TURING

New Riders

Second Edition

Stylin' with CSS: A Designer's Guide

写给大家看的CSS书

（第2版）

[美] Charles Wyke-Smith 著

李松峰 张程 等译

- 学习CSS就这么简单
- 在不知不觉中成为Web设计高手
- Amazon全五星盛誉畅销著作

人民邮电出版社

POSTS & TELECOM PRESS

前　言

很难相信，从我写本书第 1 版到现在已经过了 3 年。在此期间，我又参与了大量网站的开发并调整了使用 CSS 的方式。我原计划对本书第 2 版进行一些小的调整，涵盖 IE 7 并从整体上使内容与时代同步。但是，最终我却对前 3 章进行了大量改进，并且完全重写了书中剩余的内容。因此，也使原计划几周的工作量用了将近一年才完成。

我所做的修改反映了任何经常使用 CSS 的开发人员在技术上的进步，而且这些修改也更深入地涉及所有程序员都需要掌握的两种技能：避免重写以前曾经写过的代码和掌握以最经济的方法编写代码的方法。在新版本中，介绍了实现这两个有价值的目标的各种方式。

重用和DRY

第一个目标是代码重用，它是我们在好几章中都会涉及的一个主题。书中展示了创建功能性构建块的技术，无论是页面布局的框架结构，还是应用样式的美观的链接列表，将它们保存为可以迅速添加到标记中的 CSS 类，设计者能够对其进行个性化地修改。本书已经将这些构建块添加到一个名为 Stylib 的代码库中，其中包含各种界面元素并以 CSS 代码及其相关 XHTML 标记的形式组织。这个库及本书的示例代码也放在了网站 www.stylinwithcss.com 中。虽然 Stylib 库还不算成熟，但已经包含了很多有用的组件，这些组件曾为我节省了很多时间——但愿对你也同样适用。今后，我还要不断完善这个库并将它们放到网站上供读者下载。

另一个目标是代码节约，它是贯穿本书的另一个主题。举例来说，为文档层次中最高层的元素应用一个样式，该样式就能够影响到其他众多元素。我经常看到别人的 CSS 样式表中包含为每个标题和段落声明的相同的 font-family 规则，而通过把 font-family 属性应用给顶层的 body 标签则意味着只需在一个地方编写和维护该属性。代码节约的底层理念是一句编程名言，叫做 DRY，意思是 Don't Repeat Yourself（不要重复自己）。在本书前面几章中，我们会展示很多类似的例子。

掌握关键技术

我认为只要真正理解了少数重要的 CSS 技术，就能够使一名新手变成能够独当一面的 CSS 熟手。这些技术包括正确地使用定位和 display 属性，以及理解浮动和清除的工作原理。在本书第 4 章中，将专门介绍这些 CSS 中的重点技术，并通过简单的例子示范如何在实践中运用它们。任何对 CSS 感兴趣并希望把自己的技能提升到一个新水平的读者，都会发现第 4 章非常具

呈现的页面，在 Netscape 中也应该得到几乎相同的结果。因此，如果基于以上 4 种浏览器测试和调整页面并取得满意的结果，就可以确保页面在几乎任何用户的浏览器中都能够得到体现设计意图的呈现。这里，我们甚至不会提到在 IE 5.5 中测试；因为它当前只有不到 0.5% 的使用率，而任何使用一个具有 8 年历史的老浏览器的用户，恐怕都会有比呈现我们的网页更大的技术问题。

我的建议是绝对不要使用 IE 6 作为开发浏览器，而要使用前面 3 个 SCB 中的一种，在 SCB 中已经能得到满意的页面后再在 IE 6 中测试并调整。实际上，更进一步讲，我会强烈建议你在开发期间使用 Firefox 来预览页面。之所以作此推荐，是因为我发现 Firefox 是所有 SCB 中最符合标准的一种浏览器。此外，在 Firefox 中安装 Web Developer Toolbar（Web 开发者工具条），就可以方便地验证 XHTML 和 CSS 而无需上传到服务器，启用和禁用 CSS，查看页面中 XHTML 元素的轮廓以便实际地确定元素是否按照计划进行布局，以及使用其他一些有用而且节省时间的工具。假如不安装 Web Developer Toolbar，那么大量开发时间就会白白浪费掉。而且，还可以为 Firefox 安装 Firebug 调试工具，有了这个附加软件，差不多就具有了进行 Web 开发的专业级环境。Firefox 和这两种附加软件都可以通过 www.getfirefox.com 免费下载。

请下载代码，别再自己敲了

尽管本书在出版之前进行了认真的编辑和审校，但错误仍然在所难免。我会及时修改书中代码存在的问题，并更新到本书网站（www.stylinwithcss.com）上。如果读者要试验本书的示例代码，请直接到本书站点下载。这样，不仅可以节省手工录入的时间，而且可以得到修正后的代码。

别忘了给我写信

如果你发现了错误，请到本书网站上发电子邮件告诉我，我会在确认之后把勘误公布到网站中。同时，我也希望看到读者的评论和建议，因此在你购买本书后要记着花几分钟给我写封信。我会尽最大努力回复每一封读者的来信，也会尽量回答读者遇到的任何问题。不过，假如你真的想把 CSS 代码发送给我看，请把它嵌入到 XHTML 文档的头部，并将页面中的图像指向你的服务器的绝对 URL；然后，将 XHTML 文档通过电子邮件发送给我。这样，只要打开页面，我就能看到相应的效果。

最后，感谢你购买这本书。希望它能够给你带来非常大的帮助。

本书中使用的图标说明


说明


提示


注意

① 可以在图灵网站 www.turingbook.com 本书网页免费注册下载。——编者注

目　录

第 1 章

XHTML：为内容定义结构

本书主要讲述如何以最有效、最简捷的方式构建符合标准的网站。并且，使这些网站容易被用户访问、也容易更新。Web 标准就是由 W3C（World Wide Web Consortium，万维网联盟）发布的一组推荐规范。如果所有浏览器制造商和所有 Web 程序员都遵循这些规范，那么从理论上讲，所有网页会在每个浏览器中都具有相同的外观和行为。但是，说起来容易，实行起来却很难。

2004 年底，当我撰写本书第 1 版时，Web 标准运动正处于风起云涌的态势。今天，大多数新建立的网站都以符合 Web 标准的方式来设计和编程。因此，Web 也已经发展到了一个更好的阶段。

由 Web 标准的倡导者推动的这一运动，使得 Web 不仅发展到了更好的阶段，而且也变得更容易预知。这些倡导者同浏览器制造商共同努力，确保了新浏览器能够按照 W3C 的推荐规范来解释 3 种主要的客户端编程语言（XHTML、CSS 和 JavaScript），而不是从强调竞争优势的一已私利出发各自为战地使用专有标签和特性。

1.1 Web标准

我经常会提到 CSS2 或 CSS3。这两个术语只是引用 CSS 标准的两个特定的版本。同其他技术一样，CSS 标准也会不断得到改进。目前，CSS2 在大多数浏览器中几乎都得到了完整的实现；CSS3 虽然已经出台一段时间了，但仍然只得到 Firefox 和 Opera 的部分支持，基本上没有得到 IE 7 的支持。本书后面会更多地提到两个版本的 CSS 标准。

在遵循最佳 Web 标准实践的基础上，像你我一样的 Web 开发者能够为所有用户实现外观和性能几乎一致的网站。例如，我们可能认为微软的 IE 是大多数符合 Web 标准的浏览器中最好的。但是，不管其现有的优势怎样，事实却不是如此。

根据 W3C 推荐规范，其他一些浏览器在解释 CSS 方面做得很好。像 PC 平台上最新版的 Firefox 和 Opera，以及 Mac 平台上的 Safari 和 Firefox 等，都能够以几乎一致的方式呈现按照 CSS2 规范设计的 XHTML 样式。然而，微软的 IE 6 却包含很多的未实现特性，并且还存在其他一些不标准的实现。

相对于 IE 6 来说，微软在 2006 年 10 月发布的 IE 7 则有了极大的改进。为此，我一直盼望着人们能够迅速地从 IE 6 切换到 IE 7。可是，根据 thecounter.com 的统计，截至 2007 年 7 月，仍然有 50% 的上网用户使用 IE 6[①]。

1.1.1 时至今日，仍然会提到 IDWIMIE

如果只对 IE 6 或者对 IE 6 和 IE 7 都是这种情况，我会一一指明。

不管怎样，由于 IE 6 还迟迟不肯退出历史舞台，我仍然会在提到某个 CSS 特性时告诉你 IDWIMIE（音 id-wimmy），即 It Doesn't Work In Microsoft Internet Explorer（它在微软的 IE 中无效）。

针对 IE（及其他较早浏览器）的缺点，存在一种叫做 hack 的解决方案。所谓 hack，就是指以非标准的方式来使用 CSS，以达到欺骗特定的浏览器，使其“看到”或忽略某些样式的目的。创建 hack 既乏味又耗时，但只要 IE 6 普遍存在，hack 的创建就要持续下去。

对于网站设计者和访问者而言，Web 标准为在所有平台的所有浏览器中实现一致的网站外观和行为提供了一种预期。虽然这一预期还没有实现，但是，每种浏览器都支持不同的特性集，从而导致结果不一致，并使得跨浏览器 / 跨平台的 Web 开发效率低下、难以驾驭的时代（似乎）已经一去不复返了。今天，Web 标准已经深入人心。

① 截至 2008 年 10 月 17 日，IE 7 的市场占有率（42%）已超过 IE 6（36%），参见 http://www.thecounter.com/stats/2008/October/browser.php。——译者注

1.1.2 内容、结构和表现

因此，遵循 W3C 推荐的 Web 标准，本书将向你展示如何使用 XHTML 定义内容的结构，如何使用 CSS 定义内容的表现。

(1) 内容是一个综合的概念，它包括文本、图像、视频、声音、动画和文件（例如 PDF 文档）等，所有你想交付给读者的东西。

(2) XHTML（eXtensible HyperText Markup Language，可扩展超文本标记语言）允许你定义内容中的每个元素是什么。是标题或者段落呢？还是一个项目列表、一个超链接或者是一幅图像？一切都由你为内容添加相应的 XHTML 标记来决定。标记由用来识别内容中每个元素的标签（标签名被一对尖括号 < > 包围）组成。要创建一个 XHTML 元素（以下简称元素），既可以使用一个开始标签和一个结束标签来围绕一块内容，像这样：

```
<p>This tag defines the text content as a paragraph</p>
```

也可以对非文本内容（如本例中的图像）使用一个单独的标签：

```
<img src="images/fido.gif" alt="a picture of my dog" />
```

本章主要讨论 XHTML 及如何使用它，但此刻最重要的是要知道：XHTML 定义一个文档的结构。

(3) CSS（层叠样式表）允许你定义每一个被标记了的内容元素怎样在页面上表现出来。某个段落的字体用 Helvetica 还是 Times？是粗体还是斜体？是缩进还是与页边对齐？CSS 能够控制每个内容元素的格式及定位。要格式化一个段落中文字的大小，我可能会这样写：

```
p {font-size: 12px;}
```

这会使文字的高度为 12 像素。本书的全部内容几乎都是教你学习 CSS，但此刻最重要的事情是要知道：CSS 定义一个文档的表现。

严格来讲，XHTML 和 CSS 并不是编程语言，而是分别用来标记内容和为内容提供样式的机制，因此这里所说的“语言”只具有一般化的含义。

img 标签也包含两个属性，分别是图像的来源和替代文本——要了解更多有关属性的内容，请参考提示条“什么是属性”。

什么是属性

属性能够添加到标签中，用以进一步定义标签。每个属性由两部分组成：属性名和属性值，格式为 name="value"。例如，下面的图像标签：

```
<img src="images/fido.gif" alt="a picture of my dog" />
```

包含两个属性：图像的来源，这个属性的值 "images/fido.gif" 定义了图像在服务器上的相对位置；图像的替代描述文本，这个属性的值 "a picture of my dog" 会在图像加载失败时出现在屏幕上或者被屏幕阅读器大声地读出来。类似这些标签的属性都属于文档的结构部分。

在 Web 标准出现之前，人们经常把用于表现的属性也放到标签中，例如文本大小和颜色等属性。现在，通过把所有表现信息都转移到样式表中，可以极大地降低标记的复杂性，而属性也只用来定义文档的结构。

开发 Web 标准的一个核心目标就是提供一种将文档结构与表现分开的方法，这对于开发既具有可移植性（可以在多种设备中显示）又具有持久性（未来仍然可用）的内容是至关重要的。

基于标准编码的10大好处

你可能会感到奇怪——“为什么要费力地改变多年来标记网页的方式呢？”，以下是采用基于标准编码实践的 10 大理由。

(1) 可以交付到多用户代理。同一标记块中的内容可以容易地交付到各种用户代理（对能够阅读 XHTML 文档的设备的统称），例如浏览器、带浏览器的智能手机、蜂窝电话，以及为视力残障人士设计的阅读文本的屏幕阅读器等。只需为每种类型的设备创建一个不同的样式表，或者毫无修饰地显示 XHTML 即可。

(2) 改善性能。由于内容中只包含最少的结构化标记，网页会更轻（文件更小），因而下载更快。可以用一个样式表替换掉通常要在网站每个页面中加入的表现标记。稍后会看到，一个样式表可以为整个网站定义表现，而用户的浏览器只需下载这个样式表一次。

(3) 适合所有浏览器。只需一点努力，就可以使网页很好地兼容早期的浏览器，于是所有用户都会尽可能地获得基于其可用技术的最佳体验。

(4) 分离内容和表现。可以修改甚至彻底改变网站的内容或表现（即设计），但不会因为一方改动而影响另一方。

(5) 构建流动式（自适应的）页面。在网页中编写可变数量的动态内容变得更加简单。例如：在电子商店中，对于给定的列表或菜单可以更容易地创建出容纳可变数量项目的页面。

(6) 验证代码良好。在开发期间可以随时通过 XHTML 和 CSS 验证服务即时地报告代码中的错误。这提供了一种快速调试的手段，如果页面能够在屏幕上正确显示并通过验证，则说明页面是完整的。

(7) 流水线式生产。生产会更富有效率。因为你（设计者）是唯一知道在大量的表现标记中内容位于何处的人，故而很容易将注意力转移到内容管理上来。最后不得不自己添加内容——工作单调而且可能也不是老板要求干的。而通过采用基于 Web 标准的实践，你可以为负责内容的小组提供简单的标记规则，同时自己可以并行地进行表现设计，并且很清楚他们的内容和你的设计能够完全无缝地衔接起来。

(8) 分发内容更容易。由于内容同具体的表现规则保持了分离，因而对第三方使用分发内容会容易很多。而且，在很多情况下，以其他方式基本上是不可行的。

(9) 使内容具有可访问性。更容易让网站具有可访问性并满足法律要求［例如，通常被称为 ADA 508 的《美国残疾人法》第 508 条（Americans with Disabilities Act, Section 508）］。

(10) 更小的工作量。不仅需要编写的代码会减少，而且能够更快和更容易地获得你需要的和修改后的结果。

1.2 时代在改变

现在，Web 标准已经被相当广泛地采用。设计者们放弃了使用表格来进行页面布局，转而使用清晰的结构化标记，告别了嵌套的表格、空白图像以及数不清的
（换行标记）和 （非换行空格）。如今，这些技术只用于强制一切各就各位，同内容不再有关系。

下面，我们来看一个使用老方法的例子。

1.2.1 过去的方式

这是在 Web 标准被普遍采用之前，如何编写网站的一个经典的例子。以下标记片段截取自 2004 年 7 月 1 日微软的主页。

```
<table cellpadding="0" cellspacing="0" width="100%"
height="19" border="0" ID="Table5">
<tr>
<td nowrap="true" id="homePageLink"></td>
```

```
<td><span class="ltsep">|</span></td>

<td class="lt0" nowrap="true" onmouseenter="mhHover('localTo
olbar', 0*2+2, 'lt1')" onmouseleave="mhHover('localToolbar',
0*2+2, 'lt0')">

<a href="http://go.microsoft.com/?LinkID=508110">MSN Home</a>
</td>

<td><span class="ltsep">|</span></td>

<td class="lt0" nowrap="true" onmouseenter="mhHover('localTo
olbar', 1*2+2, 'lt1')" onmouseleave="mhHover('localToolbar',
1*2+2, 'lt0')">

<a href="http://go.microsoft.com/?linkid=317769">Subscribe</a>
</td>

<td><span class="ltsep">|</span></td>

<td class="lt0" nowrap="true" onmouseenter="mhHover('localTo
olbar', 2*2+2, 'lt1')" onmouseleave="mhHover('localToolbar',
2*2+2, 'lt0')">

<a href="http://go.microsoft.com/?linkid=317027">Manage Your
Profile</a></td>

<td width="100%"></td>
</tr>
</table>
```

以上所有代码只是用来生成页面上的一行按钮（图 1-1）。

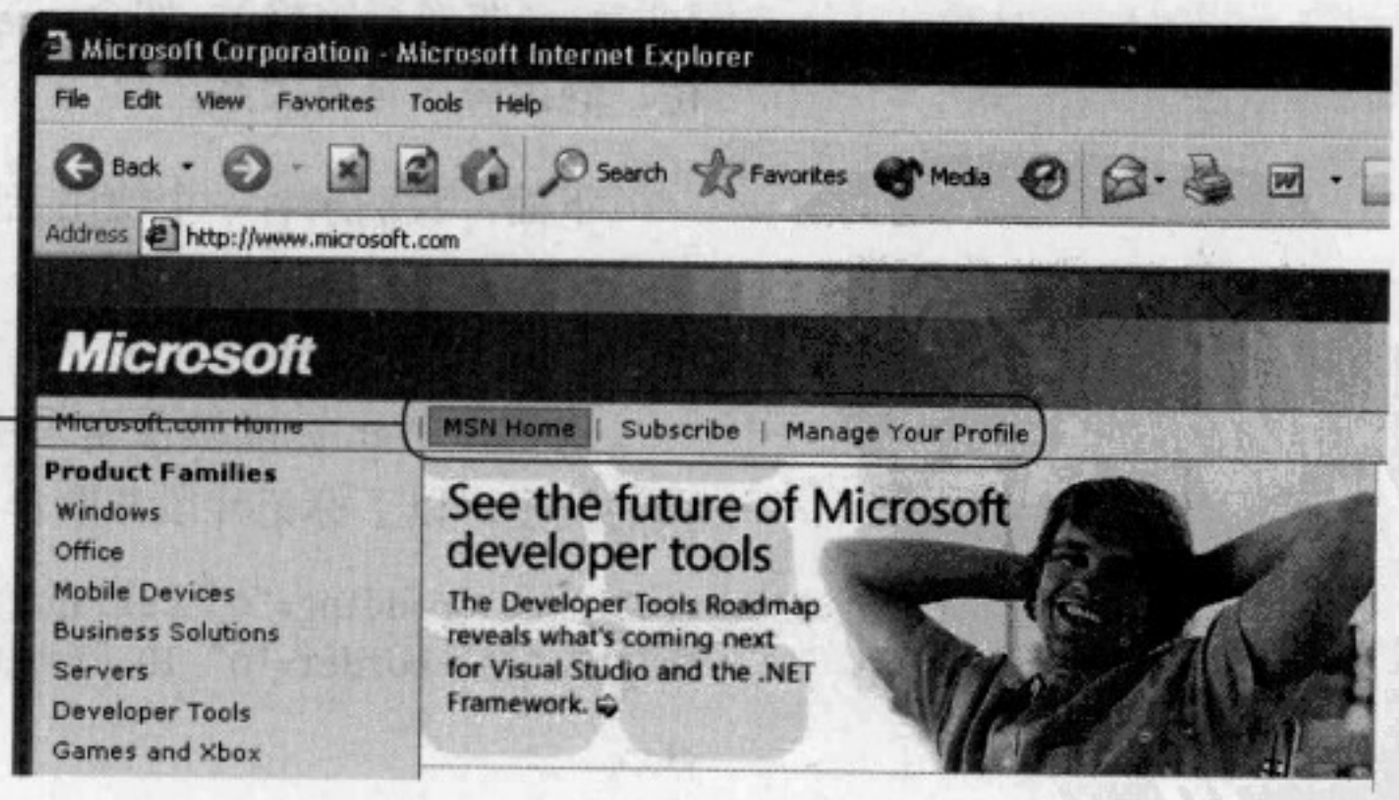


图 1-1　为了创建图中的 3 个链接，大约使用了 1 000 个代码字符

其中，关键的突出显示的代码只占 956 个字符中的 247 个，或者说不到全部代码的 26%。其他的 74% 只是“黏稠的巧克力

酱”而已。在这些标签中，除了 href 属性之外，其余全部都是表现代码。因此，这些表现代码都可以抽取出来并转换成样式表中几条简明的定义。使用表格的目的不是为了显示数据，只是用来对齐 3 个链接。剩下的代码主要用于生成翻转功能。每个链接都需要下列信息：在 JavaScript 中标识自身的类、保持文本不会换行的强制的 nowrap 属性和两个对 JavaScript 函数的调用——没错，每个链接都有（顺便说一句，通过 CSS 很容易创建翻转效果，我们在后面会介绍，只需两个简单的 CSS 样式即可）。同时，我们还注意到表格单元格中包含了一个嵌套的带类属性的 span 元素。这个 span 元素是为了显示链接之间的细竖线而专门设置的。

