格式的文件，但目前还没有得到任何浏览器的支持。

14.2.2 插入图像文件的好处

虽然可以利用img元素在画面中追加图像文件，但是也可以使用样式表来追加图像文件，这样做的好处是可以为页面的编写节省大量时间。

例如，在代码清单14-3所示的示例中，可以利用名字为“new”的类来在个别标题后面追加表示新内容的图像文件，这个功能可以被利用在购物网站的商品清单中，用来表示哪些货物是新到的，或者用在文章网站的文章列表中，用来表示哪些文章是新发表的。

代码清单14-3 使用选择器插入图像文件的示例

```
<!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>使用选择器插入图像文件示例</title>
</head>
<style type="text/css">
h1.new:after{
    content:url(new.gif);
}
</style>
<body>
<h1 class="new">标题A</h1>
<h1 class="new">标题B</h1>
<h1>标题C</h1>
<h1>标题D</h1>
<h1>标题E</h1>
```

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

这段代码的运行结果如图14-3所示。

另外，还有一种在样式表中追加图像文件的方法，就是把它作为元素的背景图像文件来追加。例如代码清单14-4的示例中，同时对两个标题元素追加图像文件，对第一个标题元素采用before选择器，对第二个标题元素采用追加背景图像的方法来追加。在浏览器中显示的时候，这两种追加的结果看不出有什么区别。

图14-3　使用选择器插入图像文件的示例

代码清单14-4　同时采用两种方法追加图像文件的示例

```
<!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>同时采用两种方法追加图像文件示例</title>
</head>
<style type="text/css">
h1.head01:before{
    content:url(new.gif);
}
h1.head02{
    background-image:url(new.gif);
    background-repeat:no-repeat;
    padding-left:28px;
}
</style>
<body>
<h1 class="head01">标题A</h1>
<h1 class="head02">标题B</h1>
</body>
</html>
```

这段代码的运行结果如图14-4所示。

但是，在打印的时候，如果设定为不打印背景的话，使用before选择器追加的图像文件能够正常打印，但是使用追加背景图像的方法追加的图像文件就不能正常打印了。

譬如，在Firefox浏览器中运行代码清单14-4中的示例代码，然后点击“文件”菜单下的“打印预览”子菜单，在弹出的打印预览对话框中，点击页面设置按钮，在弹出的页面设置对话框中将“打印背景（颜色和图片）”复选框设为非选取状态，然后关闭页面设置对话框，观察打印预览对话框中的画面，画面变为如图14-5所示。

图14-4 同时采用两种方法追加图像文件示例

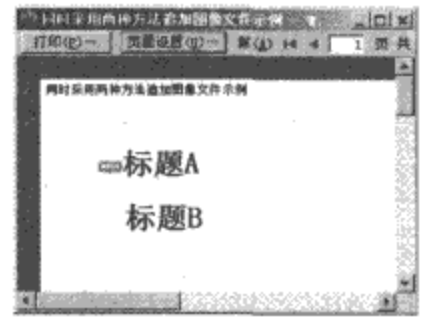

图14-5 将打印时的页面设置修改为不打印背景

14.2.3 将alt属性的值作为图像的标题来显示

如果在content属性中通过“attr(属性名)”这种形式来指定attr属性值，可以将某个属性的属性值显示出来。在代码清单14-5中，给出一个attr属性值的使用示例。在该示例中，在页面上用img元素显示一个图像文件，并且在该元素中指定alt属性的属性值，alt属性的作用是用来指定当图像不能正常显示时所显示的替代文字。在图像文件后面显示图像文件的标题，在样式中将attr属性值设定为img元素的alt属性值，这样图像文件的标题文字就是alt属性中指定的文字了。到目前为止，只有Opera 10浏览器对这个attr属性值提供支持。

代码清单14-5 attr属性值的使用示例

```
<!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>attr属性值的使用示例</title>
</head>
<style type="text/css">
img:after{
    content:attr(alt);
    display:block;
    text-align:center;
    margin-top:5px;
}
</style>
<body>
<p><img src="sky.jpg" alt="蓝天白云"/></p>
</body>
</html>
```

这段代码在Opera 10浏览器中的运行结果如图14-6所示。

图14-6 attr属性值的使用示例

14.3　使用content属性来插入项目编号

前面两节中分别介绍了利用before选择器与after选择器的content属性在元素的前面或后面插入文字与图像的方法，本节介绍当页面中具有多个项目时如何利用这个content属性来在项目前插入项目编号，在本节的最后介绍一下如何利用这个content属性在字符串两边加上括号。

到目前为止，Firefox、Safari、Opera浏览器均支持插入项目编号的功能，在Internet Explorer中从IE8开始支持这个功能。

14.3.1　在多个标题前加上连续编号

在content属性中使用counter属性值来针对多个项目追加连续编号，使用方法如下所示。

```
<元素>：before{
    content：counter(计数器名);
}
```

使用计数器来计算编号，计数器可任意命名。

另外，还需要在元素的样式中追加对元素的counter-increment属性的指定，为了使用连续编号，需要将counter-increment属性的属性值设定为before选择器或after选择器的counter属性值中所指定的计数器名。代码如下所示。

```
<元素>{
    counter-increment:before选择器或after选择器中指定的计数器名
}
```

接下来，我们在代码清单14-6中看一个对多个项目追加连续编号的示例，在该示例中具有多个标题，使用before选择器对这些标题追加连续编号。

代码清单14-6　对多个项目追加连续编号的示例

```
<!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>对多个项目追加连续编号的示例</title>
</head>
<style type="text/css">
h1:before{
    content: counter(mycounter);
}
h1{
    counter-increment: mycounter;
}
</style>
<body>
<h1>大标题</h1>
<p>示例文字。</p>
<h1>大标题</h1>
<p>示例文字。</p>
```

```
<h1>大标题</h1>
<p>示例文字。</p>
</body>
</html>
```

这部分代码的运行结果如图14-7所示。

14.3.2 在项目编号中追加文字

可以在插入的项目编号中加入文字，使项目编号变成类似“第1章”之类的带文字的编号。针对代码清单14-6，只要将before选择器中的代码修改为如下所示的代码就可以了。

```
h1:before{
content: '第'counter(mycounter)'章';
}
```

将代码清单14-6中before选择器中的代码用上面这段代码进行替代，然后重新运行该示例，运行结果如图14-8所示。

图14-7 对多个项目追加连续编号的示例

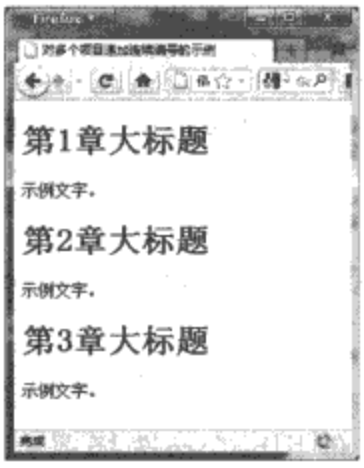

图14-8 在项目编号中追加文字的示例

14.3.3 指定编号的样式

可以指定追加编号的样式，譬如对代码清单14-6中追加的编号指定如下所示的样式，使得编号后面带一个“.”文字，编号颜色为蓝色，字体大小为42像素。

```
h1:before{
    content: counter(mycounter)'.';
    color:blue;
    font-size:42px;
}
```

将上面这段代码替换到代码清单14-6中，重新运行代码清单14-6，运行结果如图14-9所示。

14.3.4 指定编号的种类

用before选择器或after选择器的content属性，不仅可以追加数字编号，还可以追加字母编号或罗马数字编号。使用如下所示的方法指定编号种类。

```
content: counter(计数器名, 编号种类)
```

可以使用list-style-type属性的值来指定编号的种类，list-style-type为指定列表编号时所用的属性。例如，指定大写字母编号时，使用“upper-alpha”属性，指定大写罗马字母时，使用“upper-roman”属性。

将代码清单14-6中before选择器中的代码修改成如下所示的代码，然后重新运行该示例，运行结果如图14-10所示。

```
h1:before{
    content: counter(mycounter,upper-alpha)'.';
    color:blue;
    font-size:42px;
}
```

图14-9　指定编号的样式示例

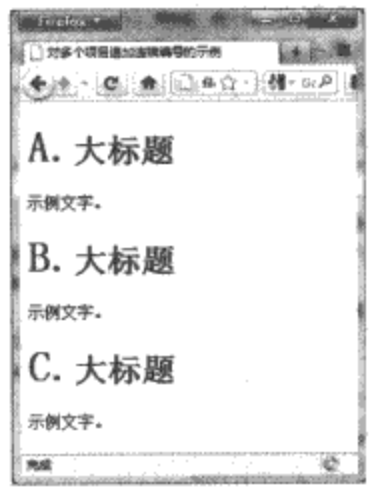

图14-10　指定编号的种类示例

14.3.5　编号嵌套

可以在大编号中嵌套中编号，在中编号中嵌套小编号。在代码清单14-7中，我们给出一个编号嵌套的示例，在该示例中，有两个大标题，每个大标题中又有三个中标题，使用编号嵌套的方式分别对大标题与中标题进行分层编号。

代码清单14-7　编号嵌套示例

```
<!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>编号嵌套示例</title>
</head>
<style type="text/css">
h1:before{
    content: counter(mycounter) '. ';
}
h1{
    counter-increment: mycounter;
}
h2:before{
```

```
    content: counter(subcounter) '. ';
}
h2{
    counter-increment: subcounter;
    margin-left: 40px;
}
</style>
<body>
<h1>大标题</h1>
<h2>中标题</h2>
<h2>中标题</h2>
<h2>中标题</h2>
<h1>大标题</h1>
<h2>中标题</h2>
<h2>中标题</h2>
<h2>中标题</h2>
</body>
</html>
```

这段代码的运行结果如图14-11所示。

在这个示例中，六个中标题的编号是连续的，如果要将第二个大标题里的中标题重新开始编号的话，需要在大标题中使用counter-reset属性将中编号进行重置。

将代码清单14-7中h1元素的样式指定的代码修改成如下代码（添加counter-reset属性），然后重新运行该示例，运行结果如图14-12所示。

```
h1{
    counter-increment: mycounter;
    counter-reset: subcounter;
}
```

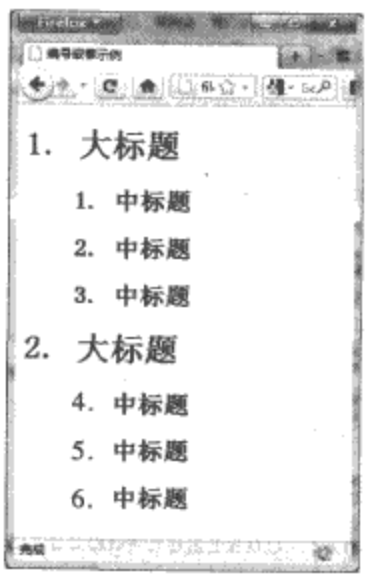

图14-11 编号嵌套示例

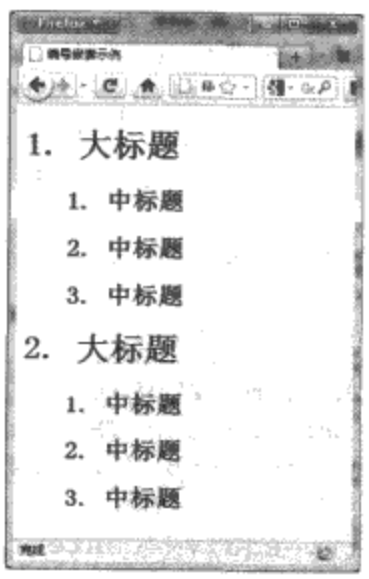

图14-12 重置中编号示例

14.3.6 中编号中嵌入大编号

可以将大编号嵌入在中编号中，譬如要将代码清单14-7中的中编号修改为“大编号-中编号”的形式，需要将中编号的before选择器中的代码修改成如下代码。

```
h2:before{
    content:  counter(mycounter) '-' counter(subcounter) '. ';
}
```

修改后在浏览器中重新运行代码清单14-7中的示例，运行结果如图14-13所示。

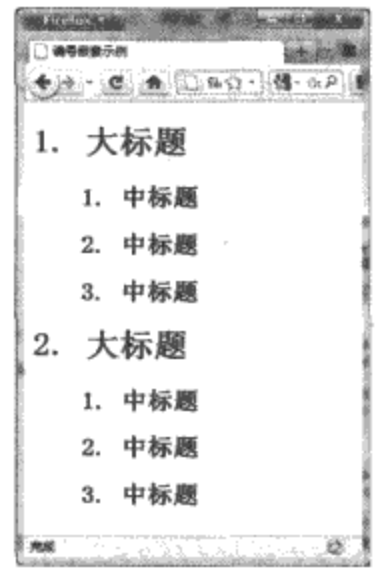

图14-13　中编号中嵌入大编号示例

同样的，可以在小编号中嵌入中编号，中编号中嵌入大编号，只需相应地在before选择器所指定的小编号中包括大编号与中编号，在before选择器所指定的中编号中包括大编号就可以了。

代码清单14-8为一个编号多层嵌入的示例，在该示例的页面中有两个大标题，每个大标题有两个中标题，每个中标题有两个小标题，小标题的编号中包括大标题的编号与中标题的编号，中标题的编号中具有大标题的编号。

代码清单14-8　编号多层嵌入的示例

```
<!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>编号多层嵌入的示例</title>
</head>
<style type="text/css">
h1:before{
    content: counter(mycounter) '. ';
}
h1{
    counter-increment: mycounter;
    counter-reset: subcounter;
}
h2:before{
    content:  counter(mycounter) '-' counter(subcounter) '. ';
}
h2{
    counter-increment: subcounter;
    counter-reset: subsubcounter;
    margin-left: 40px;
}
h3:before{
    content:counter(mycounter) '-' counter(subcounter) '-' counter(subsubcounter)'. ';
}
h3{
    counter-increment: subsubcounter;
```

```
        margin-left: 40px;
    }
    </style>
    <body>
    <h1>大标题</h1>
    <h2>中标题</h2>
    <h3>小标题</h3>
    <h3>小标题</h3>
    <h2>中标题</h2>
    <h3>小标题</h3>
    <h3>小标题</h3>
    <h1>大标题</h1>
    <h2>中标题</h2>
    <h3>小标题</h3>
    <h3>小标题</h3>
    <h2>中标题</h2>
    <h3>小标题</h3>
    <h3>小标题</h3>
    </body>
    </html>
```

这段代码的运行结果如图14-14所示。

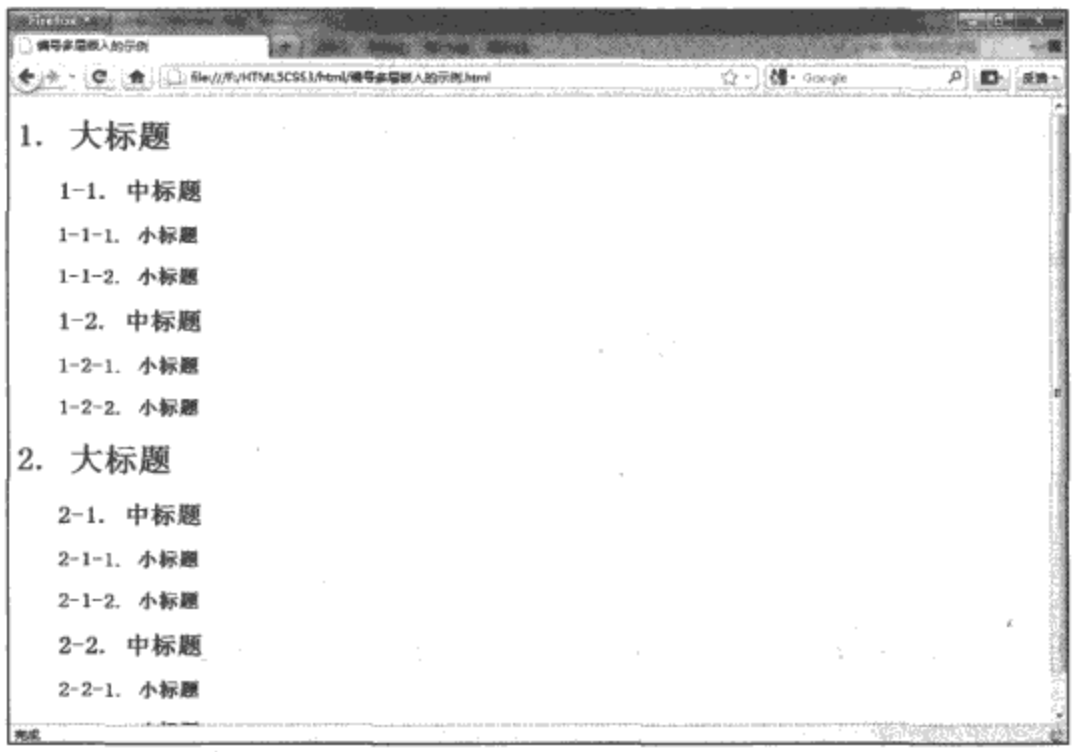

图14-14 编号多层嵌入的示例

14.3.7 在字符串两边添加嵌套文字符号

可以使用content属性的open-quote属性值与close-quote属性值在字符串两边添加诸如括号、单引号、双引号之类的嵌套文字符号。open-quote属性值用于添加开始的嵌套文字符号，close-quote属性值用于添加结尾的嵌套文字符号。

另外，在元素的样式中使用quotes属性来指定使用什么嵌套文字符号。

对于嵌套文字符号的添加功能，目前只有Firefox浏览器与Opera浏览器提供支持，Safari和IE8浏览器不提供支持。

代码清单14-9为添加嵌套文字符号的一个示例，在该示例中有一个h1标题元素，文字为“标题”，使用before选择器与after选择器在标题文字两边添加括号。

代码清单14-9　添加嵌套文字符号的示例

```
<!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>添加嵌套文字符号的示例</title>
</head>
<style type="text/css">
h1:before{
    content: open-quote;
}
h1:after{
    content: close-quote;
}
h1{
    quotes:" (" ") ";
}
</style>
<body>
<h1>标题</h1>
</body>
</html>
```

当需要添加双引号时，需要使用“\”转义字符，使用方法如下所示。

```
h1{
    quotes:"\"" "\"";
}
```

代码清单14-9的运行结果如图14-15所示。

图14-15　添加嵌套文字符号的示例

第15章
文字与字体相关样式

本章将针对CSS 3中与文字、字体相关的一些属性做详细介绍，其中包括text-shadow属性、word-break属性、word-wrap属性、Web Font和@font-face属性，以及font-size-adjust属性。

学习内容：

- 掌握如何使用text-shadow属性给页面上的文字添加阴影效果。
- 掌握如何使用word-break属性让页面上的文字可以根据自己的需要来进行换行，而不是使用浏览器默认的换行方式。
- 掌握如何使用word-wrap属性来让浏览器在长单词或很长的URL地址的中间进行换行。
- 掌握如何能够让浏览器在显示文字的时候使用服务器端的字体，而不再是只能使用客户端所安装的字体。
- 掌握如何使用font-size-adjust属性来保证在修改字体的时候不改变文字的大小，不会让页面上已经设计好的布局产生混乱。

15.1　给文字添加阴影——text-shadow属性

15.1.1　text-shadow属性的使用方法

在CSS 3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浏览器、Firefox浏览器、Chrome浏览器，以及Opera浏览器都支持该功能。

text-shadow属性是在CSS 2中定义的，在CSS 2.1中删除了，在CSS 3的Text模块中又恢复了。text-shadow的使用方法如下所示。

```
text-shadow: length length length color
```

其中，前面三个length分别指阴影离开文字的横方向距离、阴影离开文字的纵方向距离和阴影的模糊半径，color指阴影的颜色。

在代码清单15-1中，我们给出一个text-shadow属性的使用示例。在该示例中给一段红色文字绘制灰色阴影。其中阴影离开文字的横方向距离和纵方向距离均为5个像素。

代码清单15-1　text-shadow属性的使用示例

```
<!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>text-shadow属性的使用示例</title>
</head>
<style type="text/css">
div{
        text-shadow: 5px 5px 5px gray;
        color: navy;
        font-size: 50px;
        font-weight: bold;
        font-family: 宋体;
}
</style>
```

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

这段代码的运行结果如图15-1所示。

某些场合下可以通过给文字添加阴影来使页面上的文字更加容易看清楚，譬如文字与背景不能很容易地分辨时，或文字与背景图像互相重叠的时候。

在代码清单15-2的示例中，文字被显示在图片上面，通过给文字添加阴影的方法使它从背景上突出显示出来。

代码清单15-2 使用阴影突出显示文字的示例

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 ransitional//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>使用阴影突出显示文字示例</title>
</head>
<style type="text/css">
div{
        color: white;
        font-size: 25px;
        font-weight: bold;
        font-family: 宋体;
        background-image: url(sky.jpg);
        width: 140px;
        height: 45px;
        padding: 30px 0;
        text-align: center;
        text-shadow: 3px 3px 5px black;
}
</style>
<body>
<div>你好</div>
</body>
</html>
```

这段代码的运行结果如图15-2所示。

图15-1 text-shadow属性的使用示例

图15-2 使用阴影突出显示文字的示例

15.1.2　位移距离

text-shadow属性所使用的参数中，前两个参数为阴影离开文字的横方向位移距离与纵方向位移距离。使用text-shadow属性时必须要指定这两个参数，可以对这两个参数指定负数值。

将代码清单15-1中示例的div元素的样式指定代码修改为如下所示代码，然后重新运行该示例，则运行结果如图15-3所示。

```
<style type="text/css">
div{
        text-shadow: -15px 10px 5px gray;
        color: navy;
        font-size: 50px;
        font-weight: bold;
        font-family: 宋体;
}
</style>
```

图15-3　给text-shadow属性指定负数参数值

15.1.3　阴影的模糊半径

text-shadow属性所使用的参数中第三个参数是阴影的模糊半径，代表阴影向外模糊时的模糊范围。这个半径的值越大，则阴影向外模糊的范围也就越大。

将代码清单15-1中示例的div元素的样式指定代码修改为如下所示代码，其中阴影的模糊半径从5个像素修改为20个像素，重新运行该示例，则运行结果如图15-4所示。

```
<style type="text/css">
div{
        text-shadow: 5px 5px 20px gray;
        color: navy;
        font-size: 50px;
        font-weight: bold;
        font-family:宋体
}
</style>
```

图15-4　扩大模糊半径示例

模糊半径参数为可选参数，省略这个参数时，该参数默认为0，代表阴影不向外模糊。

将代码清单15-1中示例的div元素的样式指定代码修改为如下所示代码，不对模糊半径参数进行指定，然后重新运行该示例，运行结果如图15-5所示。

```
<style type="text/css">
div{
        text-shadow: 5px 5px gray;
        color: navy;
        font-size: 50px;
        font-weight: bold;
        font-family: 宋体;
}
</style>
```

图15-5　不指定模糊半径示例

15.1.4 阴影的颜色

text-shadow属性所使用的参数中第四个参数是绘制阴影时所使用的颜色，该参数可以放在其他三个参数之后，也可放在其他三个参数之前，成为第一个参数。该参数为可选参数，不对这个参数进行指定时在CSS 2中使用color属性中的颜色，也就是文字颜色，CSS 3中使用浏览器指定的默认色。

通过试验发现在CSS 3中不指定阴影颜色时，在Firefox浏览器与Opera浏览器中使用color属性中的颜色，在Safari浏览器及Chrome浏览器中不支持这个参数的省略，省略该参数时不会对阴影进行绘制。

将代码清单15-1中示例的div元素的样式指定代码修改为如下所示代码，不指定阴影的颜色，则运行结果如图15-6所示。

```
<style type="text/css">
div{
        text-shadow: 5px 5px 5px;
        color: navy;
        font-size: 50px;
        font-weight: bold;
        font-family: 宋体;
}
</style>
```

图15-6 不指定阴影颜色示例

15.1.5 指定多个阴影

可以使用text-shadow属性来给文字指定多个阴影，并且针对每个阴影使用不同颜色。指定多个阴影时使用逗号将多个阴影进行分隔。到目前为止，只有Firefox浏览器、Chrome浏览器及Opera浏览器对这个功能提供支持。

将代码清单15-1中示例的div元素的样式指定代码修改为如下所示代码，在这段代码中，为文字依次指定了桔色、黄色及绿色阴影，同时也为这些阴影指定了适当的位置。

```
<style type="text/css">
div{
        text-shadow: 10px 10px #f39800,
                     40px 35px #fff100,
                     70px 60px #c0ff00;
        color: navy;
        font-size: 50px;
        font-weight: bold;
        font-family: 宋体;
}
</style>
```

将上面这段代码替代到代码清单15-1中，然后重新运行该示例，运行结果如图15-7所示。

图15-7 为文字指定多个阴影的示例

15.2 让文本自动换行——word-break属性

在CSS 3中，使用word-break属性来让文字自动换行。这原来是Internet Explorer中独自发展出来的属性，在CSS 3中被Text模块采用，现在也得到了Chrome浏览器及Safari浏览器的支持。

15.2.1 依靠浏览器让文本自动换行

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的时候，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。图15-8是浏览器对于西方文字进行换行的一个示例。

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，图15-9是当中文当中含有西方文字时，浏览器进行换行的一个示例。

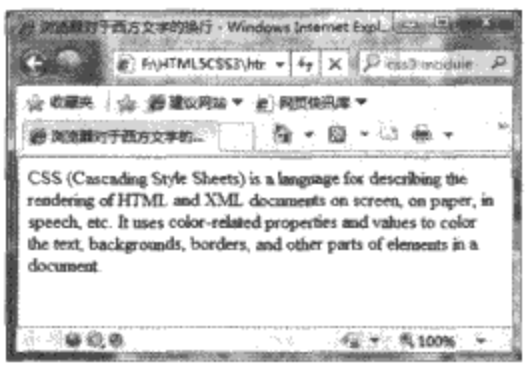

图15-8 InternetExplorer浏览器对于西方文字的换行

图15-9 Internet Explorer浏览器对于中文当中含有西方文字时进行换行的示例

当中文当中含有标点符号的时候，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。譬如图15-10的示例中，第一行文字末尾处的“首”后面是一个逗号，但是为了不让句号处于下一行的行首，浏览器将文字“首”也进行换行。

图15-10 InternetExplorer浏览器对于标点符号的换行

15.2.2 指定自动换行的处理方法

在CSS 3中，可以使用word-break属性来自己决定自动换行的处理方法。通过word-break

属性的指定，不仅仅可以让浏览器实现半角空格或连字符后面的换行，而且可以让浏览器实现任意位置的换行。word-break属性的使用方法类似如下所示。

```
<style type="text/css">
div {
    .   word-break: keep-all;
    }
</style>
```

word-break属性可以使用的值如表15-1所示。

表15-1 word-break属性可以使用的值

值	换行规则	IE 5以上版本浏览器	Safari 3与Google Chrome 6浏览器
normal	使用浏览器默认换行规则	支持	支持
keep-all	只能在半角空格或连字符处换行	支持	不支持
break-all	允许在单词内换行	支持	支持

在Internet Explorer浏览器中，当word-break属性使用keep-all参数值时，对于中文来说，只能在半角空格或连字符或任何标点符号的地方换行，中文与中文之间不能换行，如图15-11所示。

另外，Safari浏览器与Chrome浏览器对keep-all参数值不提供支持。

当word-break属性使用break-all参数值时，对于西方文字来说，允许在单词内换行，如图15-12所示。

图15-11 Internet Explorer浏览器中word-break属性使用keep-all参数值的示例

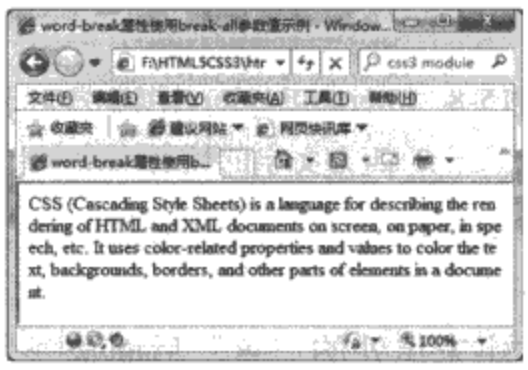

图15-12 Internet Explorer浏览器中word-break属性使用break-all参数值的示例

对于标点符号来说，当word-break属性使用break-all参数值时，在Safari浏览器与Chrome浏览器中，允许标点符号位于行首，如图15-13所示。

而在Internet Explorer浏览器中，当word-break属性使用break-all参数值时，仍然不允许标点符号位于行首。如图15-14所示。

在图15-14中，在第一行结尾处，如果浏览器允许标点符号处于行首的话，结尾处的“中”字应该可以显示在第一行，因为还有一个字的位置，但是因为Internet Explorer浏览器不允许标点符号处于行首，所以将“中”字连同后面的逗号一起在下一行中显示。

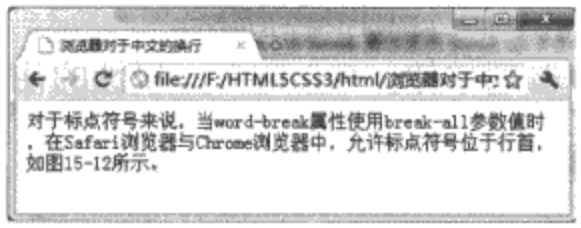

图15-13 Chrome浏览器中使用break-all参数值时对于标点符号的处理

图15-14 Internet Explorer浏览器中使用break-all参数值时对于标点符号的处理

15.3 让长单词与URL地址自动换行——word-wrap属性

对于西方文字来说，浏览器在半角空格或连字符的地方进行换行。因此，浏览器不能给较长的单词自动换行。当浏览器窗口比较窄的时候，文字会超出浏览器的窗口，浏览器下部出现滚动条，让用户通过拖动滚动条的方法来查看没有在当前窗口显示的文字。

但是，这种比较长的单词出现的机会不是很大，而大多数超出当前浏览器窗口的情况是出现在显示比较长的URL地址的时候。因为在URL地址中没有半角空格，所以当URL地址中没有连字符的时候，浏览器在显示时是将其视为一个比较长的单词来进行显示的。

在CSS 3中，使用word-wrap属性来实现长单词与URL地址的自动换行。word-wrap属性的使用方法如下所示。

```
div{
        word-wrap: break-word;
}
```

word-wrap属性可以使用的属性值为normal与break-word两个。使用normal属性值时浏览器保持默认处理，只在半角空格或连字符的地方进行换行。使用break-word时浏览器可在长单词或URL地址内部进行换行，如图15-15所示。

目前，word-wrap属性得到了所有浏览器的支持。

图15-15 使用word-wrap属性将长单词与URL地址强制换行

15.4 使用服务器端字体——Web Font与@font-face属性

在CSS 3之前，页面文字所使用的字体必须已经在客户端中被安装才能正常显示，在样式表中允许指定当前字体不能正常显示时使用的替代字体，但是如果这个替代字体在客户端中也没有安装时，使用这个字体的文字就不能正常显示了。

为了解决这个问题，在CSS 3中，新增了Web Fonts功能，使用这个功能，网页中可以使用安装在服务器端的字体，只要某个字体在服务器端已经安装，网页中就都能够正常显示了。

15.4.1 在网页上显示服务器端字体

在CSS 3中，可以使用@font-face属性来利用服务器端字体。@font-face属性的使用方法如下所示。

```
@font-face{
    font-family: WebFont;
    src: url('font/Fontin_Sans_R_45b.otf') format("opentype");
    font-weight: normal;
}
```

在上面这段代码中，在font-family属性值中使用“WebFont”来声明使用服务器端的字体。

在src属性值中，指定服务器端字体的字体文件所在的路径，在format属性值中声明字体文件的格式，可以省略文件格式的声明而单独使用src属性值。在这段代码中，使用了exljbris字体公司免费提供的Fontin Sans字体，字体文件格式为OpenType格式。到目前为止，可以使用的文件格式为OpenType格式与TrueType文件格式，使用OpenType格式时将format属性值设定为opentype，使用TrueType文件格式时将format属性值设定为truetype，OpenType格式的文件扩展名为“.otf”，TrueType文件格式的扩展名为“.ttf”。另外，在Internet Explorer浏览器中使用服务器端字体的时候，只能使用微软自带的Embedded OpenType字体文件，文件扩展名为“.eot”，同时不需要使用format属性值。下面是在Internet Explorer浏览器中使用服务器端字体时的代码示例。

```
@font-face {
    font-family: BorderWeb;
    src:url(BORDERW0.eot);
}
```

在针对元素使用这个服务器端字体的时候，还需要在元素样式中将font-family属性值指定为WebFont，指定方法类似如下所示。

```
h1{
    font-family: WebFont;
}
```

接下来，我们在代码清单15-3中看一个使用服务器端字体的示例，在该示例中对一个address元素中的字体使用exljbris字体公司免费提供的Fontin Sans字体，在运行该示例前需要下载这个字体文件并且安装在服务器端，下载地址为：http://www.josbuivenga.demon.nl/fontinsans.html。

代码清单15-3 服务器端字体的使用示例

```
<!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>服务器端字体使用示例</title>
</head>
<style type="text/css">
@font-face{
    font-family: WebFont;
```

```
    src: url('Fontin_Sans_R_45b.otf') format("opentype");
    font-weight: normal;
}
h1{
    font-family: WebFont;
    font-size: 60px;
    text-align: center;
    border: solid 1px #4488aa;
    margin: 20px;
    padding: 5px;
}
address{
    font-family: WebFont;
    font-size: 14px;
    font-style: normal;
    text-align: center;
    margin: 20px;
    padding: 5px;
}
</style>
<body>
<h1>Cascading Style Sheets</h1>
<address>
This page uses the
<a href="http://www.josbuivenga.demon.nl/fontinsans.html">
Fontin Sans font by exljbris.
</a>
</address>
</body>
</html>
```

这段代码的运行结果如图15-16所示。

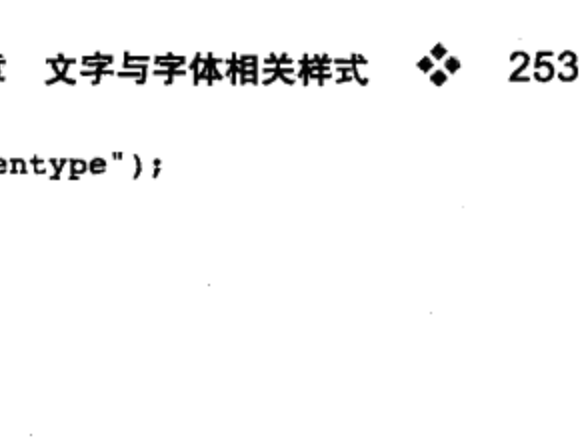

图15-16　服务器端字体的使用示例

15.4.2　定义斜体或粗体字体

在定义字体的时候，可以将字体定义为斜体字或者粗体字。在使用服务器端字体的时候，需要根据是斜体还是粗体，使用不同的字体文件。

在15.4.1节中，示例中使用到的字体为常规字体（不是斜体，不是粗体），在要使用示例中所使用到的字体之前，先要下载字体文件，下载地址在15.4.1节中已给出，这里不做赘述。在下载的压缩文件中，除了有使用Fontin Sans常规字体时所需的字体文件之外，也包含了使用粗体、斜体、粗斜体、小型大写字体时所要用到的字体文件，文件名依次为“Fontin_Sans_B_45b.otf”、“Fontin_Sans_I_45b.otf”、“Fontin_Sans_BI_45b.otf”、“Fontin_Sans_SC_45b.otf”。

接下来，我们在代码清单15-4中给出一个使用服务器端Fontin Sans字体的粗体与斜体文字的示例。在示例中，显示4个div元素，依次对这些div元素使用Fontin Sans字体的常规字体、斜体、粗体、粗斜体。

代码清单15-4 服务器端字体使用粗体与斜体的示例

```
<!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>服务器端字体使用粗体与斜体的示例</title>
</head>
<style type="text/css">
@font-face{
    font-family: WebFont;
    src: url('Fontin_Sans_R_45b.otf') format("opentype");
}
@font-face{
    font-family: WebFont;
    font-style: italic;
    src: url('Fontin_Sans_I_45b.otf') format("opentype");
}
@font-face{
    font-family: WebFont;
    font-weight: bold;
    src: url('Fontin_Sans_B_45b.otf') format("opentype");
}
@font-face{
    font-family: WebFont;
    font-style: italic;
    font-weight: bold;
    src: url('Fontin_Sans_BI_45b.otf') format("opentype");
}
div{
    font-family: WebFont;
    font-size: 40px;
}
div#div1
{
    font-style: normal;
    font-weight: normal;
}
div#div2
{
    font-style: italic;
    font-weight: normal;
}
div#div3
{
    font-style: normal;
    font-weight: bold;
}
div#div4
{
    font-weight: bold;
    font-style: italic;
}
</style>
```

```
<body>
<div id="div1">Text Sample1</div>
<div id="div2">Text Sample2</div>
<div id="div3">Text Sample3</div>
<div id="div4">Text Sample4</div>
</body>
</html>
```

这段代码的运行结果如图15-17所示。

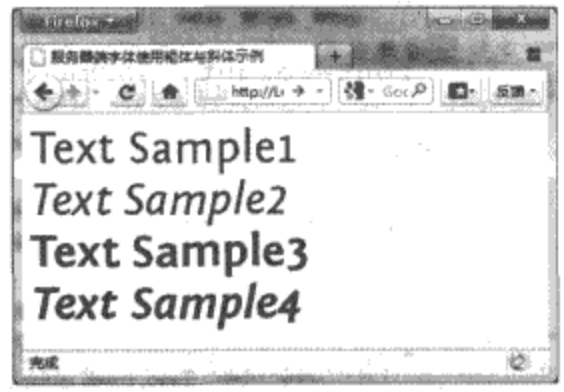

图15-17 服务器端字体使用粗体与斜体的示例

15.4.3 显示客户端本地的字体

@font-face属性不仅可以用于显示服务器端的字体，也可以用来显示客户端本地的字体。

举个例子来说，在客户端本地已经安装了Arial字体系列（font-family）。Arial字体系列由以下几个字体组成。

- Arial
- Arial Italic
- Arial Bold
- Arial Bold Italic
- Arial Black

在浏览器中，当使用Arial字体系列显示文字的正体与粗体时，分别使用Arial字体与Arial Bold字体。但是，可以通过书写样式代码的方式让浏览器在显示粗体时使用Arial Black字体。另外，使用@font-face属性显示客户端本地的字体时，需要将字体文件路径的URL属性值修改为“local()”形式的属性值,并且在“local“后面的括号中写入使用的字体。

代码清单15-5为使用@font-face属性显示客户端字体的一个示例，在该示例中，有两个div元素，分别使用Arial字体系列的常规字体与粗体字体，其中显示粗体字体的时候使用Arial Black字体。

代码清单15-5 使用@font-face属性显示客户端本地的字体示例

```
<!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>使用@font-face属性显示客户端本地的字体示例</title>
</head>
<style type="text/css">
@font-face{
    font-family: Arial;
    src: local('Arial');
}
@font-face
{
    font-family: Arial;
```

```
        font-weight: bold;
        src: local('Arial Black');
    }
    div{
        font-family: Arial;
        font-size: 40px;
    }
    div#div1
    {
        font-weight: normal;
    }
    div#div2
    {
        font-weight: bold;
    }
    </style>
    <body>
    <div id="div1">Text Sample1</div>
    <div id="div2">Text Sample2</div>
    </body>
    </html>
```

这段代码的运行结果如图15-18所示。

图15-18 使用@font-face属性显示客户端本地的字体示例

使用@font-face属性显示客户端本地字体的好处是可以让浏览器在对字体进行显示时首先在客户端本地寻找是否存在该字体，当客户端寻找不到时可以使用服务器端的字体。在如下所示的代码中，浏览器将首先在客户端本地寻找是否存在Helvetica Neue字体，如果存在则直接使用，如果不存在则使用服务器端的MyHelvetica字体。