最近，微软为了使自己的网站更符合标准投入了相当大的努力。不过，如果网站的源代码（巧克力酱？）与上面的例子类似，那么还要继续往下看。在第 6 章中，我们会讨论如何在只有简单的无序列表标记的基础上创建与此类似的导航元素。在应用了一些 CSS 规则之后，我们会得到轻量级的、容易阅读的导航元素。而最关键的是，这些元素对能够识别 XHTML 的任何设备（无论其屏幕大小甚至能否理解 CSS）都将具有同样可访问的语义。

1.2.2 未来刚刚开始

今天，随着越来越多的浏览器和其他设备围绕 XHTML 和 CSS 进行了标准化，非标准的网站在这些新型的设备和浏览器中交付内容会变得越来越困难。你最近是否在掌上电脑中查看过自己的主页呢？

尽管把你当前的网站带到现代化的时代可能需要做一些实实在在的工作，但对新 Web 标准的遵循将会使你为此感到欣慰。因为，你可以只做一次，并且一次就能做对了。如果你刚开始做一个新网站，那么第一次就会做好。

在本书中，你将学会分离网站的内容和表现，以便网站适应未来。为此，需要使用 XHTML 将内容标记为网页，然后再通过一行代码，将这些页面链接到一个叫做样式表的文件。这个单独的样式表文件中包含着一组表现规则，每条规则定义了应该如何显示对应的标记。

可以使用 link 标签的 media 属性来定义样式表与哪种类型的设备关联。

这种分离所产生的效果在于，我们可以为浏览器、手持设备、打印设备，视力残障人士使用的屏幕阅读器等分别创建不同的样式表。今天的许多用户代理（各种设备）都期望看到针对它们的样式表。比如说，像 RIM 的 Blackberrry 和 Palm 的 Treo 这样的智能手机，会寻找为手持设备的使用而定义的样式表。如果这个样式表存在，则使用它。于是，我们就可以基于同样的 XHTML 在这些小屏幕设备上，提供一种经过调整的甚至完全不同的表现。

每个样式表都会使内容在相应的设备中以尽可能最佳的方式得到呈现，但只需一个版本的 XHTML 标记。后面我们会看到，XHTML 页面能够根据显示它的设备或环境自动选择正确的样式表。因此，“编写一次，使用多次”的内容会变得真正具有可移植性和灵活性，而且，同样的内容也能够适应将来发展中出现的任何表现需要。不过，同任何对将来的美好憧憬一样，一切仍然要从处理当前的具体问题开始。

1.3 XHTML及编写规则

如果你想了解更多关于 XML 的知识，可以参考 SpiderPro 网站（www.spiderpro.com/bu/buxmlm001.html）上面的 XML 教程。

由于 CSS 是为 XHTML 添加样式的一种机制，因而如果没有坚实的 XHTML 作为基础，就谈不上使用 CSS。那么，到底什么是 XHTML 呢？ XHTML 是将 HTML 作为 XML 进行的一次重新表达——明白了吗？简而言之，XHTML 基于自由形式的 XML 结构——XML 中的标签可以按照实际包含的内容来命名。例如，<starname>Madonna</starname>。XML 的这种非常强大的能力意味着在 XHTML 中，既有一组为 XHTML 内容自定义的标签，还有一个支持性文档——称为 DTD（Document Type Definition，文档类型定义）。后者说明了解释 XHTML 的设备应该如何处理这些标签。通过遵循 XML，XHTML 超越了 HTML 的限制，能够随着时间推移而得到扩展。并且，能够作为实时的 Web 服务在其他数据系统之间共享。每个 XHTML 网页开始处的 DOCTYPE 标签，就是用于在标记与 DTD 之间建立这种关键性的关联。

XML 在各行各业都得到了普遍应用，而 XHTML 中相同的 X（表示 eXtensible，即可扩展）则强调了不可阻挡的表现和内容分离的发展趋势。

本章其余部分将围绕 HTML 的这个最新的、彻底重新表达的、完全现代的、更灵活的版本展开。

1.3.1 XHTML——规则

通过正确地编写 XHTML 标记，能够确保页面在未来数年内仍然能够在各种主要的设备中恰当地显示。XHTML 清晰、易写、灵活的本性不仅使代码加载更快、编辑时容易理解，而且，也能够保证同样的内容可以在不同的应用程序中得到重用。

如果基于有效的 XHTML 规则编写格式良好的标记，而且样式表是有效的 CSS，那么就很容易确定网站是符合 Web 标准的。

格式良好意味着 XHTML 的结构按照本章下面介绍的标记规则进行了正确的组织。

有效意味着页面中只使用了 DTD 中定义的标签，而 DTD 在每一个现代的网页中都是通过页面的 DOCTYPE 标签（稍后会详细介绍 DOCTYPE）进行关联的。某些标签过去经常使用但现在已经不推荐，这意味着它们虽然还可以使用，但已经有了不同的、通常更具有语义的标签作为替代。为了鼓励使用新标签，同时标识出浏览器虽然支持但已不推荐使用的标签，验证程序会将不推荐使用的标签作为错误标识出来。为了检查网页是否符合这两个标准，可以把页面上传到一台 Web 服务器上，然后打开 http://validator.w3.org 并输入页面的 URL。

如果你在 Firefox 中安装了相应的开发者工具条，那么不必把页面上传到 Web 服务器也能在本地机器上验证它们。可以在 http://chrispederick.com/work/web-developer/ 上下载到这样的工具条。

单击“Check”[①]按钮并稍等几秒钟，就会看到一个包含页面中所有错误的详细列表，或者令人满意的“This Page Is Valid XHTML 1.0 Strict!”信息（见图 1-2）。也可以在 http://jigsaw.w3.org/css-validator 上以相同的方式验证 CSS。

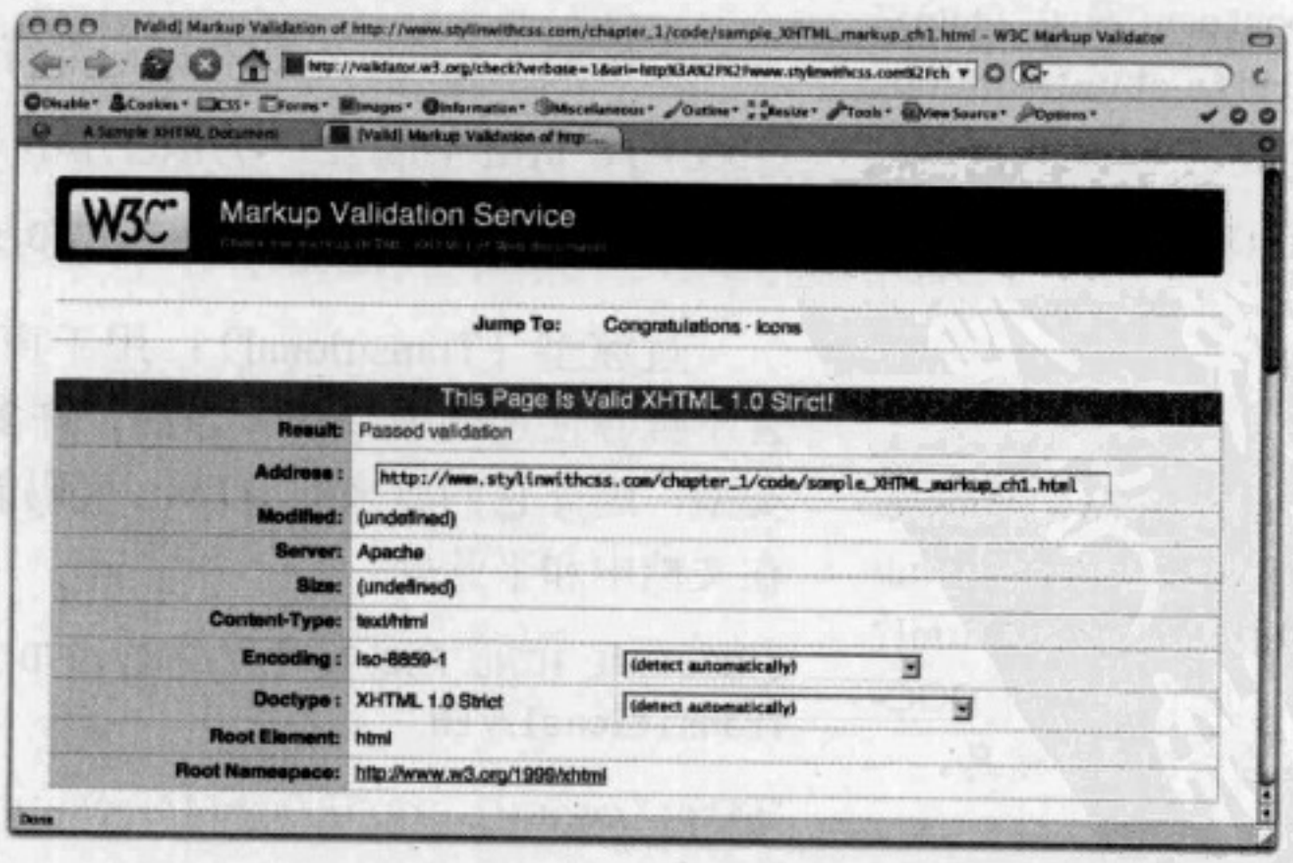


图 1-2 W3C 验证程序给出的好信息，基本上可以确保页面在任何理解 XHTML 的设备上得到有意义的呈现

① 原文 Press Submit 有误。因为上述验证页面中不存在 Submit 按钮。——译者注

页面必须要验证吗

现有的W3C验证程序（jigsaw.w3c.com）是为了确保页面有效（只使用DOCTYPE的DTD中定义的元素和属性）而且格式良好（标签的结构正确）。可以说，编写能够通过验证的页面是一种良好的习惯，而有些人可能会认为页面必须要通过验证。不可否认的是，验证程序能够及时地发现标记代码中的错误，如果不进行验证，那么要发现这些错误可能得花上数小时。

虽然不验证页面并不一定意味着页面不能在当前的Web浏览器中按照预期显示。但是，将来的设备或者其他非浏览器设备如何处理这个页面就不好说了。

我的建议是验证编写的每一个页面并纠正验证程序显示的错误。例如，关闭它发现的所有未关闭的标签，对嵌套不正确的标签（例如块级元素位于行内元素内部）进行重新编码，等等。简而言之，应该尽可能确保页面的格式良好。然而，有效则是另外一回事。

举例来说，为了使用Peter-Paul Koch的JavaScript代码在用户作出选择时显示表单的其他字段（参见http://www.quirksmode.org/dom/usableforms.html），就必须在包含着要显示元素的每个div标签中添加一个rel属性。对于div来说，rel不是一个有效的属性，因此页面不会通过验证。但是，文档仍然是格式良好的，于是我会甘心添加其他功能作为补偿以允许这个错误存在。尽管有的纯粹论者坚持认为每个页面都必须通过验证，但如果是因为这样的局部“有效”错误而导致验证失败，而文档的格式依然良好，我也认为可以忽略验证程序给出的建议。

欢迎读者就此问题给我发送电子邮件，以便共同讨论……

要了解那些应该放弃、并以相应的XHTML标签替代的不推荐标签，请参考About.com网站（http://webdesign.about.com/od/htmltags/a/bltags_deprctag.htm）。

可以在http://www.oreillynet.com/pub/a/javascript/synd/2001/08/28/doctype.html?page=1上了解到与DOCTYPE有关的更多信息。

下面是符合XHTML标准的完整（幸好不算太长）的编码要求。

(1) 声明DOCTYPE。DOCTYPE放在位于页面顶部的开始html标签前面，用于告诉浏览器页面中是包含HTML、XHTML，还是包含两者的混合，以便浏览器正确地解释页面中的标记。有3种DOCTYPE，分别用于告诉浏览器需要处理的3种主要标记类型。

严格型（Strict）：所有标记都符合XHTML标准。

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

过渡型（Transitional）：用于声明标记中既包含XHTML也包含不推荐使用的HTML。当前，许多公众瞩目的网站都使用了这种类型，因此它们原来的HTML代码能够与后来新加的XHTML代码在文档中和平共处。

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

① 根据本书源文件、Web标准网站（http://www.webstandards.org/learn/reference/templates/）提供的模板及上下文（即讲解XHTML，而不是HTML），将HTML 4.01过渡型改为XHTML 1.0过渡型。——译者注

如果你从其他站点复制 DOCTYPE 或命名空间声明，一定要确保 URL 是绝对 URL（换句话说，以 http:// 开头后跟完整文档路径的 URL）。有些网站（当然包括 W3C）会提供自己的 DOCTYPE 和命名空间文件，因此它们可能会使用相对 URL。如果你直接复制使用这些带相对 URL 的声明，那么由于不同的服务器上不存在这些文件，就会因为 URL 无效而导致页面的行为无法预知。

可以在 Dive into Mark 网站（http://diveintomark.org/archives/2002/05/29/quirks_mode）上学习到有关 Quirks 模式的更多内容。

由于输入 DOCTYPE（及规则 (2) 和规则 (3) 中讨论的 XML 命名空间和内容声明）很费事，因此本书网站上提供了相应的页面模板。可以使用这些模板作为 XHTML 文档的起点。根据你想使用的 3 种（严格型、过渡型或框架型）DOCTYPE 选择即可。[③]

框架型（Frameset）：同过渡型相同，但 XHTML 中不推荐的框架在这种类型下也将被视为有效。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">[①]
```

指定 DOCTYPE 非常重要。如果浏览器没有在标记中发现 DOCTYPE，就会假设网站中的页面编写于 Web 标准出现很早以前。我的建议是，如果你是从头开始构建网站，因此能够避免像 FONT 和 COLOR 这样不推荐的或者废弃的标签及属性，那么就应该使用前面提到的严格型的 DOCTYPE。

当遇到不包含 DOCTYPE 的页面时，许多浏览器都会进入一种被称为 Quirks 的模式中，这是一种由 Mozilla、Windows 平台中的 IE 6 和 Mac 平台中的 IE 5 支持的向后兼容特性。

在 Quirks 模式中，浏览器不会识别现代的 DOM（Document Object Model，文档对象模型），并装作从未听说过 Web 标准。浏览器的这种根据 DOCTYPE 的或有无来切换模式的能力，使得它们对符合标准或非标准网站中的代码，都会尽可能给出最好的解释。

不过，由于某些非常规的原因，DOCTYPE 标签不需要使用斜杠来关闭，并且 DOCTYPE 始终需要大写。而这同下面的规则 (4) 和规则 (7) 是完全抵触的。因此，要注意千万别混淆。

(2) 声明 XML 命名空间。注意下面例子中的新 html 标签：

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:
lang="en">
```

当浏览器在处理 XHTML 页面并希望知道相应的 DTD 中包含哪些已经定义的有效的 XHTML 标签时，就会在这里找到它——被深埋在了 W3C[②]的服务器中。

简而言之，DOCTYPE 和命名空间声明可以保证浏览器能够按照我们的期望来解释 XHTML 代码。

(3) 声明内容类型。内容类型声明出现在文档的头部，与其他 meta 标签放在一起。最常见的内容类型声明如下：

① 同上页脚注①，只不过是将 HTML 4.01 框架型改为 XHTML 1.0 框架型。——译者注
② 原文 WC3 有误。——译者注
③ 需要在本书网站上注册才能下载到这些模板及本书各章的源文件。——译者注

```
<meta http-equiv="Content-type" content="text/html;
charset=iso-8859-1" />
```

这里只是声明了文档使用什么字符进行编码。ISO-8859-1 是一种由所有标准的类似英语的语言使用的拉丁字符集。如果你的下一个站点打算使用古斯拉夫或希伯来字符，可以在微软的网站（http://msdn.microsoft.com/workshop/author/dhtml/reference/charsets/charset4.asp）上找到相应的内容类型。

(4) 关闭每个标签，无论是封闭的还是非封闭的。所谓封闭的标签，是指包含内容的标签，比如：

```
<p>This is a paragraph of text inside paragraph tags. To
be XHTML-compliant, it must, and in this case does, have
a closing tag.</p>
```

所谓非封闭标签，是指不包含文本但仍然需要在末尾使用空格加斜杠关闭的标签，比如：

```
<img src="images/siamese.jpg" alt="My cat" />
```

对于现代的浏览器来说，斜杠前面的空格并不是必需的，但为了看清标签是否正确地关闭，最好添加这个空格。

(5) 所有标签必须正确嵌套。如果某个标签在前一个标签关闭之前打开，那么这个标签必须在前一个标签关闭之前关闭。例如：

```
<p>It's <strong>very important</strong> to nest tags
correctly.</p>
```

这里，strong 标签正确地嵌套在了 <p> 中，因为它是在 p 标签关闭之前关闭的。这种一个标签位于另一个标签内的形式就叫做嵌套。

下面是错误嵌套的一个例子：

```
<p>The nesting of these tags is <strong>wrong.</p></
strong>
```

多个元素可以嵌套在一个包含元素中。比如，列表元素就可以在一个 ul 或 ol 元素中嵌套多个 li 元素，如下所示：

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

因为 CSS 依赖正确的嵌套来为元素应用样式，所以必须保证嵌套正确。使用 W3C 的验证程序可以确认所有标签嵌套正确，因此文档的格式良好。

(6) 行内标签不能包含块级标签。像 p（段落）和 div（部分）这样的块级元素，在页面中会按照从上到下的顺序自动组织自己。如果有两个段落，那么在默认情况下，第二个段落会出现在第一个段落的下方——无需使用换行符。另一方面，像 a（锚，即超链接）和 em（强调，通常显示为斜体）这样的行内标签，则会以常规文本流的形式出现，不会强制换行。

本书第 4 章将详细讨论块级元素和行内元素。现在，只要记住，如果将段落 p 这样的块级元素嵌套在像链接 a 这样的行内元素中，那么代码不会通过验证。

此外，某些块级元素也不能包含其他块级元素。例如，h1 到 h6（标题）标签不能包含段落。除了借助验证之外，运用常识也可以避免此类问题。因为，当我们撰写文章或者使用 Word 时，不会把整个段落放到段落的标题中。所以，只要不在 XTHML 文档中做出不合乎逻辑的事，那么就不会出现什么大问题。

(7) 标签全部小写。显然，这条规则的意思就是不能使用任何大写字母。我始终都是这样做的，如果你不是，那么 P 的日子结束了，从今天开始，必须写成 p。

(8) 属性必须有值，并且值必须加引号。在 HTML 中，某些标签的属性不需要值，但在 XHTML 中，所有属性都必须有值。例如，如果以前在 HTML 表单中使用 select 标签创建弹出菜单，并希望当页面加载后默认选中一个菜单项，你可能会编写如下标记代码：

```
<SELECT NAME=ANIMALS>
<OPTION VALUE=Cats>Cats</OPTION>
<OPTION VALUE=Dogs SELECTED>Dogs</OPTION>
</SELECT>
```

这样会使得在默认情况下将 Dogs 显示为下拉菜单的选项。

而等价的有效 XHTML 代码应该是：

之所以使用 a 作为链接的标签名，是因为能够在同一页面中跳到不同位置的链接叫做锚(anchor)。而同一个标签也可以用于跳到其他页面。因此，用于这一目的的 a 标签现在也被通称为链接。当然，XHTML 中也有一个链接标签（link），该标签用于向页面中关联样式表。注意不要将这两个链接标签混淆。记住，用户通过单击可以跳到新位置的“超链接”，技术上的术语叫做锚，而且总是使用 a 标签，即使人们都把这种机制称为链接。

```
<select name="animals">
<option value="cats">Cats</option>
<option value="dogs" selected="selected">Dogs</option>
</select>
```