```
@font-face {
    font-family: MyHelvetica;
    src: local("Helvetica Neue"),
    url(MgOpenModernaRegular.ttf);
}
```

15.4.4 属性值的指定

在@font-face属性中，可以指定的属性值如表15-2所示。

表15-2 @font-face属性中可以指定的属性值

属性值	说明	取值范围
font-family	设置字体系列的名称	
font-style	设置字体的样式	normal：不使用斜体 italic：使用斜体 oblique：使用倾斜体 inherit：从父元素继承

（续）

属性值	说明	取值范围
font-variant	设置字体的大小写	normal：使用浏览器默认值 small-caps：使用小型大写字母 inherit：从父元素继承
font-weight	设置字体的粗细	normal：使用浏览器默认值 bold：使用粗体字符 bolder：使用更粗字符 lighter：使用更细字符 100~900：从细到粗定义字符，使用的值必须为100的整数倍，其中400等同于normal而700等同于bold
font-stretch	设置字体是否伸缩变形	normal：默认值。把缩放比例设置为标准 wider：把伸展比例设置为更进一步的伸展值 narrower：把收缩比例设置为更进一步的收缩值 ultra-condensed extra-condensed condensed semi-condensed semi-expanded expanded extra-expanded ultra-expanded： 设置字体的缩放比例 “ultra-condensed”是最宽的值， “ultra-expanded”是最窄的值
font-size	设置字体大小	
src	设置字体文件的路径	

15.5　修改字体种类而保持字体尺寸不变——font-size-adjust属性

如果改变了字体的种类，则页面中所有使用该字体的文字大小都可能发生变化，从而使得原来安排好的页面布局产生混乱，这是网页设计者最不希望发生的一种状况。

因此，在CSS 3中，针对这种情况，增加了font-size-adjust属性。使用这个属性，可以在保持文字大小不发生变化的情况下改变字体的种类。

15.5.1　字体不同导致文字大小的不同

首先，我们在代码清单15-6中看一个示例，该示例中有7个div元素，每个div元素的字体都设定为16个像素，但是字体全都不一致，导致页面上显示出来的文字大小也不相同。

代码清单15-6 字体不同导致文字大小不同的示例

```
<!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>字体不同导致文字大小不同的示例</title>
</head>
<style type="text/css">
div#div1
{
    font-family: Comic Sans MS;
    font-size:16px;
}
div#div2
{
    font-family:Tahoma;
    font-size:16px;
}
div#div3
{
    font-family:Arial;
    font-size:16px;
}
div#div4
{
    font-family:Times New Roman;
    font-size:16px;
}
</style>
<body>
<div id="div1">Text Sample1</div>
<div id="div2">Text Sample2</div>
<div id="div3">Text Sample3</div>
<div id="div4">Text Sample4</div>
</body>
</html>
```

这段代码的运行结果如图15-19所示。

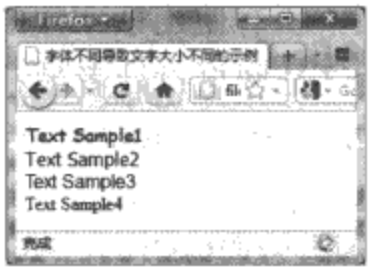

图15-19 字体不同导致文字大小不同的示例

由此可见，如果更改了字体的种类，很可能会因为文字大小的变化而导致原来的页面布局产生混乱。

15.5.2 font-size-adjust属性的使用方法

接下来，我们来看一下如何利用font-size-adjust属性达到修改字体种类而保持文字大小不会发生变化的目的。

font-size-adjust属性的使用方法很简单，但是它需要使用每个字体种类自带的一个aspect值（比例值）。font-size-adjust属性的使用方法类似如下所示，其中0.46为Times New Roman字体的aspect值。

```
div{
        font-size: 16px;
        font-family: Times New Roman,
        font-size-adjust: 0.46;
}
```

aspect值可以用来在将字体修改为其他字体时保持字体大小基本不变。这个aspect值的计算方法为x-height值除以该字体的尺寸，x-height值是指使用这个字体书写出来的小写x的高度（像素为单位）。如果某个字体的尺寸为100px时，x-height值为58像素，则该字体的aspect值为0.58，因为字体的x-height值总是随着字体的尺寸一起改变的，所以字体的aspect值都是一个常数。表15-3所示为一些常用的西方字体的aspect值。

表15-3 常用西方字体的参照值

字体种类	aspect值	字体种类	aspect值
Verdana	0 58	Times New Roman	0 46
Comic Sans MS	0 54	Gill Sans	0 46
Trebuchet MS	0.53	Bernhard Modern	0 4
Georgia	0 5	Caflisch Script Web	0 37
Myriad Web	0 48	Fjemish Script	0 28
Minion Web	0 47		

15.5.3 浏览器对于aspect值的计算方法

在font-size-adjust属性中指定aspect值并且将字体修改为其他字体后，浏览器对于修改后的字体尺寸的计算公式如下所示。

```
c = ( a / b ) s
```

其中，a表示实际使用的字体的aspect值，b表示修改前字体的aspect值，s表示指定的字体尺寸，c为浏览器实际显示时的字体尺寸。

如果想将16px的Times New Roman字体修改为Comic Sans MS字体，字体大小仍然保持16px的Times New Roman字体的大小，则需要执行如下步骤：

1）查得Times New Roman字体的aspect值为0.46。

2）查得Comic Sans MS字体的aspect值为0.54。

3）将0.54除以0.46后得到近似值1.17。

4）因为需要让浏览器实际显示的字体尺寸为16px，所以将16除以1.17，得出大约14px，然后在样式中指定字体尺寸为14px。也就是说，14px的Comic Sans MS相当于16px的Times New Roman字体。

最后要补充说明的是，在实际使用过程中，读者也可以根据需要对aspect值进行微调以达到最满意的效果，也可以将font-size-adjust属性的属性值设为“none”,设定为“none”的意思等同于不对font-size-adjust属性进行设置，按照字体原来的大小显示。

15.5.4 font-size-adjust属性的使用示例

接下来我们在代码清单15-7中看一个示例。在该示例中有三个div元素，其中一个div元素的字体使用Comic Sans MS字体，另两个div元素的字体使用Times New Roman字体。

代码清单15-7 font-size-adjust属性的使用示例

```
<!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>font-size-adjust属性的使用示例</title>
</head>
<style type="text/css">
div#div1{
    font-size: 16px;
    font-family: Comic Sans MS;
    font-size-adjust:0.54;
}
div#div2{
    font-size: 14px;
    font-family: Times New Roman;
    font-size-adjust:0.46;
}
div#div3{
    font-size: 16px;
    font-family: Times New Roman;
    font-size-adjust:0.46;
}
</style>
<body>
<div id="div1">
It is fine today. Never change your plans because of the weather.
</div>
<div id="div2">
It is fine today. Never change your plans because of the weather.
</div>
<div id="div3">
It is fine today. Never change your plans because of the weather.
</div>
</body>
</html>
```

这段代码的运行结果如图15-20所示。

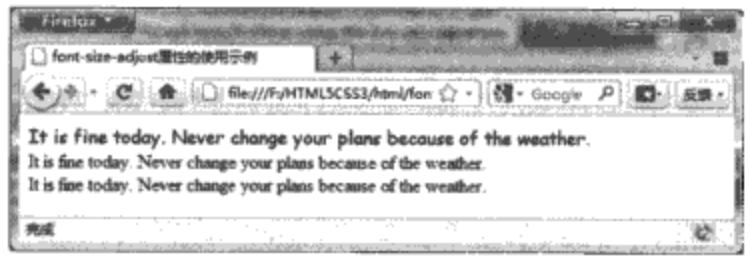

图15-20　font-size-adjust属性的使用示例（未修改字体前）

接下来，我们把第二个div元素的字体从Times New Roman字体修改为Comic Sans MS字体，但是要保持文字大小不变，于是将第二个div元素的字体改为Comic Sans MS字体，字体尺寸改为14px，font-size-adjust属性值微调为0.49。样式代码如下所示。修改后重新运行该示例，运行结果如图15-21所示。

```
div#div2{
    font-size: 14px;
    font-family: Comic Sans MS;
    font-size-adjust:0.49;
}
```

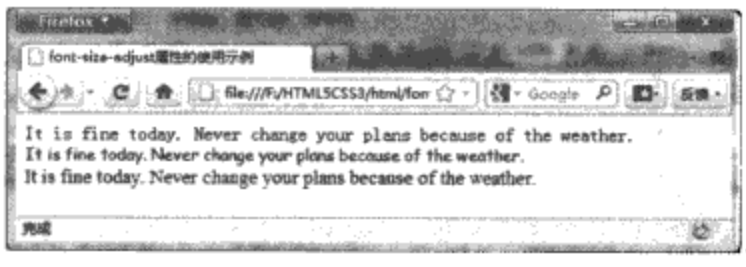

图15-21　font-size-adjust属性的使用示例（修改字体后）

第16章
盒相关样式

16.1 盒的类型

16.2 对于盒中容纳不下的内容的显示

16.3 对盒使用阴影

16.4 指定针对元素的宽度与高度的计算方法

本章将详细介绍CSS 3中各种盒的类型、概念、使用方法及浏览器的支持情况；同时，还将介绍几个属性——当盒中内容超出盒的容纳范围时，可以用它来指定浏览器如何显示这些超出部分；最后将介绍如何使用CSS 3中的属性来给盒添加阴影效果，以及如何使用CSS 3中的属性来定义元素的宽度值和高度值中是否包含内部补白区域，以及边框的宽度和高度。

学习内容：

- 掌握CSS 3中各种各样盒的类型、概念、使用方法及浏览器的支持情况。
- 当盒中内容超出容纳范围时，知道如何利用属性来让浏览器按照自己想要的方式对盒中内容进行正确显示。
- 掌握给盒添加阴影的属性及使用方法，能够使用CSS 3的属性给盒添加阴影效果。
- 掌握几种box-sizing属性值的不同含义，能够正确使用box-sizing属性来定义样式中给定的元素的宽度值和高度值中是否包含内部补白区域，以及边框的宽度和高度。

16.1 盒的类型

16.1.1 盒的基本类型

在CSS中，使用display属性来定义盒的类型。总体上来说，CSS中的盒分为block类型与inline类型。例如，div元素与p元素属于block类型，span元素与a元素属于inline类型。

接下来，我们将block类型与inline类型做一个对比。代码清单16-1中是一个将block类型与inline类型进行对比的示例。该示例中具有两个div元素与两个span元素。为了更容易辨别，我们将div元素的背景设定为绿色，将span元素的背景设定为橘色。从这个示例的运行结果中我们可以看出，div元素所代表的block类型的元素之宽度占满了整个浏览器，而span元素所代表的inline类型的元素之宽度只等于其内容所在的宽度。另外，每一行中只允许容纳一个block类型的元素，但是可以并列容纳多个inline类型的元素。

代码清单16-1 将block类型与inline类型进行对比的示例

```
<!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>将block类型与inline类型进行对比的示例</title>
</head>
<style type="text/css">
div{
        background-color: #aaff00;
}
span{
        background-color: #ffaa00
}
</style>
<body>
<div>div元素</div>
```

```
<div>div元素</div>
<span>span元素</span>
<span>span元素</span>
</body>
</html>
```

这段代码的运行结果如图16-1所示。

在这段代码中，如果样式代码使用display属性，可以将div元素与span元素的类型进行互换，将div元素变成inline类型的元素，将span元素变成block类型的元素，代码如下所示。

```
<style type="text/css">
div{
        background-color: #aaff00;
        display:inline;
}
span{
        background-color: #ffaa00;
        display:block;
}
</style>
```

将这段代码替换到代码清单16-1的示例中，然后重新运行该示例，运行结果如图16-2所示。

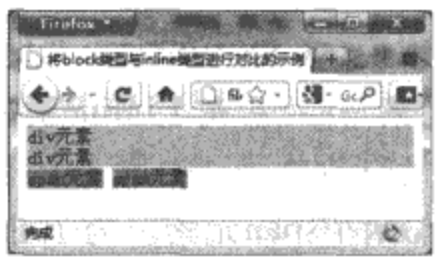

图16-1 将block类型与inline类型进行对比的示例

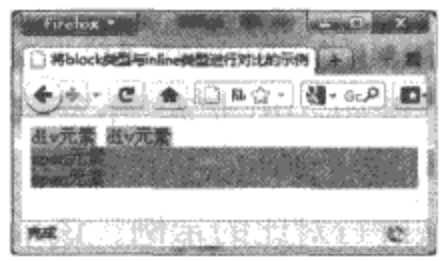

图16-2 将div元素的类型与span元素的类型进行互换

16.1.2 inline-block类型

1. inline-block类型概述

inline-block类型是在CSS 2.1中追加的一个盒类型。到目前为止，它得到了Safari浏览器、Opera浏览器、Chrome浏览器、Firefox浏览器及IE 8以上浏览器的支持。

inline-block类型盒属于block类型盒的一种，但是在显示时它具有inline类型盒的特点。例如，在div元素的样式代码中将display属性设定为“inline-block”，则div元素在显示时与将div元素的display属性设定为“inline”的显示效果相同。

代码清单16-2为inline-block类型div元素的一个示例。在该示例中，具有四个div元素，其中两个被指定为inline-block类型，背景为浅蓝色；另外两个被指定为inline类型，背景为绿色。

代码清单16-2 inline-block类型div元素的示例

```
<!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>inline-block类型div元素的示例</title>
<style type="text/css">
div.inlineblock{
    display: inline-block;
    background-color: #00aaff;
}
div.inline{
    display: inline;
    background-color: #aaff00;
}
</style>
</head>
<body>
<div>
        <div class="inlineblock">inline-block类型</div>
        <div class="inlineblock">inline-block类型</div>
</div>
<div>
        <div class="inline">inline类型</div>
        <div class="inline">inline类型</div>
</div>
</body>
</html>
```

这段代码的运行结果如图16-3所示。

如果对inline-block类型的元素使用width属性或height属性，就能看出它与inline类型的元素的区别了。width属性或height属性分别用来指定元素的宽度与高度，它们只能使用在block类型的元素上。接下来我们将代码清单16-2中的样式代码修改为如下所示的样式代码，在该代码中将设定示例中四个元素的宽度，修改后重新运行该示例。从运行结果中我们可以看出，两个inline-block类型的元素的宽度发生了变化，两个inline类型的元素的宽度没有发生任何变化。

```
<style type="text/css">
div.inlineblock{
    display: inline-block;
    background-color: #00aaff;
    width: 300px;
}
div.inline{
    display: inline;
    background-color: #aaff00;
    width: 300px;
}
</style>
```

这段代码的运行结果如图16-4所示。

2. 使用inline-block类型来执行分列显示

在CSS 2.1之前，如果需要在一行中并列显示多个block类型的元素，则需要使用float属性或position属性，但这样会使样式变得比较复杂。因此，在CSS 2.1中，追加了inline-block

类型，使得并列显示多个block类型元素的操作变得非常简单。

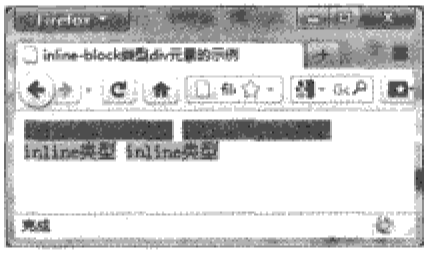

图16-3 inline-block类型div元素的示例

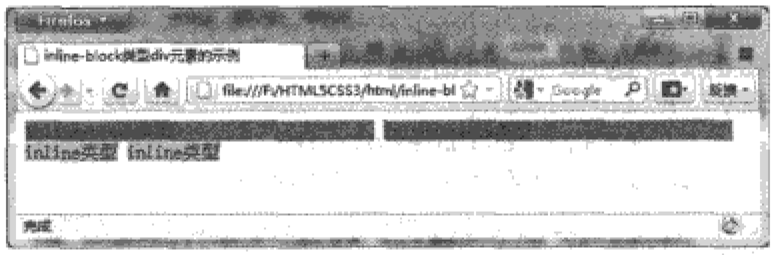

图16-4 指定inline-block类型与inline类型元素的宽度示例

接下来，我们在代码清单16-3中看一个在一行中并列显示多个block类型的元素的示例。在该示例中，具有三个block类型的div元素，使用float类型将前两个div元素并列显示，将第三个div元素显示在前两个div元素的下部，因为前两个元素的高度不一致，所以对第三个元素使用clear属性去除环形围绕方式。

代码清单16-3 使用float属性将div元素并列显示的示例

```
<!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>将div元素并列显示示例</title>
<style type="text/css">
div#a, div#b{
    display: block;
    width: 200px;
    float: left;
}
div#a{
        background-color: #0088ff;
    }
div#b{
        background-color: #00ccff;
    }
div#c{
        width: 400px;
        background-color: #ffff00;
    }
</style>
</head>
<body>
<div id="a">A A A A A A A A A A A A A A A A A A A A A A A A A A A A  A A A A A
A A A A A A A A A A A</div>
<div id="b">B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B</div>
<div id="c">C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C </div>
</body>
</html>
```

这段代码的运行结果如图16-5所示。

但是，如果使用inline-block类型的话，可以直接将两个div元素进行并列显示，不需要使用float属性，同时去除环形围绕方式的clear属性也不需要了。

将代码清单16-3中样式代码修改为如下所示代码，然后重新运行该示例，运行结果如图16-6所示。

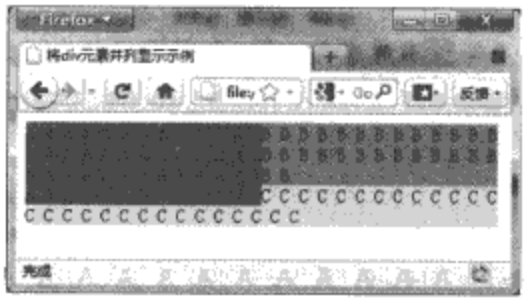

图16-5 使用float属性将div元素并列显示的示例

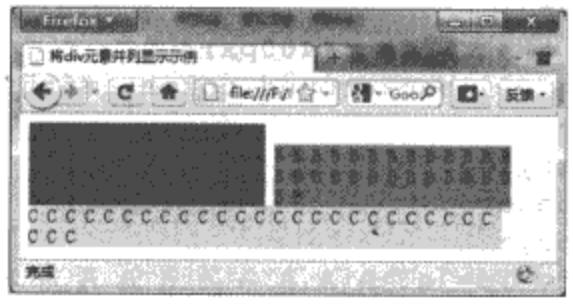

图16-6 使用inline-block类型将div元素并列显示的示例

```
<style type="text/css">
div#a, div#b{
    display: inline-block;
    width: 200px;
}
div#a{
        background-color: #0088ff;
}
div#b{
        background-color: #00ccff;
}
div#c{
        width: 400px;
        background-color: #ffff00;
}
</style>
```

默认情况下使用inline-block类型时并列显示的元素的垂直对齐方式是底部对齐，为了将垂直对齐方式改为顶部对齐，还需要在div元素的样式中加入vertical-align属性。另外，如果要让两个div元素的当中没有缝隙，还需要去除代码中两个div元素之间的换行符。最终修改后的整个页面代码如代码清单16-4所示。

代码清单16-4 使用inline-block类型将div元素并列显示的示例

```
<!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>使用inline-block类型将div元素并列显示的示例</title>
<style type="text/css">
div{
    display: inline-block;
    width: 200px;
    vertical-align:top;
}
div#a{
        background-color: #0088ff;
}
```

```
div#b{
        background-color: #00ccff;
}
div#c{
        width: 400px;
        background-color: #ffff00;
}
</style>
</head>
<body>
<div id="a">A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A</div>
<div id="b">B B B B B B B B B B B B B B B B B B B B B B B B B B B B B</div>
<div id="c"> C C C C C C C C C C C C C C C C C C C C C C C C C C C C C
</div>
</body>
</html>
```

这段代码的运行结果如图16-7所示。

3. 使用inline-block类型来显示水平菜单

在CSS 2.1之前，对于水平菜单的实现也需要使用到float属性。在大多数情况下，水平菜单是利用ul列表与li列表项目来实现的，li元素属于block类型下的list-item类型。因此，如果要让li元素并列显示，需要使用到float属性。

代码清单16-5是一个使用了ul列表与li列表项目来实现水平菜单的示例，在该示例中，使用float属性来让四个列表项目并列显示。

代码清单16-5　使用float属性实现水平菜单

```
<!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=gb2312" />
<title>使用float属性实现水平菜单示例</title>
<style type="text/css">
ul{
        margin: 0;
        padding: 0;
}
li{
        width: 100px;
        padding: 10px 0;
        background-color: #00ccff;
        border: solid 1px #666666;
        text-align: center;
        float: left;
}
a{
        color: #000000;
        text-decoration: none;
}
</style>
</head>
```

```
<body>
<ul>
<li><a href="#">菜单1</a></li>
<li><a href="#">菜单2</a></li>
<li><a href="#">菜单3</a></li>
<li><a href="#">菜单4</a></li>
</ul>
</body>
```

这段代码的运行结果如图16-8所示。

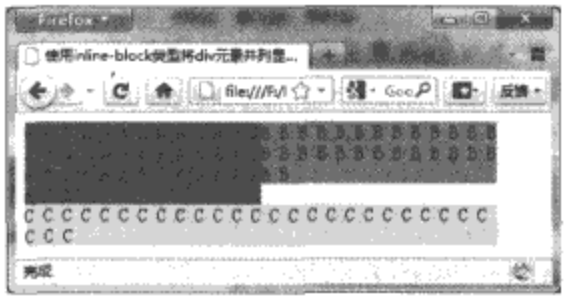

图16-7 使用inline-block类型将div元素并列显示的示例

图16-8 使用float属性实现水平菜单示例

使用inline-block类型同样可以实现代码清单16-5中所实现的水平菜单。同时可以去除列表项目中的“•”标记。使用inline-block类型实现水平菜单的代码如代码清单16-6所示。

代码清单16-6 使用inline-block类型实现水平菜单

```
<!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=gb2312" />
<title>使用inline-block类型实现水平菜单的示例</title>
<style type="text/css">
ul{
        margin: 0;
        padding: 0;
}
li{
        display:inline-block;
        width: 100px;
        padding: 10px 0;
        background-color: #00ccff;
        border: solid 1px #666666;
        text-align: center;
}
a{
        color: #000000;
        text-decoration: none;
}
</style>
</head>
<body>
<ul>
```

```
<li><a href="#">菜单1</a></li><li><a href="#">菜单2</a></li><li>
<a href="#">菜单3</a></li><li><a href="#">菜单4</a></li>
</ul>
</body>
</html>
```

这段代码的运行结果如图16-9所示。

另外，还可以让a元素也属于inline-block类型，然后使用背景色，并且指定宽度，使a元素占据整个菜单。将代码清单16-6中的样式代码修改成如下所示的代码，然后重新运行该示例，运行结果如图16-10所示。

```
<style type="text/css">
ul{
        margin: 0;
        padding: 0;
}
li{
        display: inline-block;
        background-color: #00ccff;
        border: solid 1px #666666;
        text-align: center;
}
a{
        color: #000000;
        text-decoration: none;
        background-color: #ffcc00;
        display: inline-block;
        width: 100px;
        padding: 10px 0;
}
</style>
```

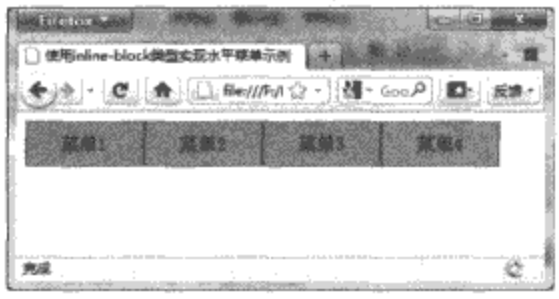

图16-9 使用inline-block类型实现水平菜单的示例

图16-10 使用inline-block类型实现水平菜单的示例

16.1.3 inline-table类型

接下来，我们介绍CSS 2中新增的另外一种盒类型——inline-table类型。到目前为止，该类型得到了Safari浏览器、Opera浏览器、Chrome浏览器、Firefox浏览器及Internet Explorer8以上浏览器的支持。

首先，我们在代码清单16-7中看一个CSS中使用table元素的示例，该示例中有一个表格，表格之前与之后都有一些文字将这个表格围绕。

代码清单16-7 CSS中使用table元素的示例

```
<!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>inline-table类型的使用示例</title>
<style type="text/css">
table{
        border: solid 3px #00aaff;
}
td{
        border: solid 2px #ccff00;
        padding: 5px;
}
</style>
</head>
<body>
你好
<table>
<tr>
        <td>A</td><td>B</td><td>C</td><td>D</td><td>E</td>
</tr>
<tr>
        <td>F</td><td>G</td><td>H</td><td>I</td><td>J</td>
</tr>
<tr>
        <td>K</td><td>L</td><td>M</td><td>N</td><td>O</td>
</tr>
</table>
你好
</body>
</html>
```

这段代码的运行结果如图16-11所示。

在代码清单16-7的运行结果中，表格前后的文字都处于不同行中，因为table元素属于block类型，所以不能与其他文字处于同一行中，但是如果将table元素修改成inline-table类型，就可以让表格与其他文字处于同一行中了。

将代码清单16-7中针对table元素指定的样式代码修改为如下所示的样式代码，然后重新运行该示例，运行结果如图16-12所示。

```
table{
        display: inline-table;
        border: solid 3px #00aaff;
}
```

但是在各个浏览器中，对于文字与表格的垂直对齐方式并不完全相同。在Safari浏览器及Chrome浏览器中，垂直对齐方式为底部对齐，在Internet Explorer浏览器、Opera浏览器及Firefox浏览器中，垂直对齐方式为顶部对齐。图16-13为在Chrome浏览器中inline-table类型元素的垂直对齐方式。

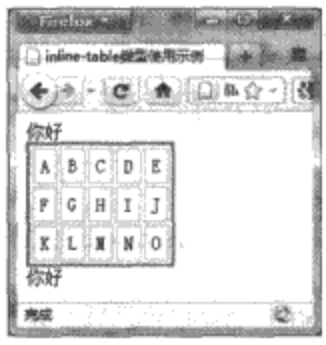

图16-11 CSS中使用table元素的示例

图16-12 inline-table类型的使用示例

可以在样式中显式指定表格与文字的对齐方式，譬如将代码清单16-7中针对table元素指定的样式代码修改为如下所示的样式代码，然后重新运行该示例，运行结果中表格与文字的垂直对齐方式将被强制设定为底部对齐，如图16-14所示。

```
table{
        display: inline-table;
        border: solid 3px #00aaff;
        vertical-align: bottom;
}
```

图16-13 在Chrome浏览器中inline-table类型元素的垂直对齐方式

图16-14 显式指定inline-table类型的元素的垂直对齐方式为底部对齐

16.1.4 list-item类型

如果在display属性中将元素的类型设定为list-item类型，可以将多个元素作为列表来显示，同时在元素的开头加上列表的标记。

代码清单16-8为list-item类型的一个使用示例，在该示例中，具有多个div元素，使用display属性将这些div元素的类型设定为list-item类型，使用list-style-type属性将列表标记设定为circle，最终显示页面中的div元素以列表形式呈现，列表标记为一个空心小圆圈。

代码清单16-8 list-item类型的使用示例

```
<!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>list-item类型使用示例</title>
<style type="text/css">
```

```
div{
        display:list-item;
        list-style-type:circle;
        margin-left:30px;
}
</style>
</head>
<body>
<div>示例div1</div>
<div>示例div2</div>
<div>示例div3</div>
<div>示例div4</div>
</body>
</html>
```

这段代码的运行结果如图16-15所示。

图16-15 list-item类型的使用示例

16.1.5 run-in类型与compact类型

将元素指定为run-in类型或compact类型的时候，如果元素后面还有block类型的元素，run-in类型的元素将被包含在block类型的元素内部，而compact类型的元素将被放置在block类型的元素左边。

代码清单16-9为run-in类型与compact类型结合使用的一个示例。示例中有两个名词解释列表用的dl元素，它会将需要解释的名词用红色框框出，将第一个dl元素中的dt元素指定为run-in类型，第二个dl元素中的dt元素指定为compact类型。将两个dl元素中的dd元素的背景色均设定为黄色。

另外，到目前为止，run-in类型只被Opera浏览器与Safari浏览器所支持，compact类型只被Opera浏览器所支持。另外，compact类型在CSS 2.1中被删除了，在CSS 3中又被恢复了。

代码清单16-9 run-in类型与compact类型的使用示例

```
<!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>run-in类型与compact类型使用示例</title>
<style type="text/css">
dl#runin dt{
```

```
        display: run-in;
        border: solid 2px red;
    }
    dl#compact dt{
        display: compact;
        border: solid 2px red;
    }
    dd{
        margin-left: 100px;
        background-color: yellow;
    }
    </style>
    </head>
    <body>
    <dl id="runin">
    <dt>名词一</dt>
    <dd>关于"名词一"的名词解释。</dd>
    </dl>
    <dl id="compact">
    <dt>名词二</dt>
    <dd>关于"名词二"的名词解释。</dd>
    </dl>
    </body>
    </html>
```

这段代码在Opera浏览器中的运行结果如图16-16所示。

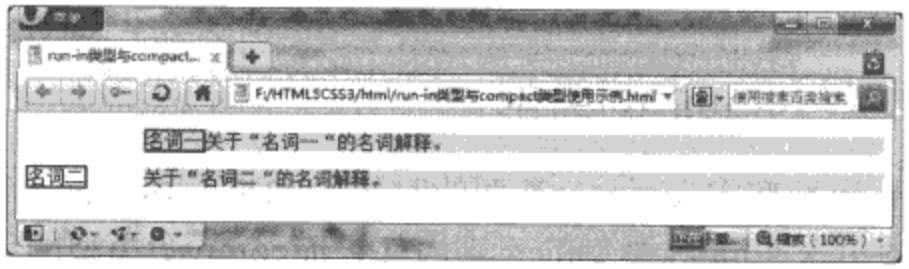

图16-16 Opera浏览器中的run-in类型与compact类型的使用示例

16.1.6 表格相关类型

在CSS 3中所有与表格相关的元素及其所属类型如表16-1所示。

表16-1 CSS 3中所有与表格相关的元素及其所属类型

元素	所属类型	说 明
table	table	代表整个表格
table	inline-table	代表整个表格，可以被指定为table类型，也可以被指定为inline-table类型
tr	table-row	代表表格中的一行
td	table-cell	代表表格中的单元格
th	table-cell	代表表格中的列标题
tbody	table-row-group	代表表格中所有行
thead	table-header-group	代表表格中的表头部分

（续）

元素	所属类型	说 明
tfoot	table-footer-group	代表表格中的脚注部分
col	table-column	代表表格中的一列
colgroup	table-column-group	代表表格中所有列
caption	table-caption	代表整个表格的标题

代码清单16-10为CSS 3中一个完整表格的构成示例，在该示例中，通过将许多div元素的类型指定为表格相关各种类型，使这些div元素共同构成了一个完整的表格。

代码清单16-10 CSS 3中完整表格的构成示例

```
<!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>CSS 3中完整表格的构成示例</title>
<style type="text/css">
.table{
        display: table;
        border: solid 3px #00aaff;
}
.caption{
        display: table-caption;
        text-align: center;
}
.tr{
        display: table-row
}
.td {
        display: table-cell;
        border: solid 2px #aaff00;
        padding: 10px;
}
.thead{
        display: table-header-group;
        background-color: #ffffaa;
}
</style>
</head>
<body>
<div class="table">
    <div class="caption">字母表</div>
    <div class="thead">
        <div class="tr">
            <div class="td">1st</div>
            <div class="td">2nd</div>
            <div class="td">3rd</div>
            <div class="td">4th</div>
            <div class="td">5th</div>
        </div>
```

```
        </div>
        <div class="tr">
            <div class="td">A</div>
            <div class="td">B</div>
            <div class="td">C</div>
            <div class="td">D</div>
            <div class="td">E</div>
        </div>
        <div class="tr">
            <div class="td">F</div>
            <div class="td">G</div>
            <div class="td">H</div>
            <div class="td">I</div>
            <div class="td">J</div>
        </div>
    </div>
    </body>
    </html>
```

这段代码的运行结果如图16-17所示。

16.1.7 none类型

将元素的类型指定为none类型后，该元素将不会被显示。

代码清单16-11为none类型的一个使用示例，示例中有四个div元素，分别显示四段文字，通过none类型的指定，使其中两段文字不被显示。

代码清单16-11 none类型的使用示例

```
<!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>none类型的使用示例</title>
<style type="text/css">
.none{
        display: none;
}
</style>
</head>
<body>
<div>示例文字1</div>
<div class="none">示例文字2</div>
<div class="none">示例文字3</div>
<div>示例文字4</div>
</body>
</html>
```

这段代码的运行结果如图16-18所示。

图16-17 CSS 3中完整表格的构成示例

图16-18 none类型的使用示例

16.1.8 各种浏览器对于各种盒类型的支持情况

最后，我们总结一下到目前为止各种浏览器对于CSS 3中各种盒类型的支持情况，如表16-2所示。

表16-2 各种浏览器对于CSS 3中各种盒类型的支持情况

盒类型	Firefox	Safari	Opera	Internet Explorer 8	Chrome
inline	✓	✓	✓	✓	✓
block	✓	✓	✓	✓	✓
inline-block	✓	✓	✓	✓	✓
list-item	✓	✓	✓	✓	✓
run-in	×	✓	✓	×	×
compact	×	×	✓	×	×
table	✓	✓	✓	✓	✓
inline-table	✓	✓	✓	✓	✓
table-row	✓	✓	✓	✓	✓
table-cell	✓	✓	✓	✓	✓
table-row-group	✓	✓	✓	✓	✓
table-header-group	✓	✓	✓	✓	✓
table-footer-group	✓	✓	✓	✓	✓
table-column	✓	✓	×	✓	×
table-column-group	✓	✓	×	✓	×
table-caption	✓	✓	✓	✓	✓
ruby	×	×	×	✓	×
ruby-base	×	×	×	✓	×
ruby-text	×	×	×	✓	×
none	✓	✓	✓	✓	✓

16.2 对于盒中容纳不下的内容的显示

如果在样式中指定了盒的宽度与高度，就有可能出现某些内容在盒中容纳不下的情况，

可以使用overflow属性来指定如何显示盒中容纳不下的内容。同时，与overflow属性相关的还有overflow-x属性、overflow-y属性及text-overflow属性，这几个属性原本是Internet Explorer浏览器独自发展出来的属性，由于在CSS 3中被采用，因而得到了其他浏览器的支持。

16.2.1 overflow属性

在CSS 3中，可以使用overflow属性来指定对于盒中容纳不下的内容的显示方法。这个属性是在CSS 2中定义的属性，目前得到了Firefox、Safari、Opera、Internet Explorer、Chrome浏览器的支持。

例如，在代码清单16-12中，div元素内的文字超出了div元素的容纳范围。

代码清单16-12 内容超出元素容纳范围的示例

```
<!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>overflow属性使用示例</title>
<style type="text/css">
div{
        width: 300px;
        height: 150px;
        border: solid 1px orange;
}
</style>
</head>
<body>
<div>
<h1>标题文字</h1>
<p>示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。
示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。
示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。
</p>
<p>示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。
示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。
示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。
</p>
</div>
</body>
</html>
```

这段代码的运行结果如图16-19所示。

这时，如果在div元素的样式代码中加入overflow属性，并且将属性值设定为hidden，则超出容纳范围的文字将被隐藏起来。div元素修改后的样式代码如下所示。

```
<style type="text/css">
div{
        overflow:hidden;
        width: 300px;
        height: 150px;
```

```
        border: solid 1px orange;
}
</style>
```

将这段样式代码替换到代码清单16-12中，然后重新运行该示例，运行结果如图16-20所示。

图16-19 内容超出元素容纳范围的示例

图16-20 overflow属性使用hidden属性值的示例

如果将overflow属性的属性值设定为scroll，则div元素中出现固定的水平滚动条与垂直滚动条，文字超出div元素的容纳范围时将滚动显示。div元素修改后的样式代码如下所示。

```
<style type="text/css">
div{
        overflow:scroll;
        width: 300px;
        height: 150px;
        border: solid 1px orange;
}
</style>
```

将这段样式代码替换到代码清单16-12中，然后重新运行该示例，运行结果如图16-21所示。

如果将overflow属性的属性值设定为auto，则当文字超出div元素的容纳范围时根据需要出现水平滚动条或垂直滚动条，并且滚动显示超出容纳范围的内容。div元素修改后的样式代码如下所示。

```
<style type="text/css">
div{
        overflow:auto;
        width: 300px;
        height: 150px;
        border: solid 1px orange;
}
</style>
```

将这段样式代码替换到代码清单16-12中，然后重新运行该示例，运行结果如图16-22所示。

图16-21　overflow属性使用scroll属性值的示例

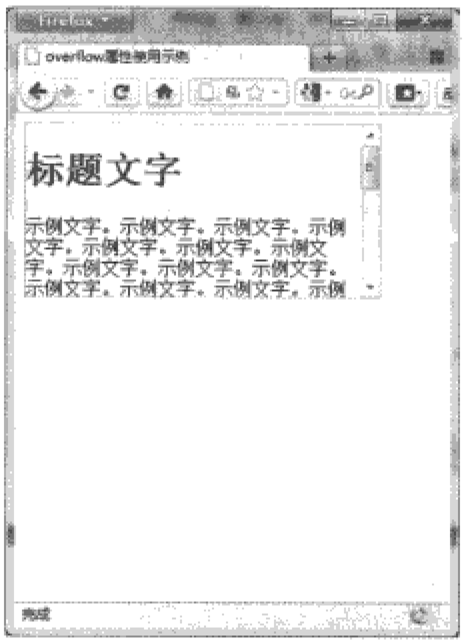

图16-22　overflow属性使用auto属性值的示例

如果将overflow属性的属性值设定为visible，则显示效果与不使用overflow属性时一样，超出容纳范围的文字依原样显示。div元素修改后的样式代码如下所示。

```
<style type="text/css">
div{
        overflow:visible;
        width: 300px;
        height: 150px;
        border: solid 1px orange;
}
</style>
```

将这段样式代码替换到代码清单16-12中，然后重新运行该示例，运行结果如图16-23所示。

图16-23　overflow属性使用visible属性值的示例

16.2.2 overflow-x属性与overflow-y属性

如果使用overflow-x属性或overflow-y属性，可以单独指定在水平方向上或垂直方向上内容超出盒的容纳范围时的显示方法。

例如将代码清单16-12中div元素的样式代码修改成如下所示的样式代码，将overflow-x属性设定为hidden，将overflow-y属性设定为scroll，则只显示垂直方向上的滚动条。