我们注意到，在这个修正版中，所有标签和属性名都是小写的，而且所有属性的值都包含在引号中。

引用的属性值不一定小写。但小写所有代码是一个好习惯——这样，就可以避免出错。唯一没有遵守这一原则的就是 alt 属性，因为这个属性的值（文本字符串）可能会出现在屏幕上。

(9) 在内容中为 < 和 & 使用等价的编码符号。当浏览器在 XHTML 中遇到一个 < 时，会理所当然地将它看成一个标签的开始。但是，假如我们想让该符号出现在内容中该怎么办呢？答案就是使用实体来对它进行编码。所谓实体，指的是用于表示单个字符的短字符串。使用实体可以保证浏览器正确地解释并显示 XHTML 中的字符，而不会同标记混淆。< 的实体编码是 <——记住 lt 代表 less than（小于）。

实体不仅有助于避免上述解析错误，而且也可以保证正确地显示某些符号，例如 © 表示的版权符号（©）。每个符号的实体都以 & 开头，以 ; 结尾。由于 XHTML 中将 & 作为实体编码的开头，所以如果我们想在内容中显示它，也必须以实体来表示它，& 的实体是 &。

有很多工具可以将旧 HTML 标记转换成 XHTML。其中，HTML Tidy 被认为是最好的一个。Infohound 网站（http://infohound.net/tidy）上有一个 HTML Tidy 的在线版，同时也包含可下载版及文档的链接。尽管在转换完成后总是需要再进行一些手工清理工作，但 HTML Tidy 等工具却能够帮你节省数小时的时间。

对此，有一条不错的经验，即当使用的字符（如 é、®、© 或£）没有印刷在键盘上面时，就需要在标记中使用实体。

实体的总数大约为 50 000 个，基本上涵盖了全世界主要语言中的大多数字符集。Web Design Group 的网站上也有一个常用的简短的实体列表（www.htmlhelp.com/reference/html40/entities）。

以上，就是 XHTML 标记的规则。虽然这些规则相对比较简单，但如果你希望自己的页面通过验证，就必须严格遵守这些规则。

1.3.2 XHTML 模板

要通过 XHTML 的有效性验证，网页中必须包含几个特定的标签。在前面规则 (1)、(2) 和 (3) 中，我们介绍了必须要告诉浏览器页面中是否包含纯 XHTML，还是也包含不推荐的标签，以及页面使用了什么字符编码。无论页面中显示什么内容，这

些标签都必须存在。另外，还需要使用标签来表示页面的 head 和 body 区域。在 Dreamweaver 中，当选择“文件”菜单中的“新建”时，就会生成一个包含所有必要元素的“页面模板”，供你向其中添加页面内容。也可以预先设置出现在页面顶部的 DOCTYPE，方法是选择“编辑→首选参数→新建文档→默认文档类型（DTD）”。

本书中使用的术语“页面模板”，只是指类似下面所示的一段代码，这样的代码是一个有效 XHTML 页面的基础，而不是指 Dreamweaver 或内容管理系统使用的包含页面布局中不可修改部分的页面模板。

为展示这种模板，我们来看一看在使用严格型 XHTML 1.0 DOCTYPE 的情况下，确保文档有效且格式良好的必要代码：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<!--the DOCTYPE-->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>XHTML 1.0 Strict template</title>
</head>
<body>
        <!--the content of your page goes here-->
</body>
</html>
```

这个模板位于本书网站中的下载包中。要得到更多 HTML 或 XHTML 模板，请访问 Web Standards 网站（www.webstandards.org/learn/templates/index.html）。

在使用这个模板时，可以将以上代码放到页面中。不过，应该注意修改 title 标签，使其中的文本能够描述页面内容。这样，无论是对屏幕阅读器还是对搜索引擎都有好处。更多信息请参见提示条“关于 title 标签”。

关于title标签

由于页面的 title 标签（页面标题）显示在浏览器窗口的最顶端，所以人们很容易忽略它。但是，title 标签对于搜索引擎来说，却占有相当大的权重。比如，能够出现在 Google 搜索结果页面中的网页，它们的页面标题中总是包含着部分或者全部搜索关键词。而且，页面标题也是作为每个搜索结果的标题显示的。保证页面标题中包含用户可能用作搜索关键词的内容，可以在该标题出现在结果中时诱使用户单击。因此，千万不要让用处不大或者太一般的“欢迎光临我们的主页”浪费 title 标签。

1.3.3 标记内容

采用 Web 标准意味着全新的工作方式。在开发过程的开始，要考虑内容的结构——它的含义是什么，而不是考虑内容的表现——它看去应该是什么样。不过，在不知道最终完成页面外观的情况下就开始编写代码是荒谬的。我通常在开始之前，先简单地制作一幅 Adobe Firewoks 效果图，取得用户对设计的认可。然后，再以这幅效果图作为参考来标记页面中的内容。当实际地标记出页面元素（如标题、段落、图像等）时，也就有了通过 CSS 添加样式的基础。因此，我的焦点就会集中在“对每一块内容应该使用哪个标签最有意义”上。

当下一章介绍了 CSS 的工作原理后，我们就可以从使用正确的标签标记内容元素和确保以 CSS 规则容易识别的方式组织这些元素的角度来观察标记。

现在，主要讨论个别的 XHTML 标签及它们的语义，以便能够对出现在页面上的任何内容片段进行斟酌，并选择最合适的标签将它们标记起来。在此期间，也需要考虑到文档流的概念。

1.3.4 文档流——块级元素和行内元素

前面提到过，多数 XHTML 标签都可以根据它们在页面上的显示方式分成两个主要的类别：块级元素和行内元素。像标题 <h1> 到 <h6> 及段落 <p> 这样的块级元素，会在无需换行符的情况下，自动在页面上堆叠起来。甚至，这些块级元素之间都会预先设置一些外边距，以便保持彼此之间的距离。行内元素没有这些外边距，它们会肩并肩地横向排布在页面上，只有当页面中没有足够的空间供它们彼此相邻时，才会折到下一行中显示。

1. XHTML 标记的简单例子

在第一个简单的 XHTML 页面的例子中，屏幕截图不仅显示了块级元素堆叠的效果，也显示了行内元素，即本例中的 <a>（链接）和 （通常显示为粗体），能够出现在块级元素中而无需换行（见图 1-3）。

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

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

<head>

<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />

<title>Block and inline XHTML elements</title>

</head>

<body>

<h1>Here's a level 1 heading</h1>

<h2>Here's a level 2 heading</h2>

<p>This paragraph is very short and contains very little
text.</p>

<p>This paragraph is longer and contains not only
<strong>bold text</strong> but also <a href="#">a link</a>
that doesn't really link to anything.</p>

</body>

</html>
```

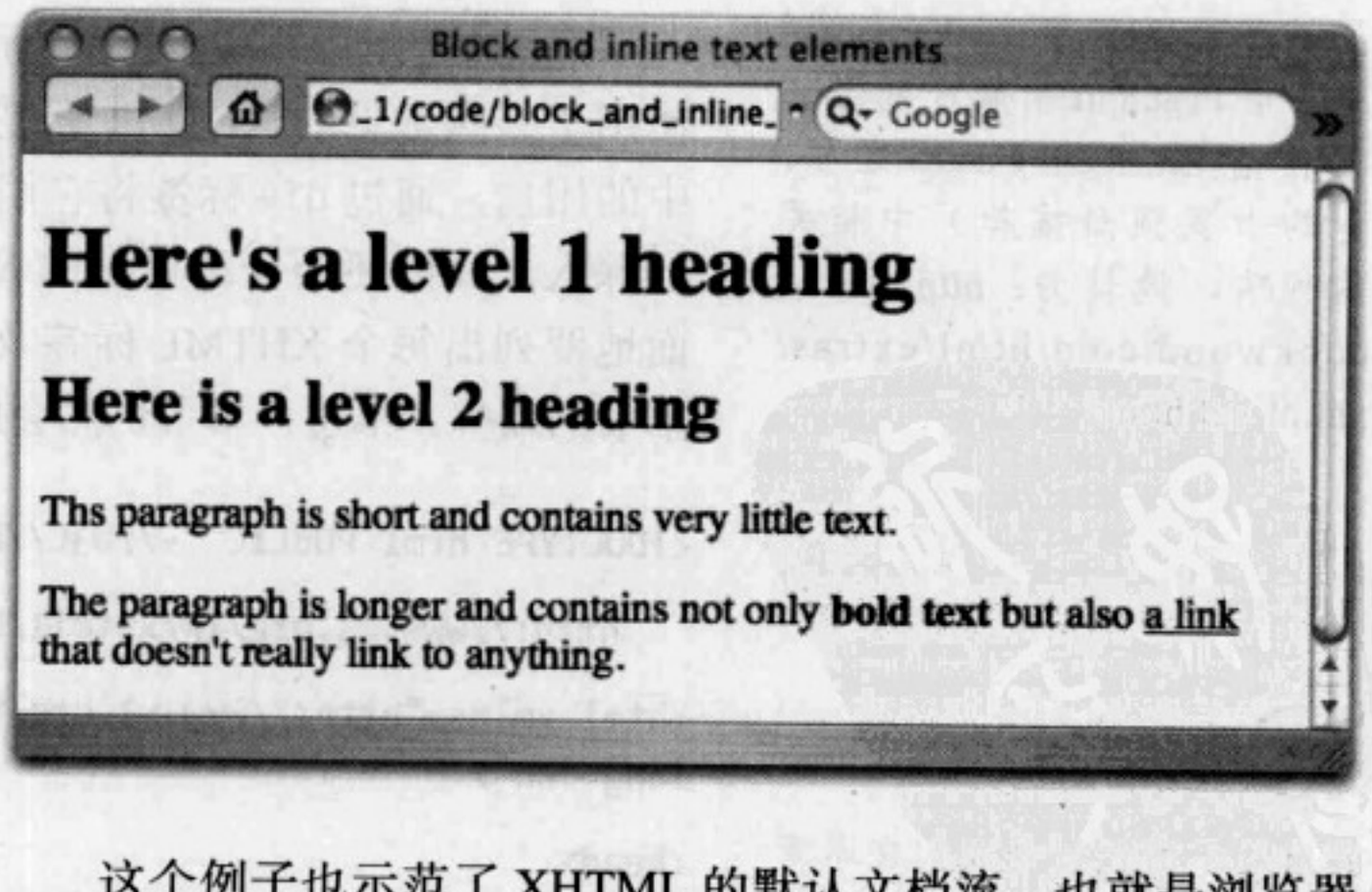


图 1-3 块级元素和行内元素文档流的例子

这个例子也示范了 XHTML 的默认文档流，也就是浏览器对块级元素和行内元素的布局方式。文档流确保了没有关联设计者样式而且正确标记的内容能够呈现为有意义的页面。后面我们会看到，使用 CSS 可以通过各种方式把默认的文档流组织

成为多种布局（例如，多栏），而且无需动用表现标签和属性来污染标记。

2. 浏览器的内部样式表

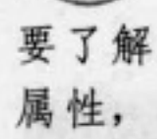
这里，要注意一件有意思的事，即在默认情况下，每个元素都有某些与它关联的样式。如图 1-3 所示，h1 标题的字体比 h2 标题的更大一些，而段落文档的字体比这两个标题的字体都小一些。这是因为浏览器有一个内置的样式表，用于设置每个元素默认的字体大小、颜色（例如，一般文字是黑色，链接是蓝色）、显示设置（块级或行内）以及很多其他样式。

当我们使用 CSS 为元素添加样式时，实际上是覆盖了浏览器样式表中为该元素设置的默认样式。这样会使问题比较简单，因为我们只需改变不符合我们期望的样式即可。然而，当由于某种原因浏览器无法读取我们的样式表时，这些默认的样式就会成为后备样式。为此，在开始添加 CSS 之前，有必要确保已标记但未应用样式的页面能够在浏览器中以有意义的方式显示。如果页面中包含格式良好的 XHTML，那么默认的文档流就可以保证这一点。

要了解基本的 XHTML 标签及属性，可以参考 Cookwood 网站（由 Peachpit 出版社的另一位作者 Elizabeth Castro 维护，他的书我强烈推荐）中相关的列表，地址为：http://www.cookwood.com/html/extras/xhtml_ref.html。

3. 复杂一些的结构化 XHTML 页面

下面是一个更接近实际的标记页面的例子（图 1-4），其中使用了一些常见的 XHTML 标签，同时也根据这些标签在页面中的用途，通过 div 标签将它们组织成了逻辑分组。随着学习的深入，我们还会介绍更多的 XHTML 标签。这里，我不会全面地罗列出每个 XHTML 标签及其相关的属性，因为这需要一本书的篇幅。不过，本书仍然会展示很多不同的标签及其用法。



突出显示的代码是我们在本章前面看到的 XHTML 模板。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:
lang="en">
<head>
<title>A Sample XHTML Document</title>
<meta http-equiv="Content-type" content="text/html;
charset=iso-8859-1" />
```

```
<meta http-equiv="Content-Language" content="en-us" />
</head>
<body>
<!--header-->
<div id="header">
<img src="images/stylin_logo1.gif" width="150" height="80"
alt="Stylin logo" />
  <h3>a New Riders book by Charles Wyke-Smith</h3>
</div>
<!--end header-->
<!--main content-->
<div id="contentarea">
  <h1>Welcome to XHTML</h1>
  <p>Good XHTML markup makes your content portable,
  accessible and future-proof. Creating XHTML-compliant pages
  requires following a few simple rules. Also, XHTML code
  can be easily validated online so you can ensure your code
  is correctly written.</p>
  <p>Here are the key requirements for successful validation
  of your XHTML code.</p>
  <ol>
    <li>Declare a DOCTYPE</li>
    <li>Declare an XML namespace</li>
    <li>Declare your content type</li>
    <li>Close every tag, enclosing or non-enclosing</li>
    <li>All tags must be nested correctly</li>
    <li>Inline tags can't contain block level tags</li>
    <li>Write tags in lowercase</li>
    <li>Attributes must have values and must be quoted</li>
    <li>Use encoded equivalents for left brace and
    ampersand</li>
  </ol>
```

```
    <a href="more.htm">more about these requirements</a> </div>
    <!--end main content-->
  <!--navigation-->
  <div id="navigation">
    <p>Here are some useful links from the Web site of the
    <acronym title="Word Wide Web Consortium">W<sup>3</sup>C
    </acronym> (World Wide Web Consortium), the guiding body of
    the Web's development.</p>
    <ul>
      <li><a href="http://validator.w3.org">W3C's XHTML
      validator</a></li>
      <li><a href="http://jigsaw.w3.org/css-validator/">W3C's
      CSS validator</a></li>
      <li><a href="http://www.w3.org/MarkUp/">XHTML Resources
      </a></li>
      <li><a href="http://www.w3.org/Style/CSS/">CSS
      Resources</a></li>
    </ul>
  </div>
  <!--end navigation-->
  <!--footer-->
  <div id="footer">
    <p>&copy; 2007 Charles Wyke-Smith.</p>
  </div>
  <!--end footer-->
  </body>
  </html>
```

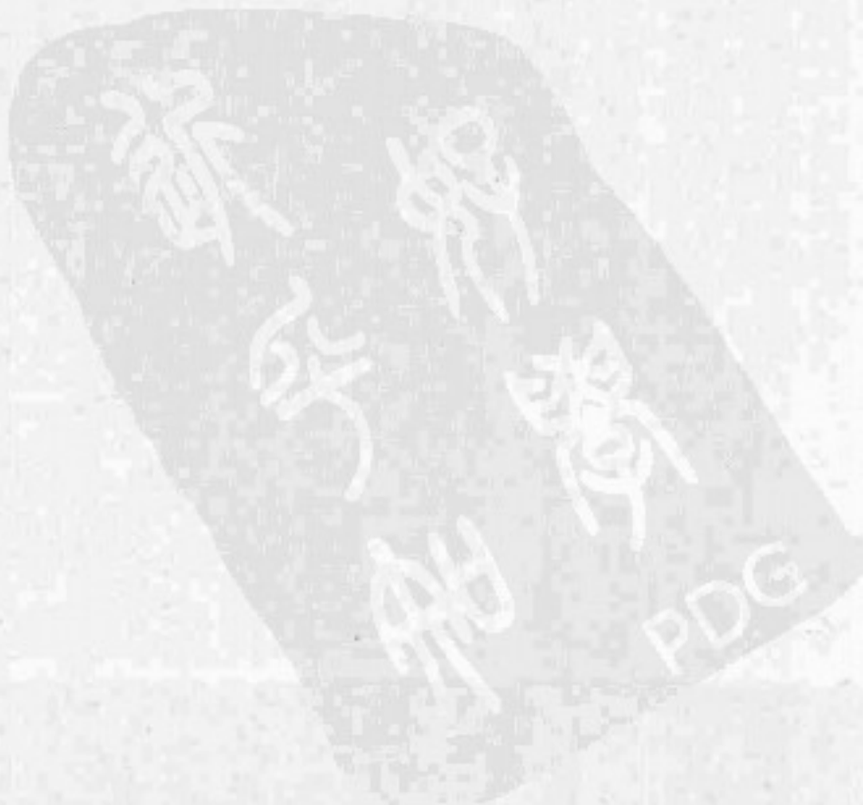
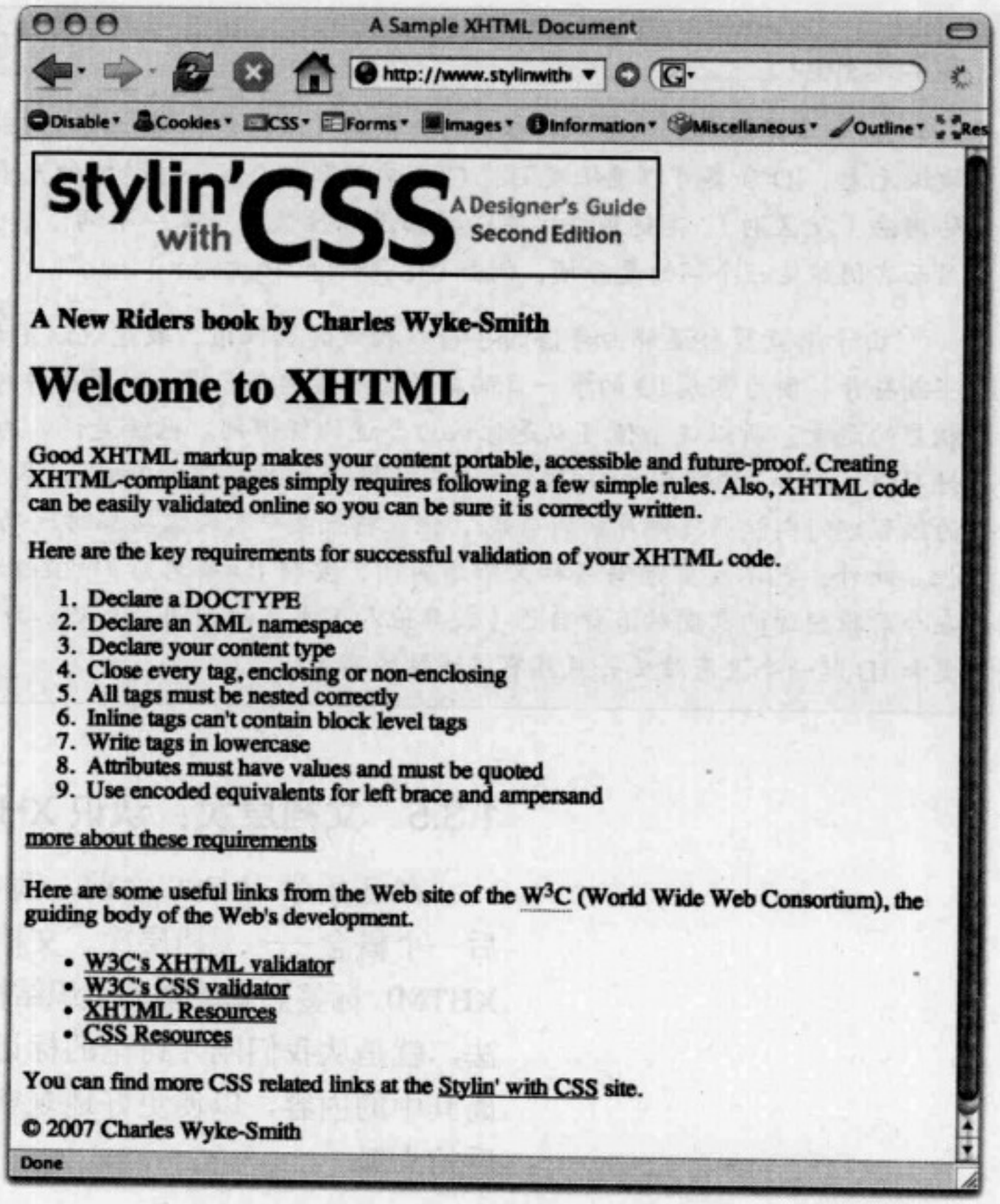


图 1-4 这是前面的代码在 Firefox 浏览器中以默认的浏览器样式呈现的结果。虽然不算漂亮，却很便于使用（另见彩插）



环绕在标题图像周围的默认蓝色边框，表示该图像是可以单击的（因为 img 标签包含在 a 标签中）。后面我们会学习到，这个相当难看的边框通过 CSS 很容易去掉。

从可以用来标识标签分组的角度来说，类属性与 ID 很相似。但是，同一个类可以在页面中出现多次，而 ID 则只能出现一次。在下一章中，我们会学习如何正确地使用类和 ID。也可以参考提示条“命名类和 ID”。

这个页面很好地示范了由正确标记的元素生成的固有的结构化的文档流。我们注意到代码中的标记通过（适当命名的）div 分成了 4 个逻辑分组：头部、内容、导航和页脚。这些 div 的 id 属性可以让我们为页面中的每个分组起一个唯一的名字。

虽然这些 div 的 ID 有助于我们对文档结构的各个部分一目了然，但将标签组织到带 ID 的 div 中的主要目的，还是为了把一组 CSS 规则应用到一组特定的标签上，而不是应用到整个文档。这样，一种元素类型（例如 p）在一个 div 中可以应用一种样式，而在另一个 div 中可以应用另一种样式。下面，我们通过分析文档层次来理解相应的工作原理。

命名类和ID

ID 和类是添加到标签中的标识符。虽然可以将类和 ID 添加给任何标签，但最常见的还是添加给块级元素。ID 和类可以确保我们将 CSS 应用到一个或一组特定的元素上。后面，我们会介绍 ID 和类的用法（及区别）。不过现在，有必要知道属性值必须是一个词，尽管可以使用下划线来连接对浏览器而言仍然是一个词的复合词，例如 class="navigation_links"。

由于浏览器会误解由奇怪的字符串构成的属性值，我建议以字母作为这个词的开头，而不是数字或符号。因为类或 ID 的唯一目的就是为元素起个名字，以便在样式表（或 JavaScript 代码）中引用相应的元素，所以这个值可以是你认为合适的任何词。也就是说，为类和 ID 指定有意义的值是一个好习惯，例如 class="navigationbar" 就好于 class="deadrat"。尽管 deadrat（死老鼠）类在折磨人的编程期间内能够提供片刻的自娱，但在将来某一时刻需要编辑代码时，这种幽默感就会成为一种负担。此外，也不要靠缩写名称来节省时间，要将 ID 命名为 "footer" 而不是 "fr"；否则，很可能会在今后猜想你的意图时浪费自己（或其他人）更多的时间。总之，为了照顾自己，也应该花点时间为类和 ID 起一个没有歧义并且具有描述性的名字。

1.3.5 文档层次：认识 XHTML 家族

在开始学习 CSS 之前，我们再来看与 XHTML 相关的最后一个概念——文档层次。文档层次就像是家谱或基于页面中 XHTML 标签嵌套关系的组织结构图。理解这个概念的一个好办法，就是从我们刚才讨论的标记主体部分中截取一个片段并剥离其中的内容，以便更好地观察标签的结构。下面就是剥离之后的头部：

```
<body>
  <div id="header">
    <img />
    <h3> </h3>
  </div>
<!-- 为了清晰起见，这里省略了其余的标签 -->
</body>
```

这样，我们就能清晰地看出标签之间的关系。例如，在标记中，body 标签包含（或嵌套）了其他所有标签。而且，div 标签（ID 为 "header"）包含两个标签：img 标签和 h3 标签。

Dreamweaver 的代码视图能够自动缩进上面嵌套的标签（命令→应用源代码格式化），因此可以帮我们更清楚地看到这些层次。

通过观察这个分层视图（见图 1-5），我们可以说 img 和 h3 标签都是 div 标签的子标签，因为 div 包含这两个元素。相应地，div 标签是这两个元素的父标签，而 img 和 h3 标签彼此之间是同辈标签，因为它们有相同的父标签。最后，body 标签是 img 和 h3 标签的祖先元素，因为后两个标签间接地源自 body。同样地，img 和 h3 标签（在同样的意义上说，也包括 div 标签）都是 body 标签的后代标签。用斯莱·斯通（Sly Stone）乐队的话说："事关一个家族……"

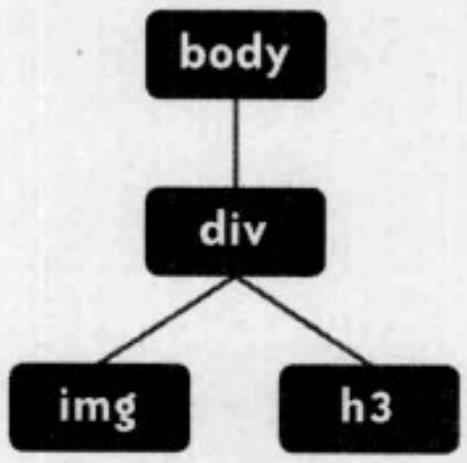


图 1-5 通过层次框图表现的文档结构

在 CSS 中，可以通过某种简写的方式来表达这些关系，例如：

```
div#header img {some CSS styling in here}
```

这条 CSS 规则只会选择位于（源自）ID 为 "header"（# 是 CSS 中表示 ID 的符号）的 div 中的 img 标签。页面中其他的 img 标签不会受这条规则的影响，因为它们没有包含在 ID 为 "header" 的 div 中。这样，我们就可以只为这幅图像添加边框，或者通过为它设置外边距使它远离周围的元素。

在下一章中，我们会深入学习如何编写类似这样的 CSS 规则，但现在需要理解的一个重要概念是，位于文档主体中的每个标签都是 body 标签的后代。并且，取决于它们在标记中的位置，每个标签都可能是文档层次中其他标签的祖先元素、父元素、子元素或同辈元素。

通过创建使用（并且通常是组合使用）ID、类及这种层次结构的规则，我们就拥有了精确地指定哪条 CSS 规则影响哪个（些）XHTML 元素的手段，这正是接下来我们将要讨论的内容。

第 2 章

CSS 的工作原理

在第 1 章中，我们介绍了为文档提供结构化层次的 XHTML 标记。也看到了由浏览器应用了其默认样式的元素，而且每个元素或者显示为块级（堆叠）元素、或者显示为行内（并排）元素。在标记正确的情况下，文档会以便于使用的方式自动在页面中自上而下排布——给人以文档流的感觉。但是，“便于使用”不等于“漂亮好看”，因此，有创造力的人（比如你）可以使用 CSS 来改变浏览器的默认样式，同时添加更多的样式，以便为访问者创建更实用也更具有审美效果的设计。在本章中，我们将探索 CSS 的机制，到本章结束时，你就可以为第 1 章中学习的示例 XHTML 标记创建自己的样式了。

2.1 为文档应用样式的 3 种方式

为网页添加样式的方式有 3 种：内联、嵌入和链接到单独的 CSS 样式表。其中，真正对开发网站有意义的是将 CSS 样式表链接到 XHTML 页面。不过，我们也会介绍另外两种添加样式的方式，因为它们对创建页面也有帮助。

一个样式表能够链接到无限个 XHTML 页面，这样既可以保证页面之间的一致性，也能够使对样式的修改立即在整个网站中得到反映。

2.1.1 内联样式

内联样式（也称为局部样式）是指通过 XHTML 的 style 属性为标签添加的样式，比如：

```
<p>This paragraph simply takes on the browser's default
paragraph style.</p>

<p style="font-size: 25pt; font-weight:bold; font-style:
italic; color:red;">By adding inline CSS styling to this
paragraph, we can override the default styles.</p>

<p>And now we are back to a regular default paragraph without
any inline styles.</p>
```

以上样式的效果如图 2-1 所示。

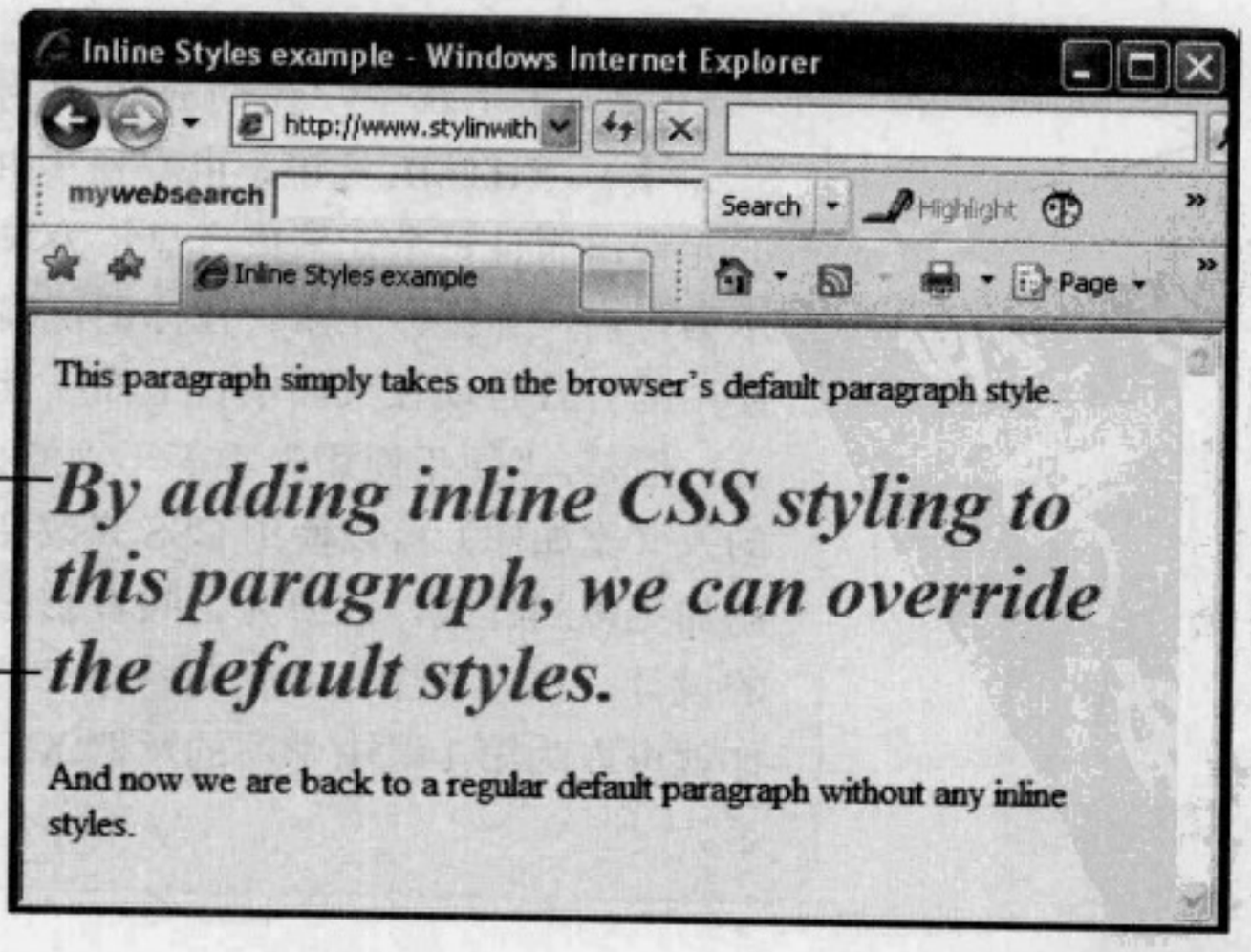


图 2-1 内联样式只应用于它所在的标签

以下是有关内联样式的一些基本常识。

- 它们的作用范围很有限。内联样式只会影响它所在的标签。
- 使用内联样式同我们很久以前的做法一样，只是把表现标记直接放在标签中的另一种方式而已。随处添加内联样式对可移植性和可维护性造成的破坏，同添加像FONT这样不推荐的HTML标记所造成的破坏没有什么区别。因此，通常应该避免使用内联样式。
- 在某些罕见的情况下，如果只需要覆盖一个特殊实例的样式但又没有其他好办法，也可以使用内联样式而不必感到不安。也就是说，通过为相关的标签添加唯一的ID或类，然后在样式表中编写对应的样式，几乎总是可以避免使用内联样式。
- 在把样式转移到样式表中之前，使用内联样式也是一种不错的试验方式（参见2.1.3节）。不过，千万别忘记在完成期望的效果之后完全清除style属性，并把相应的样式剪贴到样式表中。如果把内联样式丢在了标记中，那么它总是会覆盖在样式表中对相应标签所做的任何修改。并且，当问题实际上隐藏在标记中时，你可能会因为修复样式表而搭上数小时的时间。
- 内联样式会覆盖在嵌入样式（下面会介绍到）中定义的相同样式。而嵌入样式则会覆盖你在样式表中定义的全局样式（有关这一条的详细说明，请参见2.8节）。

2.1.2 嵌入样式

嵌入样式就是放在XHTML文档头部中的一组CSS样式。之所以把它们称为*嵌入样式*（或*页面样式*），是因为它们是页面的一部分（或者说是嵌入到了页面中）。嵌入样式的用法如下：

```
<head>
<title>Embedded Styles example</title>
<meta http-equiv="Content-type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Language" content="en-us" />
```

```
<style type="text/css">

h1 {font-size: 16px;}
p {color:blue;}

</style>
</head>
```

其中，style标签告诉浏览器它所遇到的代码不是XHTML，而标签的type属性声明了其中的代码是CSS。

以下是有关嵌入（或页面）样式的一些基本常识。

- 嵌入样式的作用范围局限于包含这些样式的页面。
- 如果只需要发布带有这些特定样式的一个页面，就可以把这些样式嵌入到文档的头部。不过，这样并没有真正做到样式与内容分离，因为样式仍然包含在同一个文档中。在实践本章介绍的单页例子的过程中，你会逐渐熟悉嵌入样式。
- 如果是为表单等复杂布局编写多个样式，有时候以嵌入方式在文档头部应用这些样式会更方便，因为这样无需频繁地在标记和样式表之间进行切换。然后，当效果达到时，可以把相应的样式转移到主样式表中，并用一行链接到主样式表的代码取代这些样式。
- 页面样式会覆盖样式表中的样式，但它们会被在内联样式中定义的属性所覆盖。
- 如果想把一个XHTML页面发送给某人征求意见，在页面中嵌入样式是比较合适的。这样，审阅者只需打开这个页面就可以看到最终效果。但是，对于任何规模的网站来说，事实上只有一种管理CSS的方式，那就是把样式放到样式表中，然后在网站的所有页面中链接该样式表。

2.1.3 链接样式

理想情况下，应该把样式放到一个单独的文档（即样式表）中，然后将它链接到多个页面以便相应的样式具有全局作用范围（整个网站）。在这个样式表中定义的样式，能够影响到网站

中的每一个页面，而不只是一个页面或一个标签。链接样式是以上3种方式中唯一一种能够真正将表现样式与结构化标记分离的方式。如果以这种方式集中管理CSS样式，那么网站的设计和修改都会变得更简单。

举例来说，如果你想修改整个网站的效果（“客户希望所有段落的文本是蓝色，而不是黑色”），那么只需轻松地修改一个CSS样式即可。这肯定要比使用CSS之前，逐个修改网站的每个页面中每个段落的FONT属性来得更容易。

在XHTML页面的头部，只需如下一行代码，就能够把样式表链接到任意多个页面中：

```
<link href="my_style_sheet.css" media="screen"
rel="stylesheet" type="text/css" />
```

当页面加载后，相应的样式就可以应用到页面中的每个标记上了。

注意，在上面的link标签中，media属性定义为"screen"，其含义是这个样式表是为屏幕显示（当前，也就是Web浏览器）而设计的。（某些用户代理会查找最适合它们显示能力的媒体属性。media属性可能的值包括：all、projection、handheld、print和aural。W3 Schools网站（www.w3schools.com/CSS/CSS_mediatypes.asp）上包含一个完整的媒体值列表。

浏览器能够理解link标签的media属性值为all和screen的样式表。不过，通过添加第2个link标签，并将其media属性设置为"print"，则可以为浏览器提供打印页面时使用的第2个样式表。用于打印的样式表可能会隐藏导航区及其他没有必要出现在纸面上的元素。

在创建了用于打印的第2个样式表后，相应的link标签大致如下所示：

```
<link href="my_style_sheet_print.css" media="print"
rel="stylesheet" type="text/css" />
```

现在，我们已经介绍了什么是样式表。下面，我们就来看一看如何编写样式规则，以及继承、针对性和层叠等机制如何影响样式规则应用到标记上。

不要在文件名中使用空格。由于空格最终会被转换成%20字符串，所以这会导致用户迷惑不解。我一般使用下划线代替空格，这样整个文件名对浏览器来说就是一个长长的词，而且也便于通过双击来选择。

什么是层叠样式表

我们把这个问题一分为二：什么是样式表和它们怎样层叠？现在，我们只回答第一个问题。虽然在上面的讨论中也包含了一些提示，但有关层叠的问题我们会放到本章后面再介绍。

所谓样式表，就是一个扩展名为 .css 的文本文件。

样式表的内容就是一组CSS规则的列表。每条规则定义了一个应用到XHTML标记的特定样式，一条规则可以定义段落文本的字体大小、图像周围边框的粗细、标题的位置、背景的颜色等。目前，通过Adobe InDesign等印刷设计程序实现的高级排印和布局功能，都可以在网页中通过CSS来模仿。Web设计者终于拥有了对页面布局全面控制的手段，从而得以将表格及空白GIF图等变通方案抛诸脑后。

2.2 CSS规则剖析

下面，我们通过分析一条简单的CSS规则来介绍如何编写CSS。为了让文档中所有段落的文本都变成红色，可以使用下面这条规则：

```
p {color:red;}
```

尽管XHTML标记中的p标签包含在一对尖括号中，但在CSS样式里，只需使用标签名，无需使用尖括号。

这样，如果文档中有下面的XHTML标记：

```
<p>This text is very important</p>
```

那么，这个段落中的文本就会是红色的。

一条CSS规则由两部分组成：选择符——用于声明规则选择哪个（些）标签（或者，用我常说的话说，就是选择符对准的标签），在这里，这个标签是一个段落p；还有声明——用于声明在应用规则后会发生什么效果，在这里，是文本显示为红色。声明本身也由两部分组成：一个是属性——声明要影响的目标，这里是文本的颜色；另一个是值——声明将属性设置成什么（或怎么样），这里是红色。下面，有必要仔细地看一看图2-2，以便绝对清楚地理解上面出现的4个术语，因为随着学习的深入，我们会频繁地提到它们。

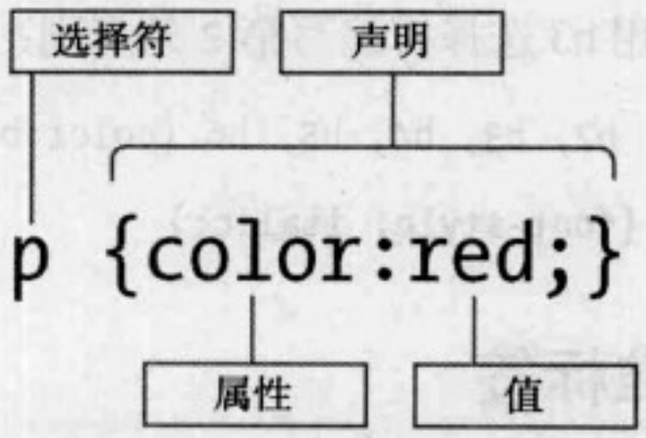


图 2-2 一条CSS规则包括两个部分——选择符和声明，声明又包括两个部分——属性和值

2.3 编写CSS规则

CSS要求绝对精确，省略一个分号会导致浏览器忽略整条规则。

你大概想知道对于字体大小和颜色属性，是否可以使用其他值。比如，是否可以使用RGB（红、绿、蓝）而不是颜色名称（答案是可以）。不过，请在学习选择符时先放下这个问题，在本章后面，我们会详细介绍规则中的声明部分。

选择符和声明的基本结构可以通过以下3种方式加以扩展。

(1) 一条规则中可以包含多个声明。