```
<style type="text/css">
div{
        overflow-x:hidden;
        overflow-y:scroll;
        width: 300px;
        height: 150px;
        border: solid 1px orange;
}
</style>
```

将这段样式代码替换到代码清单16-12中，然后重新运行该示例，运行结果如图16-24所示。

图16-24 overflow-x属性与overflow-y属性的使用示例

overflow-x属性与overflow-y属性原本是Internet Explorer浏览器中独自扩展出来的属性，后被CSS 3所采用，并被标准化。到目前为止，得到Internet Explorer浏览器、Firefox浏览器、Safari 3以上浏览器及Opera 10以上浏览器的支持。

16.2.3 text-overflow属性

当通过把overflow属性的属性值设定为“hidden”的方法，将盒中容纳不下的内容隐藏起来时，如果使用text-overflow属性，可以在盒的末尾显示一个代表省略的符号“...”。但是，text-overflow属性只在当盒中的内容在水平方向上超出盒的容纳范围时有效。

text-overflow属性目前得到了IE 6以上浏览器、Safari浏览器及Opera浏览器的支持。

例如代码清单16-13中，通过将white-space属性的属性值设定为nowrap，使得盒右端的内容不能换行显示，这样一来，盒中的内容就在水平方向上溢出了。

代码清单16-13 盒中的内容在水平方向上溢出的示例

```
<!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>text-overflow属性使用示例</title>
<style>
div{
        white-space: nowrap;
        width: 300px;
        border: solid 1px orange;
}
```

```
</style>
</head>
<body>
<div>
这是一句非常非常非常非常非常非常非常非常非常非常非常非常长的例句。
</div>
</body>
</html>
```

这段代码在Internet Explorer浏览器中的运行结果如图16-25所示。

这时，如果在div元素的样式中加入overflow属性，并且指定overflow属性的属性值为“hidden”，超出div元素部分的文字将会被隐藏起来，修改后的样式代码如下所示。

```
<style type="text/css">
div{
        overflow:hidden;
        white-space: nowrap;
        width: 300px;
        border: solid 1px orange;
}
</style>
```

将这段样式代码替换到代码清单16-13中，然后重新运行该示例，运行结果如图16-26所示。

图16-25 盒中的内容在水平方向上溢出的示例（在Internet Explorer浏览器中）

图16-26 overflow属性使用hidden属性值的示例（在Internet Explorer浏览器中）

如果在此基础上使用text-overflow属性，并且将属性值设定为“ellipsis”，则会在div元素的末尾出现一个省略号。另外，在样式代码中书写text-overflow属性的时候，在Internet Explorer浏览器中需要书写成“text-overflow”，在Safari浏览器中书写成“-webkit-text-overflow”，在Opera浏览器中书写成“-o-text-overflow”。在代码清单16-13中div元素的样式代码中加入text-overflow属性后的样式代码如下所示。

```
<style type="text/css">
div{
        overflow:hidden;
        text-overflow: ellipsis;
        -webkit-text-overflow: ellipsis;
        -o-text-overflow: ellipsis;
        white-space: nowrap;
        width: 300px;
        border: solid 1px orange;
}
</style>
```

将这段样式代码替换到代码清单16-13中，然后重新运行该示例，运行结果如图16-27所示。

图16-27　text-overflow属性使用示例（在Internet Explorer浏览器中）

另外，text-overflow属性也是由Internet Explorer浏览器扩展出来的、被CSS 3所采纳的一个属性。

16.3　对盒使用阴影

16.3.1　box-shadow属性的使用方法

在CSS 3中，可以使用box-shadow属性让盒在显示时产生阴影效果。到目前为止，该属性得到了Safari浏览器及Firefox浏览器的支持。使用Safari浏览器时，需要将样式代码书写成“-webkit-box-shadow”的形式；使用Firefox浏览器时，需要将样式代码书写成“-moz-box-shadow”的形式。box-shadow属性的指定方式如下所示。

```
box-shadow: length length length color
```

其中，前面三个length分别指阴影离开文字的横向距离、阴影离开文字的纵向距离和阴影的模糊半径，color指阴影的颜色。

代码清单16-14为box-shadow属性的一个使用示例。在该示例中，对一个橘色盒使用了灰色阴影。box-shadow属性中的前三个参数均设为10个像素。

代码清单16-14　box-shadow属性使用示例

```
<!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>box-shadow属性使用示例</title>
<style type="text/css">
div{
        background-color: #ffaa00;
        -moz-box-shadow: 10px 10px 10px gray;
        -webkit-box-shadow:10px 10px 10px gray;
        width:200px;
        height:100px;
}
</style>
</head>
<body>
```

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

这段代码的运行结果如图16-28所示。

图16-28 box-shadow属性使用示例

16.3.2 将参数设定为0

把阴影的模糊半径设定为0时，将绘制不向外模糊的阴影。将代码清单16-14中的样式代码修改为如下所示的样式代码，然后重新运行该示例，运行结果如图16-29所示。

```
<style type="text/css">
div{
        background-color: #ffaa00;
        -moz-box-shadow: 10px 10px 0 gray;
        -webkit-box-shadow:10px 10px 0 gray;
        width:200px;
        height:100px;
}
</style>
```

将阴影离开文字的横向距离与阴影离开文字的纵向距离均设定为0时，将在盒的周围绘制阴影。将代码清单16-14中的样式代码修改为如下所示的样式代码，然后重新运行该示例，运行结果如图16-30所示。

图16-29 将阴影的模糊半径设定为0

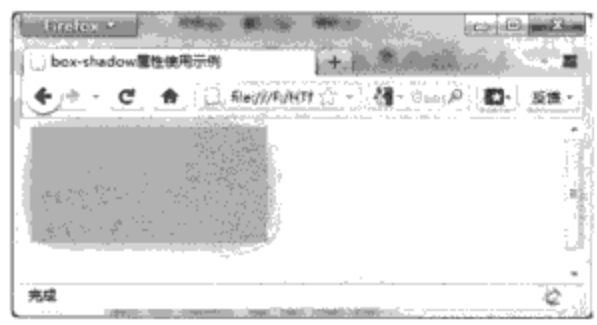

图16-30 将阴影离开文字的横向距离与阴影离开文字的纵向距离均设定为0

```
<style type="text/css">
div{
        background-color: #ffaa00;
        -moz-box-shadow: 0px 0px 50px gray;
        -webkit-box-shadow:0px 0px 50px gray;
        width:200px;
```

```
        height:100px;
}
</style>
```

可以将阴影离开文字的横向距离或阴影离开文字的纵向距离设定为负数值：将阴影离开文字的横向距离设定为负数值时，向左绘制阴影；将阴影离开文字的纵向距离设定为负数值时，向上绘制阴影。将代码清单16-14中的样式代码修改为如下所示的样式代码，然后重新运行该示例，运行结果如图16-31所示。

```
<style type="text/css">
div{
        background-color: #ffaa00;
        -moz-box-shadow: -10px -10px 10px gray;
        -webkit-box-shadow:-10px -10px 10px gray;
        width:200px;
        height:100px;
}
</style>
```

图16-31 将阴影离开文字的横向距离与阴影离开文字的纵向距离均设定为负数值

16.3.3 对盒内子元素使用阴影

可以单独对盒内的子元素使用阴影。代码清单16-15是对盒内的子元素使用阴影的一个示例，该示例中具有一个div元素，div元素内部有一个span子元素，使用box-shadow属性可以让span子元素具有阴影效果。

代码清单16-15 对盒内子元素使用阴影的示例

```
<!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>对盒内子元素使用阴影的示例</title>
<style type="text/css">
span{
      background-color: #ffaa00;
      -moz-box-shadow: 10px 10px 10px gray;
}
span{
      background-color: #ffaa00;
      -webkit-box-shadow:10px 10px 10px gray;
```

```
}
</style>
</head>
<body>
<div>示例文字示例文字示例文字示例文字示例文字<span>示例文字示例文字示例文字示例文字示
例文字示例文字示例文字示例文字示例文字示例文字</span>示例文字示例文字示例文字示例文字
示例文字</div>
</body>
</html>
```

这段代码的运行结果如图16-32所示。

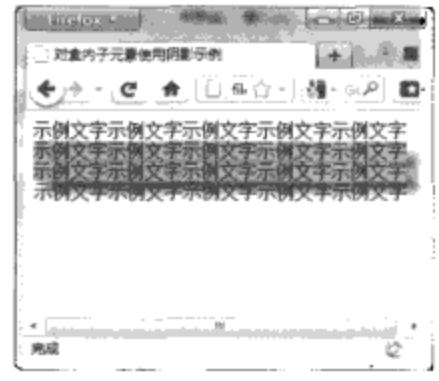

图16-32 对盒内子元素使用阴影的示例

16.3.4 对第一个文字或第一行使用阴影

可以使用first-letter选择器或first-line选择器来只让第一个文字或第一行具有阴影效果，代码清单16-16是一个只让第一个文字具有阴影效果的示例。该示例中具有一个div元素，元素内有一些文字，使用first-letter选择器来只让第一个文字具有阴影效果。

代码清单16-16 对第一个文字使用阴影的示例

```
<!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>对第一个文字使用阴影的示例</title>
<style type="text/css">
div:first-letter{
        font-size: 22px;
        float: left;
        background-color: #ffaa00;
        -webkit-box-shadow: 5px 5px 5px gray;
}
div:first-letter{
        font-size: 22px;
        float: left;
        background-color: #ffaa00;
        -moz-box-shadow: 5px 5px 5px gray;
}
</style>
</head>
<body>
<div>示例文字</div>
</body>
</html>
```

这段代码的运行结果如图16-33所示。

图16-33　对第一个文字使用阴影的示例

16.3.5　对表格及单元格使用阴影

可以使用box-shadow属性让表格及表格内的单元格产生阴影效果。代码清单16-17是使用box-shadow属性让表格及单元格产生阴影效果的一个示例。该示例中有一个表格，表格内有一些数字，使用box-shadow属性让表格及单元格都产生了阴影，使得表格看起来具有数字面板的视觉效果。

代码清单16-17　使用box-shadow属性让表格及单元格产生阴影效果的示例

```
<!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>使用box-shadow属性让表格及单元格产生阴影效果的示例</title>
<style type="text/css">
table{
        border-spacing:10px;
        -webit-box-shadow:5px 5px 20px gray;
        -moz-box-shadow:5px 5px 20px gray;
}
td{
        background-color: #ffaa00;
        -webit-box-shadow:5px 5px 5px gray;
        -moz-box-shadow:5px 5px 5px gray;
        padding:10px;
}
</style>
</head>
<body>
<table>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
```

```
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
</tr>
</table>
</body>
</html>
```

这段代码的运行结果如图16-34所示。

图16-34 使用box-shadow属性让表格及单元格产生阴影效果的示例

16.4 指定针对元素的宽度与高度的计算方法

在CSS 3中，使用box-sizing属性来指定针对元素的宽度与高度的计算方法。到目前为止，Firefox 4浏览器、Opera 10浏览器、Safari 3浏览器、Google Chrome 8浏览器与IE 8浏览器都对这个属性提供了支持，本节将针对这个属性做详细介绍。

16.4.1 box-sizing属性

在CSS中，使用width属性与height属性来指定元素的宽度与高度。但是使用box-sizing属性，可以指定用width属性与height属性分别指定的宽度值与高度值是否包含元素内部的补白区域，以及边框的宽度与高度。

可以给box-sizing属性指定的属性值为content-box属性值与border-box属性值。content-box属性值表示元素的宽度与高度不包括内部补白区域，以及边框的宽度与高度，border-box属性值表示元素的宽度与高度包括内部补白区域，以及边框的宽度与高度，在没有使用box-sizing属性的时候，默认使用content-box属性值。在样式代码中，使用Firefox浏览器的时候，需要将其书写成“-moz-box-sizing”的形式；使用Safari浏览器或Chrome浏览器的时候，需要书写成“-webkit-box-sizing”的形式；使用Opera浏览器的时候，需要书写成“box-sizing”的形式；使用Internet Explorer浏览器的时候，需要书写成“-ms-box-sizing”的形式。

代码清单16-18中的示例可以很直观地说明这两个属性值得的区别。在该示例中存在两个div元素，在第一个div元素的box-sizing属性中指定content-box属性值，在第二个div元素的box-sizing属性中指定border-box属性值，通过在浏览器中的运行结果我们可以很直观地看

出这两个属性值的区别。

代码清单16-18 box-sizing属性的使用示例

```
<!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>box-sizing属性使用示例</title>
</head>
<style type="text/css">
div{
        width: 300px;
        border: solid 30px #ffaa00;
        padding: 30px;
        background-color: #ffff00;
        margin: 20px auto;
}
div#div1{
        box-sizing: content-box;
        -webkit-box-sizing: content-box;
        -moz-box-sizing: content-box;
        -ms-box-sizing: content-box;
}
div#div2{
        box-sizing: border-box;
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        -ms-box-sizing: border-box;
}
</style>
<body>
<div id="div1">
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
</div>
<div id="div2">
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
</div>
</body>
</html>
```

代码清单16-18中示例的运行结果如图16-35所示。

在这个示例中，虽然同时指定了两个div元素的宽度都是300px，但是在第一个元素的box-sizing属性中指定了content-box属性值，所以元素内容部分的宽度为300px，元素的总宽度为：元素内容宽度300px+内部补白宽度30px × 2+边框宽度30px × 2=420px；第二个元素的box-sizing属性中指定了border-box属性值，所以元素的总宽度为300px，元素内容部分的宽度=元素总宽度300px-内部补白宽度30px × 2-边框宽度30px × 2=180px。

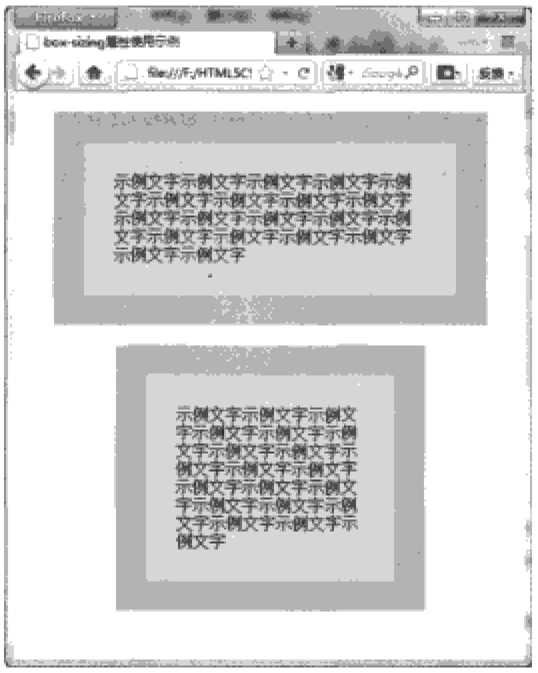

图16-35 box-sizing属性的使用示例

另外，在Firefox浏览器中，还可以在box-sizing属性中指定属性值为padding-box，意思是指定的宽度与高度包括内容的宽度与高度和内部补白区域的宽度与高度，不包括边框的宽度与高度。将代码清单16-18中的代码修改为代码清单16-19中的所示的代码，在该示例的两个div元素的后面追加一个div元素，并且指定该元素的box-sizing属性的属性值为padding-box，请在表示示例运行结果的图16-36中观察第三个div元素的宽度与前两个div元素的宽度有什么区别。

代码清单16-19 三种box-sizing属性值的使用示例

```
<!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>三种box-sizing属性值使用示例</title>
</head>
<style type="text/css">
div{
        width: 300px;
        border: solid 30px #ffaa00;
        padding: 30px;
        background-color: #ffff00;
        margin: 20px auto;
}
div#div1{
        box-sizing: content-box;
        -webkit-box-sizing: content-box;
        -moz-box-sizing: content-box;
        -ms-box-sizing: content-box;
}
div#div2{
        box-sizing: border-box;
```

```
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        -ms-box-sizing: border-box;
}
div#div3{
        -moz-box-sizing:padding-box;
}
</style>
<body>
<div id="div1">
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
</div>
<div id="div2">
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
</div>
<div id="div3">
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
</div>
</body>
</html>
```

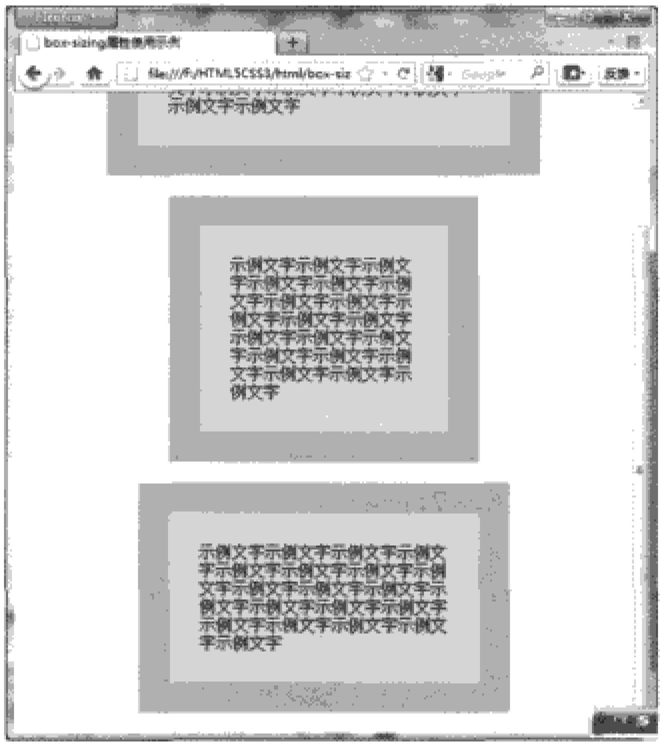

图16-36　三种box-sizing属性值的使用示例

16.4.2　为什么要使用box-sizing属性

使用box-sizing属性的目的是控制元素的总宽度，如果不使用该属性，样式中默认使用的是content-box属性值，它只对内容的宽度做了一个指定，却没有对元素的总宽度进行指定。有些场合下利用border-box属性值会使得页面布局更加方便。譬如代码清单16-20中的示例，只要将两个div元素的border-box属性值都设定为50%，就可以确保两个div元素的并列显示了。

代码清单16-20 确保两个div元素的并列显示

```
<!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=gb2312" />
<title>确保两个div元素的并列显示</title>
</head>
<style>
div{
        width: 50%;
        border: solid 30px #ffaa00;
        padding: 30px;
        background-color: #ffff00;
        float: left;
        -moz-box-sizing: border-box;
        -webkit-box-sizing: border-box;
        -ms-box-sizing: border-box;
        box-sizing: border-box;
}
div#div2{
    border: solid 30px  #00ffff;
}
</style>
<body>
<div id="div1">
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
</div>
<div id="div2">
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字
</div>
</body>
</html>
```

代码清单16-20的运行结果如图16-37所示。

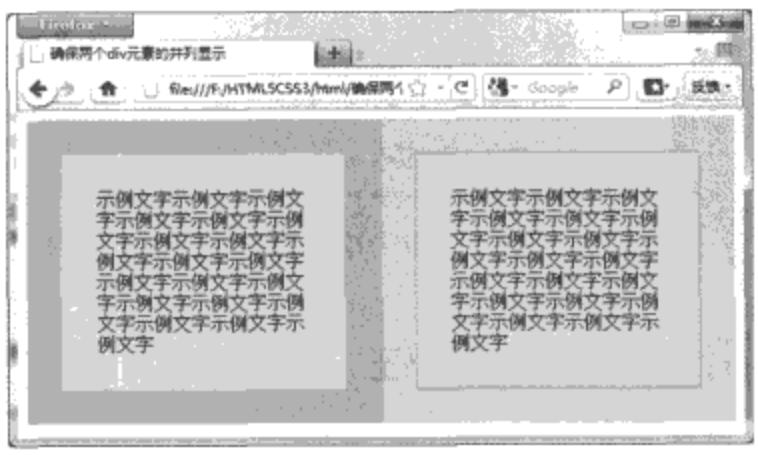

图16-37 确保两个div元素的并列显示

第17章
与背景和边框相关样式

本章主要介绍CSS 3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，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。

学习内容：

- 掌握CSS 3中新增的与背景相关的background-clip属性、background-origin属性、background-size属性及background-break属性的概念、使用方法以及各种浏览器的支持情况。
- 知道如何在一个元素的背景中使用多个图像文件来完成复杂背景图像的绘制。
- 知道如何使用CSS 3中的border-radius属性来给元素添加一个圆角边框。
- 知道如何使用CSS 3中的border-image属性来给元素添加一个可随着元素尺寸的变化而自动伸缩的图像边框。

17.1 与背景相关的新增属性

在CSS 3中，追加了一些与背景相关的属性，如表17-1所示。

表17-1 CSS 3中追加的与背景相关的属性

属　性	功　能
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属性之外的其他三个属性，在书写样式代码的时候需要在属性前面加上“-moz-”文字。但是在使用background-break属性的时候，在样式代码中不是书写“-moz-background-break”，而是书写“-moz-background-inline-policy”，这一点需要注意。

在Safari浏览器、Google Chrome浏览器及Opera浏览器中，支持除了background-break之外的其他三个属性，在书写样式代码的时候需要在属性前面加上“-webkit-”文字。

17.1.1 指定背景的显示范围——background-clip属性

在HTML页面中，一个具有背景的元素通常由元素的内容、内部补白（padding）、边框、外部补白（margin）构成，它们的结构示意图如图17-1所示。

图17-1 具有背景的元素构成图

元素背景的显示范围在CSS 2与CSS 2.1、CSS 3中并不相同。在CSS 2中，背景的显示范围是指内部补白之内的范围，不包括边框；而在CSS 2.1乃至CSS 3中，背景的显示

范围是指包括边框在内的范围。在CSS 3中，可以使用background-clip来修改背景的显示范围，如果将background-clip的属性值设定为border,则背景范围包括边框区域，如果设定为padding,则不包括边框区域。

为了更直观地说明问题，我们来看代码清单17-1中的一个示例。该示例中具有两个div元素，设定两个div元素的背景颜色均为黑色，边框均为绿色点划线。在样式代码中指定一个div元素的background-clip的属性值为border，另一个div元素的background-clip的属性值为padding。我们来看一下在示例的运行结果中两个div元素在显示上有什么区别。

代码清单17-1　两种background-clip属性值的对比示例

```
<!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>两种background-clip属性值的对比示例</title>
<style type="text/css">
div{
        background-color: black;
        border: dashed 15px green;
        padding: 30px;
        color:white;
        font-size:30px;
        font-weight:bold;
}
div.div1{
        -moz-background-clip: border;
        -webkit-background-clip: border;
}
div.div2{
        -moz-background-clip: padding;
        -webkit-background-clip: padding;
}
</style>
</head>
<body>
<div class="div1">示例文字1</div><br>
<div class="div2">示例文字2</div>
</body>
```

这段代码的运行结果如图17-2所示。

从图17-2中我们可以看出，当div元素的背景色是黑色、边框是点划线的时候，如果background-clip的属性值为border，则边框点划线中点与点之间的颜色也变为了div元素的背景色黑色，说明背景的显示范围包括了边框在内；如果background-clip的属性值为padding，则边框点划线中点与点之间的颜色为网页的背景色白色，说明背景的显示范围不包括边框在内。

另外，当背景为图像时运行结果也同样如此，当background-clip的属性值为border时，图像会占据边框点划线点与点之间的空间，而当background-clip的属性值为padding时，边框点划线点与点之间的颜色仍然为网页背景色。

将代码清单17-1中的样式代码修改为如下代码，然后重新运行该示例，运行结果如图17-3所示。

```
<style type="text/css">
div{
        background-color: black;
        background-image: url(flower-green.png);
        border: dashed 15px green;
        padding: 30px;
        color:white;
        font-size:2em;
        font-weight:bold;
}
div.div1{
        -moz-background-clip: border;
        -webkit-background-clip: border;
}
div.div2{
        -moz-background-clip: padding;
        -webkit-background-clip: padding;
}
</style>
```

图17-2 两种background-clip属性值的对比示例

图17-3 背景为图像时两种background-clip属性值的对比示例

17.1.2 指定绘制背景图像的绘制起点——background-origin属性

在绘制背景图像时，默认是从内部补白（padding）区域的左上角开始，但是可以利用background-origin属性来指定绘制时从边框的左上角开始，或者从内容的左上角开始。

在Firefox浏览器中指定绘制起点时，需要在样式代码中将background-origin属性书写成“-moz-background-origin”的形式；在Safari浏览器或Chrome浏览器中指定绘制起点时，需要在样式代码中将background-origin属性书写成“-webkit-background-origin”的形式。接下来，我们在代码清单17-2中看一个background-origin属性的使用示例，在示例中具有三个div元素，三个div元素的背景均指定为同一背景图像，分别指定其background-origin属性为

border、padding及content，分别代表从边框的左上角、内部补白区域的左上角或内容的左上角开始绘制。另外，将三个div元素的background-repeat属性均指定为no-repeat，表示不使用平铺方式。

代码清单17-2　background-origin属性的使用示例

```
<!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>background-origin属性使用示例</title>
</head>
<style type="text/css">
div{
        background-color: black;
        background-image: url(flower-green.png);
        background-repeat: no-repeat;
        border: dashed 15px green;
        padding: 30px;
        color:white;
        font-size:2em;
        font-weight:bold;
}
div.div1{
    -moz-background-origin: border;
    -webkit-background-origin: border;
}
div.div2{
    -moz-background-origin: padding;
    -webkit-background-origin: padding;
}
div.div3{
    -moz-background-origin: content;
    -webkit-background-origin: content;
}
</style>
<body>
<div class="div1">示例文字1</div><br>
<div class="div2">示例文字2</div><br>
<div class="div3">示例文字3</div>
</body>
```

这段代码的运行结果如图17-4所示。

另外，虽然将background-clip属性指定为padding的时候，边框点划线中点与点之间的图像不会显示，但是仍然可以通过将background-origin属性指定为border的方法来指定从边框的左上角开始绘制。代码清单17-3为针对这部分进行证明的一个示例，该示例中有一个div元素，对该div元素指定背景图像后同时指定它的background-clip属性值为padding，且background-origin属性值为border。

图17-4 background-origin属性的使用示例

代码清单17-3 background-clip属性与background-origin属性结合使用的示例

```
<!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>background-clip属性与background-origin属性结合使用示例</title>
</head>
<style type="text/css">
div{
        background-color: black;
        background-image: url(flower-green.png);
        background-repeat: no-repeat;
        border: dashed 15px green;
        padding: 30px;
        color:white;
        font-size:2em;
        font-weight:bold;
}
div.div1{
            -moz-background-clip: padding;
            -webkit-background-clip: padding;
            -moz-background-origin: border;
            -webkit-background-origin: border;
}
</style>
<body>
<div class="div1">示例文字1</div><br>
</body>
</html>
```

这段代码的运行结果如图17-5所示。

图17-5 background-clip属性与background-origin属性结合使用的示例

17.1.3　指定背景图像的尺寸——background-size属性

在CSS 3中，可以使用background-size属性来指定背景图像的尺寸。到目前为止，background-size属性得到了Firefox 4浏览器、Safari 3、Opera 10、Google Chrome 8浏览器的支持。在样式代码中，使用Safari 3浏览器的时候，需要将其书写成“-webkit-background-size”的形式，使用Opera 10浏览器的时候，需要书写成“background-size”的形式，使用Chrome浏览器的时候，可以书写成“background-size”的形式或“-webkit-background-size”的形式。

使用background-size属性来指定背景图像尺寸的最简单的方法类似如下所示。

```
background-size: 40px 20px;
-webkit-background-size: 40px 20px;
```

其中，40px为背景图像的宽度，20px为背景图像的高度，中间用半角空格进行分隔。

代码清单17-4为background-size属性的一个使用示例。在该示例中，具有一个div元素，使用background-size属性来指定该div元素的背景图像的宽度为40px，高度为20px。

代码清单17-4　background-size属性的使用示例

```
<!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>background-size属性的使用示例</title>
</head>
<style type="text/css">
div{
        background-color: black;
        background-image: url(flower-red.png);
        padding: 30px;
        color:white;
        font-size:2em;
        font-weight:bold;
        background-size: 40px 20px;
        -webkit-background-size: 40px 20px;
}
</style>
<body>
<div>示例文字</div><br>
</body>
</html>
```

代码清单17-4的运行结果如图17-6所示。

另外，如果要维持图像纵横比例的话，可以在设定图像宽度与高度的同时，将另一个参数设定为“auto”。例如，将代码清单17-4中的样式代码修改为如下所示的样式代码，该代码中将图像的高度设定为20像素，宽度设定为auto，修改后运行该示例，运行结果如图17-7所示。

```
<style type="text/css">
```

```
div{
        background-color: black;
        background-image: url(flower-red.png);
        padding: 30px;
        color:white;
        font-size:2em;
        font-weight:bold;
        background-size: auto 20px;
        -webkit-background-size: auto 20px;
}
</style>
```

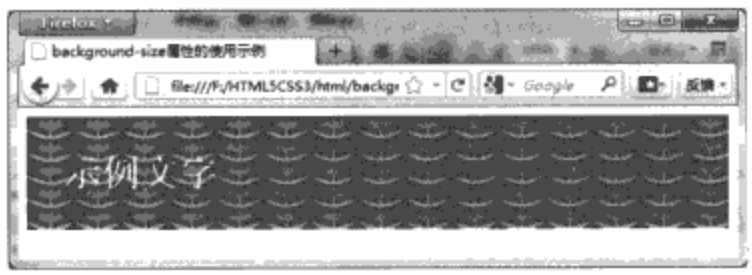

图17-6 background-size属性的使用示例

图17-7 background-size属性中使用auto参数示例

在使用background-size属性的时候，可以将宽度与高度中的一个参数省略，只指定一个参数。在这种情况下，在Safari 3浏览器中的处理方式为宽度与高度都使用这个参数，而在Google Chrome 8以上浏览器与Opera 10以上浏览器中的处理方式为宽度值使用这个参数，高度值使用auto参数。

将代码清单17-4中的样式代码修改为如下所示的样式代码，该代码只在background-size属性的参数中设定一个值20px，然后重新运行该示例，在Opera浏览器中的运行结果如图17-8所示。

```
<style type="text/css">
div{
        background-color: black;
        background-image: url(flower-red.png);
        padding: 30px;
        color:white;
        font-size:2em;
        font-weight:bold;
        background-size:20px;
        -webkit-background-size:20px;
}
</style>
```

在指定宽度与高度的时候，也可以使用百分比的值来作为参数。这时，在Safari 3以上的浏览器中将指定的百分比视为图像尺寸除以整个内部补白（padding）区域的尺寸后得出的百分比，在Opera 10以上的浏览器中将指定的百分比视为图像尺寸除以整个边框区域的尺寸后得出的百分比。

将代码清单17-4中的样式代码修改为如下所示的样式代码，该代码中设定宽度与高度均为50%，然后重新运行该示例，运行结果如图17-9所示。

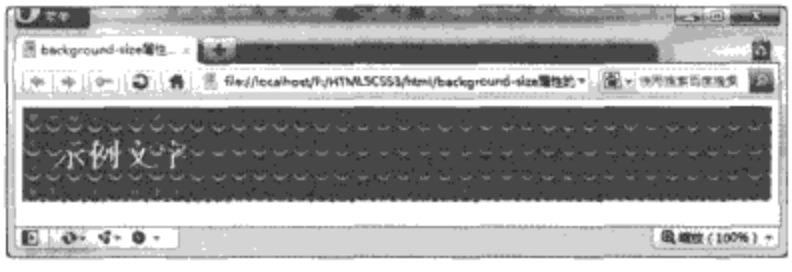

图17-8　在Opera浏览器的background-size属性中使用唯一参数的示例

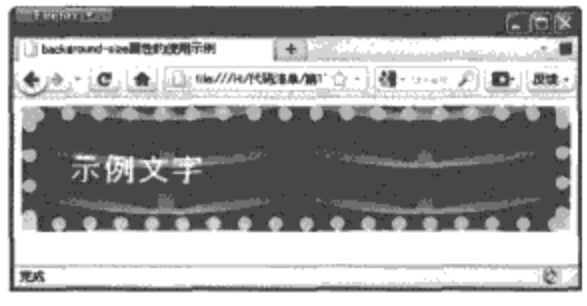

图17-9　background-size属性中使用百分比参数示例

```
<style type="text/css">
div{
        background-color: black;
        background-image: url(flower-red.png);
        border: dotted 15px yellow;
        padding: 30px;
        color:white;
        font-size:2em;
        font-weight:bold;
        background-size: 50% 50%;
        -webkit-background-size: 50% 50%;
}
</style>
```

17.1.4　指定内联元素背景图像进行平铺时的循环方式——background-break属性

在CSS 3中，可以使用background-break属性来指定平铺内联元素背景图像时的循环方式，可以指定bounding-box、each-box和continuous这三种循环方式。在使用Firefox浏览器的时候，需要在样式代码中将其书写成“-moz-background-inline-policy”的形式。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。

代码清单17-5为background-break属性的一个使用示例。该示例中具有三个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三个span元素指定三种background-break属性值，通过运行结果的显示我们可以看出这三种循环方式的区别。

代码清单17-5　background-break属性的使用示例

```
<!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>background-break属性的使用示例</title>
</head>
```

```
<style type="text/css">
span{
    background-color: #888888;
    background-image: url(flower-green.png);
    padding: 0.2em;
    color:gray;
    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>示例文字示例文字示例文字示例文字示例文字示例文字示例文
字示例文字示例文字示例文字示例文字</span></div><br/>
<div class="div2"><span>示例文字示例文字示例文字示例文字示例文字示例文字示例文
字示例文字示例文字示例文字示例文字</span></div><br/>
<div class="div3"><span>示例文字示例文字示例文字示例文字示例文字示例文字示例文
字示例文字示例文字示例文字示例文字</span></div>
</body>
</html>
```

这段代码的运行结果如图17-10所示。

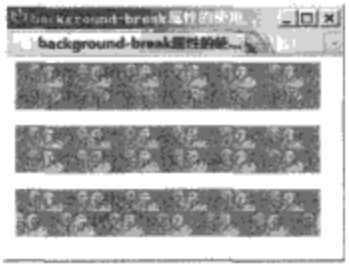

图17-10 background-break属性的使用示例

17.2 在一个元素中显示多个背景图像

在CSS 3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叠显示，从而使得背景图像中所用素材的调整变得更加容易。

首先，我们在代码清单17-6中看一个示例——在一个元素中显示多个背景图像。在该示例中具有一个div元素，我们来看一下怎样在这个div元素中显示多个背景图像。

代码清单17-6 在一个元素中显示多个背景图像的示例

```
<!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>在一个元素中显示多个背景图像的示例</title>
</head>
<style>
div{
        background-image:url(flower-red.png), url(flower-green.png),url(sky.jpg);
        background-repeat: no-repeat, repeat-x, no-repeat;
        background-position: 3% 98%,85%, center center,  top;
        width: 300px;
        padding: 90px 0px;
}
</style>
<body>
<div></div>
</body>
</html>
```

这段代码的运行结果如图17-11所示。

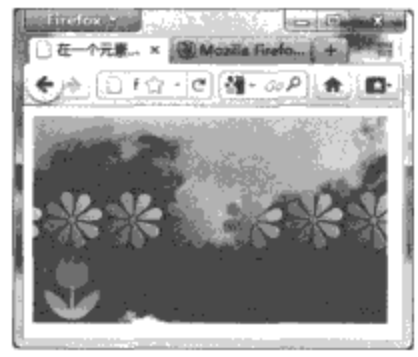

图17-11　在一个元素中显示多个背景图像的示例

在div元素的样式代码中，我们用到了几个关于背景的属性——background-image属性、background-repeat属性与background-position属性。这些属性都是CSS1中就有的属性，但是在CSS 3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。

请注意：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。

在代码清单17-6中，通过指定多个background-image属性、background-repeat属性与background-position属性，我们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着多个图像文件一起利用的属性有如下几个：

- background-image
- background-repeat
- background-position
- background-clip
- background-origin
- background-size

17.3　圆角边框的绘制

本节介绍如何使用CSS 3的样式进行圆角边框的绘制。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CSS 3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。到目前为止，Safari浏览器、Firefox浏览器、Opera浏览器及Chrome浏览器都支持这种绘制圆角边框的样式。

17.3.1 border-radius属性

在CSS 3中，只要使用border-radius属性指定好圆角的半径，就可以绘制圆角边框了。使用Firefox浏览器的时候，需要在样式代码中将其书写成“-moz-border-radius”的形式；使用Safari浏览器的时候，需要书写成“-webkit-border-radius”的形式，使用Opera浏览器的时候，需要书写成“border-radius”的形式，使用Chrome浏览器的时候，可以书写成“border-radius”或“-webkit-border-radius”的形式。

代码清单17-7是绘制圆角边框的一个示例，在该示例中具有一个div元素，使用border-radius属性将其边框绘制为圆角边框，圆角半径为20像素，边框颜色为蓝色，div元素的背景色为浅蓝色。

代码清单17-7 绘制圆角边框的示例

```
<!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>绘制圆角边框示例</title>
</head>
<style type="text/css">
div{
        border: solid 5px blue;
        border-radius: 20px;
        -moz-border-radius: 20px;
        -o-border-radius: 20px;
        -webkit-border-radius: 20px;
        background-color: skyblue;
        padding: 20px;
        width: 180px;
}
</style>
<body>
<div>
示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。示例文字。
示例文字。示例文字。示例文字。
</div>
</body>
</html>
```

这段代码的运行结果如图17-12所示。

图17-12　绘制圆角边框的示例

17.3.2　在border-radius属性中指定两个半径

在border-radius属性中，可以指定两个半径，指定方法如下所示。

```
border-radius: 40px 20px;
```

针对这种情况，各种浏览器的处理方式并不一致。在Chrome浏览器与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浏览器与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。

将代码清单17-7中div元素的样式代码修改为如下所示的样式代码（使用两个半径），然后重新运行该示例，在Firefox浏览器中的运行结果如图17-13所示，在Chrome浏览器中的运行结果如图17-14所示。

```
<style type="text/css">
div{
        border: solid 5px blue;
        border-radius: 40px 20px;
        -moz-border-radius:40px 20px;
        -o-border-radius:40px 20px;
        -webkit-border-radius: 40px 20px;
        background-color: skyblue;
        padding: 20px;
        width: 180px;
}
</style>
```

图17-13　在Firefox浏览器中使用两个半径参数的border-radius属性

图17-14　在Chrome浏览器中使用两个半径参数的border-radius属性

17.3.3 不显示边框的时候

在CSS 3中，如果使用了border-radius属性但是把边框设定为不显示的时候，浏览器将把背景的四个角绘制为圆角。

将代码清单17-7中div元素的样式代码修改为如下所示的样式代码（指定边框为不显示），然后重新运行该示例，运行结果如图17-15所示。

```
<style type="text/css">
div{
        border: none;
        border-radius: 20px;
        -moz-border-radius:20px;
        -o-border-radius:20px;
        -webkit-border-radius: 20px;
        background-color: skyblue;
        padding: 20px;
        width: 180px;
}
</style>
```

图17-15 使用了border-radius属性但是设定边框为不显示

17.3.4 修改边框种类的时候

使用了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。将代码清单17-7中div元素的样式代码修改为如下所示的样式代码（将边框修改为虚线，颜色修改为红色，并指定边框宽度为10px），然后重新运行该示例，运行结果如图17-16所示。

```
<style type="text/css">
div{
        border: dashed 5px blue;
        border-radius: 20px;
        -moz-border-radius: 20px;
        -o-border-radius: 20px;
        -webkit-border-radius: 20px;
        background-color: skyblue;
        padding: 20px;
        width: 180px;
}
</style>
```

图17-16 使用了border-radius属性并且指定边框为虚线时

17.3.5 绘制四个角不同半径的圆角边框

如果要绘制的圆角边框的四个角半径各不相同，可以将border-top-left-radius属性、border-top-right-radius属性、border-bottom-right-radius属性、border-bottom-left-radius属性结合起来使用。其中border-top-left-radius属性指定左上角半径，border-top-right-radius属性指定右上角半径，border-bottom-right-radius属性指定右下角半径，border-bottom-left-radius属性指定左下角半径。将代码清单17-7中div元素的样式代码修改为如下所示的样式代码（指定四个角为不同半径），然后重新运行该示例，运行结果如图17-17所示。