```
p {color:red; font-size:12px; line-height:15px;}
```

这样，段落文本会变成红色、12像素高，行间距[①]是15像素（像素，就是构成屏幕上显示内容的小点）。

而且，每个声明都以一个分号结尾，以便与下一个声明分开。位于结束花括号之前的最后一个分号是可选的，但是，为了便于今后再添加新声明，我建议不要省略它。

(2) 多个选择符可以组合起来。假设你想让h1到h5标签的文本变成蓝色和粗体。如果不怕麻烦，你可以输入下列规则：

```
h1 {color:blue; font-weight:bold;}
h2 {color:blue; font-weight:bold;}
h3 {color:blue; font-weight:bold;}
```

但是，通过把选择符组合到一条规则中，就可以避免这一麻烦：

```
h1, h2, h3, h4, h5, h6 {color:blue; font-weight:bold;}
```

只需记住在每个选择符后面加一个逗号即可——除了最后一个选择符。选择符之间的空格是可选的，但适当的空格可以使代码更容易阅读。

(3) 一个选择符可以在多条规则中使用。如果在编写完上面的规则之后，你想让h3标签中的文本显示为斜体，那么还可以

① 注意，这里的行间距不是指上一行底部到下一行顶部之间的距离，而是指两行基线之间的距离。在这个例子中，真正的行与行之间的距离是15−12=3像素。——译者注

使用 h3 选择符编写第 2 条规则：

```
h1, h2, h3, h4, h5, h6 {color:blue; font-weight:bold;}
h3 {font-style: italic;}
```

2.4 在文档层次中对准标签

如果你对上一章最后介绍的文档层次的概念还不清楚，请重新看一看 1.3.5 节的内容，在此不再赘述。

2.4.1 使用上下文选择符

如果在规则中以标签名作为选择符，那么规则就会对准该类型的每一个标签。例如，下面的规则：

```
p {color:red;}
```

会使每个段落的文本都变成红色。

但是，如果只想让一个特殊段落中的文本是红色怎么办呢？要更有选择性地对准标签，可以使用上下文选择符。比如：

```
div p {color:red;}
```

只会使位于 div 标签中的段落的文本显示为红色。

如果你不太熟悉 XHTML，需要注意 span 和 div 一样，都是没有默认样式的中立容器。换句话说，如果不明确地为 span 添加样式，它对标记不会有任何影响。通常，将在 XHTML 中没有定义但对我们有意义的内容标记出来是很有用的。而使用 span 标签，可以保证在用户代理不识别样式表的情况下不影响文档的表现。与作为块级元素、会强制换行的 div 不同，span 是行内元素，因此它不会强制换行。在默认情况下，strong 标签会导致粗体文本，而 em（emphasis，强调）会导致斜体——当然，这些样式都可以使用 CSS 来修改。

我们注意到，在上面的例子中，上下文选择符中使用了不止一个标签名（这里使用了 div 和 p）。距离声明最近的标签（即 p）是我们要对准的标签。其他的标签用于声明包含要对准的、由规则影响的标签的祖先标签。下面我们来详细介绍这个概念。

为此，需要使用以下面的示例标记：

```
<h1>Contextual selectors are <em>very</em> selective.</h1>
<p>This example shows how to target a <em>specific</em> tag
using the document hierarchy.</p>
<p>Tags only need to be descendants <span>in the <em>order
stated</em> in the selector</span>; other tags can be in
between and the selector still works.</p>
```

注意，第一段中包含一个 em 元素，第二段中的 em 元素被

嵌套在 span 标签中。图 2-3 展示了在使用浏览器默认样式的情况下，这些代码的外观。

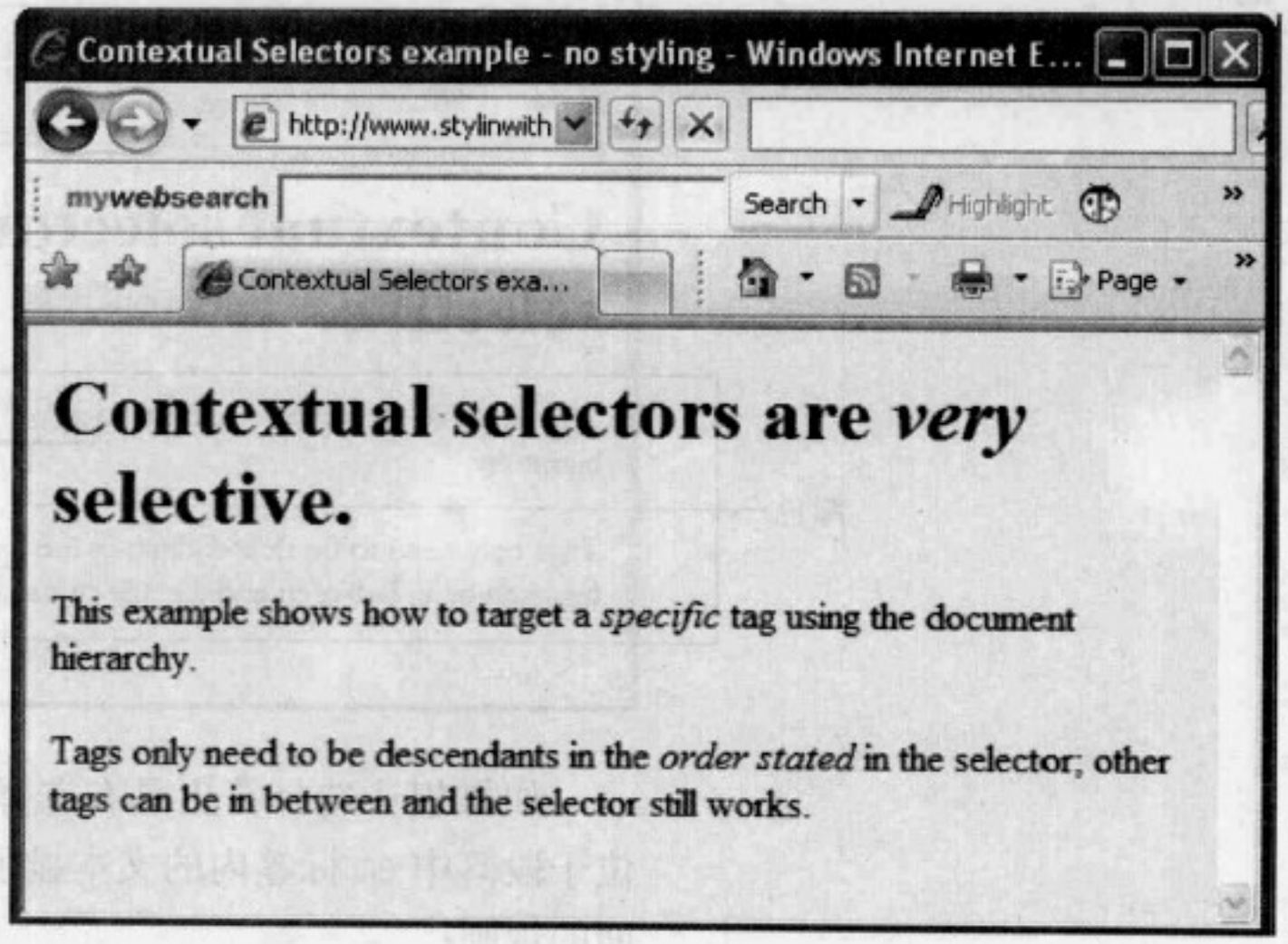


图 2-3 这里只应用了浏览器的默认样式

图 2-4 展示了标记中的层次。

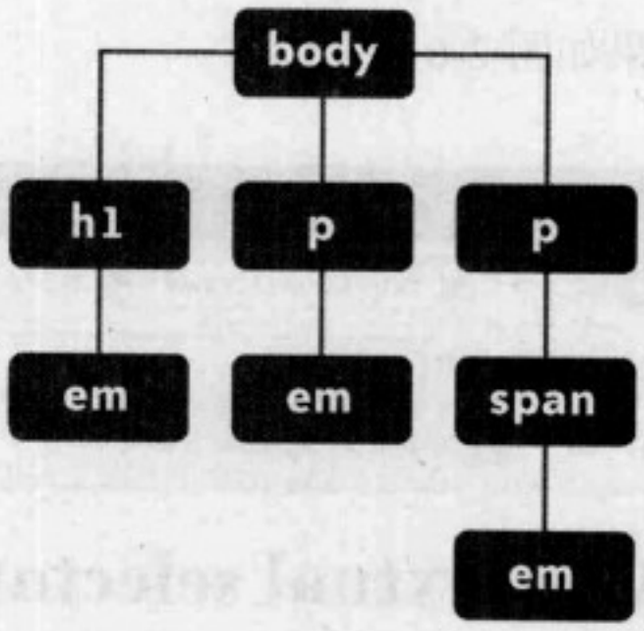


图 2-4 这是页面中代码的层次

这个层次图展示了标签之间的嵌套关系。如果把下面的规则：

```
em {color:green;}
```

放在上面那个页面文档头部的 style 标签中，所有位于 em 标签中的文本都会变成绿色（如图 2-5 所示）。

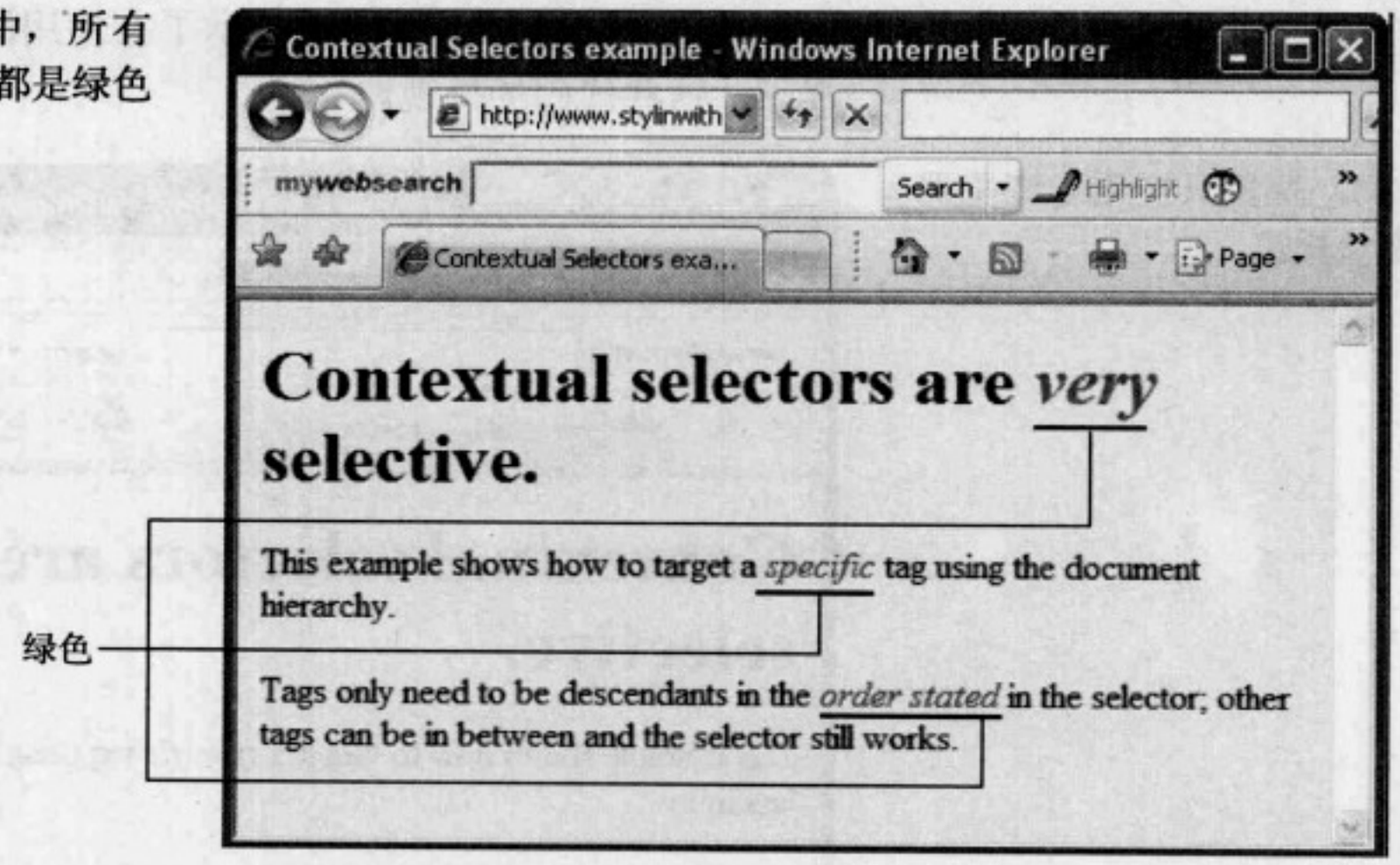


图 2-5 在这个例子中，所有位于 em 标签中的文本都是绿色的

如果想让选择符更具有选择性怎么办呢？我们假设只想让位于段落中 em 标签内的文本显示为绿色。那么，就应该编写下面的规则：

```
p em {color:green;}
```

结果如图 2-6 所示。

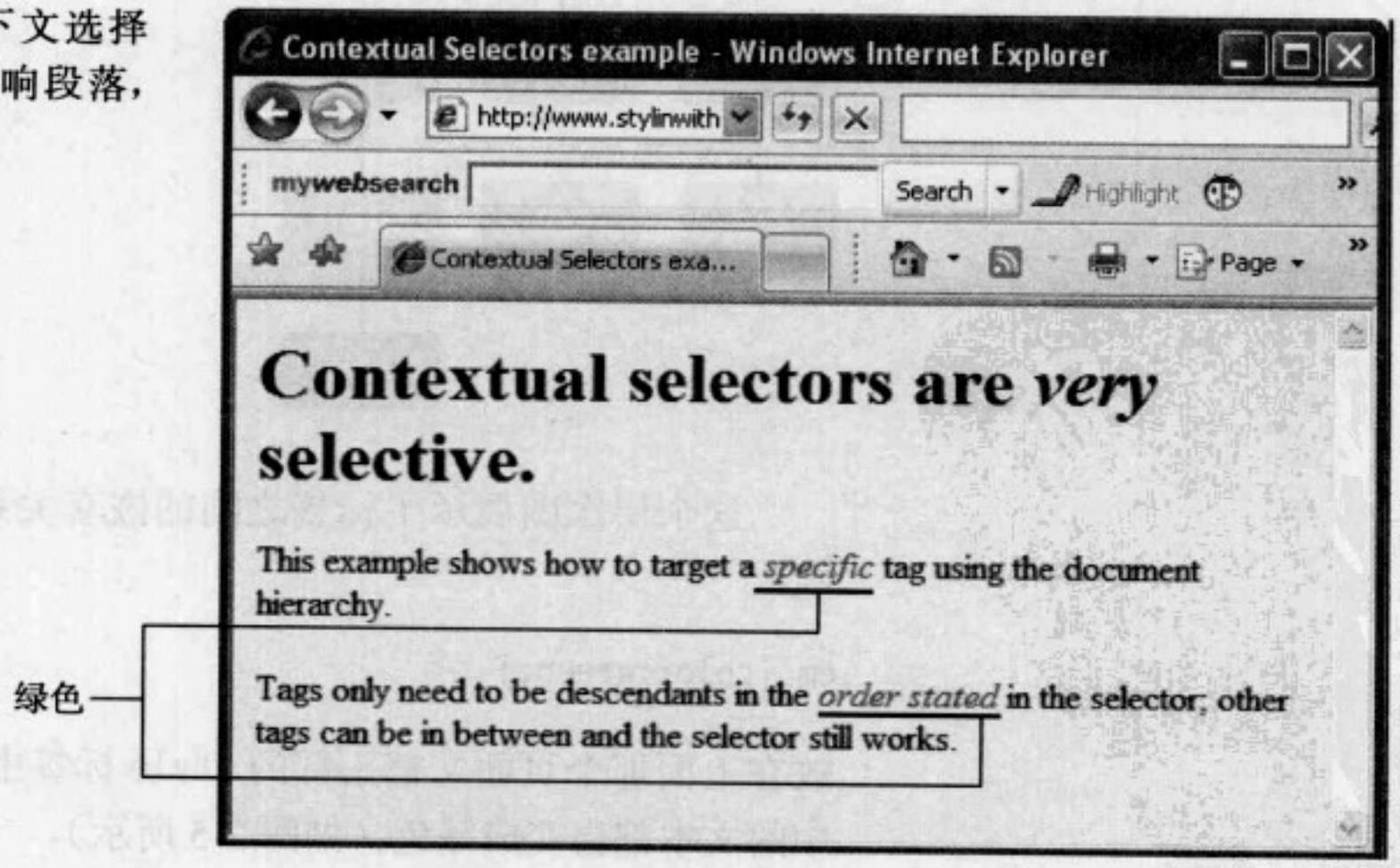


图 2-6 通过使用上下文选择符，可以使规则只影响段落，不影响标题

由于在选择符中的 em 前面添加了一个 p，因而规则只会对准位于 p 标签中的 em 标签；这次，位于 h2 标签中的 em 标签不

会受到影响。不过，同我们前面看到的分组选择符不同，上下文选择符中的每个选择符之间以空格分隔，而不是逗号。

记住，带上下文选择符的规则只会应用到最后一个标签，而且，只有当该标签前面的选择符在文档层次中位于它上方某处，并以相同顺序出现时才会有效。至于它们之间隔着多少标签都不是问题。

因此，位于 span 标签中的 em 标签也会受到这条规则的影响。即使该标签不是 p 标签的直接子元素，但规则仍然会应用，因为它是 p 标签的后代元素。下面的例子示范了通过在选择符中声明更多的标签，可以使选择更有针对性：

```
p span em {color:green;}
```

这条规则的结果如图 2-7 所示。

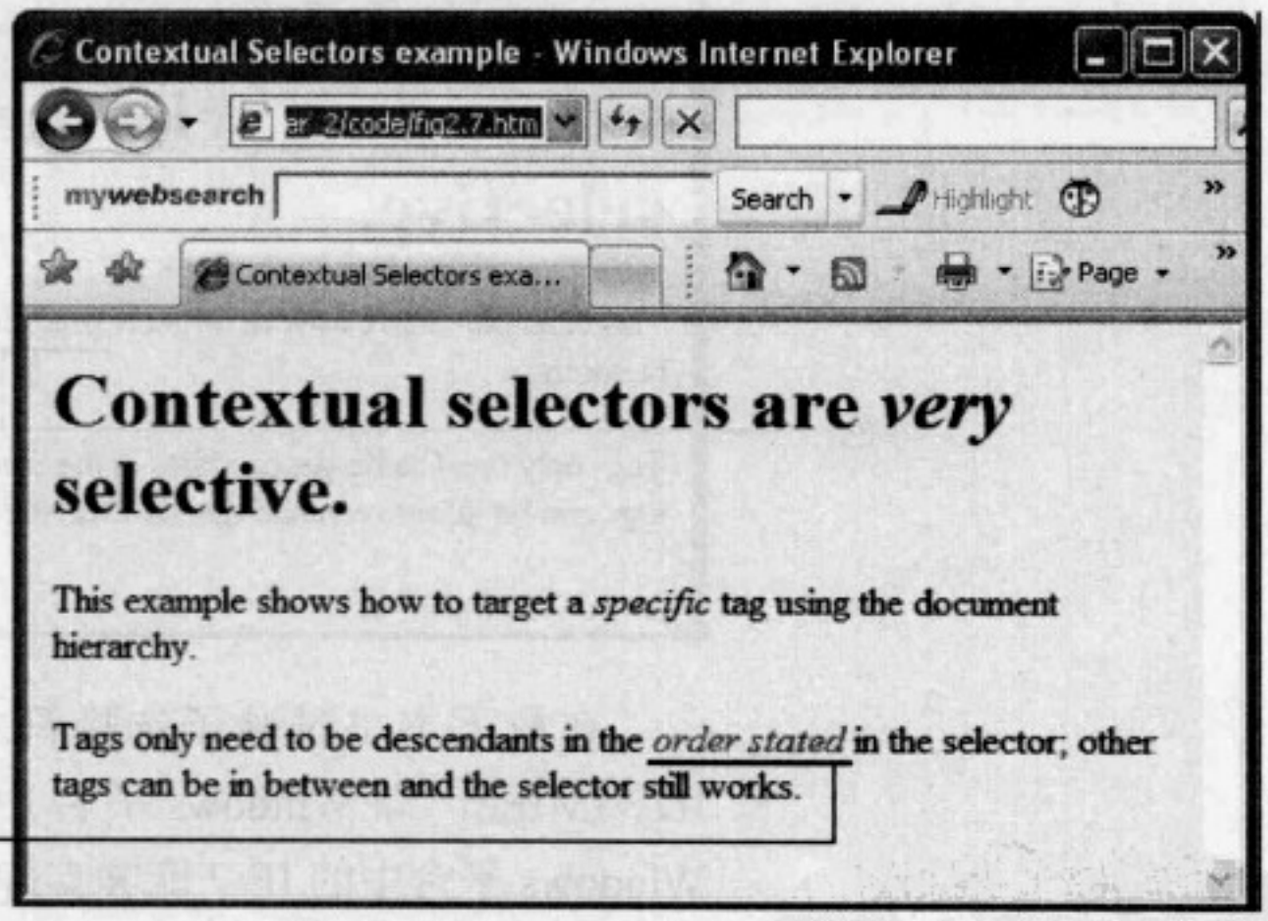


图 2-7 在选择符中使用 3 个元素的情况下，可以非常具体地指定哪些文本应该是绿色的

对于现在的规则声明，只有位于 p 标签中的 span 标签中的 em 才会被选中。由于这个选择符为规则生效设置了非常具体的上下文，所以只有一个标签符合该标准。在像这样的上下文选择符中，可以按照需要罗列任意多个选择符，以确保对准你想修改的那个标签。

然而，当想要对准单词“speific”时，问题会更加复杂。我们在图 2-5 中看到，规则 p em {color: green;} 选择了位于两个段落中的 em 标签。可见，通过标准的上下文选择符无法对准这个特定的标签。为此，我们需要使用一个子选择符。

2.4.2 使用子选择符

在第 1 章中，我们提到过子标签是一个包含标签的直接后代标签。如果想让规则必须对准一个特定标签的子标签，那么通过使用 > 符号也可以做到，比如：

用在两个选择符之间的 > 符号的意思是“的子元素”。