```
<style type="text/css">
div{
        border: solid 5px blue;
        border-radius-topleft: 10px;
        border-radius-topright: 20px;
        border-radius-bottomright: 30px;
        border-radius-bottomleft: 40px;
        -moz-border-radius-topleft: 10px;
        -moz-border-radius-topright: 20px;
        -moz-border-radius-bottomright: 30px;
        -moz-border-radius-bottomleft: 40px;
        -o-border-radius-topleft: 10px;
        -o-border-radius-topright: 20px;
        -o-border-radius-bottomright: 30px;
        -o-border-radius-bottomleft: 40px;
        -webkit-border-top-left-radius: 10px;
        -webkit-border-top-right-radius: 20px;
        -webkit-border-bottom-right-radius: 30px;
        -webkit-border-bottom-left-radius: 40px;
        background-color: skyblue;
        padding: 20px;
        width: 180px;
}
</style>
```

图17-17　绘制四个角不同半径的圆角边框

17.4　使用图像边框

17.4.1　border-image属性

在CSS 3之前，如果要使用图像边框，若元素的长或宽是随时可变的，页面制作者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也就变得比较多了。

针对这种情况，CSS 3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，这样就不需要页面制作者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。关于浏览器对于边框所用到的图像的自动分割内容，会在17.4.2节中进行详细介绍。

使用border-image属性的时候，如果使用的是Firefox浏览器，需要在样式代码中将其书写成“-moz-border-image”的形式；如果使用的是Safari浏览器或Chrome浏览器，需要书写成“-webkit-border-image”的形式；如果使用的是Opera浏览器，需要书写成“border-image”的形式。

接下来，我们在代码清单17-8中看一个border-image属性的使用示例，该示例中有一个div元素，使用border-image属性为该div元素添加了一个图像边框。

代码清单17-8 border-image属性的使用示例

```
<!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>border-image属性的使用示例</title>
</head>
<style type="text/css">
div{
        border-image: url(borderimage.png) 20 20 20 20 / 20px;
        -webkit-border-image: url(borderimage.png) 20 20 20 20 / 20px;
        -moz-border-image: url(borderimage.png) 20 20 20 20 / 20px;
}
</style>
<body>
<div>
示例文字
</div>
</body>
</html>
```

这段代码的运行结果如图17-18所示。

代码清单17-8中没有对div元素指定宽度，所以图像边框的宽度等于浏览器的宽度，如果对div元素指定宽度，则图像边框也会自动伸缩成指定的宽度，而且能够正常显示。将代码清单17-8中的样式代码修改成如下所示的样式代码（指定div元素的宽度），然后重新运行该示例，运行结果如图17-19所示。

```
<style type="text/css">
div{
        border-image: url(borderimage.png) 20 20 20 20 / 20px;
        -webkit-border-image: url(borderimage.png) 20 20 20 20 / 20px;
        -moz-border-image: url(borderimage.png) 20 20 20 20 / 20px;
        width:200px;
}
</style>
```

图17-18 border-image属性的使用示例

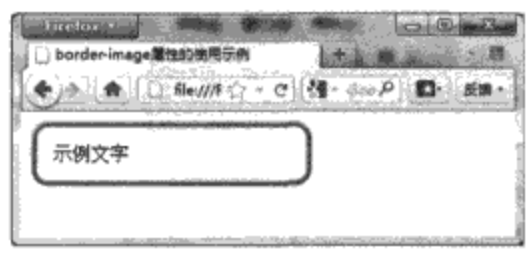

图17-19 添加图像边框后修改div元素的宽度

17.4.2 border-image属性最简单的使用方法

border-image属性最简单的使用方法如下所示。

```
border-image: url(图像文件的路径) A B C D
-webkit-border-image: url(图像文件的路径) A B C D
```

```
-moz-border-image: url(图像文件的路径) A B C D
```

border-image属性值中至少必须指定五个参数，其中第一个参数为边框所使用的图像文件的路径，A、B、C、D四个参数表示当浏览器自动把边框所使用到的图像进行分隔时的上边距、右边距、下边距及左边距。图17-20用图示的方法对这四个参数进行了说明。

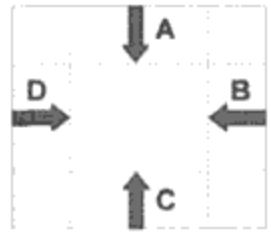

图17-20　A、B、C、D四个参数的图示

接下来，让我们看一下如果在border-image属性值中指定了这四个参数，浏览器对于边框所使用的图像是如何进行分割的。

首先，当在样式代码中书写如下所示的代码时，浏览器对于边框所使用的图像分割方法如图17-21所示。

```
border-image: url(borderimage.png) 18 18 18 18
-webkit-border-image: url(borderimage.png) 18 18 18 18
-moz-border-image: url(borderimage.png) 18 18 18 18
```

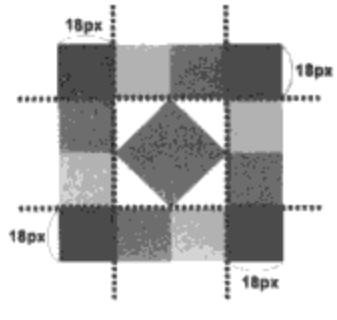

图17-21　浏览器对于图像文件的分割

如图17-21所示，图像被自动分割为9部分。分割后的图像在CSS 3中的名称如表17-2所示。

表17-2　被分割为9部分的图像名称（显示顺序与图17-21相对应）

border-top-left-image	border-top-image	border-top-right-image
border-left-image		border-right-image
border-bottom-left-image	border-bottom-image	border-bottom-right-image

具体显示的时候，四个角上的border-top-left-image、border-top-right-image、border-bottom-left-image、border-bottom-right-image这四部分是没有任何展示效果的，不会平铺、不会重复、也不会拉伸，类似于视觉中盲点的意思。

对于border-top-image、border-left-image、border-right-image、border-bottom-image这四部分，浏览器分别作为上边框使用图像、左边框使用图像、右边框使用图像、下边框使用图像来进行显示，必要时可以将这四部分图像进行平铺或伸缩。

在代码清单17-9中，我们给出一个border-image属性最简单的使用方法的示例，在该示例的运行结果中，我们可以看出浏览器最终是如何将分割后的各部分图像显示在一个边框宽度为5px的div元素中的。

代码清单17-9　border-image属性最简单的使用方法的示例

```
<!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>border-image属性的最简单的使用方法的示例</title>
</head>
<style type="text/css">
```

```
div{
        border: solid 5px;
        border-image: url(borderimage.png) 18 18 18 18;
        -webkit-border-image: url(borderimage.png) 18 18 18 18;
        -moz-border-image: url(borderimage.png) 18 18 18 18;
        width:300px;
}
</style>
<body>
<div>
示例文字
</div>
</body>
</html>
```

这段代码的运行结果如图17-22所示。

图17-22 border-image属性最简单的使用方法示例

17.4.3 使用border-image属性来指定边框宽度

在代码清单17-9中，使用了border属性来指定边框的宽度。在CSS 3中，除了可以使用border属性或border-width属性来指定边框的宽度外，使用border-image属性同样可以指定边框的宽度，指定方法如下所示。

```
border-image: url(图像文件的路径) A B C D/border-width
```

将代码清单17-9中的样式代码修改为如下所示的样式代码（在border-image属性中将边框宽度修改为18px），然后重新运行该示例，运行结果如图17-23所示。

```
<style type="text/css">
div{
        border:solid;
        border-image: url(borderimage.png) 18 18 18 18/18px;
        -webkit-border-image: url(borderimage.png) 18 18 18 18/18px;
        -moz-border-image: url(borderimage.png) 18 18 18 18/18px;
        width:300px;
}
</style>
```

可以在border-image属性中将四条边的边框指定为不同宽度，将代码清单17-9中的样式代码修改为如下所示的样式代码（指定4条边分别为5px、10px、15px、20px），然后重新运行该示例，运行结果如图17-24所示。

另外，在这段代码中A、B、C、D四个参数只指定了一个参数18px，这是因为在CSS 3中，如果此处的四个参数完全相同，可以只写一个参数，将其他三个参数省略。

```
<style type="text/css">
div{
      border:solid;
      border-image: url(borderimage.png) 18/5px 10px 15px 20px;
      -webkit-border-image: url(borderimage.png) 18/5px 10px 15px 20px;
      -moz-border-image: url(borderimage.png) 18/5px 10px 15px 20px;
      width:300px;
}
</style>
```

图17-23　使用border-image属性来指定边框宽度

图17-24　指定4条边为不同宽度

17.4.4　中央图像的自动拉伸

浏览器将边框所用图像自动分割为9部分后，除了将border-top-image、border-left-image、border-right-image、border-bottom-image这4部分自动分配为四条边所用的图像之外，将位于中间部分的图像分配给元素边框所包围的中间区域，随着div元素的内容变化的同时，或者在样式代码中修改div元素的宽度或高度的同时，中间部分的图像也会自动进行伸缩，以填满该中间区域。

将代码清单17-9中的样式代码修改为如下所示的样式代码（增加div元素中内容），同时增加div元素中的文字，然后重新运行该示例，运行结果如图17-25所示。

```
<style type="text/css">
div{
        border:solid;
        border-image: url(borderimage.png) 18/5px;
        -webkit-border-image: url(borderimage.png) 18/5px;
        -moz-border-image: url(borderimage.png) 18/5px;
        width:300px;
}
</style>
```

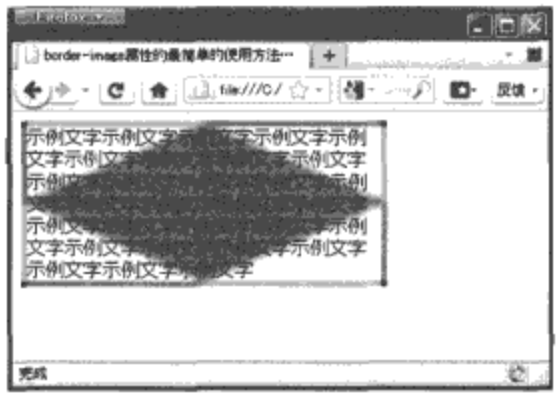

图17-25　中间图像的自动伸缩

17.4.5 指定四条边中图像的显示方法

可以在border-image属性中指定元素四条边中的图像是以拉伸的方式显示，还是以平铺的方式显示，指定方法如下所示。

```
border-image: url(文件路径) A B C D/border-width topbottom leftright
```

其中，topbottom表示元素的上下两条边中图像的显示方法，leftright表示元素的左右两条边中的显示方法。在显示方法中可以指定的值为repeat、stretch与round三种。

❑ repeat

将显示方法指定为repeat时，图像将以平铺的方式进行显示。

代码清单17-10为在border-image属性中指定四条边中的图像以平铺的方式进行显示的示例。如果四条边中图像均以平铺方式进行显示，则中间图像也以平铺方式进行显示。

代码清单17-10 指定四条边中图像为平铺显示

```
<!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>指定四条边中图像为平铺显示</title>
</head>
<style type="text/css">
div{
        border-image: url(borderimage.png) 18/5px repeat repeat;
        -webkit-border-image: url(borderimage.png) 18/5px repeat repeat;
        -moz-border-image: url(borderimage.png) 18/5px repeat repeat;
        width:300px;
        height:200px;
}
</style>
<body>
<div></div>
</body>
</html>
```

这段代码的运行结果如图17-26所示。

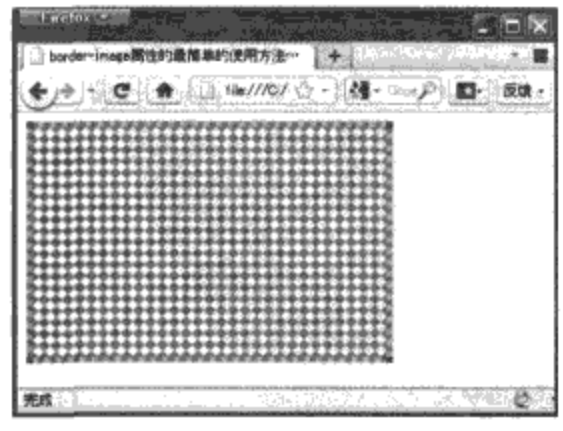

图17-26 指定四条边中的图像为平铺显示

❑ stretch

将显示方法指定为stretch时，图像将以拉伸的方式进行显示。

代码清单17-11为在border-image属性中指定四条边中的图像以拉伸的方式进行显示的示例。如果四条边中的图像均以拉伸方式进行显示，则中间图像也以拉伸方式进行显示。

代码清单17-11 指定四条边中的图像为拉伸显示

```
<!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>指定四条边中的图像为拉伸显示</title>
</head>
<style>
div{
      border-image: url(borderimage.png) 18/5px stretch stretch;
      -webkit-border-image: url(borderimage.png) 18/5px stretch stretch;
      -moz-border-image: url(borderimage.png) 18/5px stretch stretch;
      width:300px;
      height:200px;
}
</style>
<body>
<div></div>
</body>
</html>
```

这段代码的运行结果如图17-27所示。

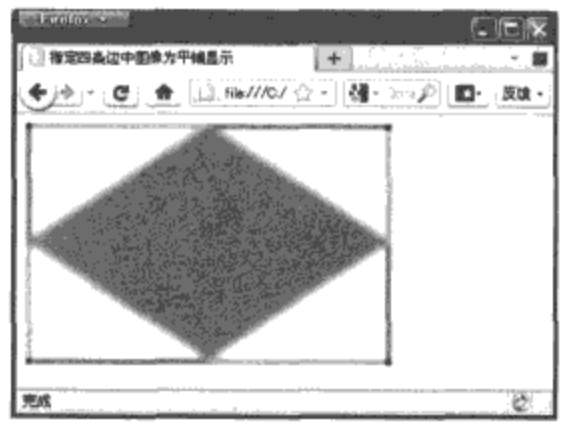

图17-27　指定四条边中图像为拉伸显示

❑ repeat+stretch

可以将上下两条边中的图像的显示方式指定为平铺显示，左右两条边中的图像的显示方式指定为拉伸显示，或者将上下两条边中的图像的显示方式指定为拉伸显示，左右两条边中的图像的显示方式指定为平铺显示。使用第一种指定方式时，中央图像在水平方向为平铺显示，垂直方向为拉伸显示；使用第二种指定方式时，中央图像在水平方向为拉伸显示，垂直方向为平铺显示。

代码清单17-12为平铺显示方式与拉伸显示方式结合使用的一个示例，示例中元素垂直方向的显示方式为拉伸显示，水平方向的显示方式为平铺显示。

代码清单17-12 平铺显示与拉伸显示结合使用的示例

```
<!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>平铺显示与拉伸显示结合使用</title>
</head>
<style type="text/css">
div{
        border-image: url(borderimage.png) 18/5px repeat stretch;
        -webkit-border-image: url(borderimage.png) 18/5px repeat stretch;
        -moz-border-image: url(borderimage.png) 18/5px repeat stretch;
        width:300px;
        height:200px;
}
```

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

这段代码的运行结果如图17-28所示。

图17-28 平铺显示与拉伸显示结合使用

❑ round

将显示方法指定为round时与将显示方法指定为repeat时类似，都是将图像进行平铺显示，区别在于如果最后显示的一幅图像不能被完全显示，且能够显示的部分不到图像的一半，就不显示最后的图像，然后扩大前面的图像，使显示区域正好完整平铺全部图像；如果能够显示的部分超过图像的一半，就显示最后的图像，但是将全部显示的图像缩小，使显示区域正好完整平铺全部图像。代码清单17-13为将显示方法指定为round时的一个示例，示例中四条边的显示方式全部指定为round，元素中各条边中的图像与中央图像全部为完整显示，没有残缺图像。另外需要注意的是，在Safari浏览器与Chrome浏览器中，将显示方法指定为round与将显示方法指定为repeat时的处理方法完全一致，图像不完全时也照原样显示，不做特别处理。

代码清单17-13 将显示方法指定为round时的示例

```
<!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>将显示方法指定为round</title>
</head>
<style type=" text/css" >
div{
        border-image: url(borderimage.png) 18/5px round round;
        -webkit-border-image: url(borderimage.png) 18/5px round round;
        -moz-border-image: url(borderimage.png) 18/5px round round;
        width:300px;
        height:200px;
}
</style>
<body>
<div></div>
</body>
</html>
```

这段代码在Firefox浏览器中的运行结果如图17-29所示。

这段代码在Chrome浏览器中的运行结果如图17-30所示。

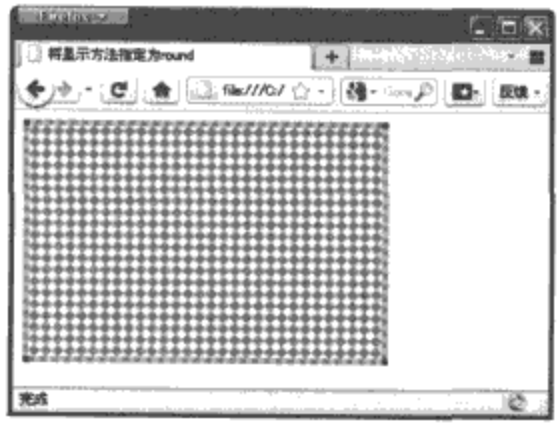

图17-29　在Firefox浏览器中将显示方式指定为round

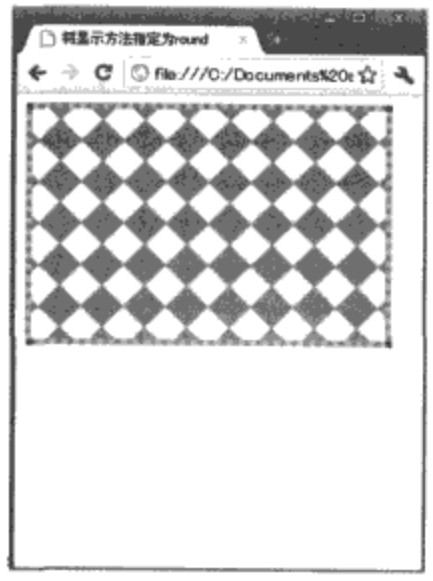

图17-30　在Chrome浏览器中将显示方式指定为round

17.4.6　使用背景图像

在使用border-image属性的时候，仍然可以正常使用背景图像，但是为了不让边框图像挡住背景图像，需要使用中间为透明的边框图像，否则背景图像就有可能被边框图像的中央部分挡住部分或全体了。

代码清单17-14为同时让元素具有边框图像和背景图像的示例，其中所使用到的边框图像如图17-31所示。

图17-31　代码清单17-14示例中使用的边框图像——borderimage.png

代码清单17-14　元素同时具有边框图像和背景图像的示例

```
<!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>元素同时具有边框图像和背景图像</title>
</head>
<style type="text/css">
div{
        background-image: url(bk.jpg);
        background-repeat: no-repeat;
        border-image: url("borderimage.png") 20 20 20 20 / 5px;
        background-origin: border;
        border-radius: 18px;
        -webkit-border-image: url("borderimage.png") 20 20 20 20 / 5px;
        -webkit-background-origin: border;
```

```
        -webkit-border-radius: 18px;
        -moz-border-image: url("borderimage.png") 20 20 20 20 / 5px;
        -moz-background-origin: border;
        -moz-border-radius: 18px;
        width:711px;
        height:404px;
}
</style>
<body>
<div></div>
</body>
</html>
```

这段代码的运行结果如图17-32所示。

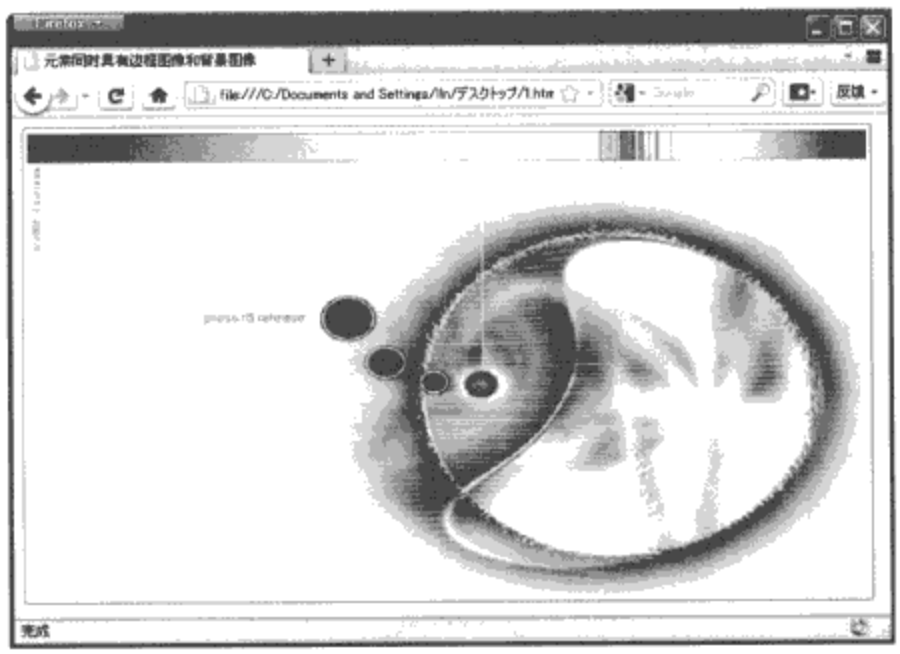

图17-32 让元素同时具有边框图像和背景图像

第18章

CSS 3中的变形处理

在CSS 3中，可以利用transform功能来实现文字或图像的旋转、缩放、倾斜、移动这四种类型的变形处理，本章将对此做详细介绍。

学习内容：

- 掌握CSS 3中transform功能的使用方法，能够使用transform功能来实现文字或图像的旋转、缩放、倾斜与移动的变形效果。
- 能够将旋转、缩放、倾斜与移动这四种变形效果结合使用，并知道如果使用的先后顺序不同，页面显示结果会有什么样的区别。

18.1 transform功能的基础知识

18.1.1 如何使用transform功能

在CSS 3中，通过transform属性来使用transform功能。到目前为止，Safari 3.1以上浏览器、Google Chrome 8以上浏览器、Firefox 4以上浏览器以及Opera 10以上浏览器都对该属性提供支持。在样式代码中，使用Safari浏览器或Chrome浏览器的时候，需要将其书写成“-webkit-transform”的形式，使用Firefox浏览器的时候，需要书写成“-moz-transform”的形式，使用Opera浏览器的时候，需要书写成“-o-transform”的形式。

首先，我们在代码清单18-1中看一个简单地使用transform属性实现变形处理的示例。在示例中有一个黄色的div元素，通过在样式代码中使用“transform: rotate(45deg)”语句使该div元素顺时针旋转45度。deg是CSS 3的“Values and Units”模块中定义的一个角度单位。

代码清单18-1 transform属性的使用示例

```
<!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>transform属性的使用示例</title>
</head>
<style type="text/css">
div{
        width: 300px;
        margin: 150px auto;
        background-color: yellow;
        text-align: center;
        -webkit-transform: rotate(45deg);
        -moz-transform: rotate(45deg);
        -o-transform: rotate(45deg);
}
</style>
<body>
<div>示例文字</div>
</body>
</html>
```

这段代码的运行结果如图18-1所示。

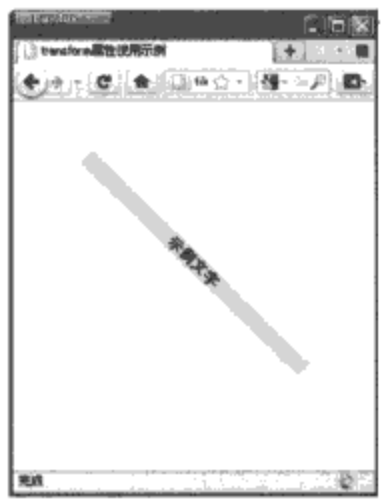

图18-1　transform属性的使用示例

18.1.2　transform功能的分类

在CSS 3中，可以使用transform功能实现四种文字或图像的变形处理，分别是旋转、缩放、倾斜以及移动。

旋转功能的实现方法前面已经介绍过，使用rotate方法，在参数中加入角度值，角度值后面跟表示角度单位的“deg”文字即可，旋转方向为顺时针旋转。

1. 缩放

使用scale方法来实现文字或图像的缩放处理，在参数中指定缩放倍率。譬如“scale(0.5)”表示缩小50%。代码清单18-2为使用scale方法实现缩放处理的一个示例。该示例中有一个div元素，使用scale方法使该元素缩小了50%。

代码清单18-2　scale方法的使用示例

```
<!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>scale方法使用示例</title>
</head>
<style type="text/css">
div{
        width: 300px;
        margin: 150px auto;
        background-color: yellow;
        text-align: center;
        -webkit-transform: scale(0.5);
        -moz-transform: scale(0.5);
        -o-transform: scale(0.5);
}
</style>
<body>
<div>示例文字</div>
</body>
</html>
```

这段代码的运行结果如图18-2所示。

另外，可以分别指定元素水平方向的放大倍率与垂直方向的放大倍率。例如，把代码清单18-2中的样式代码修改成如下所示的样式代码（即，使水平方向缩小50%，垂直方向放大一倍），修改后重新运行该示例，结果如图18-3所示。

```
<style type="text/css">
div{
        width: 300px;
        margin: 150px auto;
        background-color: yellow;
        text-align: center;
        -webkit-transform: scale(0.5,2);
        -moz-transform: scale(0.5,2);
        -o-transform: scale(0.5,2);
}
</style>
```

图18-2 scale方法的使用示例

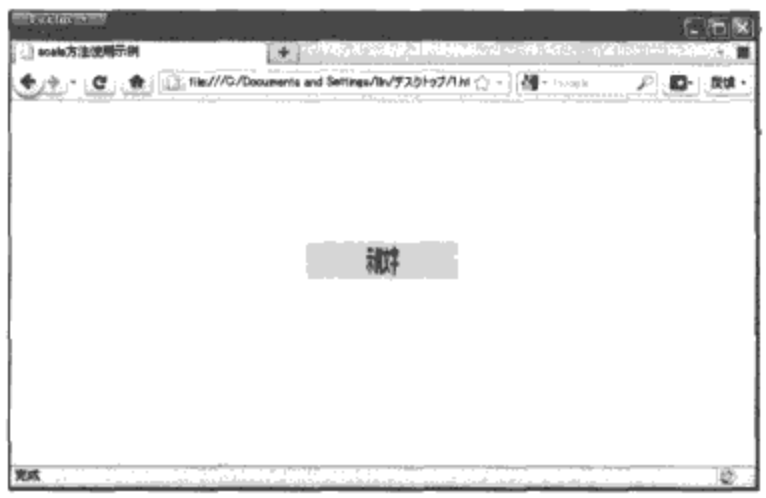

图18-3 分别指定水平方向与垂直方向的放大倍率

2. 倾斜

使用skew方法来实现文字或图像的倾斜处理，在参数中分别指定水平方向上的倾斜角度与垂直方向上的倾斜角度。例如“skew(30deg,30deg)”表示水平方向上倾斜30度，垂直方向上倾斜30度。代码清单18-3为skew方法的一个使用示例，该示例中有一个div元素，通过skew方法把元素水平方向上倾斜30度，垂直方向上倾斜30度。

代码清单18-3 skew方法使用示例

```
<!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>skew方法使用示例</title>
</head>
<style>
div{
        width: 300px;
        margin: 150px auto;
        background-color: yellow;
        text-align: center;
        -webkit-transform: skew(30deg,30deg);
```

```
        -moz-transform: skew(30deg,30deg);
        -o-transform: skew(30deg,30deg);
}
</style>
<body>
<div>示例文字</div>
</body>
</html>
```

这段代码的运行结果如图18-4所示。

另外，skew方法中的两个参数可以修改成只使用一个参数，省略另一个参数——这种情况下视为只在水平方向上进行倾斜，垂直方向上不倾斜。

将代码清单18-3中的样式代码修改成如下所示的样式代码（只指定一个参数），修改后重新运行该示例，结果如图18-5所示。

```
<style type="text/css">
div{
        width: 300px;
        margin: 150px auto;
        background-color: yellow;
        text-align: center;
        -webkit-transform: skew(30deg);
        -moz-transform: skew(30deg);
        -o-transform: skew(30deg);
}
</style>
```

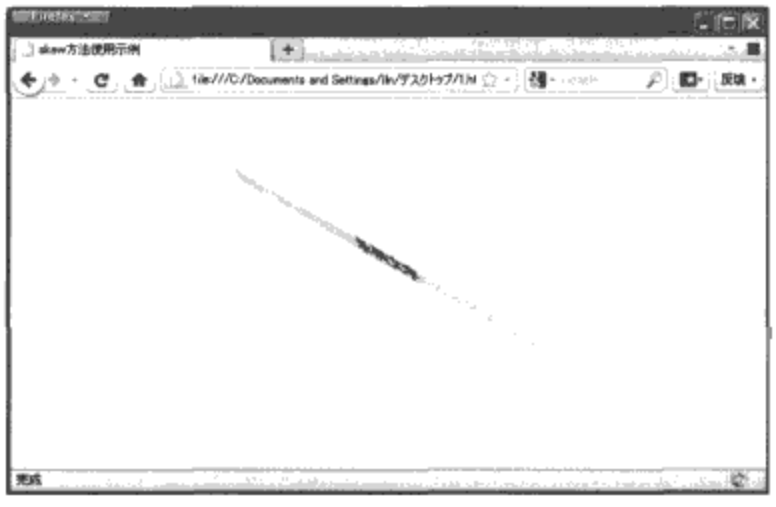

图18-4　skew方法使用示例

图18-5　skew方法中只使用一个参数

3. 移动

使用translate方法来将文字或图像进行移动，在参数中分别指定水平方向上的移动距离与垂直方向上的移动距离。例如“translate (50px，50px)”表示水平方向上移动50个像素，垂直方向上移动50个像素。代码清单18-4为translate方法的一个使用示例，该示例中有一个div元素，通过translate方法把元素水平方向上移动50个像素，垂直方向上移动50个像素。

代码清单18-4　translate方法的使用示例

```
<!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>translate方法使用示例</title>
</head>
<style type="text/css">
div{
        width: 300px;
        margin: 150px auto;
        background-color: yellow;
        text-align: center;
        -webkit-transform: translate(50px,50px);
        -moz-transform:translate(50px,50px);
        -o-transform: translate(50px,50px);
}
</style>
<body>
<div>示例文字</div>
</body>
</html>
```

这段代码的运行结果如图18-6所示。

另外，translate方法中的两个参数可以修改成只使用一个参数，省略另一个参数，这种情况下视为只在水平方向上进行移动，垂直方向上不移动。

将代码清单18-4中的样式代码修改成如下所示的样式代码（只指定一个参数），修改后重新运行该示例，结果如图18-7所示。

```
<style type="text/css">
div{
        width: 300px;
        margin: 150px auto;
        background-color: yellow;
        text-align: center;
        -webkit-transform: translate(50px);
        -moz-transform:translate(50px);
        -o-transform: translate(50px);
}
</style>
```

图18-6 translate方法的使用示例

图18-7 translate方法中只使用一个参数

18.2 对一个元素使用多种变形的方法

18.2.1 两个变形示例

在上一节我们介绍了使用transform对元素进行旋转、缩放、倾斜以及移动的方法，本节将介绍如何综合使用这几种方法来对一个元素进行多重变形。

首先，我们来看两个示例。代码清单18-5是一个对元素先移动，然后旋转最后缩放的示例；代码清单18-6是一个对元素先旋转，然后缩放，最后移动的示例。这两个示例都是对同一个页面中同一个元素进行多重变形的示例，而且各种变形方法中所使用的参数也都相同，旋转时都是顺时针旋转45度，缩放时都是将元素放大1.5倍，移动时都是向右移动150px，向下移动200px——两个示例的差别只是使用三种变形方法的先后顺序不一样而已，我们来看一下两种示例在浏览器中的运行结果是否相同。

代码清单18-5 对元素使用多重变形的示例（先移动，然后旋转，最后缩放）

```
<!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>对元素使用多重变形的示例</title>
</head>
<style type="text/css">
div{
      width: 300px;
      background-color: yellow;
      text-align: center;
      -webkit-transform:  translate(150px, 200px) rotate(45deg) scale(1.5);
      -moz-transform: translate(150px, 200px)  rotate(45deg)  scale(1.5) ;
      -o-transform:  translate(150px, 200px) rotate(45deg)  scale(1.5);
}
</style>
<body>
<div>示例文字</div>
</body>
</html>
```

代码清单18-5的运行结果如图18-8所示。

代码清单18-6 对元素使用多重变形示例（先旋转，然后缩放，最后移动）

```
<!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>对元素使用多重变形示例</title>
</head>
<style type="text/css">
div{
        width: 300px;
```

```
            background-color: yellow;
            text-align: center;
            -webkit-transform:rotate(45deg) scale(1.5) translate(150px, 200px);
            -moz-transform:rotate(45deg) scale(1.5) translate(150px, 200px);
            -o-transform:rotate(45deg) scale(1.5) translate(150px, 200px);
    }
    </style>
    <body>
    <div>示例文字</div>
    </body>
    </html>
```

代码清单18-6的运行结果如图18-9所示。

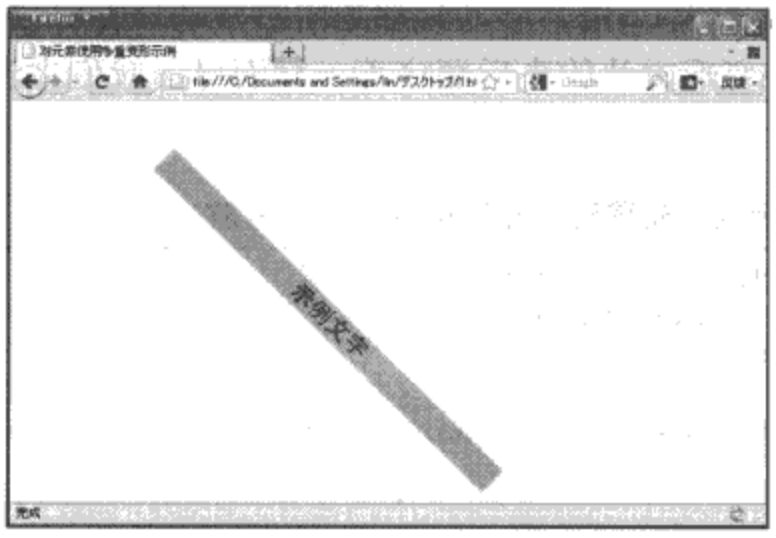

图18-8 对元素使用多重变形示例（先移动，然后旋转，最后缩放）

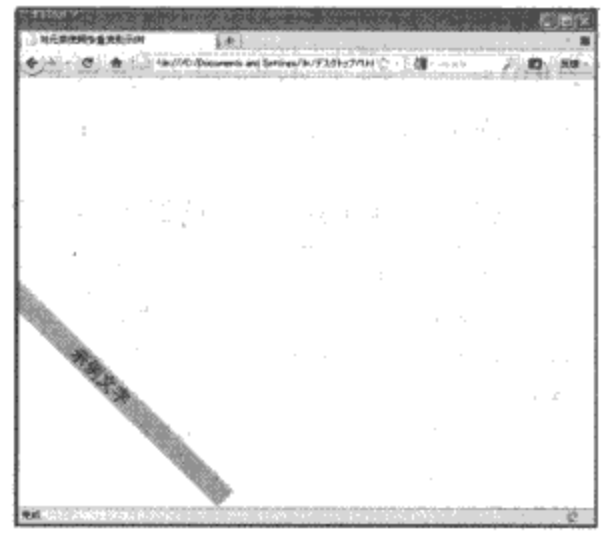

图18-9 对元素使用多重变形示例（先旋转，然后缩放，最后移动）

从两个示例的运行结果中我们可以看出，元素在两个页面上所处位置并不相同，为什么会这样？

首先，我们来详细地看一下代码清单18-5的示例中做变形处理的详细步骤。

1）首先向右移动150px，向下移动200px，如图18-10所示（图中黑点为元素的中心点）。

2）然后旋转45度，并且放大1.5倍，如图18-11所示（图中黑点为元素的中心点）。

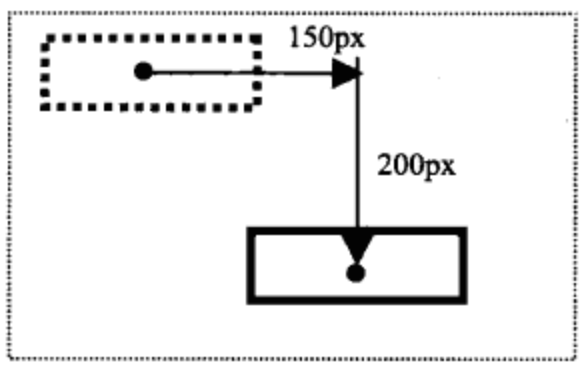

图18-10 元素向右移动150px，向下移动200px

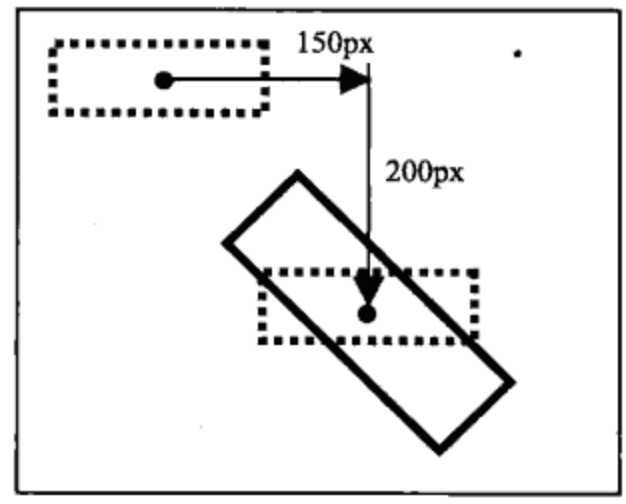

图18-11 元素经过移动后旋转并放大

接下来，我们来详细地看一下代码清单18-6的示例中所做变形处理的详细步骤。

1）首先旋转45度，并且放大1.5倍，如图18-12所示（图中黑点为元素的中心点）。

2）然后向右移动150px，向下移动200px，如图18-13所示。

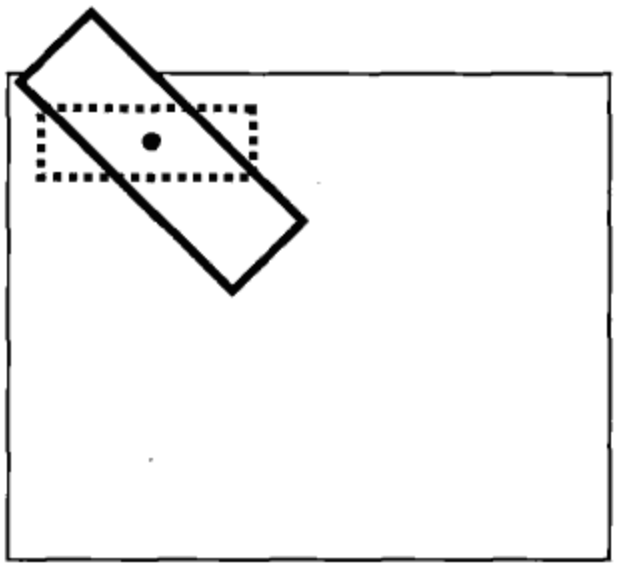

图18-12 元素旋转45度，并且放大1.5倍

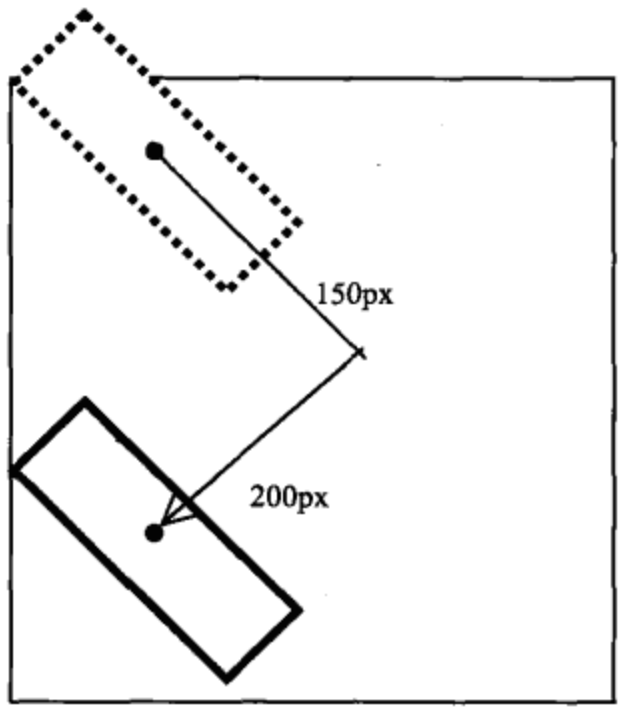

图18-13 元素经过旋转并放大后移动

18.2.2 指定变形的基准点

在使用transform方法进行文字或图像的变形时，是以元素的中心点为基准点进行的。使用transform-origin属性，可以改变变形的基准点。在样式代码中，使用Safari浏览器或Chrome浏览器的时候，需要将其书写成“-webkit-transform-origin”的形式；使用Firefox浏览器的时候，需要将其书写成“-moz-transform-origin”的形式；使用Opera浏览器的时候，需要书写成“-o-transform-origin”的形式。

接下来，我们在代码清单18-7中看一个示例，该示例中有两个div元素。首先我们不改变变形的基准点，并且将第二个div元素进行旋转，然后看一下该示例的运行结果。

代码清单18-7 不改变变形的基准点

```
<!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>不改变变形的基准点</title>
</head>
<style type="text/css">
div{
        width: 200px;
        height:200px;
        display:inline-block;
}
```

```
div#a{
        background-color: pink;
}
div#b{
        background-color: green;
        -webkit-transform: rotate(45deg);
        -moz-transform: rotate(45deg);
        -o-transform: rotate(45deg);
}
</style>
<body>
<div id="a"></div>
<div id="b"></div>
</body>
</html>
```

代码清单18-7的运行结果如图18-14所示。

接下来，我们使用transform-origin属性把变形的基准点修改为第二个元素的左下角处，样式代码如下所示。

```
<style type="text/css">
div{
        width: 200px;
        height:200px;
        display:inline-block;
}
div#a{
        background-color: pink;
}
div#b{
        background-color: green;
        -webkit-transform: rotate(45deg);
        -moz-transform: rotate(45deg);
        -o-transform: rotate(45deg);
        //修改变形基准点
        -webkit-transform-origin: left bottom;
        -moz-transform-origin: left bottom;
        -o-transform-origin: left bottom;
}
</style>
```

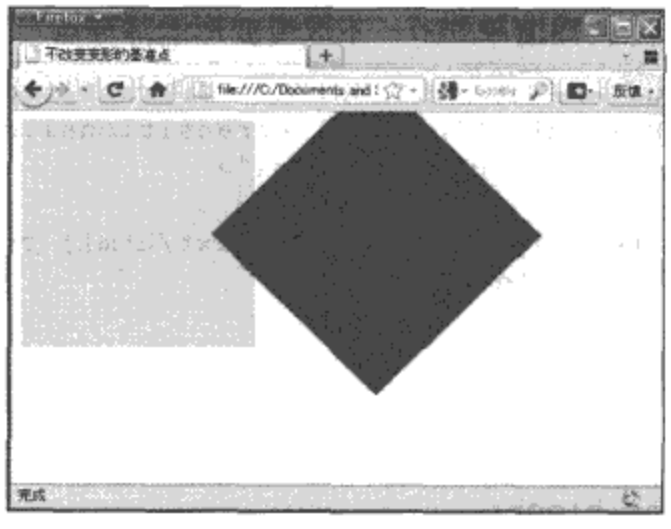

图18-14 不改变变形的基准点

将这段样式代码替换到代码清单18-7的样式代码中，然后重新运行该示例，运行后的结果如图18-15所示。

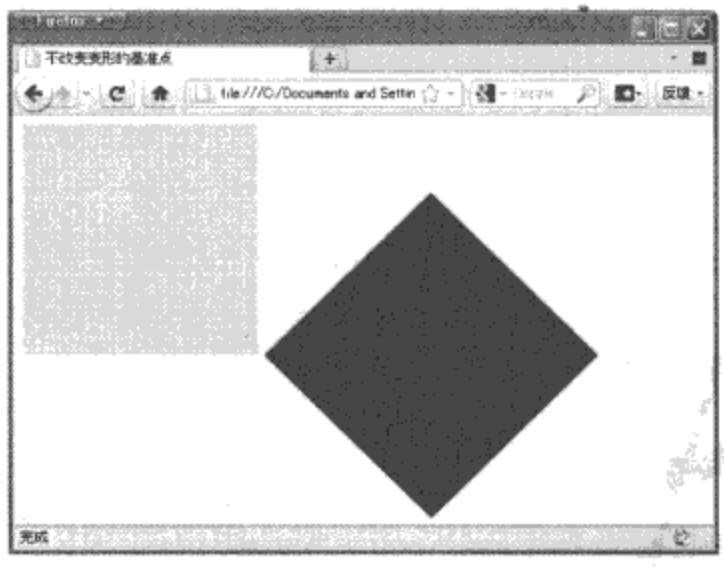

图18-15　修改变形的基准点为元素的左下角处

指定transform-origin属性值的时候，采用“基准点在元素水平方向上的位置，基准点在元素垂直方向上的位置”的方法，其中“基准点在元素水平方向上的位置”中可以指定的值为left、center、right，“基准点在元素垂直方向上的位置”中可以指定的值为top、center、bottom。

第19章
CSS 3中的动画功能

在CSS 3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。

CSS 3中的动画功能分为Transitions功能与Animations功能，这两种功能都可以通过改变CSS中的属性值来产生动画效果。例如，通过改变background-color属性的属性值来让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来在页面上产生更复杂的动画效果。

本章将针对Transitions功能与Animations功能做详细介绍。

学习内容：

- 掌握CSS 3中Transitions功能的使用方法，能够使用Transitions功能来实现在属性值的开始值与属性的结束值之间进行平滑过渡的动画。
- 掌握CSS 3中Animations功能的使用方法，能够在样式中创建多个关键帧，在这些关键帧之中编写样式，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。

19.1 Transitions功能

本节将针对CSS 3中的Transitions功能做详细介绍，到目前为止，Firefox 4以上浏览器、Opera 10以上浏览器、Safari 3.1以上浏览器以及Google Chrome 8以上浏览器都对Transitions功能提供了支持。

19.1.1 Transitions功能的使用方法

在CSS 3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。在样式代码中，如果使用Firefox浏览器，需要将其书写成“-moz-transition”的形式；如果使用Opera浏览器，需要书写成“-o-transition”的形式；如果使用Safari浏览器或Chrome浏览器，需要书写成“-webkit-transition”的形式。

transitions属性的使用方法如下所示。

```
transition: property duration timing-function
```

其中property表示对哪个属性进行平滑过渡，duration表示在多长时间内完成属性值的平滑过渡，timing-function表示通过什么方法来进行平滑过渡。

接下来，我们在代码清单19-1中看一个Transitions功能的使用示例。该页面中有一个div元素，背景色为黄色，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浅蓝色，通过transitions属性指定：当鼠标指针移动到div元素上时，在1秒钟内让div元素的背景色从黄色平滑过渡到浅蓝色。

代码清单19-1 Transitions功能的使用示例

```
<!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>Transitions功能的使用示例</title>
</head>
<style type="text/css">
div{
        background-color: #ffff00;
        -webkit-transition: background-color 1s linear;
        -moz-transition: background-color 1s linear;
        -o-transition: background-color 1s linear;
}
div:hover{
        background-color: #00ffff;
}
</style>
<body>
<div>示例文字</div>
</body>
</html>
```

代码清单19-1所示代码的运行结果如图19-1所示。

图19-1 Transitions功能的使用示例

在CSS 3中，还有另外一种使用Transitions功能的方法，就是将transitions属性中的三个参数改写成transition-property属性、transition-duration属性、transition-timing-function属性，这三个属性的含义及属性值的指定方法与transitions属性中的三个参数的含义及指定方法完全相同，样式代码类似于如下代码。

```
-webkit-transition-property: background-color;
-webkit-transition-duration: 1s;
-webkit-transition-timing-function: linear;
-moz-transition-property: background-color;
-moz-transition-duration: 1s;
-moz-transition-timing-function: linear;
-o-transition-property: background-color;
-o-transition-duration: 1s;
-o-transition-timing-function: linear;
```

19.1.2 使用Transitions功能同时平滑过渡多个属性值

可以使用Transitions功能同时对多个属性值进行平滑过渡。代码清单19-2为使用Transitions功能实现多个属性平滑过渡的示例。该示例中有一个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。

代码清单19-2 使用Transitions功能实现多个属性的平滑过渡

```
<!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>使用Transitions功能实现多个属性的平滑过渡</title>
</head>
<style type="text/css">
div{
        background-color: #ffff00;
        color: #000000;
        width: 300px;
        -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: 400px;
}
</style>
<body>
<div>示例文字</div>
</body>
</html>
```

代码清单19-2中的示例的运行结果分为如下三种情况：

1）当鼠标指针没有停留在div元素上时，页面显示如图19-2所示。

2）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时的页面如图19-3所示。

图19-2 鼠标指针没有停留在div元素上时的页面显示

图19-3 鼠标指针停留在div元素上，div元素的属性值处于变化状态中

3）当鼠标指针停留在div元素上，div元素的几个属性的属性值变化结束后的页面显示如图19-4所示。

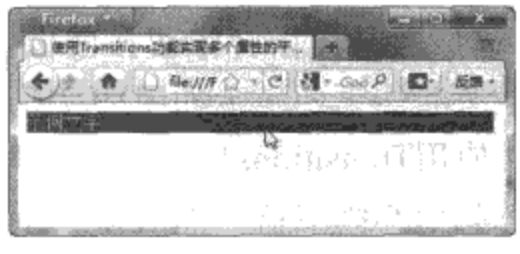

图19-4 鼠标指针停留在div元素上，div元素的属性值已终止变化

另外，可以通过改变元素的位置属性值、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。

代码清单19-3为使用Transitions功能实现元素的移动与旋转动画的一个示例，该示例中有一个div元素，

div元素中有一幅图像，当鼠标指针停留在图像上时，图像会向右移动30px，并且顺时针旋转720度。

代码清单19-3 使用Transitions功能实现元素的移动与旋转动画

```
<!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>使用Transitions功能实现元素的移动与旋转动画</title>
</head>
<style type=" text/css" >
img{
        position: absolute;
        top: 70px;
        left: 0;
        -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: 30px;
        -webkit-transform: rotate(720deg);
        -moz-transform: rotate(720deg);
        -o-transform: rotate(720deg);
}
</style>
<body>
<div>
<img src=" flower-red.png" alt=" *" title="" />
</div>
</body>
</html>
```

代码清单19-3中的示例的运行结果分为如下三种情况：

1）当鼠标指针没有停留在图像上时，页面显示如图19-5所示。

2）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19-6所示。

3）当鼠标指针停留在图像上，图像向右移动和旋转结束后的页面显示如图19-7所示。

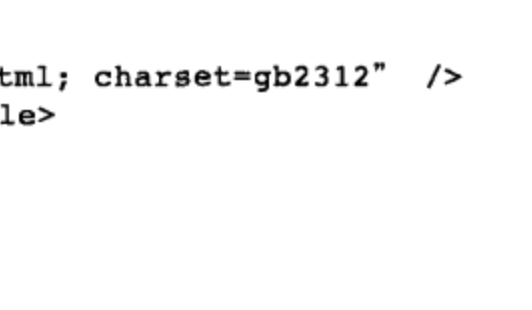

图19-5 鼠标指针没有停留在画像上

使用Transitions功能实现动画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动画效果。在CSS 3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，下一节我们将针对这个Animations功能做详细介绍。

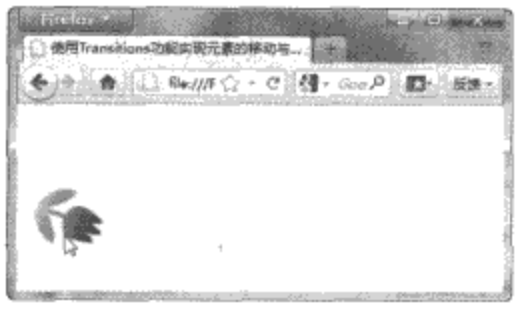

图19-6 鼠标指针停留在图像上，图像正在向右移动和旋转

图19-7 鼠标指针停留在图像上，图像向右移动和旋转结束后的页面显示

19.2 Animations功能

在CSS 3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以上浏览器与Google Chrome 2以上浏览器对该功能提供支持。本节将针对Animations功能做详细介绍。

19.2.1 Animations功能的使用方法

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。它们的区别在于：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。

首先，我们在代码清单19-4中看一个Animations功能的使用示例，该示例中有一个div元素，其背景色为红色，当鼠标指针移动到div元素上时，元素的背景色将经历从红色到深蓝色，从深蓝色到黄色，从黄色回到红色这样一系列的变化。在代码清单19-4之后，我们将结合这个示例对Animations功能的使用方法做详细介绍。

代码清单19-4 Animations功能的使用示例

```
<!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>Animations功能的使用示例</title>
</head>
<style type="text/css">
div{
    background-color: red;
}
@-webkit-keyframes mycolor{
    0%{
            background-color: red;
    }
    40%{
```

```
                background-color: darkblue;
        }
        70%{
                background-color: yellow;
        }
        100%{
                background-color: red;
        }
    }
    div:hover{
            -webkit-animation-name: mycolor;
            -webkit-animation-duration: 5s;
            -webkit-animation-timing-function: linear;
    }
    </style>
    <body>
    <div>
    示例文字
    </div>
    </body>
    </html>
```

在代码清单19-4所实现的动画中带有如下几个关键帧，通过这些关键帧之间的平滑过渡完成了动画的实现。

1. 开始帧

在Chrome浏览器中开始帧的页面显示如图19-8所示。

2. 背景色为深蓝色的关键帧

在整个动画过程的40%处有一帧为背景色是深蓝色的关键帧，在Chrome浏览器中这个关键帧的页面显示如图19-9所示。

3. 背景色为黄色的关键帧

在整个动画过程的70%处有一帧为背景色是黄色的关键帧，在Chrome浏览器中这个关键帧的页面显示如图19-10所示。

图19-8 代码清单19-4中实现动画的开始帧

图19-9 代码清单19-4所示动画中背景色为深蓝色的关键帧

图19-10 代码清单19-4所示动画中背景色为黄色的关键帧

4. 结束帧

整个动画中最后的一帧为结束帧，在结束帧之后，元素的属性不再发生变化。在代码清单19-4的示例中，动画的结束帧与开始帧的页面显示完全相同，背景色都是红色。

使用Animations功能的时候，如果使用的是Safari浏览器或Chrome浏览器，会使用如下所示的方法来创建关键帧的集合。

```
@-webkit-keyframes 关键帧集合名{创建关键帧的代码}
```

在代码清单19-4的示例中，关键帧集合的名称是mycolor。

创建关键帧的代码类似如下所示。

```
40%{
        本关键帧中的样式代码
}
```

这里的“40%”表示该帧位于整个动画过程中的40%处，开始帧为0%，结束帧为100%。在代码清单19-4的示例中，除了开始帧与结束帧外，在整个动画的40%处与70%处创建了两个关键帧。在表示过程百分比后的中括号中书写各关键帧中的样式代码，在代码清单19-4中通过在关键帧中设定div元素不同的背景色来完成三种背景色之间的平滑过渡。

关键帧的集合创建好之后，在元素的样式中使用该关键帧的集合。代码类似如下所示。

```
div:hover{
        -webkit-animation-name: mycolor;
        -webkit-animation-duration: 5s;
        -webkit-animation-timing-function: linear;
}
```

在animation-name属性中指定关键帧集合的名称，在使用Safari浏览器或Chrome浏览器时，需要在Animations功能所使用到的各属性前加上“-webkit-”前缀。在animation-duration属性中指定完成整个动画所花费的时间，在animation-timing-function属性中指定实现动画的方法。

19.2.2 实现多个属性值同时改变的动画

如果要想实现让多个属性值同时变化的动画，只需在各关键帧中同时指定这些属性值就可以了。代码清单19-5给出了让多个属性值同时变化的动画示例，该示例由代码清单19-4修改而来，在动画中不仅完成了三种背景色之间的平滑过渡，而且在背景色为深蓝色的关键帧中，让div元素顺时针旋转了30度，在背景色为黄色的关键帧中，让div元素逆时针旋转了30度。

代码清单19-5 让多个属性值同时变化

```
<!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>让多个属性值同时变化</title>
</head>
<style type="text/css">
div{
    position: absolute;
    background-color: yellow;
    top:100px;
```

```
        width:500px;
    }
    @-webkit-keyframes mycolor{
        0%{
                background-color: red;
                -webkit-transform: rotate(0deg);
        }
        40%{
                 background-color: darkblue;
                 -webkit-transform: rotate(30deg);
        }
        70%{
                 background-color: yellow;
                 -webkit-transform: rotate(-30deg);
        }
        100%{
                 background-color: red;
                 -webkit-transform: rotate(0deg);
        }
    }
    div:hover{
            -webkit-animation-name: mycolor;
            -webkit-animation-duration: 5s;
            -webkit-animation-timing-function: linear;
    }
    </style>
    <body>
    <div>
    示例文字
    </div>
    </body>
    </html>
```

在Chrome浏览器中，代码清单19-5中开始帧与结束帧的页面显示如图19-11所示。

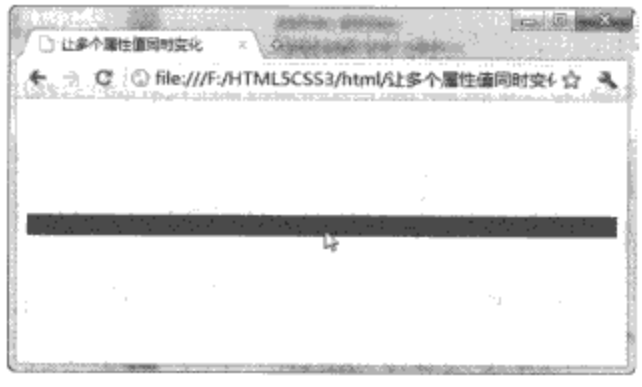

图19-11　代码清单19-5中开始帧与结束帧的页面显示

在Chrome浏览器中，代码清单19-5中背景色为深蓝色的关键帧的页面显示如图19-12所示。

在Chrome浏览器中，代码清单19-5中背景色为黄色的关键帧的页面显示如图19-13所示。

在元素的样式代码中，可以通过animation-iteration-count属性来指定动画的播放次数，也可通过对该属性指定infinite属性值来让动画不停地循环播放。将鼠标指针停留在div元素

上时的样式修改为如下所示的代码，动画将不停地循环播放。

```
div:hover{
        -webkit-animation-name: mycolor;
        -webkit-animation-duration: 5s;
        -webkit-animation-timing-function: linear;
        -webkit-animation-iteration-count: infinite;
}
```

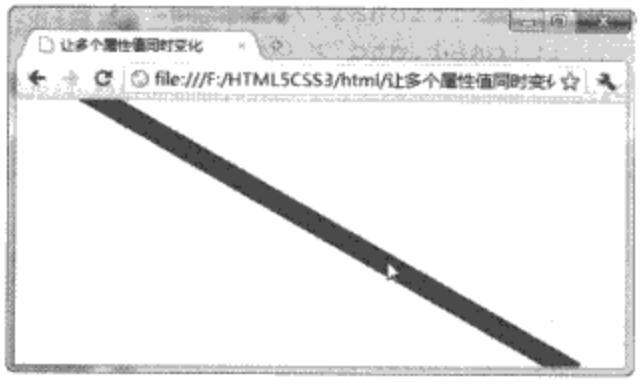

图19-12 代码清单19-5中背景色为深蓝色的关键帧的页面显示

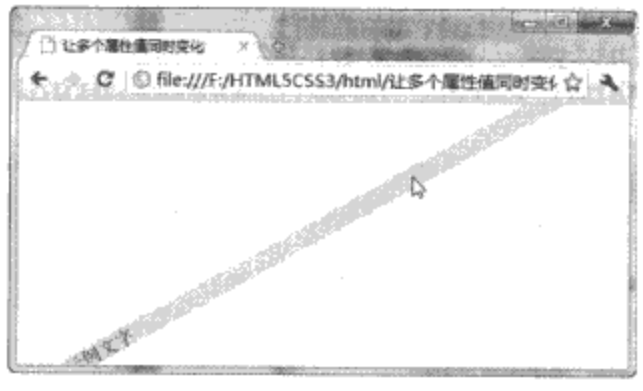

图19-13 代码清单19-5中背景色为黄色的关键帧的页面显示

如果将animation-iteration-count属性的属性值设定为某个整数值，则动画播放的次数就等于该整数值。

如果去除鼠标指针停留在div元素上时的样式，并把样式中的代码改写为div元素本身的样式，成为如下所示的代码，则动画将在页面打开时进行播放。

```
div{
        -webkit-animation-name: mycolor;
        -webkit-animation-duration: 5s;
        -webkit-animation-timing-function: linear;
        -webkit-animation-iteration-count: infinite;
}
```

19.2.3 实现动画的方法

在前面几个Animations功能的使用示例中，我们只使用到了一种实现动画的方法——linear。linear方法的含义是在动画从开始到结束时使用同样的速度进行各种属性值的改变，在一个动画过程中不改变各种属性值的改变速度。除了linear方法外，还有其他几种实现动画的方法，如表19-1所示。

表19-1 Animations功能中实现动画的方法

方法	属性值的变化速度
linear	在动画开始时到结束时以同样速度进行改变
ease-in	动画开始时速度很慢，然后速度沿曲线值进行加快
ease-out	动画开始时速度很快，然后速度沿曲线值进行放慢
ease	动画开始时速度很慢，然后速度沿曲线值进行加快，然后再沿曲线值放慢
ease-in-out	动画开始时速度很慢，然后速度沿曲线值进行加快，然后再沿曲线值放慢

接下来，我们可以通过代码清单19-6看出Animations功能中各种实现动画的方法的区别。该示例中有一个div元素，页面打开时，该div元素在5秒钟内从长100px、宽100px扩大到长500px、宽500px，通过改变animation-timing-function属性的属性值，然后观察div元素的长度与宽度在整个动画中的变化速度，可以看出实现动画的各种方法之间的区别。

代码清单19-6 实现动画的各种方法的比较示例

```
<!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>实现动画的各种方法的比较示例</title>
</head>
<style type="text/css">
@-webkit-keyframes mycolor{
    0%{
            width:100px;
            height:100px;
    }
    100%{
             width:500px;
             height:500px;
    }
}
div{
        background-color: red;
        width:500px;
        height:500px;
        -webkit-animation-name: mycolor;
        -webkit-animation-duration: 5s;
        -webkit-animation-timing-function: ease-out;
}
</style>
<body>
<div>
</div>
</body>
</html>
```

代码清单19-6的运行结果如图19-14所示。

图19-14 实现动画的各种方法的比较示例

19.2.4 实现网页的淡入效果

在本节的最后，介绍一下如何使用Animations功能来实现网页设计中的一种经常使用的动画效果——网页的淡入效果。在代码清单19-7中给出一个实现网页淡入效果的示例，该示例的页面很简单，只有几个文字，通过在开始帧与结束帧中改变页面的opacity属性的属性值来实现页面的淡入效果。

代码清单19-7 实现网页淡入效果的示例

```
<!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>实现网页淡入效果的示例</title>
</head>
<style type="text/css">
@-webkit-keyframes fadein{
    0%{
        opacity: 0;
        background-color: white;
    }
    100%{
        opacity: 1;
        background-color: white;
    }
}
body{
        -webkit-animation-name: fadein;
        -webkit-animation-duration: 5s;
        -webkit-animation-timing-function: linear;
        -webkit-animation-iteration-count: 1;
}
</style>
<body>
示例文字
</body>
</html>
```

第20章
布局相关样式

Web页面中的布局，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CSS 3之前，主要使用float属性或position属性进行页面中的简单布局，但是使用它们也存在一些缺点，譬如两栏或多栏中如果元素的内容高度不一致则有底部很难对齐的问题。因此，在CSS 3中追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，还可以进行更为便捷、更为复杂的页面布局。

本章将针对CSS 3中的布局做详细介绍，主要介绍多栏布局与盒布局。到目前为止，这两种布局方式得到了Firefox浏览器、Safari浏览器以及Chrome浏览器的支持。

学习内容：

- 掌握CSS 3中多栏布局的使用方法，知道为什么要使用多栏布局，它可以解决使用float属性或position属性时出现的哪些问题。
- 掌握CSS 3中盒布局的使用方法，知道为什么要使用盒布局，它可以解决使用float或position属性时出现的哪些问题，盒布局与多栏布局有什么区别，什么场合下应该使用盒布局，什么场合应该使用多栏布局。
- 掌握CSS中弹性盒布局的基本概念及使用方法，能够指定容器中元素水平方向或垂直方向上的排列方式，能够将容器中元素的宽度与高度指定为容器的宽度与高度,能够使用box-pack属性以及box-align属性将子元素放置在父元素的中央。

20.1 多栏布局

20.1.1 使用float属性或position属性的缺点

本节将针对CSS 3中的多栏布局做详细介绍，在介绍多栏布局之前，先来回顾一下CSS 3之前是如何使用float属性或position属性进行页面中的简单布局的。代码清单20-1是使用float属性进行页面布局的一个示例。

代码清单20-1 使用float属性进行页面布局的示例

```
<!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>使用float属性进行页面布局的示例</title>
<style type="text/css">
div{
    width: 20em;
    float:left;
}
div#div1{
        margin-right:2em;
}
div#div3{
        width:100%;
        background-color:yellow;
        height:200px;
}
```

```
</style>
</head>
<body>
<div id="div1">
<p>示例文字1。相对来说比较长的示例文字。示例文字。相对来说比较长的示例
文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的示
例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的
示例文字。示例文字。</p>
</div>
<div id="div2">
<p>示例文字2。相对来说比较长的示例文字。示例文字。相对来说比较长的示
例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的
示例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长
的示例文字。示例文字。</p>
</div>
<div id="div3">
页面中其他内容
</div>
</body>
</html>
```

这段代码的运行结果如图20-1所示。

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，就是第一个div元素与第二个div元素是各自独立的，因此如果在第一个div元素中加入一些内容的话，将会使得两个元素的底部不能对齐，导致页面中多出一块空白区域。譬如在代码清单20-1的第一个div元素的开头加上一幅图像，不改变p元素中的文字内容，代码如下所示。

```
<div id="div1">
<img src="test.jpg">
<p>示例文字1。相对来说比较长的示例文字。示例文字。相对来说比较长的示例
文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的示
例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的
示例文字。示例文字。</p>
</div>
```

将以上这段代码替换到代码清单20-1的定义第一个div元素的html代码处，然后重新运行该示例，结果如图20-2所示。

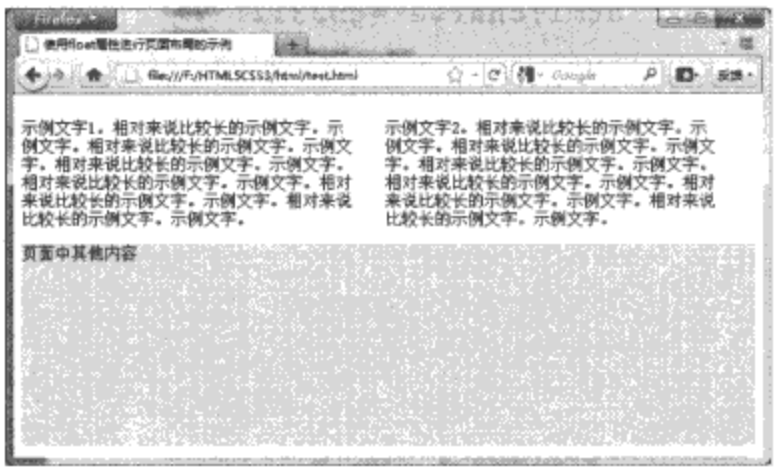

图20-1 使用float属性进行页面布局的示例

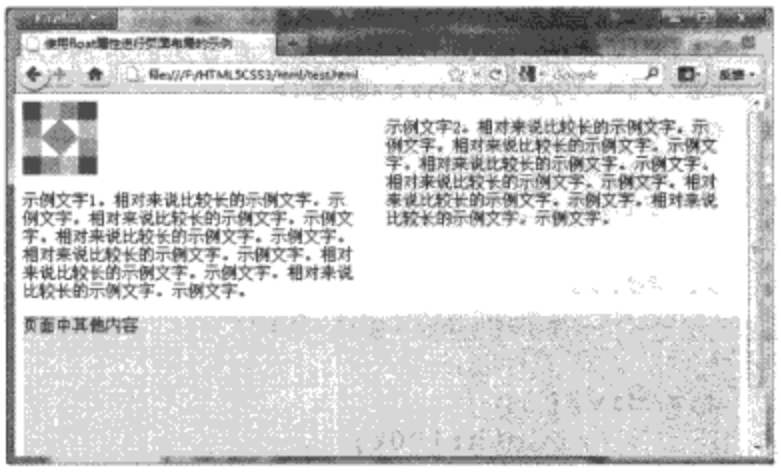

图20-2 在第一个div元素的第一段中插入一幅图像

20.1.2 使用多栏布局方式

针对代码清单20-1的示例中所展示的使用float属性或position属性时的缺点，在CSS 3中加入了多栏布局方式。使用多栏布局可以将一个元素中的内容分为两栏或多栏显示，并且确保各栏中内容的底部对齐。代码20-2为一个多栏布局方式的使用示例，该示例由代码清单20-1修改而来，可以查看插入图像后左右两栏的底部是否对齐。

代码清单20-2 多栏布局方式的使用示例

```
<!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>多栏布局方式的使用示例</title>
<style type="text/css">
div#div1{
        width:40em;
        -moz-column-count: 2;
        -webkit-column-count: 2;
}
div#div3{
        width:100%;
        background-color:yellow;
        height:200px;
}
</style>
</head>
<body>
<div id="div1">
<img src="test.jpg">
<p>示例文字1。相对来说比较长的示例文字。示例文字。相对来说比较长的示例
文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的示
例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的
示例文字。示例文字。</p>
<p>示例文字2。相对来说比较长的示例文字。示例文字。相对来说比较长的示
例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的
示例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长
的示例文字。示例文字。</p>
</div>
<div id="div3">
页面中其他内容
</div>
</body>
</html>
```

这段代码运行结果如图20-3所示。

在CSS 3中，通过column-count属性来使用多栏布局方式，该属性的含义是将一个元素中的内容分为多栏进行显示。在Firefox浏览器中，需要将其书写成“-moz-column-count”的形式；在Safari浏览器或Chrome浏览器中，需要书写成“-webkit-column-count”的形式。

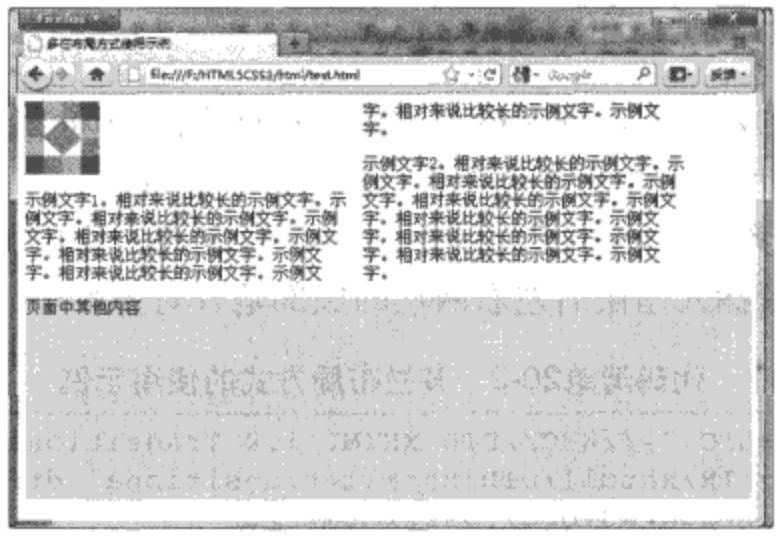

图20-3 多栏布局方式的使用示例

使用多栏布局的时候，需要将元素的宽度设置成多个栏目的总宽度，它与使用float属性和position属性时的区别是：使用两个属性时只需单独设定每个元素的宽度即可，而使用多栏布局时需要设定元素中多个栏目相加后的总的宽度。

我们也可以使用column-width属性单独设置每一栏的宽度而不设定元素的宽度。在Firefox浏览器中，需要将其书写成“-moz-column-width”的形式；在Safari浏览器或Chrome浏览器中，需要书写成“-webkit-column-width”的形式。具体代码书写方法类似如下所示。

```
div#div1{
            -moz-column-count: 2;
            -webkit-column-count: 2;
            -moz-column-width:20em;
            -webkit-column-width:20em;
}
```

另外，笔者通过实验发现，到目前为止，如果在最新的Firefox 4浏览器中使用-moz-column-width属性指定每栏宽度而不设定元素的宽度的话，则需要在元素外面单独设立一个容器元素，然后指定该容器元素的宽度，即改成代码清单20-3中的代码，否则指定的每栏宽度被浏览器视为未设定，在Safari浏览器或Chrome浏览器中则不需要做此修改。

代码清单20-3 在Firefox浏览器中设定每栏宽度

```
<!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>在Firefox浏览器中设定每栏宽度</title>
<style type="text/css">
div#container{
    width:42em;
}
div#div1{
            -moz-column-count: 2;
            -webkit-column-count: 2;
            -moz-column-width:20em;
```

```
            -webkit-column-width:20em;
}
div#div3{
            width:100%;
            background-color:yellow;
            height:200px;
}
</style>
</head>
<body>
<div id="container">
<div id="div1">
<img src="test.jpg">
<p>示例文字1。相对来说比较长的示例文字。示例文字。相对来说比较长的示例
文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的示
例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的
示例文字。示例文字。</p>
<p>示例文字2。相对来说比较长的示例文字。示例文字。相对来说比较长的示
例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长的
示例文字。示例文字。相对来说比较长的示例文字。示例文字。相对来说比较长
的示例文字。示例文字。</p>
</div>
<div id="div3">
页面中其他内容
</div>
</div>
</body>
</html>
```

这段代码的运行结果如图20-4所示。

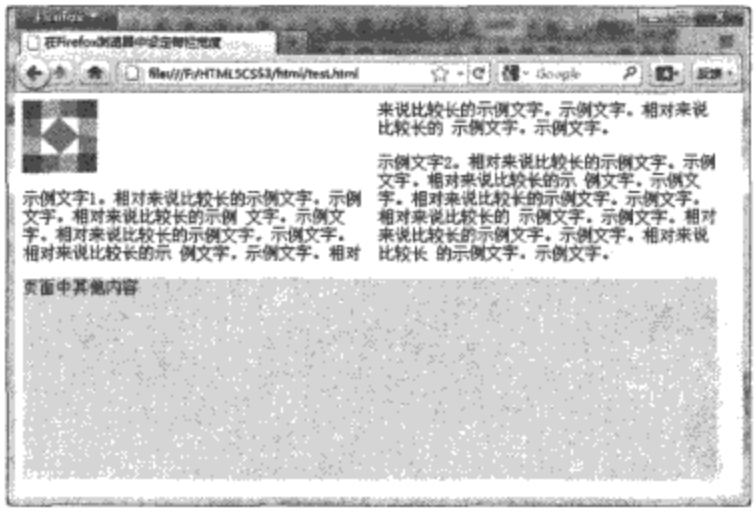

图20-4 在Firefox浏览器中设定每栏宽度

可以使用column-gap属性来设定多栏之间的间隔距离。在Firefox浏览器中，需要将其书写成“-moz-column-gap”的形式；在Safari浏览器或Chrome浏览器中，需要书写成“-webkit-column-gap”的形式。将代码清单20-3中设定div元素的样式代码修改为如下所示的样式代码（增加两栏之间的间隔距离），修改后重新运行该示例，运行结果如图20-5所示。

```
div#div1{
            -moz-column-count: 2;
            -webkit-column-count: 2;
```

```
            -moz-column-width:20em;
            -webkit-column-width:20em;
            -moz-column-gap:2em;
            -webkit-column-gap:2em;
}
```

可以使用column-rule属性在栏与栏之间增加一条间隔线，并且设定该间隔线的宽度、颜色等，该属性的属性值的指定方法与CSS中的border属性的属性值指定方法相同。将代码清单20-3中设定div元素的样式代码修改为如下所示的样式代码，修改后重新运行该示例，页面中两栏之间将增加一条红色间隔线，宽度为1px，如图20-6所示。

```
div#div1{
            -moz-column-count: 2;
            -webkit-column-count: 2;
            -moz-column-width:20em;
            -webkit-column-width:20em;
            -moz-column-gap:2em;
            -webkit-column-gap:2em;
            -moz-column-rule: 1px solid red;
            -webkit-column-rule: 1px solid red;
}
```

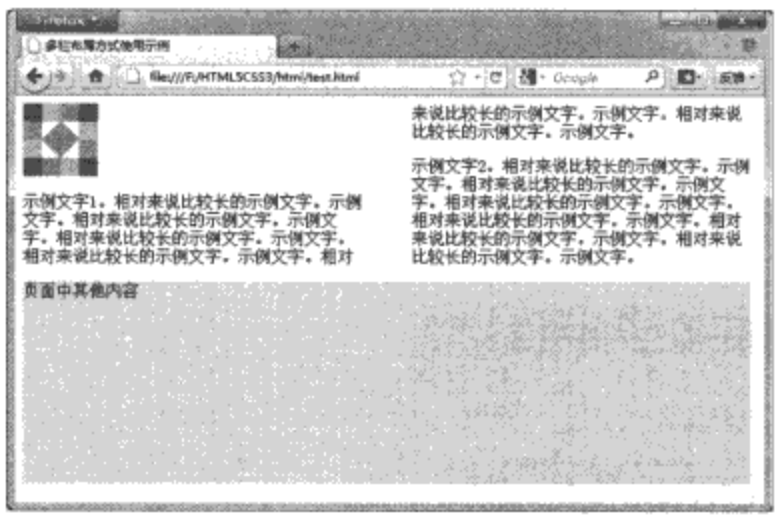

图20-5 使用column-gap属性来设定多栏之间的间隔距离

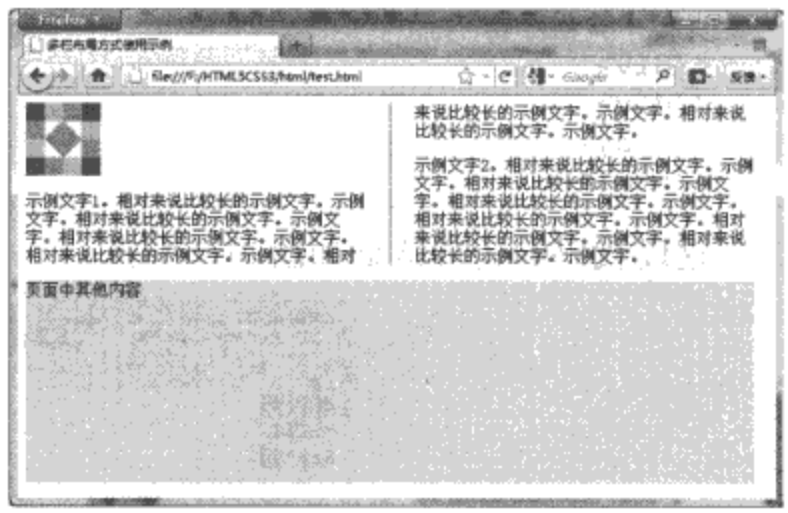

图20-6 在栏与栏之间增加间隔线

20.2 盒布局

20.2.1 盒布局的基础知识

1. 使用float属性或position属性时的缺点

在CSS 3中，除了多栏布局之外，还可以使用盒布局解决前面所说的使用float属性或position属性时左右两栏或多栏中底部不能对齐的问题。

接下来，我们在代码清单20-4中看一个使用float属性进行布局的示例，该示例中有三个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。

代码清单20-4　使用float属性进行布局的示例