```
p > em {color:green;}
```

这样，就可以成功地对准单词“specific”而不会影响其他 em 文本。因为“specific”被包含在一个作为 p 标签子元素的 em 标签中，而词组“order stated”则不是（图 2-8）。

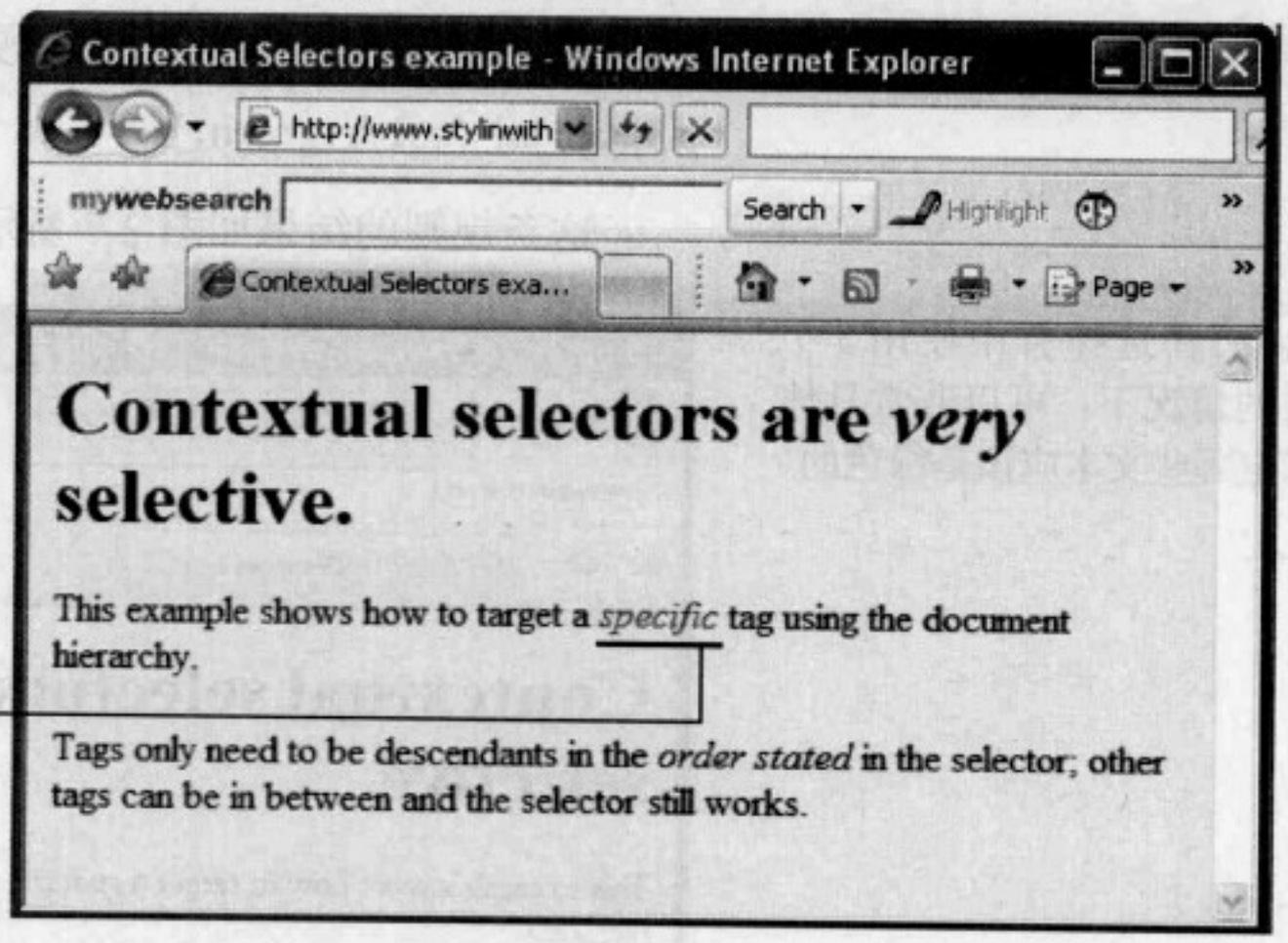


图 2-8 子选择符为选择标记中的单词“specific”提供了必须的上下文

在放下本书试验子选择符之前，请记住，在本书写作时 IDWIMIE6，即 Windows 平台中的 IE 6 不支持子选择符（但 Windows 平台中的 IE 7 能够支持）。不过，如果你遇到了非使用子选择符不可的情况，那么我们也有解决方案。很快你就会看到，通过类和 ID 可以使规则对准任何单个标签，但为了使用它们，也需要更多的标记。

因此，在 IE 6 的使用率降低到可以忽略不计之前，我们应该主要使用子选择符在样式表中创建差异效果，以便绕过 IE 的各种不符合标准的行为。或者，利用 IE 6 的这一缺陷来显示一种不同的但却能够接受的结果。在本书后面的章节中，我们就以这种方式来使用子选择符。

2.4.3 添加类和 ID

到目前为止，我们注意到当在选择符中简单地声明一个标签名（如 p 或 h1）时，相应的规则会应用到该标签的每一个实例。我们也知道了，要增强选择过程中的针对性，可以使用上下文选择符来指定包含要对准标签的标签。

事实上，我们还可以通过向 XHTML 标记中添加 ID 和类属性，来对准文档中的特定的区域。ID 和类为我们提供了将样式应用到文档中的另一种途径——一种无需考虑文档层次的方式。

1. 类的简单用法

下面的标记用于示范类的用法：

```
<h1 class="specialtext">This is a heading with the <span>same class</span> as the second paragraph</h1>

<p>This tag has no class.</p>

<p class="specialtext"> When a tag has a class attribute, we can target it <span>regardless</span> of its position in the hierarchy.</p>
```

在编写类选择符时，需要以 .（英文句点）开头。注意不要在句点和选择符之间加空格。

注意，有两个标签都添加了值为 specialtext 的类属性。接下来，我们就给类为 specialtext 的标签应用粗体样式（图 2-9）。

```
p {font-family: Helvetica, sans-serif;}

.specialtext {font-weight:bold;}
```

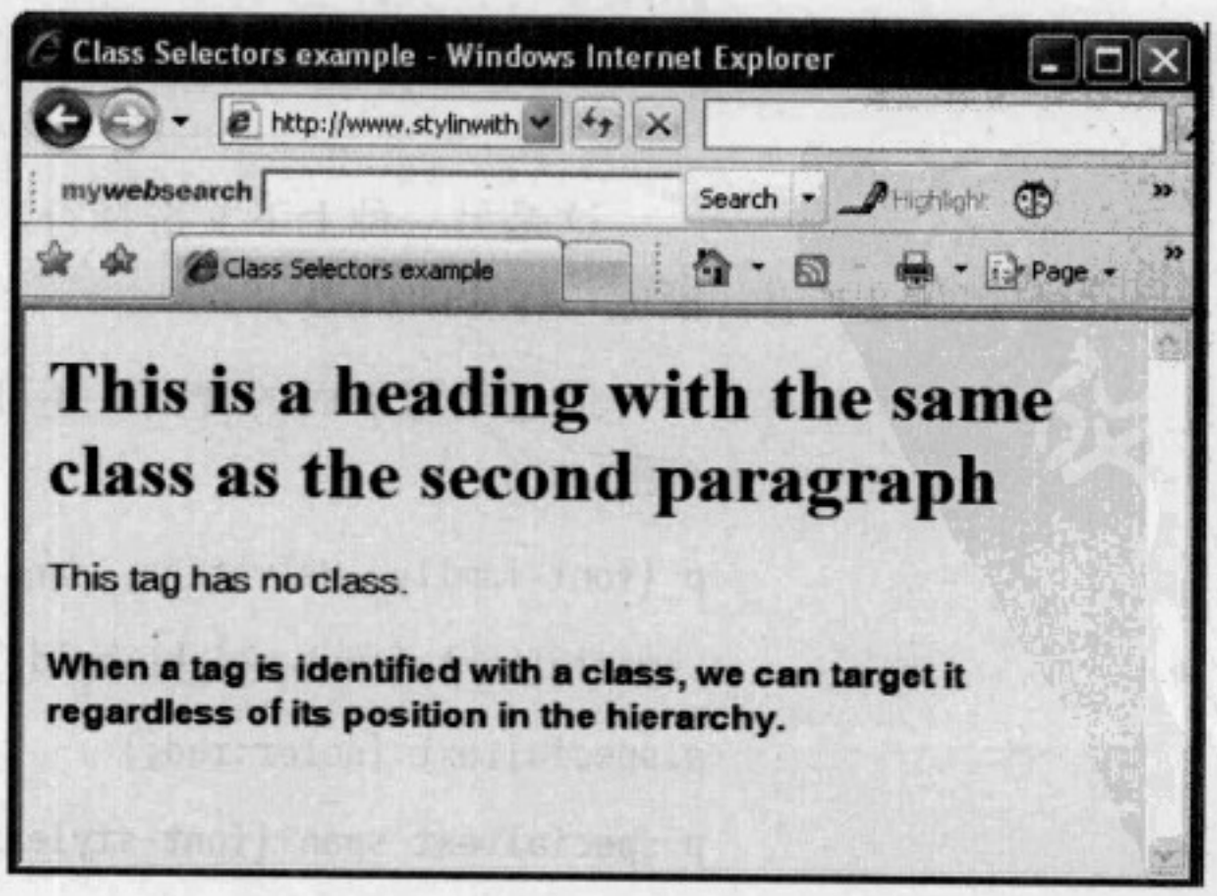


图 2-9 这里我们使用一个类选择符来加粗了两个不同的标签

以上规则会使两个段落的字体都变成 Helvetica（或者在 Helvetica 字体不可用时，浏览器中的 sans-serif 字体），而带有 specialtext 类的段落则以粗 Helvetica 字体显示。h1 标签中的文本保持为浏览器的默认字体（通常是 Times），但由于它也带有 specialtext 类，因而也显示为粗体。注意 span 这个没有默认样式的标签，因为我们没有明确为它添加样式，所以它对显示结果没有任何影响。

2. 上下文中的类选择符

如果只想对准带有这个类的段落，我们可以创建一个组合了标签名和类的选择符，即（图 2-10）：

```
p {font-family: Helvetica, sans-serif;}
.specialtext {font-weight:bold;}
p.specialtext {color:red;}
```

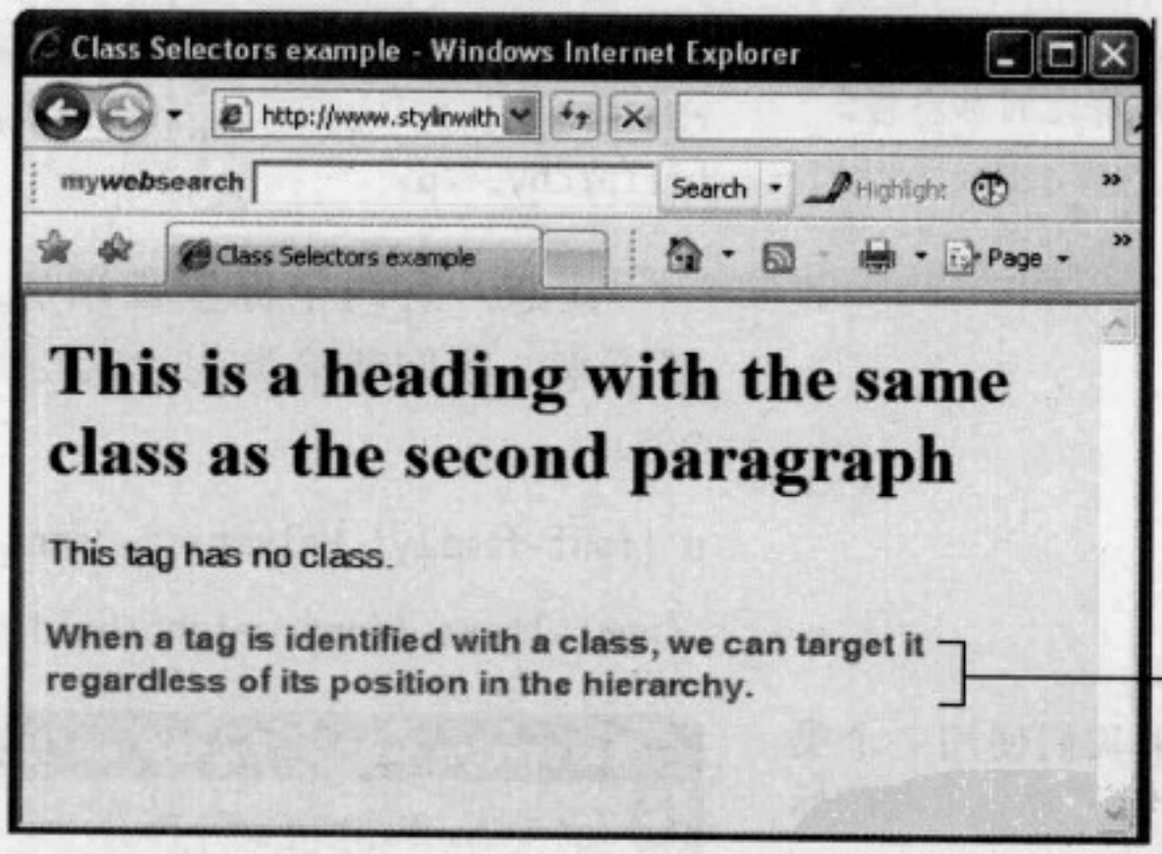


图 2-10　通过组合标签名和类名，可以使选择符更有针对性

这是另一种上下文选择符，因为类必须位于一个段落中才能应用相应的规则。

通过编写下面的规则，还可以进一步增强选择符的针对性（图 2-11）：

```
p {font-family: Helvetica, sans-serif;}
.specialtext {font-weight:bold;}
p.specialtext {color:red;}
p.specialtext span {font-style:italic;}
```

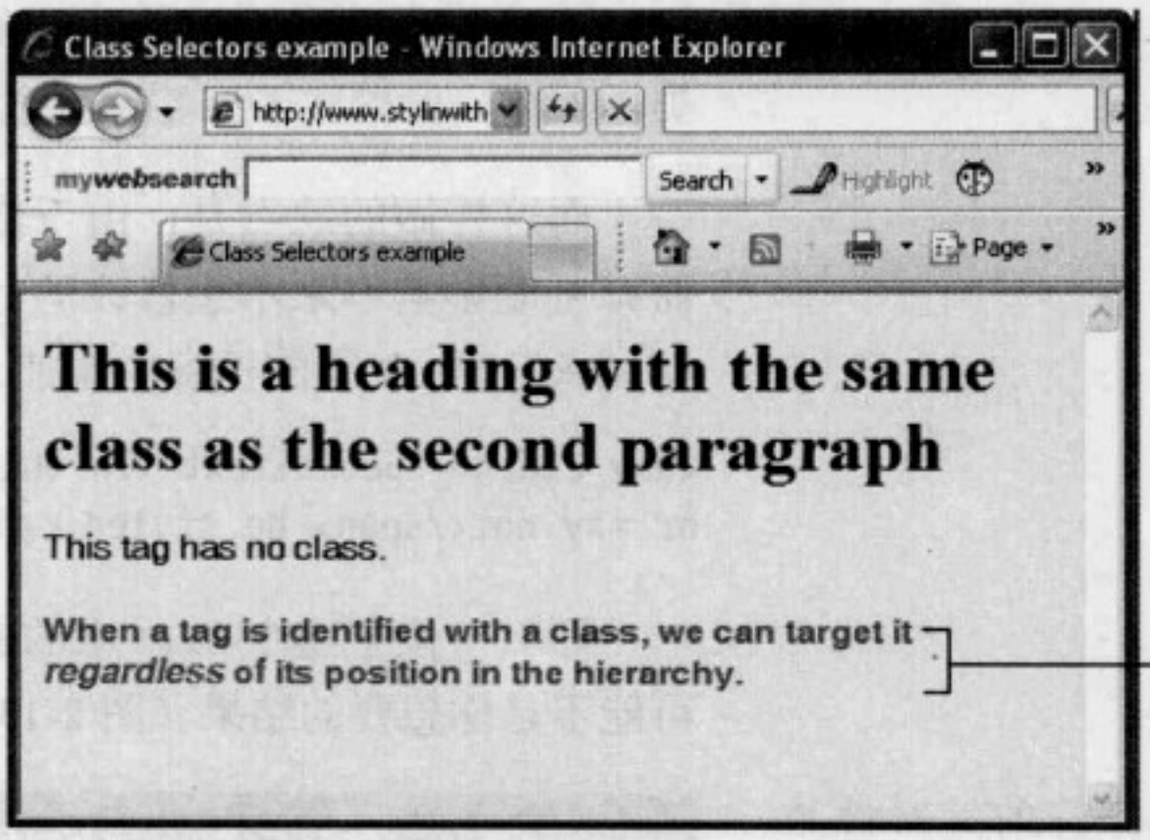


图 2-11 通过添加第 2 个选择符，可以更具体地指定为哪个标签添加样式

现在，单词“regardless”既是粗体也是斜体，因为它位于带有 specialtext 类的段落中的 span 标签内（正如选择符所指定的）。如果你想让这条规则也对准 h1 标签中的 span，那么可以通过两种方式来修改它。最简单的方式就是去掉类与特定的标签关联（图 2-12）。