```
<!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>使用float属性进行布局的示例</title>
<style type="text/css">
#left-sidebar{
        float: left;
        width: 200px;
        padding: 20px;
        background-color: orange;
}
#contents{
        float: left;
        width: 300px;
        padding: 20px;
        background-color: yellow;
}
#right-sidebar{
        float: left;
        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>左侧边栏</h2>
<ul>
<li><a href="">超链接</a></li>
<li><a href="">超链接</a></li>
<li><a href="">超链接</a></li>
<li><a href="">超链接</a></li>
<li><a href="">超链接</a></li>
</ul>
</div>
<div id="contents">
<h2>内容</h2>
<p>示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字。示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字。示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字。示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字示例文字。</p>
</div>
<div id="right-sidebar">
<h2>右侧边栏</h2>
```

```
<ul>
<li><a href="">超链接</a></li>
<li><a href="">超链接</a></li>
<li><a href="">超链接</a></li>
</ul>
</div>
</div>
</body>
</html>
```

代码清单20-4的运行结果如图20-7所示。

从图20-7中我们可以看出，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。

2. 使用盒布局

如果改为使用盒布局的话，这个问题将很容易得到解决。在CSS 3中，通过box属性来使用盒布局，在Firefox浏览器中，需要将此属性书写成“-moz-box”的形式；在Safari浏览器或Chrome浏览器中，需要书写成“-webkit-box”的形式。

在代码清单20-4中示例最外层的id为“container”的div元素的样式中使用了box属性，并去除了代表左侧边栏（id为left-sidebar）、中间内容（id为contents）、右侧边栏（id为right-sidebar）的div元素的样式中的float属性，修改代码如下所示。

```
<style type="text/css">
#container{
    display: -moz-box;
    display: -webkit-box;
}
#left-sidebar{
    width: 200px;
    padding: 20px;
    background-color: orange;
}
#contents{
    width: 300px;
    padding: 20px;
    background-color: yellow;
}
#right-sidebar{
    width: 200px;
    padding: 20px;
    background-color: limegreen;
}
#left-sidebar, #contents, #right-sidebar{
     -moz-box-sizing: border-box;
     -webkit-box-sizing: border-box;
}
</style>
```

将以上这段样式代码替换到代码清单20-4的示例样式代码中，然后重新运行该示例，运行结果如图20-8所示。

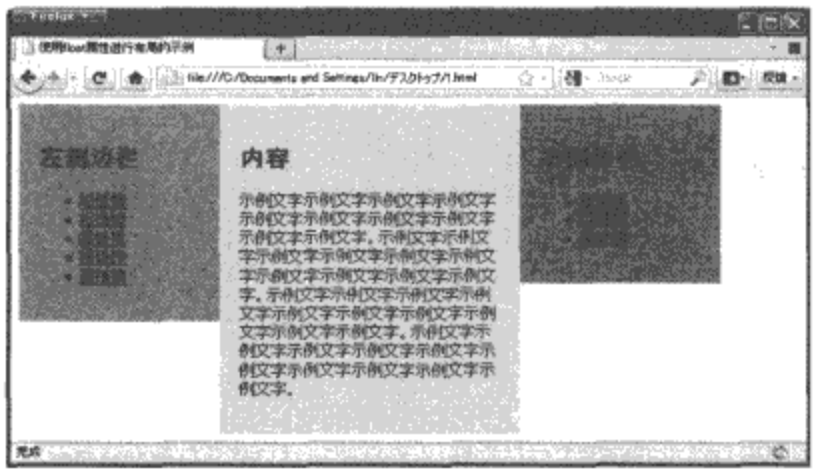

图20-7　使用float属性进行布局的示例

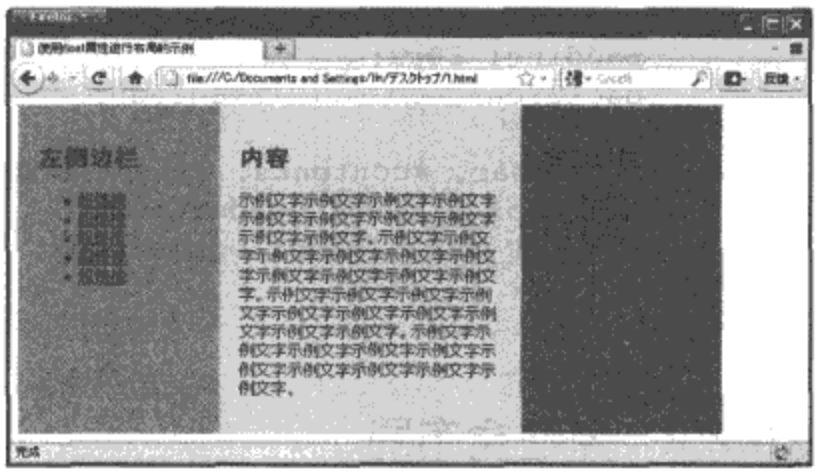

图20-8　盒布局使用示例

3. 盒布局与多栏布局的区别

盒布局与多栏布局的区别在于：使用多栏布局时，各栏宽度必须是相等的，在指定每栏宽度时，也只能为所有栏指定一个统一的宽度，栏与栏之间的宽度不可能是不一样的。另外，使用多栏布局时，也不可能具体指定什么栏中显示什么内容，因此比较适合使用在显示文章内容的时候，不适合用于安排整个网页中由各元素组成的网页结构时。

如果将代码清单20-4中示例最外层的id为“container”的div元素的样式改为通过column-count属性来使用多栏布局，并去除代表左侧边栏、中间内容、右侧边栏的id分别为“left-sidebar”、“contents”、“right-sidebar”的div元素的样式的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，则运行结果将如图20-9所示。

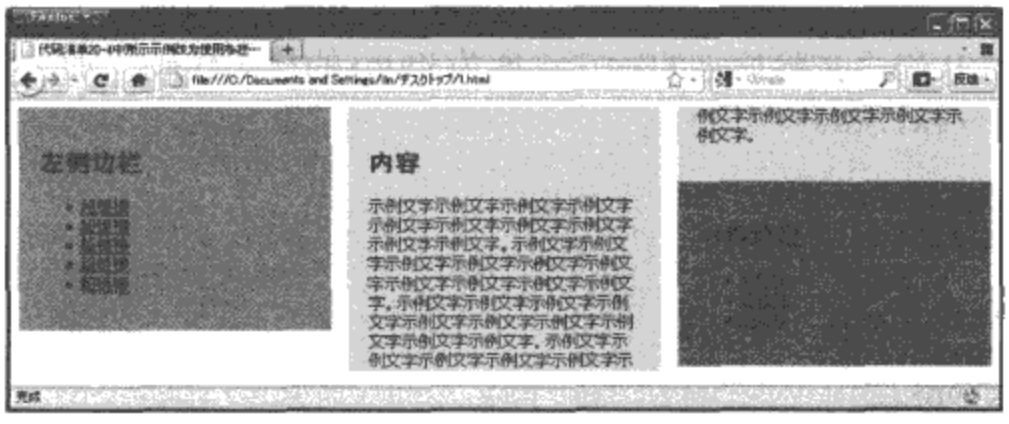

图20-9　通过修改代码清单20-4的示例，改为使用多栏布局

```
<style type="text/css">
#container{
    -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;
}
#left-sidebar, #contents, #right-sidebar, #footer{
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
}
</style>
```

20.2.2 弹性盒布局

1. 使用自适应窗口的弹性盒布局

在20.2.1节介绍的盒布局中，我们对代表左侧边栏、中间内容、右侧边栏的三个div元素的宽度都进行了设定，如果我们想让这三个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，我们需要使用包括负外边距（margin）在内的比较复杂的指定方法才能够达到这个要求，但是如果使用盒布局的话，我们只要使用一个box-flex属性，使盒布局变为弹性盒布局就可以了。在样式代码中，如果使用Firefox浏览器，需要将其书写成“-moz-box-flex”的形式；如果使用Safari浏览器或Chrome浏览器，需要书写成“-webkit-box”的形式。

针对代码清单20-4中的示例，将样式代码修改为如下所示的样式代码，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200px，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300px的样式代码，加入box-flex属性。

```
<style type="text/css">
#container{
    display: -moz-box;
    display: -webkit-box;
}
#left-sidebar{
    width: 200px;
    padding: 20px;
    background-color: orange;
}
#contents{
    -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{
      -moz-box-sizing: border-box;
      -webkit-box-sizing: border-box;
```

```
}
</style>
```

将上面这段样式代码替换到代码清单20-4中，然后重新运行该示例，运行结果如图20-10所示。

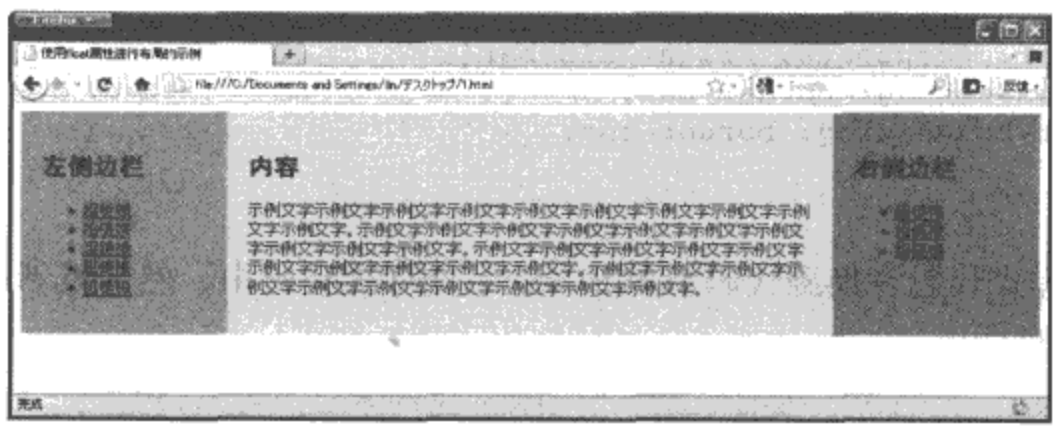

图20-10　弹性盒布局的使用示例

2. 改变元素的显示顺序

使用弹性盒布局的时候，可以通过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。在样式代码中，如果使用Firefox浏览器，需要将其书写成“-moz-box-ordinal-group”的形式；如果使用Safari浏览器或Chrome浏览器，需要书写成“-webkit-box-ordinal-group”的形式。

例如，针对代码清单20-4中的示例，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、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，以观察元素显示顺序是否会发生改变。

```
<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>
```

将上面这段样式代码替换到代码清单20-4的示例样式代码中，然后重新运行该示例，运行后结果如图20-11所示。

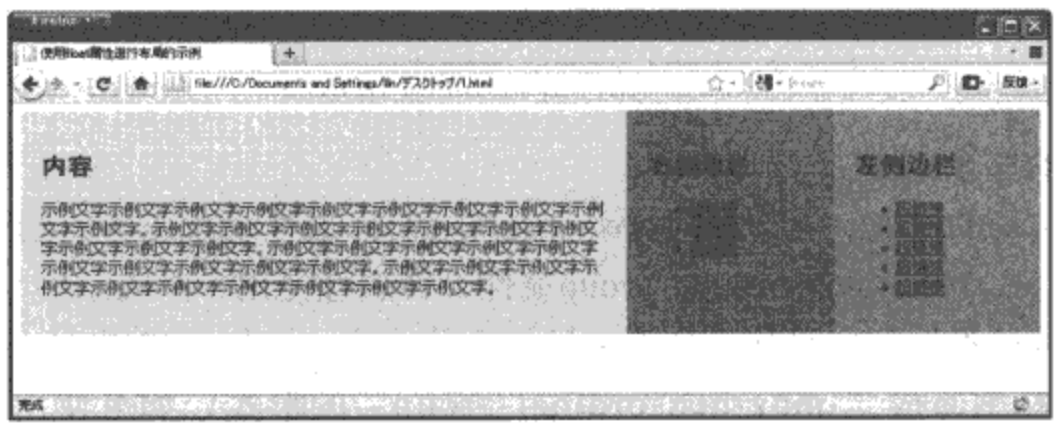

图20-11 改变元素的显示顺序

从图20-11可以看出，虽然没有改变HTML 5的页面代码，但是通过使用弹性盒布局，使用-moz-box-ordinal-group属性，我们同样可以改变元素的显示顺序，这样可以大大提高页面布局时的工作效率。

3. 改变元素的排列方向

使用弹性盒布局的时候，我们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CSS 3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，需要将其书写成“-moz-box-orient”的形式；如果使用Safari浏览器或Chrome浏览器，需要书写成“-webkit-box-orient”的形式。

例如，针对代码清单20-4中的示例，可以将该示例中的样式代码修改为如下所示的样式代码，在多个div元素的容器元素——id为container的div元素中加入box-orient属性，并设定属性值为vertical（表示垂直方向排列），则代码示例中代表左侧边栏，中间内容，右侧边栏的三个div元素的排列方向将从水平方向改变为垂直方向，如图20-12所示。

```
<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>
```

图20-12　改变元素的排列方向

4. 元素宽度与高度的自适应

使用盒布局的时候，元素的宽度与高度具有自适应性，即元素的宽度与高度可以根据排

列方向的改变而改变。通过代码清单20-5的示例，我们可以很清楚地看出这个特性。该示例中有一个容器元素，元素中有三个div元素，它们只对容器元素指定了宽度与高度，从运行结果中我们可以看出，当排列方向被指定为水平方向时，三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，三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。

代码清单20-5 元素宽度与高度的自适应示例

```
<!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>元素宽度与高度的自适应示例</title>
<style type="text/css">
#container{
            display: -moz-box;
            display: -webkit-box;
            border: solid 5px blue;
            -moz-box-orient: horizontal;
            -webkit-box-orient: horizontal;
            width: 500px;
            height: 300px;
}
#text-a{
    background-color: orange;
}
#text-b{
    background-color: yellow;
}
#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">示例文字A</div>
<div id="text-b">示例文字B</div>
<div id="text-c">示例文字C</div>
</div>
</body>
</html>
```

这段示例中元素的排列方向被设定为水平方向，运行结果如图20-13所示。

在代码清单20-5的示例中，不修改其他代码，只在容器元素的样式代码中把排列方向改变为垂直方向，代码如下所示。

```
#container{
    display: -moz-box;
    display: -webkit-box;
    border: solid 5px blue;
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
    width: 500px;
    height: 300px;
}
```

将以上这段代码替换到代码清单20-5的示例中，然后重新运行该示例，运行结果如图20-14所示。

从图20-13与图20-14中我们可以看出，虽然使用盒布局的时候，元素的高度与宽度具有一定程度的适应性，但是容器中总还是会留出一大片空白的区域，应该要怎么样来消除这块空白区域呢？

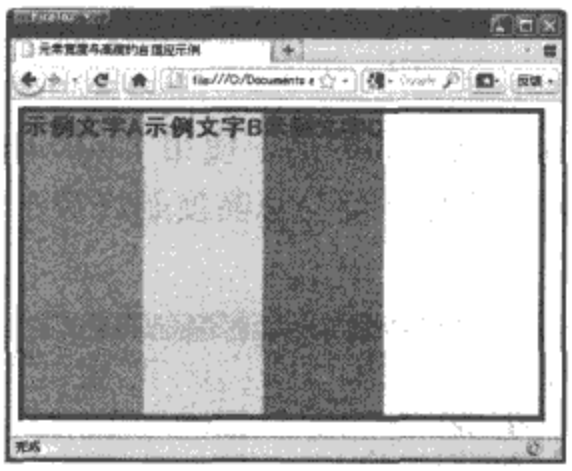

图20-13 元素宽度与高度的自适应示例（水平方向排列）

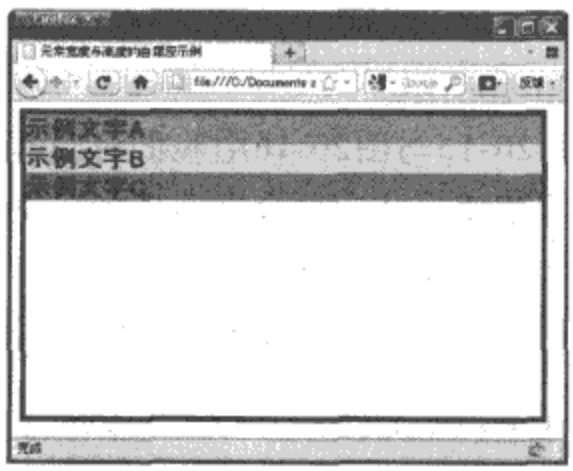

图20-14 元素宽度与高度的自适应示例（垂直方向排列）

5. 使用弹性盒布局来消除空白

针对代码清单20-5的示例中所产生的问题，我们可以改用弹性盒布局来解决，使得多个参与排列的元素的总宽度与总高度始终等于容器的宽度和高度。

将代码清单20-5的示例中的样式代码修改如下，将排列方向设定为水平方向，在中间的一个div子元素的样式代码中加入box-flex属性。

```
<style type="text/css">
#container{
            display: -moz-box;
            display: -webkit-box;
            border: solid 5px blue;
            -moz-box-orient: horizontal;
            -webkit-box-orient:horizontal;
            width: 500px;
            height: 300px;
}
#text-a{
```

```
        background-color: orange;
}
#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>
```

将上面这段样式代码替换到代码清单20-5的示例中，然后重新运行该示例，运行结果如图20-15所示。

在容器元素的样式代码中把排列方向修改为垂直方向，修改后重新运行该示例，运行结果如图20-16所示。

从图20-15与图20-16中我们可以看出，如果使用弹性盒布局的话，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。

图20-15 使用弹性盒布局来修改元素宽度与高度的自适应性（水平方向排列）

图20-16 使用弹性盒布局来修改元素宽度与高度的自适应性（垂直方向排列）

6. 对多个元素使用box-flex属性

前面我们所讲的示例中，都是只对一个元素使用box-flex属性，使其宽度和高度自动扩大，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。在CSS 3中，也可以对多个元素使用box-flex属性，例如我们可以把代码清单20-5的示例中容器的前两个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。

```
#container{
        display: -moz-box;
        display: -webkit-box;
```

```
        border: solid 5px blue;
        -moz-box-orient: vertical;
        -webkit-box-orient: vertical;
        width: 500px;
        height: 300px;
}
#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;
}
```

修改后重新运行该示例，从运行结果中我们可以看出，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，而第三个div元素的高度仍保持为元素内容的高度，如图20-17所示。

如果三个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，如图20-18所示。

图20-17 对前两个div元素使用box-flex属性

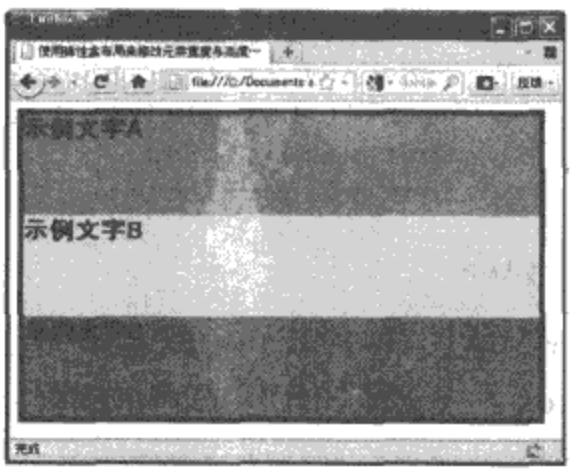

图20-18 对三个div元素都使用box-flex属性

到现在为止，我们在样式中所使用的box-flex属性的属性值都一直是1，到底这个box-flex属性代表了什么含义？如果某个div元素的box-flex属性值大于1（譬如为2）时，页面显示会是什么情况？

首先，我们将代码清单20-5中示例的样式代码修改为如下所示的样式代码，修改容器高度为200px，在每个div子元素的样式代码中均使用box-flex属性，但是将第一个div元素的box-flex属性设定为2，其他两个div子元素的box-flex属性仍保留为1，元素排列方向为垂直排列，修改完毕后重新运行该示例，运行结果如图20-19所示。

图20-19 将第一个div子元素的box-flex属性设定为2

```
<style type="text/css">
#container{
        display: -moz-box;
        display: -webkit-box;
        border: solid 5px blue;
        -moz-box-orient: vertical;
        -webkit-box-orient: vertical;
        width: 500px;
        height: 200px;
}
#text-a{
        background-color: orange;
        -moz-box-flex: 2;
        -webkit-box-flex: 2;
}
#text-b{
        background-color: yellow;
        -moz-box-flex: 1;
        -webkit-box-flex: 1;
}
#text-c{
        background-color: limegreen;
        -moz-box-flex: 1;
        -webkit-box-flex: 1;
}
#text-a, #text-b, #text-c{
        -moz-box-sizing: border-box;
        -webkit-box-sizing: border-box;
        font-size: 1.5em;
        font-weight: bold;
}
</style>
```

首先，从图20-19我们可以看出，第一个div子元素的高度并不等于其他两个div子元素的两倍。box-flex属性的属性值的正确含义如图20-20所示。

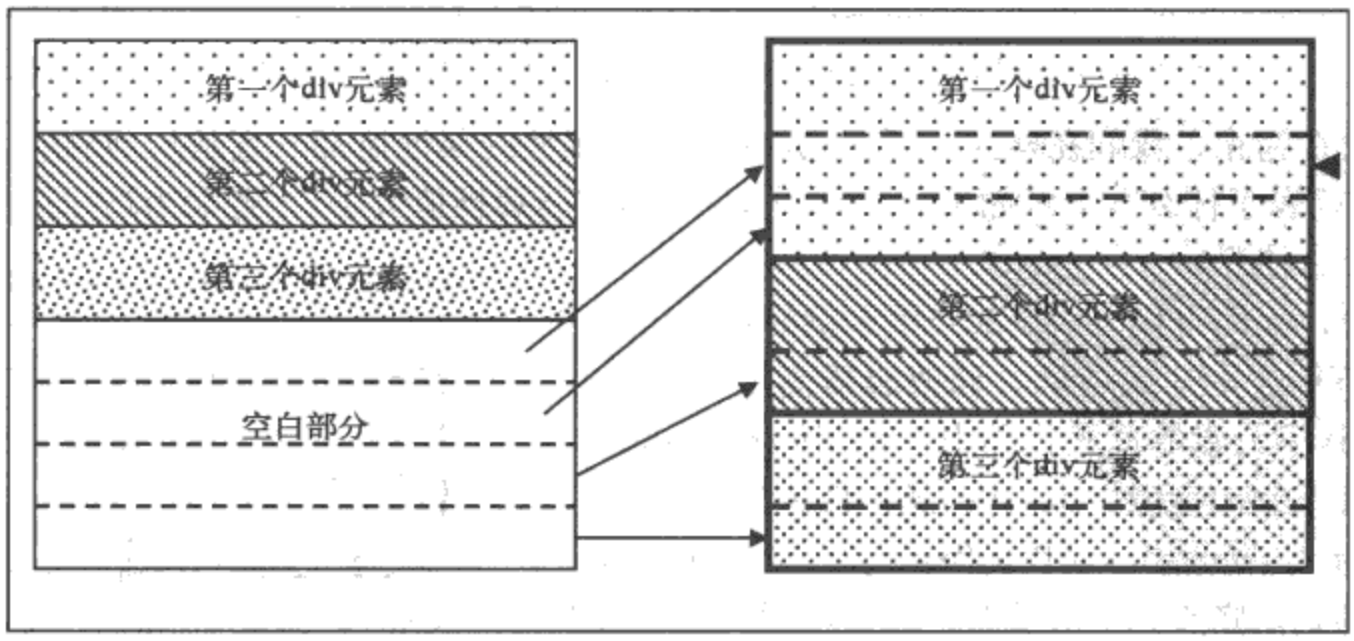

图20-20　容器的空白部分按元素的box-flex属性值进行分配

7. 指定水平方向与垂直方向的对齐方式

使用盒布局的时候，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。使用box-pack属性的时候，在Firefox浏览器中需要将其书写成“-moz-box-pack”的形式；在Safari浏览器或Chrome浏览器中需要书写成“-webkit-pack”的形式。使用box-align属性的时候，在Firefox浏览器中需要书将其写成“-moz-box-align”的形式；在Safari浏览器或Chrome浏览器中需要书写成“-webkit-align”的形式。

可以为box-pack属性及box-align属性指定的属性值和各种属性值的含义如表20-1所示。

使用box-pack属性及box-align属性能够很容易地将文字、图像或子元素放置在元素内的各个部位中，同时也解决了一些在CSS中放置元素内容时出现的一些问题。

表20-1 box-pack属性及box-align属性的属性值及其含义

属性值	排列方向	box-pack属性值的含义	box-align属性值的含义
start	horizontal	左对齐，文字、图像或子元素被放置在元素最左边	顶部对齐，文字、图像或子元素被放置在元素最顶部
center	horizontal	中对齐，文字、图像或子元素被放置在元素中部	中部对齐，文字、图像或子元素被放置在元素中部
end	horizontal	右对齐，文字、图像或子元素被放置在元素最右边	底部对齐，文字、图像或子元素被放置在元素最底部
start	vertical	顶部对齐，文字、图像或子元素被放置在元素最顶部	左对齐，文字、图像或子元素被放置在元素最左边
center	vertical	中部对齐，文字、图像或子元素被放置在元素中部	中对齐，文字、图像或子元素被放置在元素中部
end	vertical	底部对齐，文字、图像或子元素被放置在元素最底部	右对齐，文字、图像或子元素被放置在元素最右边

在使用CSS1.0或CSS 2.0的时候，在div元素内部直接放置文字的场合下，如果想让文字水平居中，只要使用text-align属性就可以了，但是若要让文字垂直居中，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CSS 3中，只要让div元素使用box-align属性（排列方向默认为horizontal），文字就可以垂直居中了。

代码清单20-6为一个让文字位于div元素正中央的简单示例，该示例中有一个div元素，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元素正中央。

代码清单20-6 让文字位于div元素正中央的简单示例

```
<!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>让文字位于div元素正中央的简单示例</title>
<style type="text/css">
```

```
div{
        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;
}
</style>
</head>
<body>
<div>示例文字</div>
</body>
</html>
```

代码清单20-6的运行结果如图20-21所示。

图20-21 让文字位于div元素正中央

在使用CSS1.0或CSS 2.0的时候，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属性及box-align属性，同样也使该问题很容易就得到了解决。

代码清单20-7为一个在浏览器窗口正中央显示图像的示例，该示例中有一幅图像，通过对body元素使用box-pack属性及box-align属性，不管浏览器窗口如何变化，该图像将始终位于浏览器正中央。

代码清单20-7 在浏览器窗口正中央显示图像的示例

```
<!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>在浏览器窗口正中央显示图像的示例</title>
<style type="text/css">
html, body{
    width:100%;
    height:100%;
    margin:0;
    padding: 0;
```

```
}
body{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-align: center;
    -webkit-box-align: center;
    -moz-box-pack: center;
    -webkit-box-pack: center;
}
</style>
</head>
<body>
<img src="tyl.jpg"/>
</body>
</html>
```

代码清单20-7的运行结果如图20-22所示。

图20-22 在浏览器窗口正中央显示图像的示例

第21章

Media Queries相关样式

在CSS 3的众多模块中，有一个与各种媒体相关的重要模块——Media Queries，本章将对这一模块进行详细介绍。

学习内容：

- 掌握CSS 3中Media Queries模块的基本概念，以及使用Media Queries模块可以实现的功能。
- 掌握如何编写媒体查询表达式来让浏览器根据当前窗口尺寸自动在样式表中挑选一种样式并使用。了解iPhone或iPod touch设备在支持Media Queries时有何特殊之处，以及应该用何种方法来指定iPhone或iPod touch设备中的safari浏览器在处理页面时根据多少像素的窗口宽度来处理。
- 掌握媒体查询表达式的编写方法，熟悉CSS 3中定义的所有设备类型及设备特性、媒体查询表达式中各种关键字的含义，以及结合使用设备类型、设备特性和各种关键字来正确编写媒体查询表达式。

21.1 根据浏览器的窗口大小来选择使用不同的样式

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，等等。

CSS 3中加入了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。换句话说，允许我们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。

接下来，我们看看在CSS 3中如何使用Media Queries模块中的有关功能来根据浏览器的窗口尺寸选择使用不同的样式。我们知道，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，CSS 3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。

代码清单21-1是一个根据不同的窗口尺寸来选择使用不同样式的示例，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1000px以上时，将3个div元素分为三栏并列显示；当窗口宽度在640px以上、999px以下时，3个div元素分两栏显示；当窗口宽度在639px以下时，3个div元素从上往下排列显示。

代码清单21-1 根据不同的窗口尺寸来选择使用不同样式的示例

```
<!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>根据不同的窗口尺寸来选择使用不同的样式的示例</title>
<style type="text/css">
body{
      margin: 20px 0;
}
#container{
      width: 960px;
      margin: auto;
}
#wrapper{
      width: 740px;
      float: left;
}
p{
      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; /* 绿色 */
}
/* 窗口宽度在1000px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640px以上、999px以下 */
@media screen and (min-width: 640px) and (max-width: 999px) {
        /* 2栏显示 */
        #container{
                width: 640px;
        }
        #wrapper{
                width: 640px;
                float: none;
        }
        p{
                line-height: 400px;
        }
        #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639px以下 */
@media screen and (max-width: 639px) {
        /* 1栏显示  */
        #container{
                width: 100%;
        }
        #wrapper{
                width: 100%;
                float: none;
        }
        body{
                margin: 20px;
        }
        p{
                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">
<p id="main">
MAIN
</p>
<p id="sub01">
SUB 01
</p>
</div>
<p id="sub02">
SUB 02
</p>
</div>
</body>
</html>
```

代码清单21-1的运行结果分为如下3种情况：

❑ 当窗口宽度在1000px以上时，将3个div元素分为三栏并列显示，如图21-1所示。

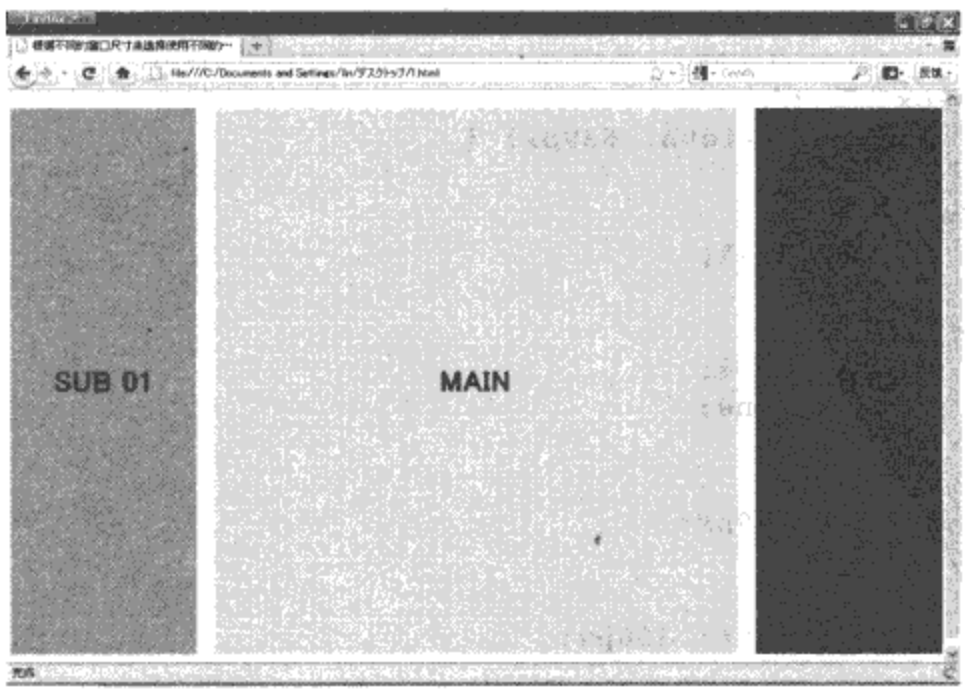

图21-1 窗口宽度在1000px以上时的页面显示

- 当窗口宽度在640px以上、999px以下时，3个div元素分两栏显示，如图21-2所示。
- 当窗口宽度在639px以下时，3个div元素从上往下排列显示，如图21-3所示。

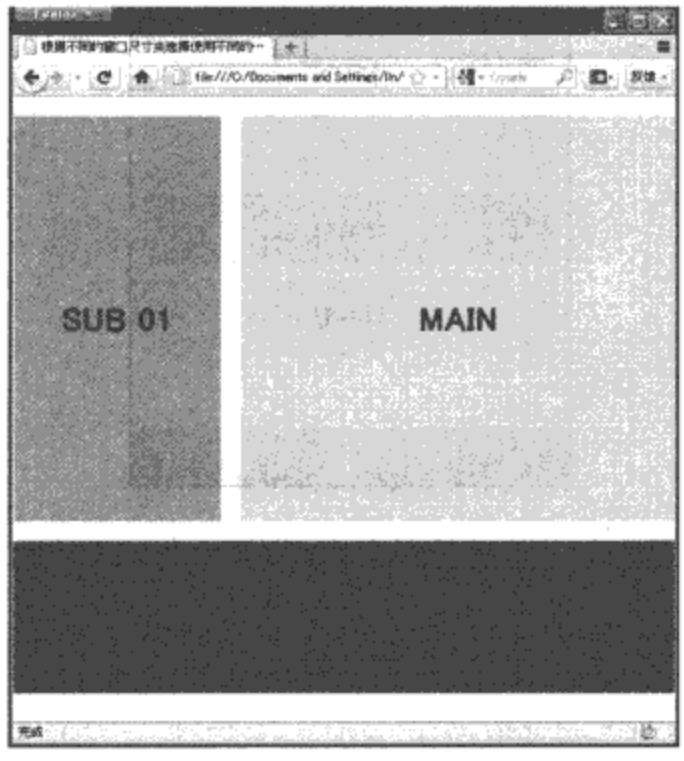

图21-2 窗口宽度在640px以上、999px以下时的页面显示

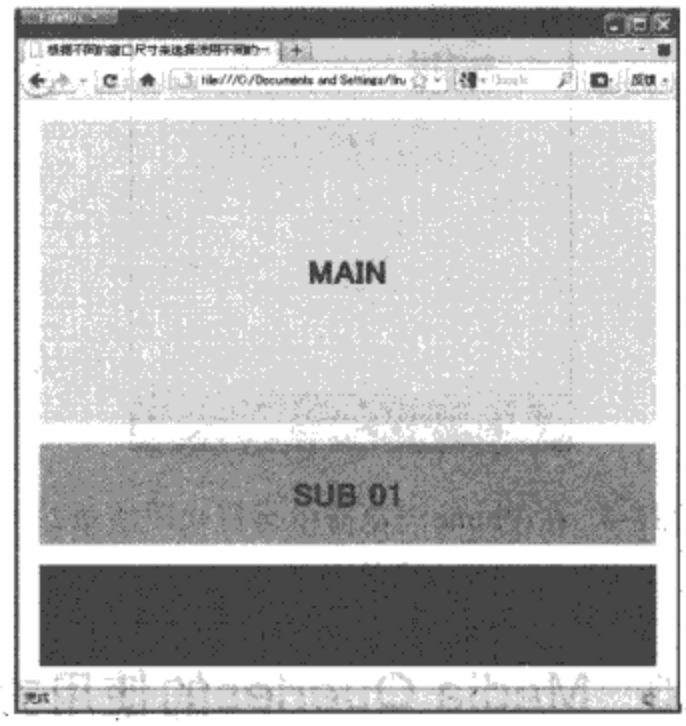

图21-3 窗口宽度在639px以下时的页面显示

21.2 在iPhone中的显示

在iPhone 3GS和iPod touch中使用的Safari浏览器也对CSS 3的媒体查询表达式提供了支持。iPhone的分辨率是320px × 480px，所以，如果运行代码清单21-1中的示例，示例页面中的3个div元素本来应该是从上往下排列显示的，但是真正运行的时候，浏览器中显示结果却为两栏显示，如图21-4所示。

为什么会是这样？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980px进行显示的。因为现在的网页大多是按照宽度为800px左右的标准进行制作的，所以Safari浏览器如果按照980px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980px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，指定方法类似如下所示：

```
<meta name="viewport" content="width=600px" />
```

在代码清单21-1中加入这段代码，并且在iPhone中重新运行该示例，Safari浏览器将窗口宽度作为600px来处理，将3个div元素从上往下并排显示，如图21-5所示。

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，请不要忘了加入<meta>标签并在标签中写入指定的窗口宽度。

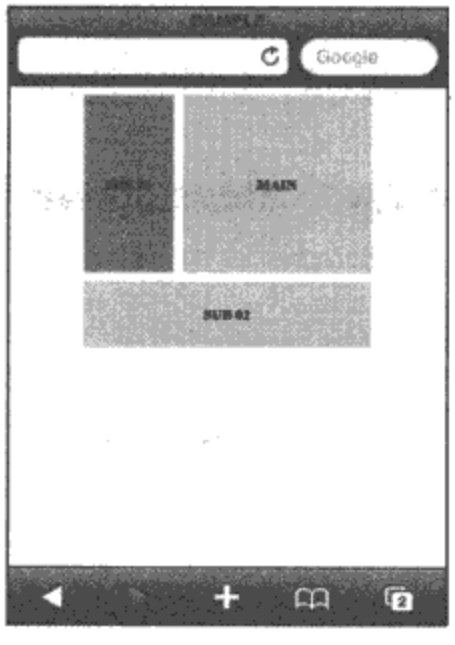

图21-4 在iPhone浏览器中运行代码清单21-1中的示例

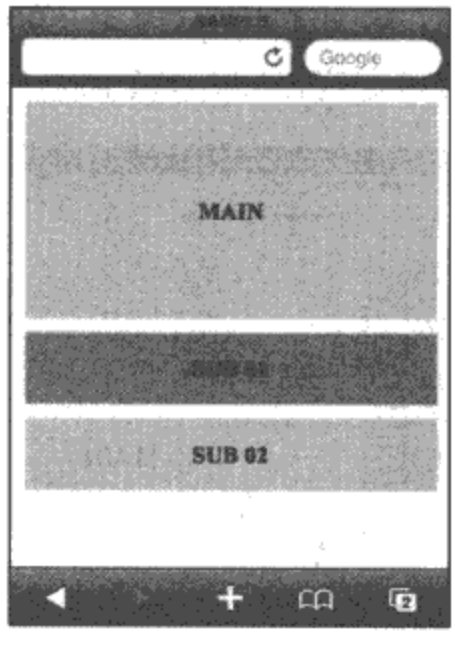

图21-5 iPhone中的Safari浏览器将窗口宽度作为600px来处理

21.3 Media Queries的使用方法

在代码清单21-1中，我们使用Media Queries来根据3种不同尺寸的窗口使用3种不同的样式，具体来说，Media Queries的使用方法如下所示：

```
@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，在此处可以指定的值与该值所代表的设备类型如表21-1所示。

表21-1 在Media Queries中可以指定的值与该值所代表的设备类型

可以指定的值	设备类型
all	所有设备
screen	电脑显示器
print	打印用纸或打印预览视图
handheld	便携设备
tv	电视机类型的设备
speech	语音和音频合成器
braille	盲人用点字法触觉回馈设备
embossed	盲文打印机
projection	各种投影设备
tty	使用固定密度字母栅格的媒介，比如电传打字机和终端

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400px时所使用的样式，书写方法如下所示：

```
(min-width: 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。

对于这13种设备特性的说明如表21-2所示。

表21-2 13种设备特性的说明

特性	可指定值	是否允许使用min/max前缀	特性说明
width	带单位的长度数值 例如：400px	允许	浏览器窗口的宽度
height	带单位的长度数值 例如：200px	允许	浏览器窗口的高度
device-width	带单位的长度数值 例如：400px	允许	设备屏幕分辨率的宽度值
device-height	带单位的长度数值 例如：200px	允许	设备屏幕分辨率的高度值
orientation	只能指定两个值： portrait或landscape	不允许	浏览器窗口的方向是纵向还是横向。 当窗口的高度值大于等于宽度值时， 该特性值为portrait，否则为landscape
aspect-ratio	比例值 例如16/9	允许	浏览器窗口的纵横比，比例值为浏览器窗口的宽度值/高度值
device-aspect-ratio	比例值 例如16/9	允许	屏幕分辨率的纵横比，比例值为设备屏幕分辨率的宽度值/高度值
color	整数值	允许	设备使用多少位的颜色值,如果不是彩色设备，该值为0
color-index	整数值	允许	色彩表中的色彩数
monochrome	整数值	允许	单色帧缓冲器中每像素的字节数
resolution	分辨率值，譬如300dpi	允许	设备的分辨率
scan	只能指定两个值： progressive或interlace	不允许	电视机类型设备的扫描方式。 progressive表示逐行扫描， interlace表示隔行扫描
grid	只能指定两个值：0或1	不允许	设备是基于栅格还是基于位图。 基于栅格时该值为1，否则该值为0

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，譬如以下语句指定了当设备窗口宽度小于640px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：

```
/*对not后面的语句执行取反操作
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)
```

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，对于如下的语句来说：

```
@media only screen and (color) {样式代码}
```

对于支持Media Queries的设备来说，将能够正确地应用样式，就仿佛only不存在一样。对于不支持Media Queries但能够读取Media Type的设备（譬如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（譬如IE8之前的浏览器）来说，无论是否有only，都将忽略这个样式。

最后要说的是，CSS 3中的Media Queries模块中也支持对外部样式表的引用，使用方法类似如下所示：

```
@import url(color.css) screen and (min-width: 1000px);
<link rel="stylesheet" type="text/css" media="screen and (min-width: 1000px)"
href="style.css" />
```

第22章

CSS 3的其他重要样式和属性

本章将对CSS 3中的一些内容比较少，但也非常重要的样式和属性进行详细介绍。

学习内容：

- 掌握CSS 3中与颜色相关的样式，掌握alpha通道的使用方法，掌握CSS 3中新增的RGBA颜色、HSL颜色与HSLA颜色的概念及使用方法。
- 掌握opacity属性的含义和使用方法，了解使用alpha来指定透明度与使用opacity属性来指定透明度这两者之间的区别，掌握transparent颜色值的含义及其使用方法。
- 掌握outline属性的含义及其使用方法，能够使用outline属性在元素周围绘制一条轮廓线并指定该轮廓线的线宽、颜色、线的样式，以及线与边框的位移距离。
- 掌握resize属性的含义及其使用方法，能够使用resize属性来定义一个允许用户自己调节尺寸的元素。
- 掌握initial属性值的含义及其使用方法，能够使用initial属性来取消对元素的样式设定。

22.1 颜色相关样式

在CSS 3之前，在样式中指定的颜色值只能为RGB颜色值，并且只能通过opacity属性来设置元素的透明度。CSS 3中增加了3种颜色值—RGBA颜色值、HSL颜色值及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。

本节将对CSS 3中与颜色相关的样式进行详细介绍。

22.1.1 利用alpha通道来设定颜色

1. 对RGB颜色设定alpha通道

在CSS 3中，可以通过对RGB颜色设定alpha通道的方法来定义RGBA颜色。所谓RGBA颜色，是指利用红色值(R)、绿色值(G)、蓝色值(B)、alpha通道值(A)来定义的颜色。其中，alpha通道值的范围为0~1.0，0表示完全透明，1表示不透明。RGBA颜色的使用方法如下所示：

```
background-color:rgba(255,0,0,0.5);
```

接下来，我们来看一个使用RGB颜色和RGBA颜色的示例，如代码清单22-1所示。在该示例中，有两个div元素，其中第一个div元素使用的是RGB颜色，所以是不透明的；第二个div元素使用的是RGBA颜色，其中alpha通道值为0.5，表示半透明。

代码清单22-1 RGB颜色和RGBA颜色的使用示例

```
<!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>RGB颜色与RGBA颜色使用示例</title>
</head>
<style type="text/css">
```

```
body{
    background-image:url(back.gif);
}
div{
    width:100%;
    height:100px;
    color:white;
}
div#div1{
        background-color:rgb(255,0,0);
}
div#div2{
        background-color:rgba(255,0,0,0.5);
}
</style>
<body>
<div id="div1">示例文字1</div>
<div id="div2">示例文字2</div>
</body>
</html>
```

这段代码的运行结果如图22-1所示。

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。在诸如IE等不支持RGBA颜色的浏览器中，将忽视对RGBA颜色值的指定，譬如代码清单22-1所示的示例，在IE中的运行结果如图22-2所示。

图22-1　RGB颜色与RGBA颜色的使用示例

图22-2　Internet Explorer浏览器中的RGBA颜色的使用示例

2. 对HSL颜色设定alpha通道

在CSS 3中，除了可以使用RGB颜色外，还可以使用HSL颜色。HSL颜色使用色调（H）、饱和度（S）、亮度（L）来定义颜色。其中，色调值中用0或360表示红色，120表示绿色，240表示蓝色，当取值大于360时，实际的值等于该值除以360之后的余数，例如，如果色调值为480，则实际的颜色值为480除以360之后的余数，等于120。饱和度和亮度的取值范围均为0%到100%。可以通过对HSL颜色设定alpha通道的方法来定义HSLA颜色。HSLA颜色是指利用色调（H）、饱和度（S）、亮度（L）、alpha通道值（A）来定义颜色。

接下来，我们来看一个HSL颜色与HSLA颜色的使用示例，如代码清单22-2所示。在该示例中，有两个div元素，这两个div元素的背景色都是绿色，其中第一个div元素使用的是

HSL颜色，所以是不透明的；第二个div元素使用的是HSLA颜色，其中alpha通道值为0.5，表示半透明。

代码清单22-2 HSL颜色与HSLA颜色的使用示例

```
<!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>HSL颜色与HSLA颜色的使用示例</title>
</head>
<style>
body{
    background-image:url(back.gif);
}
div{
    width:100%;
    height:100px;
    color:white;
}
div#div1{
        background-color: hsl(120,100%,50%);
        color: hsl(0,100%,100%);
}
div#div2{
        background-color: hsla(120,100%,50%,0.5);
        color: hsla(0,100%,100%,0.5);
}
</style>
<body>
<div id="div1">示例文字1</div>
<div id="div2">示例文字2</div>
</body>
</html>
```

这段代码的运行结果如图22-3所示。

图22-3 HSL颜色与HSLA颜色的使用示例

到目前为止，有Firefox、Opera和Chrome等几个浏览器支持HSL颜色和HSLA颜色。

22.1.2 alpha通道与opacity属性的区别

在CSS 3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Google Chrome，本节主要介绍opacity属性与alpha通道的区别。

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。

接下来，我们来看一个alpha通道与opacity属性结合使用的示例，如代码清单22-3所示。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。

代码清单22-3 alpha通道与opacity属性结合使用的示例

```
<!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>alpha通道与opacity属性结合使用的示例</title>
</head>
<style type="text/css">
body{
        background-image:url(back.gif);
}
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>
<body>
<div id="div1">示例文字1</div>
<div id="div2">示例文字2</div>
<div id="div3">示例文字3</div>
<div id="div4">示例文字4</div>
</body>
</html>
```

这段代码的运行结果如图22-4所示。

从上图中我们可以看出，对第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半透明的了。

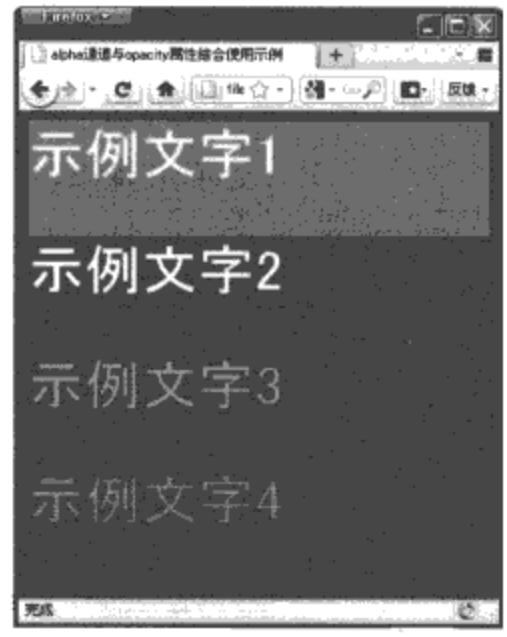

图22-4 alpha通道与opacity属性结合使用的示例

22.1.3 指定颜色值为transparent

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CSS 1中，只能在background-color属性中指定transparent值；在CSS 2中，可以在background-color及border-color属性中指定transparent值；在CSS 3中，可以在一切指定颜色值的属性中指定transparent值。

现在，transparent值已经得到了Firefox、Safari、Opera及Chrome等浏览器的支持。代码清单22-4为transparent值的一个使用示例，该示例中有三个div元素，第一个div元素的背景色被设定为transparent值，第二个div元素的边框颜色被设定为transparent值，第三个div元素的文字颜色被设定为transparent值。三个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。

代码清单22-4 transparent值的使用示例

```
<!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>transparent值的使用示例</title>
<style type="text/css">
div{
    background-color: white;
    border: solid 3px yellow;
    width:100%;
    height:100px;
}
body{
        background-image:url(back.gif);
}
div#div1{
    background-color: transparent;
}
div#div2{
    border-color: transparent;
}
div#div3{
    color: transparent;
}
```

```
</style>
</head>
<body>
<div id="div1">示例文字1</div>
<div id="div2">示例文字2</div>
<div id="div3">示例文字3</div>
</body>
</html>
```

这段代码的运行结果如图22-5所示：

到目前为止，IE对transparent值的支持并不完全，当把代码清单22-4中的示例运行在IE 8中时，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，如图22-6所示。

图22-5 transparent值的使用示例

图22-6 IE 8中的transparent值的使用示例

22.2 用户界面相关样式

在CSS 3的基本用户界面模块（Basic User Interface Module）中，定义了很多为了提高用户体验而新增的属性和功能，本节将对这些新增的属性和功能做一个详细介绍。

22.2.1 轮廓相关样式

1. CSS 2中的outline属性

在CSS 2中定义了一个outline属性，用来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。代码清单22-5是一个使用outline属性给radio单选框增加轮廓线的简单示例。

代码清单22-5 outline属性的使用示例

```
<!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>outline属性使用示例</title>
<style type="text/css">
input#male{
    outline:thin solid red
}
</style>
</head>
<body>
性别：<input type="radio" id=male name=sexRadio/>男
<input type="radio"  id=female name=sexRadio/>女
</body>
</html>
```

代码清单22-5的运行结果如图22-7所示。

图22-7 outline属性的使用示例

outline属性的使用方法如下所示：

outline：outline-color outline-style outline-width

- outline-color参数表示轮廓线的颜色，属性值为CSS中定义的颜色值，譬如red或#FF0000，可以将该参数省略，省略时该参数默认值为黑色。
- outline-style参数表示轮廓线的样式，属性值为CSS中定义的线的样式，譬如solid或dashed。可以将该参数省略，但是省略时该参数默认值为none，省略后不对该轮廓线进行绘制。
- outline-width参数表示轮廓线的宽度，属性值可以为一个宽度值，譬如40px,或者thin、medium、thick中的任意一个值。可以将该参数省略，省略时该参数默认值为medium，表示绘制中等宽度的轮廓线。

outline属性的属性值中三个参数的顺序可以互换，可以分开书写成如下所示的形式：

```
<style type="text/css">
input#male{
    outline-color:red;
    outline-style:solid;
    outline-width:thin;
}
</style>
```

2. CSS 3中新增的out-offset属性

在默认情况下，对带有边框的元素来说，使用outline属性将紧贴着边框外围绘制一条轮廓线。从代码清单22-6示例运行结果中，我们可以看到这个页面的显示效果。

代码清单22-6 给带边框元素绘制轮廓线

```
<!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>给带边框元素绘制轮廓线</title>
<style type="text/css">
div{
    border:blue solid thin;
    outline:red solid thin;
}
</style>
</head>
<body>
<div>示例文字</div>
</body>
</html>
```

代码清单22-6中示例的运行结果如图22-8所示。

图22-8 给带边框元素绘制轮廓线

有时，我们不想让这条轮廓线紧贴着边框外围，想让轮廓线稍微向外偏离几个像素，以绘制出双层边框的效果。针对这种情况，CSS 3中新增了一个outline-offset属性，可以使用该属性来实现这个效果。

outline-offset属性的使用方法非常简单，只要给该属性指定一个带像素单位的整数值即可，该整数值表示向外偏离多少个像素，书写方法类似如下所示：

```
outline-offset:2px;
```

将代码清单22-6中示例的样式代码修改为如下所示，在div元素的样式代码中加入outline-offset属性，修改后重新运行该示例，运行结果如图22-9所示。

```
<style type="text/css">
div{
    border:blue solid thin;
    outline:red solid thin;
    outline-offset:2px;
}
</style>
```

可以给outline-offset属性指定一个为负数的属性值，指定为负数的属性值后，轮廓线将向内偏移，绘制在边框内部。

将代码清单22-6中示例的样式代码修改为如下所示，给div元素的样式代码中的outline-offset属性指定一个负数属性值，重新运行该示例，运行结果如图22-10所示。

```
<style type="text/css">
div{
    padding:5px;
    border:blue solid thin;
    outline:red solid thin;
    outline-offset:-5px;
}
</style>
```

图22-9 outline-offset属性的使用示例
（外围的红色为轮廓线）

图22-10 给outline-offset属性指定负数属性值
（内部的红色为轮廓线）

22.2.2 resize属性

为了增强用户体验，CSS 3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。

到目前为止，resize属性得到了Firefox、Safari及Chrome等浏览器的支持。

代码清单22-7是一个resize属性的使用示例，该示例中有两个div元素，其中一个div元素的样式中使用了resize属性来允许用户自己修改它的尺寸，另一个div元素用来查看改变了元素尺寸后会对页面中其他元素产生什么影响。

代码清单22-7 resize属性的使用示例

```
<!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>resize属性的使用示例</title>
<style type="text/css">
#div1{
    background-color:pink;
    overflow: auto;
    resize:both;
    width:150px;
    height:150px;
}
#div2{
    background-color:orange;
    width:100%;
    height:150px;
}
</style>
</head>
<body>
<div id=div1>示例文字</div>
<div id=div2>页面中其他内容</div>
</body>
</html>
```

代码清单22-7的运行结果如图22-11所示。

在CSS 3中，可以为resize属性指定的值分为以下几种：

- none：用户不能修改元素尺寸。
- both：用户可以修改元素的宽度和高度。
- horizontal：用户可以修改元素的宽度，但不能修改元素的高度。
- vertical：用户可以修改元素的高度，但不能修改元素的宽度。
- inherit：继承父元素的resize属性值。

图22-11　resize属性的使用示例

22.3　取消对元素的样式指定——initial属性值

在CSS 3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。

22.3.1　取消对元素的样式指定

要取消对元素的样式指定，可以通过几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，一个样式写好了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属性取消掉就可以了，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CSS 3中已经不推荐使用它，取而代之的是将样式与元素或元素id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。

针对这种情况，CSS 3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。

接下来，我们来看一段简单的页面代码（如代码清单22-8所示），该页面中有三个p元素，在样式指定代码中指定了页面中这些p元素所使用的样式。

代码清单22-8 initial属性值的示例页面

```
<!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>initial属性值的使用示例</title>
</head>
<style type="text/css">
p{
    color:blue;
    font-family:宋体;
}
</style>
<body>
<p id="text01">示例文字1</p>
<p id="text02">示例文字2</p>
<p id="text03">示例文字3</p>
</body>
</html>
```

这段代码的运行结果如图22-12所示。

在这段代码中，三个p元素的文字颜色都是蓝色,字体都是宋体。这时，如果我们不想让id为“text01”的p元素使用这个样式，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，具体如下所示。

```
<style type="text/css">
p{
    color:blue;
    font-family:宋体;
}
p#text01{
    color:-moz-initial;
    color:initial;
    font-family:宋体;
}
</style>
```

把上面这段代码替换到代码清单22-8的示例样式代码中，然后重新运行该示例，运行结果如图22-13所示。

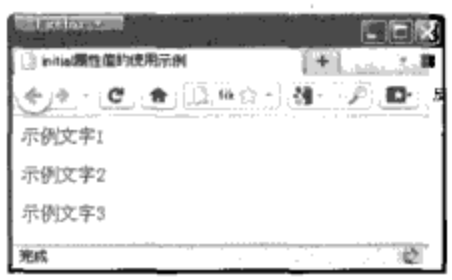

图22-12 initial属性值的使用示例（使用initial属性值前）

图22-13 initial属性值的使用示例（使用initial属性值后）

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id为“text01”的p元素中的文字颜色会变为黑色。

22.3.2　使用initial属性值并不等于取消样式设定的特例

个别情况下，对元素使用initial属性值后的显示结果并不等于将该元素的样式设定直接删除后的结果。例如一个文字为“你好”的h1元素，在页面中的显示结果如图22-14所示。

图22-14中，默认使h1元素变成了比较大的黑体文字。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h1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>
```

图22-14　h1元素的显示结果

可以在样式中对h1元素重新定义，例如对h1元素定义如下所示的样式：

```
<style type="text/css">
h1{
        font-size: 3em;
        font-weight: normal;
}
</style>
```

重新定义后，这个h1元素在浏览器中的显示结果如图22-15所示。

接着，在这段样式后面追加一段h1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h1元素的样式代码如下所示：

```
<style type="text/css">
h1{
        font-size: 3em;
        font-weight: normal;
}
h1{
        font-size: initial;
        font-weight: initial;
}
h1{
        font-size: -moz-initial;
        font-weight: -moz-initial;
}
</style>
```

图22-15　修改h1元素样式后的显示结果

使用这个追加了initial属性值的样式设定后，h1元素在浏览器中的显示结果如图22-16所示。

这个显示结果与不使用任何样式设定时的显示结果并不相同。

图22-16　对h1元素使用initial属性值

为什么在h1元素的样式代码中追加了initial属性值后

的显示结果与不使用任何样式设定时的显示结果会不一样呢？因为追加了initial属性值的样式设定后，h1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h1元素追加了什么样式。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h1元素使用的样式并不一致，如果要想让h1元素的字号和字体粗细的默认值使用浏览器的默认值，还是不要在追加的样式代码中使用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”这个网页中可以查到浏览器对HTML 4中元素所做的追加样式清单，目前各主流浏览器均遵照这个清单来对元素追加默认样式。

第23章
综合实例

本章是全书的结尾部分，主要是想通过两个综合实例来帮助读者更好地理解全书的内容，让读者能够从总体上掌握应该如何综合运用HTML 5以及CSS 3来创建一个具有现代风格的Web网站和Web应用程序。

学习内容：

- 如何使用HTML 5中新增的结构元素来构建一个结构更清晰、更加具有语义和现代风格的Web网站。
- 如何使用HTML 5 + CSS 3来构建一个现代的Web应用程序。

23.1 实例1：使用HTML 5中新增结构元素来构建网页

HTML 5中新增了几个结构元素：section元素、article元素、nav元素、aside元素、header元素和footer元素，通过运用这些结构元素，可以让网页的整体结构更加直观和明确、更加富有语义化和更具有现代风格。

本节将以一个企业网站为实例来讲解如何综合运用HTML 5中的这些结构元素、页面中每个结构元素所起的作用，以及应该展现哪些内容。这里会将实现页面的HTML 5代码与CSS 3的样式代码一起进行讲解，以便让读者同时了解在HTML 5实现的网页中应该如何使用CSS 3样式来对页面中的元素进行页面布局及视觉美化。

23.1.1 组织网页结构

在一个用HTML 5实现的网页中，每一个网页都将由一些主体结构元素构成。在大型的网站中，一个网页通常由以下几个结构元素组成。

- header结构元素：通常用来展示网站的标题、企业或公司的logo图片、广告（Flash等格式）、网站导航条，等等。
- aside结构元素：通常用来展示与当前网页或整个网站相关的一些辅助信息。例如，在博客网站中，可以用来显示博主的文章列表和浏览者的评论信息等；在购物网站中，可以用来显示商品清单、用户信息、用户购买历史等；在企业网站中，可以用来显示产品信息、企业联系方式、友情链接等。aside结构元素可以有多种形式，其中最常见的形式是侧边栏。
- section结构元素：一个网页中要显示的主体内容通常被放置在section结构元素中，每个section结构元素都应该有一个标题来显示当前要展示的主要内容的标题信息。每个section结构元素中通常还应该包括一个或多个section元素或article元素，用来显示网页主体内容中每一个相对独立的部分。
- footer结构元素：通常，每一个网页中都具有footer结构元素，用来放置网站的版权声明和备案信息等与法律相关的信息，也可以放置企业的联系电话和传真等联系信息。

接下来，让我们来看一下本节中介绍的企业网站在浏览器中的显示效果。该企业网站是江苏省常州市蓝博纺织机械有限公司的门户网站，它的首页显示效果如图23-1所示。

图23-1 江苏省常州市蓝博纺织机械有限公司的企业网站

另外，因为篇幅关系，本实例中具体讲解的是该企业网站中用来显示联系方式的“联系我们”网页，该网页在浏览器中的显示如图23-2所示。

图23-2 江苏省常州市蓝博纺织机械有限公司的企业网站的“联系我们”网页

该网页的主体结构如图23-3所示。

该网页中有几个主要的HTML 5结构元素，分别是：header元素、aside元素、section元素及footer元素。在还没有加入任何实际内容之前，该网页的页面代码如代码清单23-1所示。

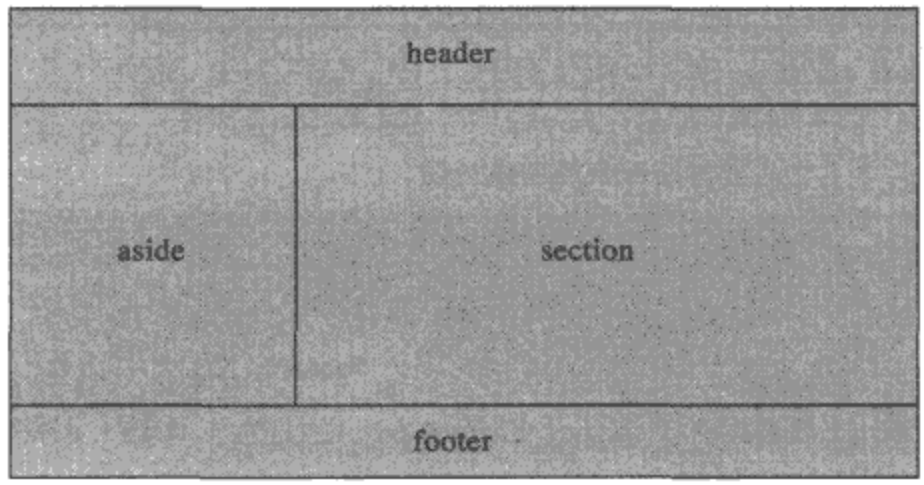

图23-3 “联系我们”网页的主体结构

代码清单23-1 示例页面的主体结构代码

```
<!DOCTYPE html>
<head>
<meta charset=UTF-8 />
<link href="style.css" rel="stylesheet" type="text/css"/>
<title>常州蓝博纺织机械有限公司|污水热能回收系统|布面含潮率控制仪|丝光机自动配碱系统|常
州蓝博纺织机械有限公司</title>
</head>
<body>
<header id="webTitle"></header>
<aside></aside>
<section id="main" ></section>
<footer></footer>
</body>
</html>
```

页面开头使用了HTML 5中的“<!DOCTYPE html>”语句来声明页面中将使用HTML 5。在head标签中，除了meta标签中使用了更简洁的编码指定方式之外，其他代码均与HTML 4中的head标签中的代码完全一致。此页面中使用了很多结构元素，用来替代HTML 4中的div元素，因为div元素没有任何语义性，而HTML 5中推荐使用具有语义性的结构元素，这样做的好处是可以让整个网页结构更加清晰，浏览器、屏幕阅读器及其他阅读代码的人也可以直接从这些结构元素上分析出网页中什么部位放置了什么内容。

23.1.2 header元素中的内容

前面我们介绍过，header元素是一个具有引导和导航作用的结构元素。很多企业网站中都有一个非常重要的header元素，一般位于网页的开头，用来显示企业名称、企业logo图片、整个网站的导航条，以及flash形式的广告条等。

在本实例中，header元素中的内容在浏览器中的显示结果如图23-4所示。

示例页面中header元素的结构示意图如图23-5所示。

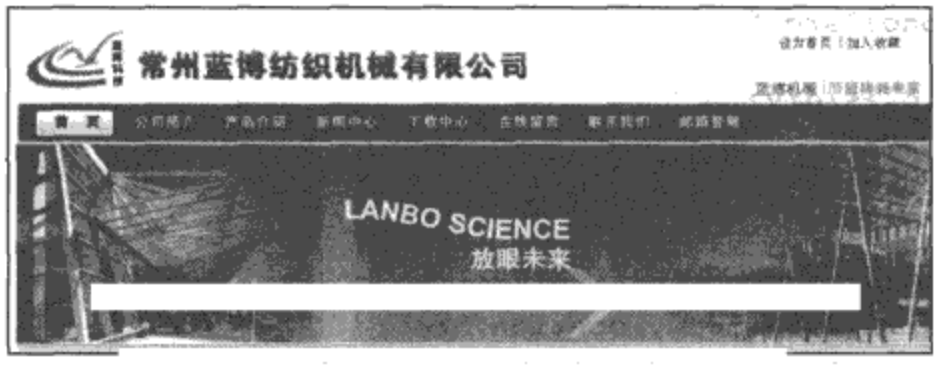

图23-4 示例页面中header元素的内容在浏览器中的显示结果

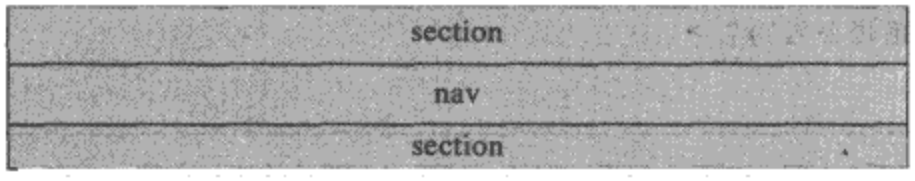

图23-5 示例页面中header元素的结构示意图

示例页面中的header元素在HTML 5页面中的代码结构如下所示：

```
<header id="webTitle">
<section id="sectionTop1">
……
</section>
<nav>
……
</nav>
<section id="sectionTop2">
……
</section>
```

1. header元素中第一个section元素的代码分析

在第一个section元素中存放了网站的名称及logo图片，它在浏览器中的页面显示如图23-6所示。

图23-6 第一个section元素在浏览器中的显示

第一个section元素内部放置了三个div元素：第一个div元素用来显示页面左边的logo图片，图片名称为“logo.jpg”,公司名称也是通过该图片来显示的；第二个div元素是一个内容为空白的空元素，用来将左边的logo图片与右边的广告图片进行分隔（通过在CSS 3样式代码中指定宽度的办法）；第三个div元素用来显示广告图片，该广告图片中也放置了“设为首页”与“加入收藏”的文字，通过链接地图的方式实现点击这两段文字后将页面设为首页或加入收藏夹的功能。section元素的HTML 5页面代码如代码清单23-2所示。

代码清单23-2 第一个section元素的HTML 5页面代码

```
<section id="sectionTop1">
<div id="TopLeft">
```

```
<img id="imglogo1"src="images/logo.jpg"/>
</div>
<div id="TopMid"></div>
<div id="TopRight">
<img id="imglogo2" src="images/logo_2.jpg" usemap="#Map">
</div>
<map name="Map">
<area shape="rect" coords="32,5,93,21" href="#"
onClick="var strHref=window.location.href;
this.style.behavior='url(#default#homepage)';
this.setHomePage('http://www.czlbkj.cn')">
<area shape="rect" coords="103,3,167,22"
href="javascript:window.external.AddFavorite('http://www.czlbkj.cn',
'常州蓝博纺织机械有限公司')" >
</map>
</section>
```

接下来，我们来看一下这部分HTML 5页面代码所使用到的CSS样式代码。在这之前，我们首先来看一下对整个网页所使用的CSS 3样式代码，如代码清单23-3所示。

代码清单23-3 整个网页使用的CSS 3样式代码

```
<style type="text/css">
header article section footer nav aside{
    display:block;
}
body{
    font-weight: normal;
    font-size: 12px;
    color:#666666;
}
.....
</style>
```

网页整体使用了两个样式：第一个样式是为了解在IE 8或之前的浏览器中能正常显示HTML 5中新增的结构元素而使用的样式，目的是将这些结构元素显示为block类型；第二个样式指定了整个网页中默认使用的字体、文字粗细及文字颜色。

接下来，我们来看一下header元素所使用的样式，如代码清单23-4所示。

代码清单23-4 header元素所使用的样式

```
<style type="text/css">
.....
header#webTitle{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-pack:center;
    -webkit-box-pack:center;
    width:100%;
}
.....
</style>
```

header元素使用box-pack属性将3个div元素显示在网页中央，并设定header元素的宽度使

之占据整个网页。另外，从CSS 2开始就使用“#元素id”或“元素#元素id”形式的选择器会将样式与元素id进行绑定，或使用“#元素id子元素”形式的选择器会将样式与某个元素的子元素进行绑定,不推荐使用没有任何语义的class属性。

最后，我们来看一下在第一个section元素中使用到的样式，如代码清单23-5所示。

代码清单23-5 在第一个section元素中使用到的样式

```
<style type="text/css">
……
#sectionTop1{
    margin-top:10px;
    display: -moz-box;
    display: -webkit-box;
    width:900px;
}
#TopLeft{
    width:600px;
}
#imglogo1{
    width:600px;
    height:74px;
}
#TopMid{
    width:100px;
    background:url(images/logo_bg.jpg);
}
#TopRight{
    width:200px;
}
#imglogo2{
    width:200px;
    height:74px;
}
……
</style>
```

这段样式代码中需要注意的是，第一个section元素中使用了盒布局的方式将三个div元素并列显示。

2. header元素中nav元素的代码分析

我们前面介绍过，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。nav元素可以被放置在header元素中，作为整个网站的导航条来使用。nav元素中可以存放列表或导航地图，或其他任何可以放置一组超链接的元素。在本示例中，网站标题部分的nav元素中放置了一个导航地图，如图23-7所示。

图23-7 nav元素在浏览器中的显示

nav元素的HTML 5页面代码如代码清单23-6所示。

代码清单23-6 nav元素的HTML 5页面代码

```
<nav>
<ul id="topNavUrl">
<li>
    <img id="img1" src="images/menu/menu_1.jpg">
</li><li>
    <a href="index.asp"><img id="img_even" src="images/menu/menu_2.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_3.jpg">
</li><li>
    <a href="about.asp"><img id="img_even" src="images/menu/menu_4.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_5.jpg">
</li><li>
    <a href="product.asp"><img id="img_even" src="images/menu/menu_6.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_7.jpg">
</li><li>
    <a href="news.asp"><img id="img_even" src="images/menu/menu_8.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_9.jpg">
</li><li>
    <a href="download.asp"><img id="img_even" src="images/menu/menu_10.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_11.jpg">
</li><li>
    <a href="gbook.asp"><img id="img_even" src="images/menu/menu_12.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_13.jpg">
</li><li>
    <a href="contact.asp"><img id="img_even" src=" images/menu/menu_14.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_15.jpg">
</li><li>
    <a href="http://mail.czlbkj.cn">
    <img id="img_even" src="images/menu/menu_16.jpg">
</li><li>
    <img  id="img_old" src="images/menu/menu_17.jpg">
</li>
</ul>
</nav>
```

nav元素中用ul列表的形式来显示网页中对其他页面的超链接，并且将超链接嵌入在一组图片之间，构成导航地图的形式。书写代码的时候，第一个li列表项目的结束标签与第二个li列表项目的开始标签之间不能换行，否则页面上显示的图片与图片之间会出现裂缝。

接下来，我们来看一下nav元素所使用到的样式代码，如代码清单23-7所示。

代码清单23-7 nav元素使用的CSS 3样式代码

```
<style type="text/css">
. ...
header nav{
```

```
    width:900px;
}
ul#topNavUrl{
    width:900 px;
    height:40px;
    margin: 0;
    padding: 0;
}
ul#topNavUrl li{
   display: inline-block;
}
ul#topNavUrl img{
    margin: 0;
    padding: 0;
    height:40px;
}
ul#topNavUrl #img1{
    width:16px;
}
ul#topNavUrl img$\_old:{
    width:7px;
}
ul#topNavUrl img$\_even:{
    width:80px;
}
ul#topNavUrl img:nth-last-child{
    width:180px;
}
……
</style>
```

这段样式代码中需要注意的是：

- 将ul列表的列表项目li元素的盒类型设定为inline-block类型，目的是在一行中可以显示多个列表项目。
- 将列表项目中每个链接图片左半边的图片id设定为结尾为“_old”的文字，然后使用CSS 3中的[att$=val] 选择器设定这些图片的宽度为7px，因为这些图片中不包含超链接文字；将列表项目中每个链接图片右半边的图片id设定为结尾为“_even”的文字，然后使用CSS 3中的[att$=val] 选择器设定这些图片的宽度为80px，因为这些图片中包含超链接文字。
- 使用CSS 3中的nth-last-child选择器设定最后一幅图片的宽度为180px。

3. header元素中第二个section元素的代码分析

接下来，我们来看一下在header元素中第二个section元素里存放的内容。该元素中存放了三个div元素：第一个div元素用来展示flash动画与导航地图之间的分隔图片，第二个div元素用来显示企业的广告（Flash格式），第三个div用来显示将header元素与网页中的header元素下部的其他内容进行分隔用的图片。第二个section元素在浏览器中的显示结果如图23-8所示。

图23-8 第二个section元素在浏览器中的显示结果

第二个section元素在HTML 5中的页面代码如代码清单23-8所示。

代码清单23-8 第二个section元素在HTML 5中的页面代码

```
<section id="sectionTop2">
<div id="Bottom1"><img src="images/index_4.jpg"/></div>
<div id="FlashDiv">
    <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
        codebase="http://download.macromedia.com/pub/shockwave/
        cabs/flash/swflash.cab#version=6,0,29,0"
        width="900" height="200">
        <param name="movie" value="images/p.swf">
        <param name="quality" value="high">
        <param name=wmode value=transparent>
        <embed src="images/p.swf" quality="high"
           pluginspage="http://www.macromedia.com/go/getflashplayer"
           type="application/x-shockwave-flash"
           width="900"
           height="200">
       </embed>
    </object>
</div>
<div id="Bottom2"><img src="images/index_6.jpg"/></div>
</section>
```

第二个section元素所使用的样式代码如代码清单23-9所示。

代码清单23-9 第二个section元素所使用的样式代码

```
<style type="text/css">
……
#sectionTop2{
    width:900px;
}
#sectionTop2 #Bottom1{
    width:900px;
    background:url(images/index_4.jpg);
    height:3px;
}
#sectionTop2 #Bottom2{
    width:900px;
    background:url(images/index_6.jpg);
    height:11px;
}
#sectionTop2 #FlashDiv{
    width:900px;
    background:url(images/news.jpg);
```

```
    height:200px;
}
.....
</style>
```

23.1.3 aside元素中的内容

aside元素用来显示当前网页主体内容之外的、与当前网页显示内容相关的一些辅助信息。例如，可以将网站经营者或管理者认为比较重要的、想让用户经常能看见的一些内容显示在aside元素中。aside元素的显示形式可以是多种多样的，其中最常用的形式是侧边栏的形式。

在示例页面中，aside元素中的内容在浏览器中的显示如图23-9所示。

示例页面中aside元素的结构示意图如图23-10所示。

图23-9 示例页面中aside元素在浏览器中的显示

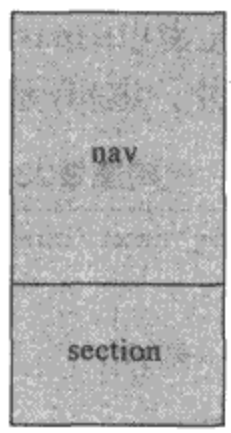

图23-10 示例页面中aside元素的结构示意图

示例页面中aside元素在HTML 5页面中的代码结构如下所示：

```
<aside>
<nav>
.....
</nav>
<section>
.....
</section>
</aside>
```

示例页面中aside元素所使用的样式代码如下所示：

```
aside{
    margin-left:50px;
    float:left;
    width:180px;
}
```

1. aside元素中nav元素的代码分析

在示例页面中，aside元素中的第一部分为nav元素，用来显示企业生产的所有产品的产品分类。该nav元素在HTML 5中的页面代码如代码清单23-10所示。

代码清单23-10 aside元素中nav元素的页面代码

```
<nav>
<img id="img1" src="images/in_left1.gif">
<ul>
<li><a href="show.asp?cid=29">定型机尾气余热回收</a></li>
<li><a href="show.asp?cid=27">GTG-600-X盘式水热能交换器</a></li>
<li><a href="show.asp?cid=25">GTL热能回收系统</a></li>
<li><a href="show.asp?cid=18">GTL-10热能回收系统</a></li>
<li><a href="show.asp?cid=19">GTL-20热能回收系统</a></li>
<li><a href="show.asp?cid=17">布面含潮率控制仪</a></li>
<li><a href="show.asp?cid=16">丝光机自动配碱系统</a></li>
<li><a href="show.asp?cid=20">自动加料系统主图</a></li>
</ul>
<img id="img2" src="images/in_left3.gif">
</nav>
```

它与header元素中的nav元素相同，也是使用ul列表显示一组超链接，并在ul列表的上下两边放置装饰图片。该nav元素使用的CSS样式代码如代码清单23-11所示。