```
.specialtext span {font-style:italic;}
```

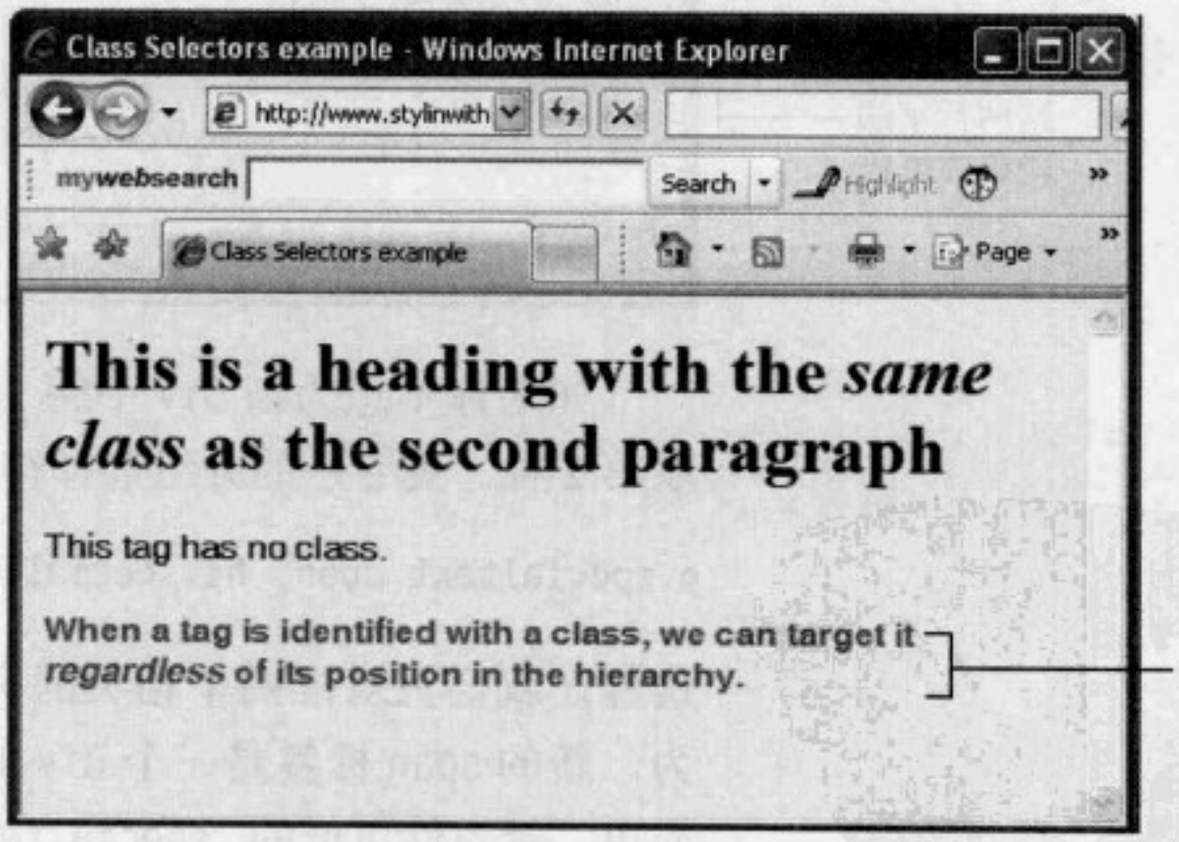


图 2-12 通过降低选择符的针对性，会选择标题中的 span 文本

位于标题中的词组“same class”变成了斜体。通过删除选择符开头的 p，实际上是去掉了类必须附属于某个特定标签的限制，因此两个 span 标签都被选中了。由于没有了标签限制，现在的规则声明了要对准的 span 标签，可以是带有 specialtext 类的任何标签的后代标签。

这种方法的好处是，可以在不考虑标签层次的情况下使用类。因此，就摆脱了标签层次之间固有的制约。

而这样做的缺点是，由于修改后的规则降低了针对性，因而有可能影响本来不想修改的标签。假设你后来又在另一个带有 specialtext 类的标签中添加了一个 span，如下所示：

```
<div class="specialtext">In this div, the span tag <span>may
or may not</span> be styled.</div>
```

那么，位于这个 span 中的文本也会变成斜体。这可能是也可能不是你想要的结果（图 2-13）。

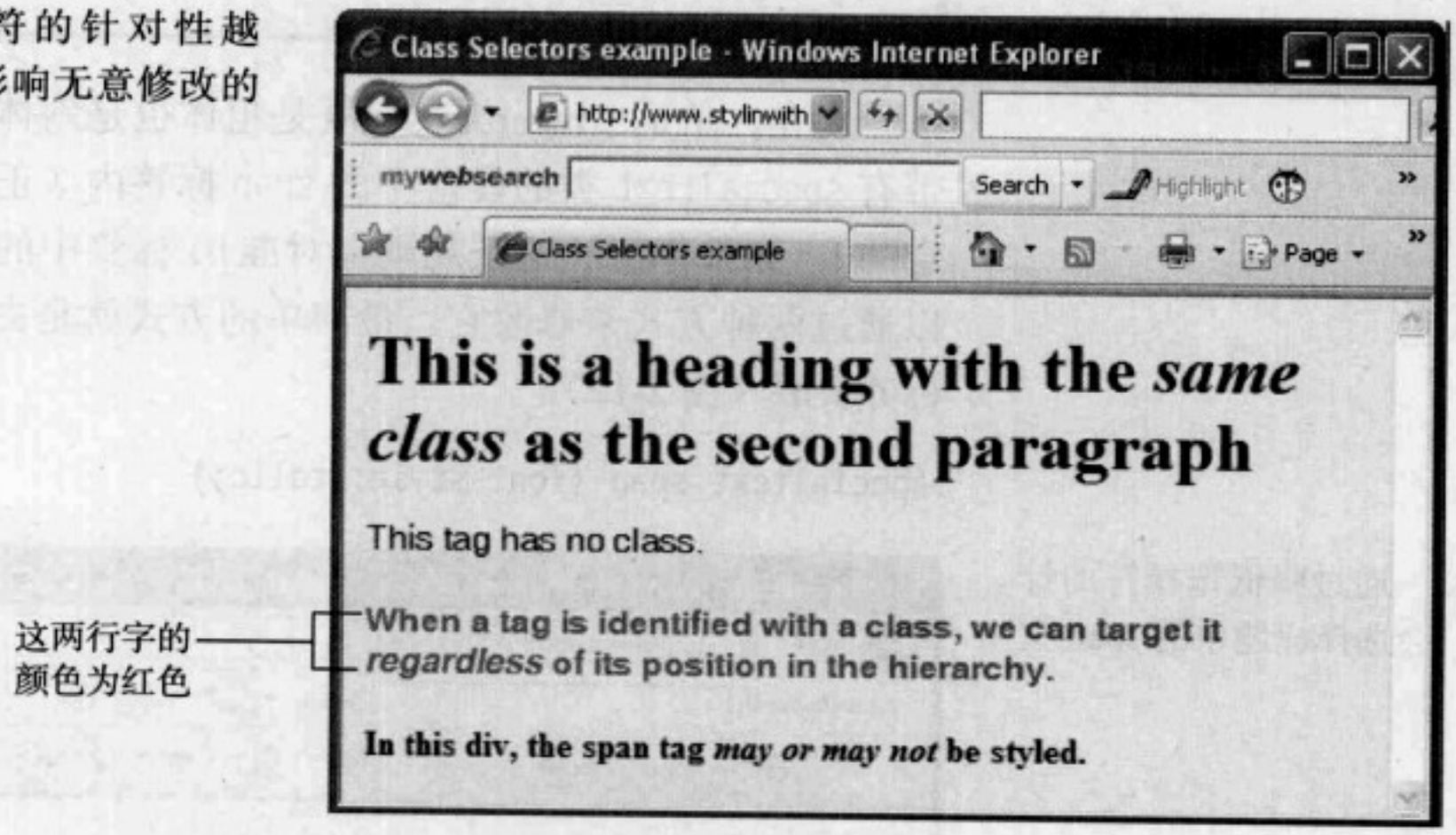


图 2-13 选择符的针对性越低，越有可能影响无意修改的标签

如果你不想为新 div 中的 span 标签添加样式，那么可以采取第 2 种、更专一的分组选择符的方式，如下所示（图 2-14）：

```
p.specialtext span, h1.specialtext span {font-style:italic;}
```

这样，就会只对准两个相关的标签，而不影响新标签。这是因为，新的 span 标签是一个 div 的后代。但是，如果我们使用更简单、针对性更低的 .specialtext span，就会对准新的 span 标签。

即使在为这个只有 4 行代码的例子添加样式的过程中，你都会觉得很费心思，但在编写一个包含数十甚至上百行的样式表时（如本书后面章节中所做的），仍然也需要把这些问题考虑全面。

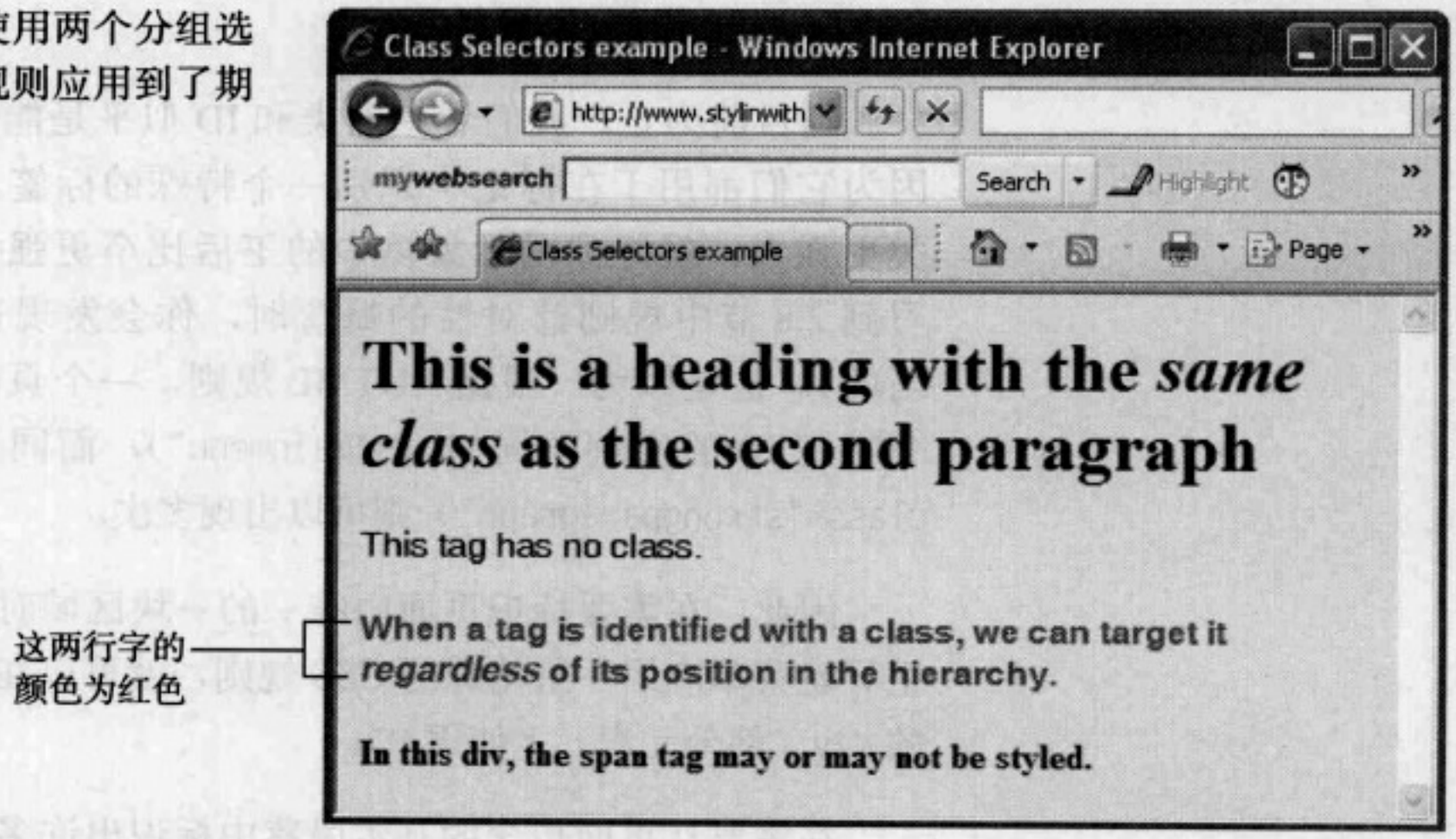


图 2-14 通过使用两个分组选择符，我们把规则应用到了期望的标签上

3. 多个类

对于类需要注意的最后一个问题是，可以为一个标签应用多个类，比如：

```
<div class="specialtext featured">In this div, the span tag
<span>may or may not</span> be styled.</div>
```

这里，我们在引号中放入了以空格分隔的 specialtext 和 featured 类。虽然乍一看这有点奇怪，但 W3C 的确就是这样规定的。在本书后面章节的例子中，我们会介绍多个类的用途。

2.4.4 ID 简介

ID 与类的写法相似，只不过在 CSS 中要使用 #（散列符号）来表示 ID，而不是类的 .（句点）。

如果像下面这样为段落添加了一个 ID：

```
<p id="specialtext">This is the special text</p>
```

那么，相应的上下文选择符应该是：

```
p#specialtext {CSS声明}
```

除些之外，ID 与类的工作方式相同。而且，我们在前面讨论的一切与类有关的内容，也都适用于 ID。那么，到底 ID 与类有什么区别呢？

2.4.5 ID和类之间的区别

到目前为止，我们看到的类和ID似乎是能够互换的——因为它们都用于在标记中识别一个特殊的标签。但是，ID比类更强大，就好像国际象棋中的王后比卒更强大一样（当学习到2.8节中规则针对性的概念时，你会发现这个比喻是恰当的）。这是因为，根据XHTML规则，一个页面中只能有一个特定ID的实例（例如id="mainmenu"），而同一个类（例如class="strongparagraph"）则可以出现多次。

因此，在需要标识页面中唯一的一块区域时——例如要为主导航菜单应用一组特殊的CSS规则，就可以在包含菜单元素的div（部分元素）上使用ID。

在需要从页面包含的基本段落中标识出许多应该具有相同样式的特殊段落时，应该使用类。

另外，使用ID也可以让JavaScript对准一个标签（例如，当用户鼠标悬停在某个链接上时激活DHTML动画）。为此，JavaScript代码编程人员更希望用id属性代替已经不推荐的name属性（XHTML验证程序会将它标识为无效）。因为这样可以确保与JavaScript相关的ID在一个页面中只会出现一次（这一点尤其重要），否则JavaScript代码可能会变得不可预知。

不要疯狂地使用类

一般来说，应该有节制地使用ID和类，正确的用法是把它们添加到包含标记主要部分的div中，然后再使用以相应的ID和类名开头的上下文选择符来访问位于该div中的标签。

这里需要引以为戒的是Jeffrey Zeldman所说的“classitis——标签中的麻疹”的问题。这个问题是指为标记中的几乎每个标签都添加一个唯一的类或者ID，然后再为每个标签编写一条规则的做法。这种做法同在标记中使用FONT及其他无关紧要的标记并没有太大的分别。好医生Zeldman已经帮助我和其他很多患此“病症”的人恢复了健康。如果你也有为每个标签都添加类的习惯（就像狂热地投入CSS怀抱的多数人一样），那么请按照本章讲述的内容再仔细研究一下第1章中的标记示例。你会发现不必添加任何多余的ID或类，同样能够便捷地为每个标签应用样式。

如果你只使用ID标识页面中的主要部分，而使用类来标识那些偶尔不能通过基于标签的上下文选择符对准的标签，那么就不会造成太大问题。而且，这样也会使样式表更加简洁。

总而言之，我们可以在一个页面中使用多个id，但每个id必须有唯一的值（名称）来标识它。而同一个类名则可以应用

到任意多个标签中。

2.4.6 特殊的选择符

尽管不是CSS的正式类别，但这些“特殊的”选择符能够以不同于前面介绍的选择符的方式来对准标签。除了*（星号）选择符之外，下面介绍的选择符能够准确地分析标记，并在某个条件匹配时（例如一种标签类型紧跟在另一种标签类型的后面）对准相应的标签。虽然这些选择符提供了某种强大的能力，但在比较旧的浏览器（特别是IE，包括IE 6）中没有得到很好的支持。不过，IE 7在这方面有了较大的改进。如果要测试浏览器是否支持伪类选择符，可以访问本书网站上提供的测试页面，地址为www.stylinwithcss.com/chapter_2/code/pseudo_tests.htm。通过这个页面你可以对自己使用的浏览器是否支持下面这些“特殊的”选择符一目了然。

1. 通配选择符

通配选择符*（也常常叫做星号选择符）的含义是“任何标签”，因此在样式表中包含如下规则：

```
* {color:green;}
```

那么，所有标签中的文本都会变成绿色——除非在其他规则中为标签指定了不同的颜色。这个选择符的另一种有意思的用法，是利用它进行子选择符的非运算，即非子选择符（如果你愿意）。

```
p * em {font-weight:bold;}
```

这里，选择符对准的至少应该是p标签的孙子标签，而不是子标签，至于em的父标签是什么则无关紧要。无论新旧，所有浏览器都能够支持星号选择符。

2. 相邻同辈选择符

这种选择符会对准紧跟在一个特殊的同辈标签后面的标签（同辈标签是指在标记层次中位于同一层次的标签——换句话说，它们都有同一个父标签）。如果将下面的规则：

```
h1 + p {font-variant:small-caps}
```

应用到下列标记：

```
<div>
 <h1>All about siblings selectors</h1>
 <p>There must be at least two selectors, with a + sign
 before the last one.</p>
 <p>The targeted selector will only be affected if it is a
 sibling to, and preceded by, the one before the + sign.</p>
</div>
```

就会导致如图 2-15 所示的结果，因为只有第一个段落存在一个同辈的 h1 标签。

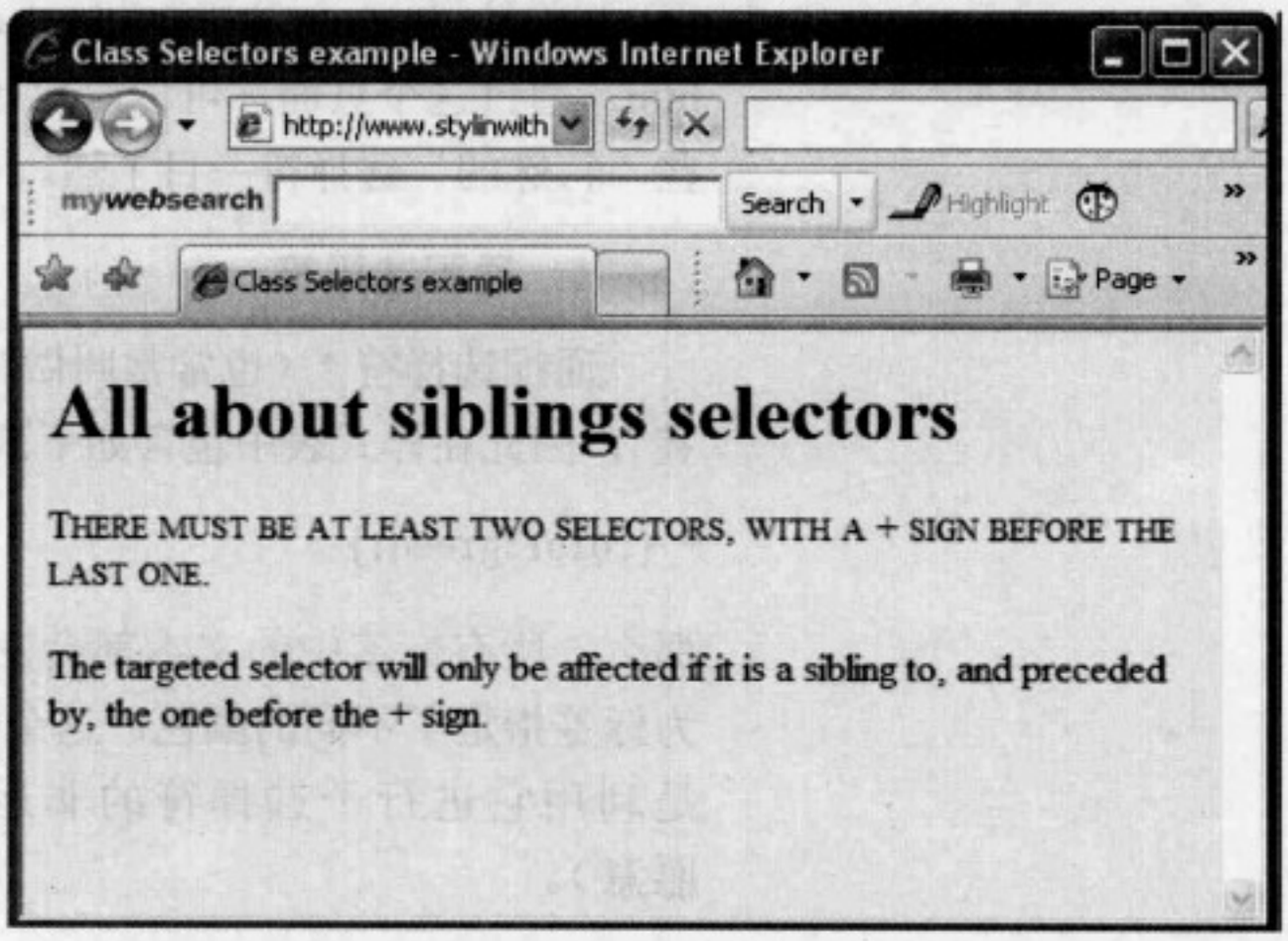


图 2-15 同辈选择符基于标记中前面的标签起作用，而且两个标签必须被嵌套在同一层次中。这是一个理解起来需要一些技巧的选择符

如图 2-15 所示，紧跟在 h1 后面的 p 标签符合了规则的条件，因此其中的文本变成了小型大写字母形式。但是，因为第 2 个 p 标签与 h1 不相邻，所以没有受到影响。这种选择符很适合为第一个列表项添加粗体样式（相邻同辈选择符在 SCB[①]和 IE 7 中有效，在 IE 6 中无效）。

3. 属性选择符

属性选择符使用的是标签的属性。这是将不同的 CSS 对准类似元素的另一种方式；即只要标签中的属性存在差别，就可以为这些标签应用不同的规则。但是，由于 IE 6 及新改进后的

① SCB，即 Standards Compliant Browser（符合标准的浏览器）。——译者注

IE 7 都不支持它，因而限制了这种选择符的用途。为此，目前我们只能使用属性选择符来增强使用 SCB 的用户体验。

下面的规则：