代码清单23-11 aside元素中的nav元素所使用的CSS样式代码

```
<style type="text/css">
……
aside nav{
    width: 180px;
}
aside nav ul{
    list-style: none;
    margin-left:5px;
    margin-top:0px;
    margin-right:15px;
}
aside nav ul li
{
    background: url("images/left_bg.gif");
    margin-left:-45px;
    padding-left:20px;
    width:160px;
}
aside nav ul li a{
    margin: 0px;
    font-size: 12px;
    line-height: 24px;
    text-align: left;
    text-decoration: none;
    color: #777;
}
aside #img1{
    width:180px;
    height:40px;
}
aside #img2{
    width:180px;
    height:20px;
```

```
}
.....
</style>
```

2. aside元素中section元素的代码分析

aside元素中的第二部分是一个section元素，其中存放了企业的联系方式。该section元素在HTML 5中的页面代码如代码清单23-12所示。

代码清单23-12 aside元素中section元素的HTML 5页面代码

```
<section>
    <ul>
        <li><img  id="img1" src="images/in_left5.gif"/></li>
        <li>
            <address>地址  常州市青洋北路1号天宁经济开发区新动力创业中心</address>
        </li>
        <li><address>电话  86-519-88257109、<br/>88116975</address></li>
        <li><address>传真  86-519-88254120</address></li>
        <li><address>E-mail:lanbo@czlbkj.cn</address></li>
        <li><img  id="img2" src="images/12.gif"/></li>
    </ul>
</section>
```

在section元素中使用ul列表的形式显示企业的联系信息，在显示联系方式等文字信息的地方使用了HTML 5中专门用来显示地址信息的address元素，在ul列表元素的开头与结尾部分使用装饰图片。

这个section元素所使用的CSS样式代码如代码清单23-13所示。

代码清单23-13 显示地址用section元素的CSS 3样式代码

```
<style type="text/css">
..
aside section{
    width:180px;
    margin:0px;
}
aside section #img1{
    width:180px;
    height:40px;
}
aside section #img2{
    width:180px;
    height:10px;
}
aside section ul{
    height:22px!important:height:24px;
    background:url(images/11.gif);
    padding-left:8px;
    list-style:none;
}

</style>
```

23.1.4 section元素中的内容

在HTML 5网站中，每个网页所展示的主体内容通常都存放在section结构元素中，而且通常带有一个标题元素header。在本示例页面中，section元素中的内容在浏览器中的显示结果如图23-11所示。

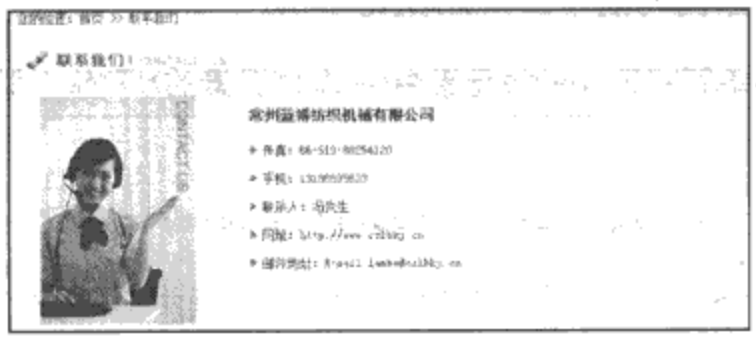

图23-11 section元素在浏览器中的显示结果

示例页面中的页面主体section元素中显示了该企业的联系方式，其结构相对来说也比较简单，由一个显示标题用的header元素及显示联系信息用的article元素组成，在HTML 5页面中的代码如代码清单23-14所示。

代码清单23-14 section元素的HTML 5页面代码

```
<section id="main" >
<header>您的位置：首页 &gt;&gt; 联系我们</header>
<article>
<header><img  id="img1" src="images/contact.gif"/></header>
<div id="left"><img  id="img2" src="images/contactImg.gif"/></div>
<div id="right">
    <ul>
        <li></li>
        <li><h3>常州蓝博纺织机械有限公司<h3></li>
        <li id="space"></li>
        <li>
            <img  id="img" src="images/ico.gif"/> 
            传真：86-519-88254120</li>
        <li><img  id="img_line" src="images/line.gif"/></li>
        <li>
            <img  id="img" src="images/ico.gif"/> 手机：13186695633</li>
        <li><img  id="img_line" src="images/line.gif"/></li>
        <li><img  id="img" src="images/ico.gif"/> 联系人：冯先生</li>
        <li><img  id="img_line" src="images/line.gif"/></li>
        <li>
            <img  id="img" src="images/ico.gif"/> 网址：
            http://www.czlbkj.cn</li>
        <li><img  id="img_line" src="images/line.gif"/></li>
        <li>
            <img  id="img" src="images/ico.gif"/>
             邮件地址：E-mail:lanbo@czlbkj.cn
                    </li>
        <li><img  id="img_line" src="images/line.gif"/></li>
```

```
        </ul>
    </div>
</article>
</section>
```

在header元素中显示该页面主体内容的标题，显示的文字为“您的位置：首页 >> 联系我们”。

在article元素中同样可使用一个header元素显示联系信息的标题，在本页面中使用了图片的方式来显示联系信息的标题，以使得页面显示效果更加美观。在标题下面使用两个div元素，分左右两列显示，左边的div元素中显示一幅图片，用来加强页面的美术效果，在右边的div元素中放置一个ul列表，在列表中显示企业的联系信息。每一个列表项目都有一幅背景图片，同时每一行联系信息的中间也使用了图片进行分隔，以加强页面的美术效果。

这个section元素所使用的CSS样式代码如代码清单23-15所示。

代码清单23-15 section元素所使用的CSS 3样式代码

```
<style type="text/css">
……
section#main{
    width:720px;
}
section#main header{
    width:720px;
    height:30px;
    background:url(images/in_c1.gif);
    padding-left:10px;
}
article{
    width:720px;
}
article #img1{
    width:720px;
    height:35px;
}
article #img2{
    width:154px;
    height:229px;
}
article #left{
    padding-left:30px;
    float:left;
    height:62px;
    margin-top:21px;
}
article #right{
    float:left;
    margin-top:21px;
}
article ul{
    list-style:none;
}
article #right li:nth-first-child(1){
```

```
        height:21px;
    }
    article #right li{
        padding-left:10px;
    }
    article #right #space{
        height:6px;
    }
    article #img{
        width:7px;
        height:8px;
    }
    article #img_line{
        height:1px;
    }
    ......
    </style>
```

值得注意的是，这段样式代码使用了CSS 3中的nth-first-child选择器来单独指定第一个列表项目的高度，因为该列表项目的作用是将列表上部的元素与列表进行分隔，所以需要根据页面显示情况单独对这个列表项目的高度进行调整。

23.1.5 footer元素中的内容

footer元素专门用来显示网站、网页或内容区块的脚注信息，在企业网站中的footer结构元素通常用来显示版权声明、备案信息、企业联系电话及网站制作单位等内容。

本节中，示例页面的footer元素在浏览器中的显示结果如图23-12所示。

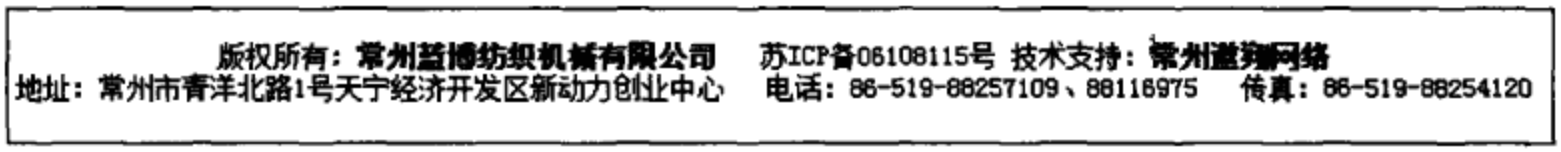

图23-12 footer元素在浏览器中的页面显示

footer元素中的内容相对来说比较简单，它存放了两个div元素，其中上面的div元素仅用来显示将footer元素与网页中其他内容进行分隔的美术图片，在第二个div元素中存放版权信息、备案信息、企业地址等内容，页面中的HTML 5代码如代码清单23-16所示。

代码清单23-16 footer元素的HTML 5页面代码

```
<footer>
    <div id="top"></div>
    <div id="main">
        版权所有：<strong>常州蓝博纺织机械有限公司</strong>苏ICP备06108115号
        技术支持：<strong>常州遨翔网络</strong><br>
        地址：常州市青洋北路1号天宁经济开发区新动力创业中心
        电话：86-519-88257109、88116975
        传真：86-519-88254120
    </div>
</footer>
```

footer元素所使用的CSS样式代码如代码清单23-17所示。

代码清单23-17 footer元素所使用的CSS 3样式代码

```
<style type="text/css">
……
footer{
    position:absolute;
    width:900px;
    left:50px;
    top:750px;
}
footer #top{
    width:900px;
    height:37px;
    background:url(images/index_8.jpg);
}
footer #main{
    width:900px;
    height:63px;
    text-align:center;
    color:black;
}
……
</style>
```

23.2 实例2：使用HTML 5 + CSS 3来构建Web应用程序

上一节讲述了如何将HTML 5中新增的结构元素与CSS 3样式结合起来创建一个结构更加清晰、更加具有语义化、更加具有现代风格的网站。本节将结合Web应用程序中的一个示例页面来向读者介绍在下一代Web应用程序的页面中应该如何使用HTML 5中新增的表单元素和其他功能及CSS 3的样式。

本节介绍的示例页面是在Web应用程序中经常会使用的信息输入页面，该页面分为上下两部分，在页面上半部分的表单中输入信息，点击表单中的保存按钮后在下半部分的一览表中显示所有信息，包括刚才输入的这条信息。

首先，我们来看一下该页面在浏览器中的显示结果。为了能够同时讲解HTML 5对本地数据库进行操作的功能，本示例中会将数据全部保存在本地数据库中。此举对于HTML 5时代的Web应用程序的Demo版来说，这也将是一个不错的选择。现在很多Demo版的Web应用程序都是使用直接将数据写死在页面中的办法来演示对数据的操作流程的。如果使用本地数据库，Demo版中的数据将会更加具有实时特征。因为目前只有Safari、Chrome和Opera等浏览器对本地数据库提供支持，所以本节中使用Opera浏览器来运行示例页面，该页面在Opera浏览器中的运行结果如图23-13所示。

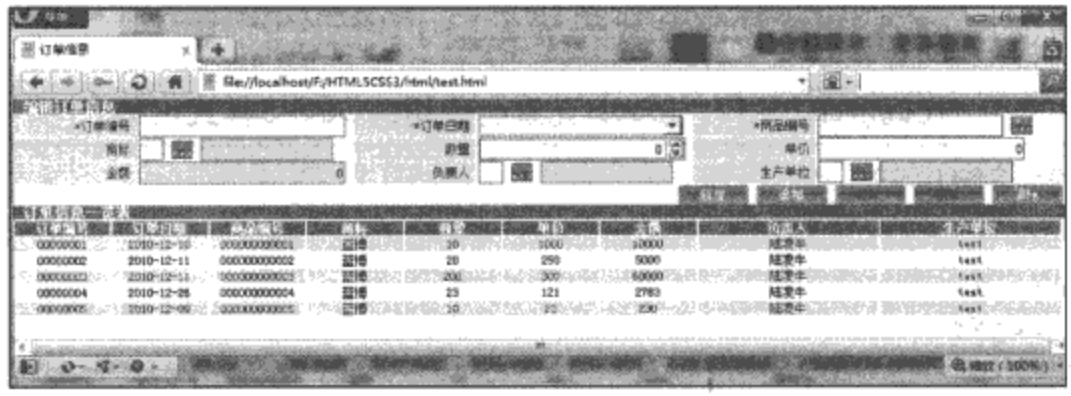

图23-13 订单信息输入页面

23.2.1 HTML 5页面代码分析

首先，我们来详细分析该示例页面的HTML 5页面代码，该页面为某个公司所使用的订单信息输入页面，该页面中的所有控件在页面打开时的状态及其功能描述如表23-1所示（暂不涉及CSS样式部分和JavaScript脚本代码部分）。

表23-1 示例页面中的所有控件及其功能描述

控件名称	显示文字	控件使用元素	最大输入位数	是否有效	是否只读
编辑订单信息文字	编辑订单信息	页面文字			
订单编号文字	*订单编号	页面文字			
订单编号文本框		Input,type=required	8	有效	非只读
订单日期文字	*订单编号	页面文字			
订单日期文本框		Input,type=date		有效	非只读
商品编号文字	*商品编号	页面文字			
商品编号文本框		Input,type=required	12	有效	非只读
商品检索按钮	...	Input,type=button			
商标文字	商标	页面文字			
商标编号文本框		Input,type=text	3	有效	非只读
商标检索按钮	...	Input,type=button			
商标文本框		Input,type=text		有效	只读
数量文字	数量	页面文字			
数量文本框		Input,type=number	6	有效	非只读
单价文字	单价	页面文字			
单价文本框		Input,type=text	6	有效	非只读
金额文字	金额	页面文字			
金额文本框		Input,type=text		有效	只读
负责人文字	负责人	页面文字			
负责人编号文本框		Input,type=text	3	有效	非只读
负责人检索按钮	...	Input,type=button			
负责人文本框		Input,type=text		有效	只读
生产单位文字	生产单位	页面文字			
生产单位编号文本框		Input,type=text	3	有效	非只读

（续）

控件名称	显示文字	控件使用元素	最大输入位数	是否有效	是否只读
生产单位检索按钮	...	Input,type=button			
生产单位文本框		Input,type=text		有效	只读
新增按钮	新增	Input,type=button		有效	
追加按钮	追加	Input,type=button		有效	
修改按钮	修改	Input,type=button		无效	
删除按钮	删除	Input,type=button		无效	
清除按钮	清除	Input,type=button		有效	
订单信息一览表文字	订单信息一览表	页面文字			
订单信息一览表		table			
订单信息列标	订单信息	th			
订单日期列标题	订单日期	th			
商品编号列标题	商品编号	th			
商标列标题	商标	th			
数量列标题	数量	th			
单价列标题	单价	th			
金额列标题	金额	th			
负责人列标题	负责人	th			
生产单位列标题	生产单位	th			

另外，一览表中各列所展示的订单信息是由JavaScript脚本代码生成出来的，不是写在HTML 5页面代码中的，因此没有在表23-1中列举出来。

接下来，我们来看一下该示例页面的HTML 5代码，如代码清单23-18所示。

代码清单23-18 订单信息输入页面的HTML 5代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="gb2312">
<title>
订单信息
</title>
<link href="mainstyle.css" rel="stylesheet" type="text/css" />
<script src="script.js" type="text/javascript"></script>
</head>
<body onload="showAllData();">
<section>
<header id="div_head_title_big">
编辑订单信息
</header>
<form id="form1" action="*">
<table cellspacing="1" cellpadding="0">
<tbody>
    <tr>
        <td id="td_title_1"><span>*</span>订单编号</td>
        <td id="td_content_1">
            <input type="required" name="tbxCode" id="tbxCode" maxlength="8"/>
```

```
</td>
<td id="td_title_2"><span>*</span>订单日期</td>
<td id="td_content_2">
    <input type="date" name="tbxDate" id="tbxDate"/>
</td>
<td id="td_title_3"><span>*</span>商品编号</td>
<td id="td_content_3">
    <input type="required" name="tbxGoodsCode" id="tbxGoodsCode"
    maxlength="12"/>
    <input type="button" name="btnSearchGoods" id="btnSearchGoods"
     value="..." onclick="btnSearchGoods_onClick();"/>
</td>
</tr>
<tr>
<td id="td_title_4">商标</td>
<td id="td_content_4">

    <input type="text" name="tbxBrand" id="tbxBrand" maxlength="3"
    onblur="tbxBrand_onblur()"/>
    <input type="button" name="btnSearchBrand" id="btnSearchBrand"
    value="..." onclick="btnSearchBrand_onclick();"/>
    <input  type="text"name="tbxBrandName" id="tbxBrandName"
    readonly="readonly" />
</td>
<td id="td_title_5">数量</td>
<td id="td_content_5">
    <input type="number" name="tbxNum" id="tbxNum" maxlength="6"
    value="0" onblur="tbxNum_onblur()"
    onkeypress=" Cancel_Input()"/>
    </td>
    <td id="td_title_6">单价</td>
    <td id="td_content_6">
        <input type="text" name="tbxPrice" id="tbxPrice" maxlength="6"
        value="0" onblur="tbxPrice_onblur()"/>
    </td>
</tr>
<tr>
    <td id="td_title_7">金额</td>
    <td id="td_content_7">
        <input  type="text"name="tbxMoney" id="tbxMoney"
        readonly="readonly"  value="0" />
    </td>
    <td id="td_title_8">负责人</td>
    <td id="td_content_8">
        <input type="text" name="tbxPerson" id="tbxPerson" maxlength="3"
        onblur="tbxPerson_onblur()"/>
        <input type="button" name="btnSearchPerson" id="btnSearchPerson"
        value="..." onclick="btnSearchPerson_onclick();"/>
        <input  type="text"name="tbxPersonName" id="tbxPersonName"
        readonly="readonly" />
    </td>
    <td id="td_title_9">生产单位</td>
    <td id="td_content_9">
        <input type="text" name="tbxProduct" id="tbxProduct" maxlength="3"
        onblur="tbxProduct_onblur()" />
```

```
            <input type="button" name="btnSearchProduct" id="btnSearchProduct"
            value="..." onclick="btnSearchProduct_onclick();"/>
            <input  type="text"name="tbxProductName" id="tbxProductName"
            readonly="readonly" />
        </td>
    </tr>
</tbody>
</table>
<div id="buttonDiv">
    <input type="button" name="btnNew" id="btnNew" value="新增"
    onclick="btnNew_onclick()"/>
    <input type="button" name="btnAdd" id="btnAdd" value="追加"
    onclick="btnAdd_onclick()"/>
    <input type="button" name="btnUpdate" id="btnUpdate" value="修改" disabled
    onclick="btnUpdate_onclick()"/>
    <input type="button" name="btnDelete" id="btnDelete" value="删除" disabled
    onclick="btnDelete_onclick()"/>
    <input type="button" name="btnClear" id="btnClear" value="清除"
    onclick="btnClear_onclick()" />
</div>
</form>
<section>
<section>
<header id="div_head_title_big">
订单信息一览表
</header>
<div id="infoTable">
<table id="datatable">
    <tr>
        <th>订单编号</th>
        <th>订单日期</th>
        <th>商品编号</th>
        <th>商标</th>
        <th>数量</th>
        <th>单价</th>
        <th>金额</th>
        <th>负责人</th>
        <th>生产单位</th>
    </tr>
    </table>
    <input type="hidden" id="hiddenBrand"/>
    <input type="hidden" id="hiddenPerson"/>
    <input type="hidden" id="hiddenProduct"/>
    </div>
    </section>
    </body>
    </html>
```

23.2.2 CSS 3样式代码分析

接下来，我们来看一下该页面所使用的CSS 3样式代码，该样式代码中值得特别注意的有以下几点：

- 将表单中所有放置页面文字的td元素的id设定为以“td_title”开头的文字，然后使用CSS 3中的[att^=val] 属性选择器统一设定此td元素的背景颜色、字体颜色等属性；将表单中所有放置输入控件的td元素的id设定为以“td_content”开头的文字，然后使用CSS 3中的[att^=val] 属性选择器统一设定放此td元素的宽度和高度、背景颜色、内边距等属性。
- 将表单中的商品编号检索按钮、商标检索按钮、负责人检索按钮、生产单位检索按钮的id都设定为以“btnSearch”开头的文字，然后统一设定这些检索按钮的背景图片、宽度及文字颜色。
- 使用CSS 3中的read-only选择器来设定只读控件的背景色为黄色。
- 将表单中的商标文本框、负责人文本框及生产单位文本框的id都设定为以“Name”结尾的文字，然后统一设定这几个文本框的宽度为60%。
- 使用nth-child(old)选择器来指定一览表中除标题行之外的所有奇数行背景色为浅灰色。
- 使用nth-last-child选择器单独指定一览表中最后两列的宽度为15%。

该页面所使用的CSS 3样式代码如代码清单23-19所示。

代码清单23-19 订单信息输入页面所使用的CSS 3样式代码

```
<style type ="text/css">
body {
    margin-left: 0px;
    margin-top: 0px;
}
header{
    font-size: 14px;
    font-weight: bold;
    color:white;
    background-color:#7088AD;
    text-align:left;
    padding-left:10px;
    display:block;
    width:100%;
}
[id^=td_title]{
    font-size: 12px;
    color: #333333;
    background-color:#E6E6E6;
    width:12%;
    text-align:right;
    padding-right:5px;
}
[id^=td_content]{
      width: 21%;
      height:22;
      background-color:#FAFAFA;
      text-align:left;
      padding-left:2px;
}
```

```
span{
      color: #ff0000;
}
input{
    width: 95%;
    border-top-style: solid;
    border-right-style: solid;
    border-bottom-style: solid;
    border-left-style: solid;
    border-top-color: #426C7C;
    border-right-color: #CCCCCC;
    border-bottom-color: #CCCCCC;
    border-left-color: #426C7C;
    border-top-width: 1px;
    border-right-width: 1px;
    border-bottom-width: 1px;
    border-left-width: 1px;
    border:1px solid #0066cc;
    height: 18px;
}
[id^=btnSearch]{
      background-image:url(image/btn_bg1.jpg);
      width:20px;
      color:white;
}
input:read-only{
      background-color:yellow;
}
input:-moz-read-only{
      background-color:yellow;
}
input#tbxNum{
      text-align:right;
}
input#tbxPrice{
      text-align:right;
}
input#tbxMoney{
      text-align:right;
}
#tbxGoodsCode{
      width: 85%;
}
#tbxBrand,#tbxPerson,#tbxProduct{
      width:20px;
}
[id$=\Name]{
      width:60%;
}
div{
      text-align:right;
}
div#buttonDiv{
      width:100%;
}
div input[type="button"]{
```

```
        font-size: 12px;
        width: 68px;
        height: 20px;
        cursor: hand;
        border:none;
        font-family:宋体;
        background-color:White;
        background-image: url(image/but_bg.gif);
        color: white;
}
div input[type="reset"]{
        font-size: 12px;
        width: 68px;
        height: 20px;
        cursor: hand;
        border:none;
        font-family:宋体;
        background-color:White;
        background-image: url(image/but_bg.gif);
        color: white;
}
input[type="text"]#tbxSearchCode{
        width:100px;
}
div#infoTable{
        overflow:auto;
        width:100%;
        height:100%;
}
div table{
        width:100%;
        background-color:white;
        cellpadding:1;
        cellspacing:1;
        font-size: 12px;
        text-align: center;
}
div table th{
        height:22;
        background-color:#7088AD;
        color: #FFFFFF;
        width:8%;
}
div table tr{
        height:30;
}
div table tr:nth-child(even){
        background-color:#E6E6E6;
}
div table th:nth-last-child(1){
        width: 15%;
}
div table th:nth-last-child(2){
        width: 15%;
}
</style>
```

23.2.3 JavaScript脚本代码分析

最后，我们来看一下示例页面中的JavaScript脚本代码，该页面的JavaScript脚本代码中的所有函数说明如表23-2所示。

表23-2 订单信息输入页面的JavaScript脚本代码中的所有函数说明

函数	函数调用条件	函数实现功能
init()	打开网页时调用	调用showAllData函数在一览表中显示全部订单信息
tbxBrand_onblur()	商标编号文本框失去光标焦点时调用	1. 清除商标编号文本框中内容。 2. 从本地数据库中取得输入的商标编号所对应的商标并填入商标文本框，如果不能获取则商标文本框内容保持为空
tbxPerson_onblur()	负责人编号文本框失去光标焦点时调用	1. 清除负责人编号文本框中的内容。 2. 从本地数据库中取得输入的负责人编号所对应的负责人，并填入负责人文本框，如果不能获取则负责人文本框的内容保持为空
tbxProduct_onblur()	生产单位编号文本框失去光标焦点时调用	1. 清除生产单位编号文本框中的内容。 2. 从本地数据库中取得输入的生产单位编号所对应的生产单位，并填入生产单位文本框，如果不能获取则生产单位文本框的内容保持为空
tbxNum_onblur()	数量文本框失去光标焦点时调用	1. 如果数量文本框中内容不为数字，则将其内容设定为0。 2. 如果数量文本框中数字不为整数，则将其内容设定为0。 3. 如果数量文本框中数字为整数，则将金额文本框中内容设定为数量*单价
tbxPrice_onblur()	单价文本框失去光标焦点时调用	1. 如果单价文本框中内容不为数字，则将其内容设定为0。 2. 如果单价文本框中内容为数字，则将金额文本框中内容设定为数量*单价
btnSearchGoods_onClick()	点击商品检索按钮时调用	1. 打开商品检索页面检索商品。 2. 商品检索页面关闭时将检索到的商品编号填入商品编号文本框
btnSearchBrand_onclick()	点击商标检索按钮时调用	1. 打开商标检索页面检索商标。 2. 商标检索页面关闭时将检索到的商标编号填入商标编号文本框。 3. 调用tbxBrand_onblur()函数在本地数据库中检索该商标编号对应的商标，并填入商标文本框
btnSearchPerson_onclick()	点击负责人检索按钮时调用	1. 打开负责人检索页面检索负责人。 2. 负责人检索页面关闭时将检索到的负责人编号填入负责人编号文本框。 3. 调用tbxPerson_onblur()函数在本地数据库中检索该负责人编号对应的负责人，并填入负责人文本框

（续）

函数	函数调用条件	函数实现功能
btnSearchProduct_onclick()	点击生产单位检索按钮时调用	1. 打开生产单位检索页面检索生产单位。 2. 生产单位检索页面关闭时将检索到的生产单位编号填入生产单位编号文本框。 3. 调用tbxProduct_onblur()函数在本地数据库中检索该生产单位编号对应的生产单位，并填入生产单位文本框
btnAdd_onclick()	点击追加按钮时调用	1. 将输入的订单信息追加到本地数据库中，如果追加失败则弹出窗口显示错误信息。如果追加成功则调用showAllData函数重新显示所有订单信息。 2. 追加数据成功后调用btnNew_onclick()函数清除表单中所有输入的订单信息，取消订单编号文本框的只读属性，将追加按钮设为有效状态，修改和删除按钮设定为无效状态
btnUpdate_onclick()	点击修改按钮时调用	1. 将当前订单编号所对应的订单信息修改为页面上当前用户输入的订单信息，如果修改失败则弹出窗口显示错误信息。如果修改成功则调用showAllData函数重新显示所有订单信息
btnDelete_onclick()	点击删除按钮时调用	1. 将当前订单编号所对应的订单信息从本地数据库中进行删除，如果删除失败则弹出窗口显示错误信息。如果删除成功则调用showAllData函数重新显示所有订单信息。 2. 数据删除成功后调用btnNew_onclick()函数清除表单中所有输入的订单信息，取消订单编号文本框的只读属性，将追加按钮设为有效状态，修改和删除按钮设定为无效状态
btnNew_onclick()	点击新增按钮时调用	清除表单中所有输入的订单信息，取消订单编号文本框的只读属性，将追加按钮设为有效状态，修改和删除按钮设定为无效状态
btnClear_onclick()	点击清除按钮时调用	1. 如果用户正在新增订单信息，则清除订单编号文本框中内容。 2. 清除表单中其他控件内容。 3. 设定数量文本框、单价文本框、金额文本框中内容为0
function tr_onclick (tr,i)	点击一览表中除标题行之外的其他行时调用	1. 将用户点击行的订单信息填入表单各控件中。 2. 设定追加按钮为无效状态，修改和新增按钮为有效状态。 tr表示被点击的一行数据，i表示该行行号

（续）

函数	函数调用条件	函数实现功能
showAllData(loadPage)	（内部函数，供其他函数调用）	1. 从本地数据库中查询出所有订单信息。如果查询失败则弹出窗口显示错误信息。 2. 如果该函数不是在页面打开时被调用的，则调用removeAllData函数清除一览表中所有数据。 3. 调用showData函数显示所有查询出来的订单信息。 Loadpage参数代表该函数是否在页面打开时被调用，参数值为true表示该函数在页面打开时被调用，参数值为false表示该函数在追加、修改或删除订单信息时被调用
removeAllData()	（内部函数，供其他函数调用）	清除一览表中所有数据
showData(row,i)		将从本地数据库中查出的一行数据显示在一览表中。row代表查询到的一行数据，i代表该行行号。

示例页面中的完整的JavaScript脚本代码如代码清单23-20所示。

代码清单23-20 示例页面中的完整的JavaScript脚本代码

```
<script>
var data = new Object;
var datatable;
var db = openDatabase('MyData', '', 'My Database', 102400);
function init()
{
    datatable= document.getElementById("datatable");
    showAllData(true);
}
function tbxBrand_onblur()
{
    var brand=document.getElementById("tbxBrand").value;
    document.getElementById("tbxBrandName").value="";
    var name;
    db.transaction(function(tx)
    {
        tx.executeSql('SELECT * FROM brand where code=?', [brand],
        function(tx, rs)
        {
            name=rs.rows.item(0).name;
            document.getElementById("tbxBrandName").value=name;
        });
    });
}
function tbxPerson_onblur()
{
    var person=document.getElementById("tbxPerson").value;
    document.getElementById("tbxPersonName").value="";
    var name;
    db.transaction(function(tx)
    {
        tx.executeSql('SELECT * FROM person where code=?', [person],
```

```
            function(tx, rs)
            {
                name= rs.rows.item(0).name;
                document.getElementById("tbxPersonName").value= name;
            });
        });
    }
    function tbxProduct_onblur()
    {
        var product=document.getElementById("tbxProduct").value;
        document.getElementById("tbxProductName").value="";
        var name;
        db.transaction(function(tx)
        {
            tx.executeSql('SELECT * FROM product where code=?',[product],
            function(tx, rs)
            {
                name= rs.rows.item(0).name;
                document.getElementById("tbxProductName").value=name;
            });
        });
    }
    function tbxNum_onblur()
    {
        var num,price;
        if(isNaN(parseInt(document.getElementById("tbxNum").value)))
            document.getElementById("tbxNum").value="0";
        num= document.getElementById("tbxNum").value;
        if(parseInt(document.getElementById("tbxNum").value)!=num)
            document.getElementById("tbxNum").value="0";
        num= parseInt(num);
        price=parseFloat(document.getElementById("tbxPrice").value);
        document.getElementById("tbxMoney").value= num * price;
    }
    function tbxPrice_onblur()
    {
        var num,price;
        num= parseInt(document.getElementById("tbxNum").value);
        price= document.getElementById("tbxPrice").value;
        if(isNaN(parseFloat(price)))
            document.getElementById("tbxPrice").value="0";
        price= parseFloat(price);
        document.getElementById("tbxMoney").value=num*price;
    }
    function btnSearchGoods_onClick(){
            var rc,w;
            rc=window.showModalDialog('SearchGoods.html','',
            'dialogHeight:720px;dialogWidth:700px;scroll:no');
            if(rc==null)
                return;
            document.all.item("tbxGoodsCode").value=rc;
    }
    function btnSearchBrand_onclick()
    {
            var rc,w;
            rc=window.showModalDialog('SearchBrand.html','',
            'dialogHeight:720px;dialogWidth:700px;scroll:no');
```

```
        if(rc==null)
            return;
        document.all.item("tbxBrand").value=rc;
        tbxBrand_onblur();
}
function btnSearchPerson_onclick()
{
        var rc,w;
        rc=window.showModalDialog('SearchPerson.html','',
        'dialogHeight:720px;dialogWidth:700px;scroll:no');
        if(rc==null)
            return;
        document.all.item("tbxPerson").value=rc;
        tbxPerson_onblur();
}
function btnSearchProduct_onclick()
{
        var rc,w;
        rc=window.showModalDialog('SearchProduct.html','',
        'dialogHeight:720px;dialogWidth:700px;scroll:no');
        if(rc==null)
            return;
        document.all.item("tbxProduct").value=rc;
        tbxProduct_onblur();
}
function btnAdd_onclick()
{
    data.Code=document.getElementById("tbxCode").value;
    data.Date=document.getElementById("tbxDate").value;
    data.GoodsCode=document.getElementById("tbxGoodsCode").value;
    data.Brand=document.getElementById("tbxBrand").value;
    data.Num=document.getElementById("tbxNum").value;
    data.Price=document.getElementById("tbxPrice").value;
    data.Person=document.getElementById("tbxPerson").value;
    data.Product=document.getElementById("tbxProduct").value;
    db.transaction(function(tx)
    {
        tx.executeSql('CREATE TABLE IF NOT EXISTS orders(code TEXT,
        date TEXT,goodscode TEXT,brand TEXT,num INTEGER,price FLOAT,
        person TEXT,product TEXT)',[]);
        tx.executeSql('INSERT INTO orders VALUES(?,?,?,?,?,?,?,?)',
        [data.Code,data.Date,data.GoodsCode,data.Brand,data.Num,
        data.Price,data.Person,data.Product],
        function(tx, rs)
        {
            alert("成功保存数据!");
            showAllData(false);
            btnNew_onclick();
        },
        function(tx, error)
        {
            alert(error.source + "::" + error.message);
        });
    });
}
function btnUpdate_onclick()
{
```

```
        data.Code=document.getElementById("tbxCode").value;
        data.Date=document.getElementById("tbxDate").value;
        data.GoodsCode=document.getElementById("tbxGoodsCode").value;
        data.Brand=document.getElementById("tbxBrand").value;
        data.Num=document.getElementById("tbxNum").value;
        data.Price=document.getElementById("tbxPrice").value;
        data.Person=document.getElementById("tbxPerson").value;
        data.Product=document.getElementById("tbxProduct").value;
        db.transaction(function(tx)
        {
            tx.executeSql('update orders set date=?,goodscode=?,brand=?,num=?,
            price=?,person=?,product=? where code=?',
            [data.Date,data.GoodsCode,data.Brand,data.Num,data.Price,
            data.Person,data.Product,data.Code],
            function(tx, rs)
            {
                alert("成功修改数据!");
                showAllData(false);
            },
            function(tx, error)
            {
                alert(error.source + "::" + error.message);
            });
        });
    }
    function btnDelete_onclick()
    {
        data.Code=document.getElementById("tbxCode").value;
        db.transaction(function(tx)
        {
            tx.executeSql('delete from orders where code=?',[data.Code],
            function(tx, rs)
            {
                alert("成功删除数据!");
                showAllData(false);
            },
            function(tx, error)
            {
                alert(error.source + "::" + error.message);
            });
        });
        btnNew_onclick();
    }
    function btnNew_onclick()
    {
        document.getElementById("form1").reset();
        document.getElementById("tbxCode").removeAttribute("readonly");
        document.getElementById("btnAdd").disabled="";
        document.getElementById("btnUpdate").disabled="disabled";
        document.getElementById("btnDelete").disabled="disabled";
    }
    function btnClear_onclick()
    {
        if(document.getElementById("btnAdd").disabled==false)
            document.getElementById("tbxCode").value="";
        document.getElementById("tbxDate").value="";
        document.getElementById("tbxGoodsCode").value="";
```

```
    document.getElementById("tbxBrand").value="";
    document.getElementById("tbxBrandName").value="";
    document.getElementById("tbxNum").value="0";
    document.getElementById("tbxPrice").value="0";
    document.getElementById("tbxMoney").value="0";
    document.getElementById("tbxPerson").value="";
    document.getElementById("tbxPersonName").value="";
    document.getElementById("tbxProduct").value="";
    document.getElementById("tbxProductName").value="";
}
function tr_onclick(tr,i)
{
    var tempArray1,tempArray2,tempArray3;
    var tc=tr.children;
    tempArray1= document.getElementById("hiddenBrand").value.split(";");
    tempArray2= document.getElementById("hiddenPerson").value.split(";");
    tempArray3= document.getElementById("hiddenProduct").value.split(";");
    document.getElementById("tbxCode").value=tc.item(0).innerHTML;
    document.getElementById("tbxDate").value=tc.item(1).innerHTML;
    document.getElementById("tbxGoodsCode").value=tc.item(2).innerHTML;
    document.getElementById("tbxBrand").value=tempArray1[i];
    document.getElementById("tbxBrandName").value=tc.item(3).innerHTML;
    document.getElementById("tbxNum").value=tc.item(4).innerHTML;
    document.getElementById("tbxPrice").value=tc.item(5).innerHTML;
    document.getElementById("tbxMoney").value=tc.item(6).innerHTML;
    document.getElementById("tbxPerson").value=tempArray2[i];
    document.getElementById("tbxPersonName").value=tc.item(7).innerHTML;
    document.getElementById("tbxProduct").value=tempArray3[i];
    document.getElementById("tbxProductName").value=tc.item(8).innerHTML;
    document.getElementById("tbxCode").setAttribute("readonly",true);
    document.getElementById("btnAdd").disabled="disabled";
    document.getElementById("btnUpdate").disabled="";
    document.getElementById("btnDelete").disabled="";
}
function showAllData(loadPage)
{
    db.transaction(function(tx)
   {
        tx.executeSql('SELECT orders.*,brand.name as brandName,
        product.name as productName,person.name as personName
        FROM orders
        inner join brand on orders.brand=brand.code
        inner join person on orders.person=person.code
        inner join product on orders.product=product.code', [],
        function(tx, rs)
        {
            if(!loadPage)
                removeAllData();
            for(var i = 0; i < rs.rows.length; i++)
            {
                showData(rs.rows.item(i),i);
            }
        },
        function(tx, error)
        {
            alert(error.source + "::" + error.message);
        });
```

```
        });
    }
    function removeAllData()
    {
        datatable= document.getElementById("datatable");
        for (var i =datatable.childNodes.length-1; i>1; i--)
        {
            datatable.removeChild(datatable.childNodes[i]);
        }

    }
    function showData(row,i)
    {
        var tr = document.createElement('tr');
        tr.setAttribute("onclick","tr_onclick(this,"+i+")");
        var td1 = document.createElement('td');
        td1.innerHTML = row.code;
        var td2 = document.createElement('td');
        td2.innerHTML = row.date;
        var td3 = document.createElement('td');
        td3.innerHTML = row.goodscode;
        var td4 = document.createElement('td');
        td4.innerHTML = row.brandName;
        var td5 = document.createElement('td');
        td5.innerHTML = row.num;
        var td6 = document.createElement('td');
        td6.innerHTML = row.price;
        var td7 = document.createElement('td');
        td7.innerHTML = parseInt(row.num)*parseFloat(row.price);
        var td8 = document.createElement('td');
        td8.innerHTML = row.personName;
        var td9= document.createElement('td');
        td9.innerHTML = row.productName;
        tr.appendChild(td1);
        tr.appendChild(td2);
        tr.appendChild(td3);
        tr.appendChild(td4);
        tr.appendChild(td5);
        tr.appendChild(td6);
        tr.appendChild(td7);
        tr.appendChild(td8);
        tr.appendChild(td9);
        datatable.appendChild(tr);
        if(document.getElementById("hiddenBrand").value!="")
            document.getElementById("hiddenBrand").value+=";";
        document.getElementById("hiddenBrand").value+=row.brand;
        if(document.getElementById("hiddenPerson").value!="")
            document.getElementById("hiddenPerson").value+=";";
        document.getElementById("hiddenPerson").value+=row.person;
        if(document.getElementById("hiddenProduct").value!="")
            document.getElementById("hiddenProduct").value+=";";
        document.getElementById("hiddenProduct").value+=row.product;
    }
    </script>
```