```
img[title] {border: 2px solid blue;}
```

会为像下面这样带有 title 属性的 img 标签

```
<img src="../images/Windsor-castle_walls.jpg" title="Windsor-
castle walls" alt="Windsor-castle walls" />
```

为了简明起见，我们故意简化了图像的 alt 属性。但是，出于可访问性的考虑，总是为看不到这幅图像的用户提供有意义的 alt 文本。

添加蓝色的、两像素宽的边框；至于 title 中包含什么值都无所谓，只要存在该属性即可。利用这个选择符可以提醒用户当他们的鼠标指向这幅图像时会显示提示条（根据 title 属性生成的弹出文本）。在实践中，一种常见的做法是复制 alt 和 title 属性的值——alt 属性中的文本会在图像未加载时显示，也可以由屏幕阅读器读取，title 属性会导致当用户将鼠标指向这幅图像时出现一个提示条。

也可以更具体地指定属性的值应该是什么。例如，下面的规则：

```
img[alt="Dartmoor-view of countryside"] {border:3px green
solid;}
```

只会为 alt 属性的值为 "Dartmoor-view of countryside" 的图像添加边框；换句话说，图像的标记必须如下所示：

```
<img src="../images/dartmoor-view.jpg" title="Dartmoor-view
of countryside" alt="Dartmoor-view of countryside" />
```

这个选择符还有一种更强大的用法，即可以只指定属性值中第一个字符。不过，属性值的“公共”部分必须同“差异”部分以连字符进行分隔。因此，如果你为此仔细地标记了 img 标签的属性，比如（注意连字符）：

```
<img src="../images/dartmoor-cottage.jpg" title="Dartmoor-
small cottage" alt="Dartmoor-small cottage" />
<img src="../images/dartmoor-view.jpg" title="Dartmoor-view
of countryside" alt="Dartmoor-view of countryside" />
```

那么，就可以通过在选择符中添加一条竖线（一般是通过 Shift + \ 输入）来选择这两幅图像，比如：

```
img[alt|="Dartmoor"] {border:3px blue solid;}
```

顺便说一下，这条规则也会选择下面这幅图像：

```
<img src="../images/dartmoor-view.jpg" title="Dartmoor "
alt="Dartmoor " />
```

即使这幅图像的 alt 属性不包含通过连字符扩展的值。

图 2-16 是在 Firefox 中查看这些代码示例的屏幕截图。

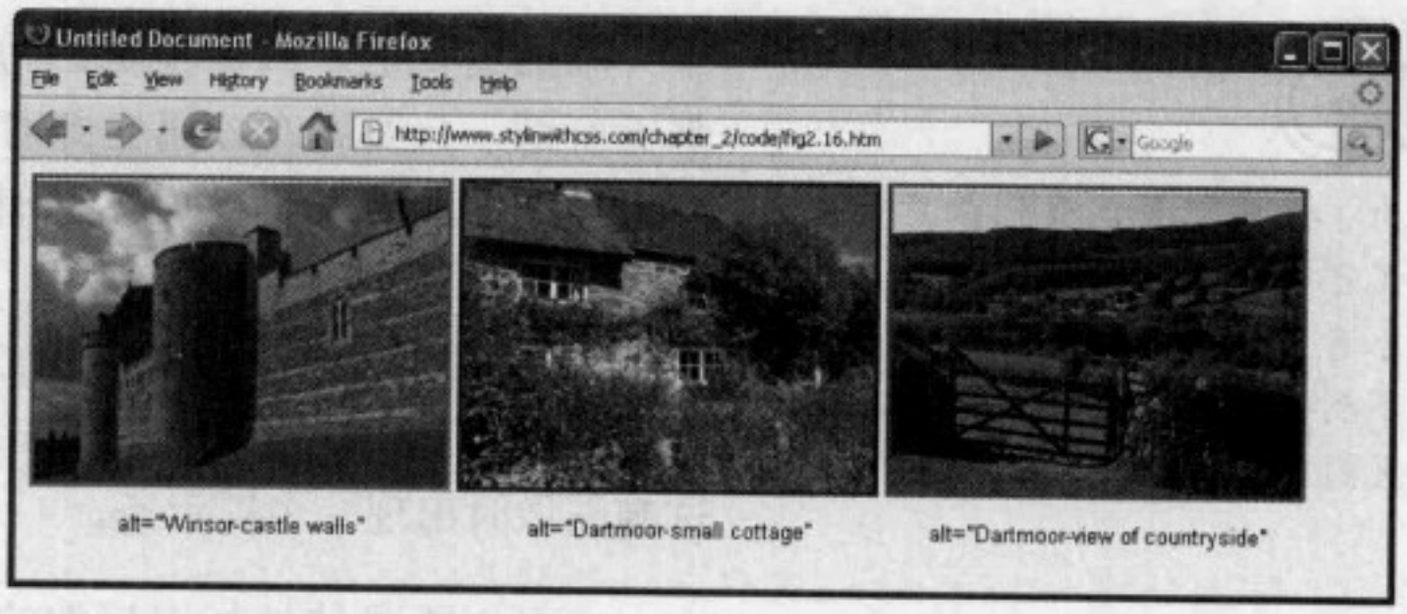


图 2-16 Firefox 正确地显示了属性选择符。图像的 title 属性与屏幕截图中显示的 alt 属性相同

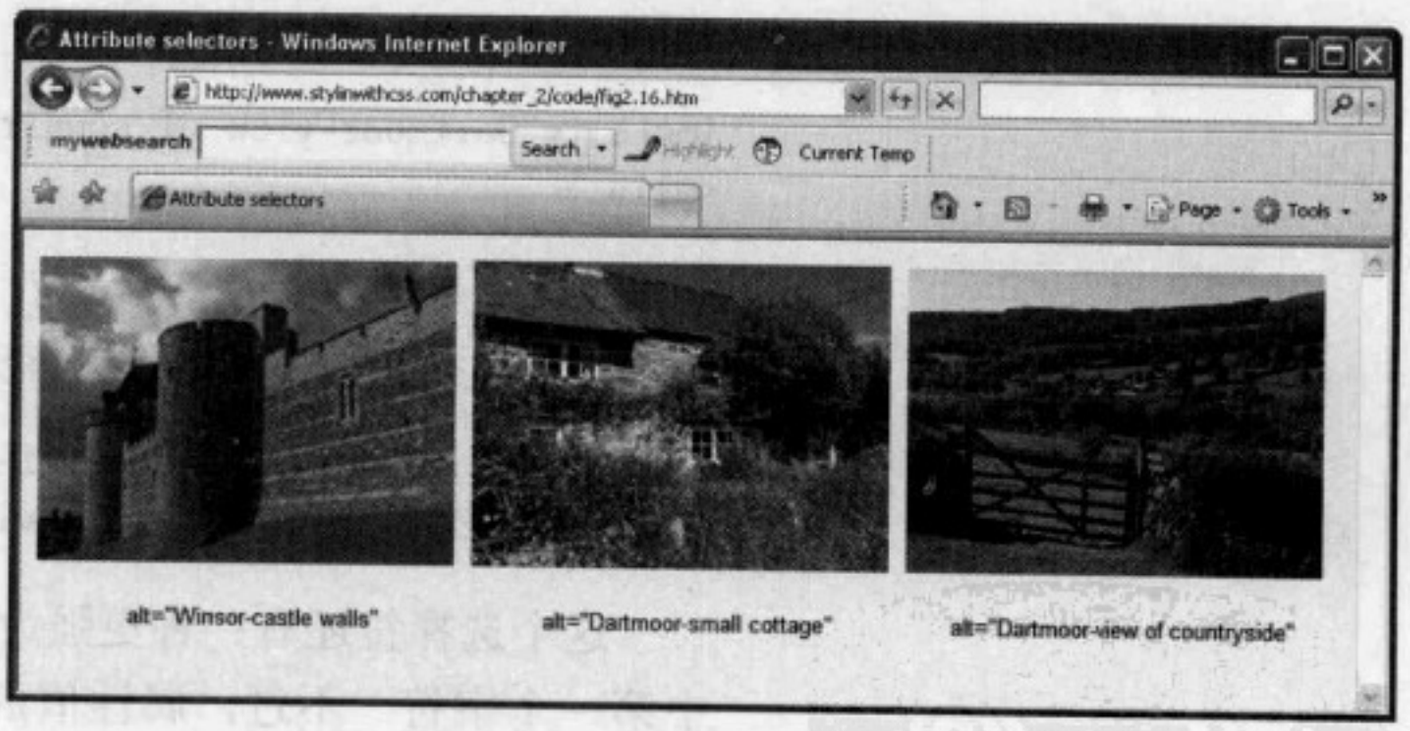


图 2-16A 不过，IE 7 不支持属性选择符

2.4.7 选择符小结

到现在为止，我们介绍的对准 CSS 规则的方式有以下几种：使用标签选择符、使用类和 ID 选择符、使用前面两种选择符的组合选择符以及使用基于标签属性的选择符。

这些选择符的共性是它们都对准标记中的某些目标——标签名、类或 ID、属性或者属性值等。但是，如果我们想在某种

事件发生时（例如用户鼠标指向链接）为标签添加样式该怎么办呢？也就是说，我们还需要一种基于事件来应用规则的方式。在学习完下面的内容之后，你就会明白有一种方式可以做到这一点。

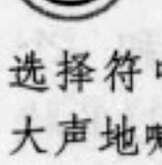
2.5 伪类

伪类指的是同样也是类但并没有实际添加到标签中的类。通过伪类可以在某种事件发生时，将规则应用到标签上。最常见的事件就是用户鼠标指向或单击某个页面元素。在较新的浏览器中（遗憾的是，不包括 IE 6 及更早的版，至少在未添加特殊的 JavaScript 功能 hover.htc[①]之前如此），很容易让屏幕上显示的页面元素响应翻转效果。所谓翻转，就是当鼠标指针悬停在某个元素上导致的变化，也称为悬停效果。例如，使用 :hover 伪类，可以在用户鼠标移动到一幅图像上面时，为图像周围添加边框。

可以通过 http://www.stylinwith-css.com/view_stylib.php[②] 下载 Stylib CSS 库，并在 js_tools[③] 文件夹中找到 csshover.htc[④]。

2.5.1 锚链接的伪类

伪类在使用超链接（a 标签）时是最常见的。当鼠标悬停在链接上时，可以通过伪类改变链接文本的颜色或者去掉文本底部的下划线。



选择符中与众不同的：（冒号）大声地喊（好吧，表明）“我是伪类！”

因为链接可能的状态有 4 种，所以针对锚链接的伪类也有 4 种。

- **链接（link）**。链接显示在页面上并等待用户单击的状态。
- **已访问（visited）**。链接已经在过去某个时刻被用户单击了的状态。
- **悬停（hover）**。用户鼠标当前位于链接上的状态（悬停）。
- **激活（active）**。链接正被鼠标单击时的状态。

下面就是针对这些状态的伪类选择符（使用了 a 选择符及一些示例声明）：

① 作者在随书的源代码中已经将这个文件改名为 csshover.htc，请读者注意。下同。——译者注
② 原文 www.stylinwithcss.com/stylib 有误。——译者注
③ 原文 JavaScript folder 有误。——译者注
④ 原文 hover.htc 有误。——译者注

```
a:link {color:black;}
a:visited {color:gray;}
a:hover {text-decoration:none;}
a:active {color:navy;}
```

首先，我们推迟对链接的颜色和行为是否合适的争论，只聚焦于这些伪类。根据上面的声明，链接在初始状态下是黑色的（并且拥有默认的下划线）。当鼠标移到它们上方时（悬停状态），下划线会消失，但仍然是黑色，因为没有为悬停状态定义颜色。当用户按下鼠标时（激活状态），链接会变成深蓝色（navy）。而在此后（或更精确地说，到浏览器中保存的访问该链接 URL 的历史记录过期或被用户删除之前），链接都会显示为灰色（gray）。通过使用这些伪类选择符时，我们可以对链接在这 4 种状态下的外观拥有完全的控制。

虽然这已经很不错了，但锚链接伪类选择符的真正威力事实上体现在将它们用作上下文选择符的情形下。这样，就可以为设计中不同部分的链接创建不同的外观和行为。比如导航区、页脚、侧边栏及正文中的链接等。我们将在本书后面介绍更复杂（或者也是更令人着迷）的例子时，使用这些伪类来为链接及其他元素添加样式。不过现在，我们要明确以下事项，然后继续。

(1) **不必定义全部 4 种状态**。比如，只定义一个链接和一个悬停状态也没有问题。有时，将链接显示为已经被访问过的状态并没有多大意义。

(2) **浏览器可能会跳过其中某些不按照下列顺序声明的规则：链接、已访问、悬停、激活**。如果你是个愤世嫉俗的人，那么助记法“LoVe-HA！[①]”是记住这一顺序的好办法。

为了创建各种翻转效果，可以将任何元素与这些伪类连用（不限于 a）。例如

```
p:hover {background-color:gray;}
```

这条规则会……好，我想对于你这么聪明的人来说，没有必要明确地告诉你当鼠标放在段落上时会发生什么。

链接（a）的伪类得到了本书测试的所有浏览器的支持——IE 5 及更新的浏览器。但是，正如前面所提到的，IE 6 不支持

① LoVe-HA！即 love hate（爱恨）！的意思。——译者注

除链接之外其他元素上的悬停伪类，除非将一个名为 hover.htc 的 JavaScript 文件链接到页面中修改 IE 行为。IE 7 虽然支持任何元素上的悬停行为，但页面的 DOCTYPE 必须是严格型。虽然听起来有点乱，但我们会在后续章节中介绍确保悬停行为对所有元素都有效的方法。

2.5.2 其他有用的伪类

伪类的目的是在特定的条件匹配时，模仿向标记中添加类的行为。通过伪类，不仅能对用户操作给出响应（如鼠标指向或单击），而且也能基于标记中某些匹配的条件来应用样式。

1. :first-child

其中 x 是一个标签名

```
x:first-child
```

这个伪类用于选择标签名为 x 的元素中的第一个子元素。例如，如果将下面的规则：

```
div.weather strong:first-child {color:red;}
```

应用到下面的标记中：

```
<div class="weather">
It's <strong>very</strong> hot and <strong>incredibly</strong>
strong> humid.
</div>
```

会使 very 变成红色，而 incredibly 不会受到影响。

如图 2-17 所示。（SCB 和 IE 7 支持 :first-child。）

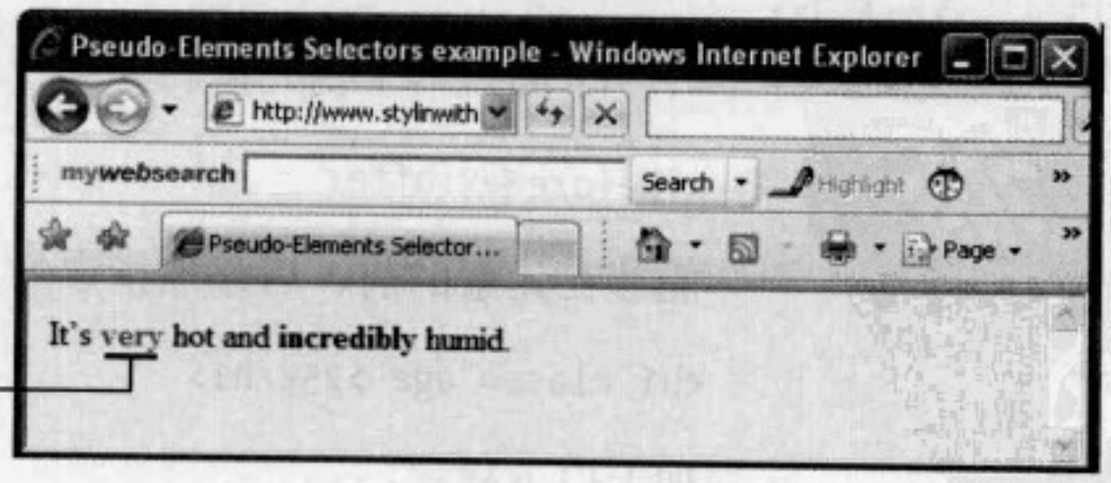


图 2-17 通过 :first-child 选择符，可以对准某种特定类型标签中的第 1 个子标签

2. :focus

其中 x 是一个标签名

```
x:focus
```

当用户单击一个表单字段时，我们说该字段会获得焦点，获得焦点的字段可以让用户在其中输入字符。例如，下面的规则：

```
input:focus {border: 1px solid blue;}
```

会在用户单击表单输入字段时为字段添加蓝色的边框（IE 6、IE 7和Safari不支持 :focus 伪类）。

2.6 伪元素

伪元素提供了在文档中魔术般地添加额外标记的效果，尽管事实上并没有添加任何额外的标记。下面来看一些例子。

使用这个伪元素：[①]

其中x是一个标签名 — `x:first-letter`

比如：

```
p:first-letter {font-size:300%; float:left;}
```

可以在段落的开头创建首字放大效果。

而通过下面这个伪元素：

其中x是一个标签名 — `x:first-line`

可以为（通常是）一段文本的第一行添加样式。例如：

```
p:first-line{font-variant:small-caps;}
```

会使段落的第一行变成小型大写字母。如果页面布局是流动的（或者说宽度自适应的），那么当调整浏览器窗口大小时，段落中的文本会自动重排，此时只有第一行会按照要求应用样式（所有SCB和IE 7支持 :first-letter 和 :first-line）。

因为搜索引擎不会收录伪元素中的内容（它不会出现在标识中），所以不要使用这些元素来添加希望由搜索引擎索引的重要内容。

下面这两个伪元素[②]：

`x:before`和`x:after`

能够在元素的前、后添加指定的文本。因此，下面的标记：

```
<h1 class="age">25</h1>
```

加上以下样式：

```
h1.age:before {content:"Age: "}
h1.age:after {content:" years old."}
```

① 原文 pseudo-class 有误。——译者注
② 原文 pseudo-class 有误。——译者注

不要依赖伪类和伪元素生成网站的关键内容，因为IE 6不支持它们，而IE 7也只是部分支持它们（这两种浏览器目前的占有率大约是70%）。应该在支持它们的浏览器中利用它们来增强用户的体验——例如，使用 :focus 选择符在用户输入时，为当前的表单字段添加粗边框。即使用户看不到这种增强效果，也仍然是可以接受的。

会导致文本变成“Age: 25 years old.”。引号中添加的空格确保了在结果字符串中输出适当的间隔。这两个选择符在标签内容通过数据库查询结果生成的情况下特别有用，假如结果中只包含数字，那么就可以通过这两个选择符为用户提供某些有意义的上下文（IE 7不支持 :before 和 :after）。

CSS3

大多数现代的浏览器都支持CSS2。CSS2在最初发布于90年代中期的CSS推荐规范基础上添加了大量新特性。CSS3是CSS的最新升级版。实际上，CSS3在2000年前后就已经发布了，而且在之后的5年中经过了不断地完善。但是，浏览器开发商并没有积极地采用CSS3。

CSS3的目标是把对文档中更多的表现控制转移到CSS中，并且进一步模仿印刷设计领域中由Adobe InDesign和QuarkXPress等排版程序所实现的高级控制功能。

由于CSS3规范覆盖面很广，因此被分割成了几个模块，包括颜色模块、背景和边框模块以及多栏布局模块等。可以访问http://www.css3.info来了解CSS3中的全部模块。

我想告诉大家的是，在3年前我写作本书第一版时，CSS3规范就同今天差不多，而且它在所有浏览器中获得支持的情况也同样如此——仍然少得可怜。其中，属性选择符是CSS3中唯一有所进展的特性，尽管很有用，但也只代表了整个规范中一个极小的方面。

CSS3真是那么难以实现吗？即使Web 2.0的整体思想从视觉上完全被封装到了带圆角的盒子中，但作为CSS3规范一部分的XHTML元素的圆角特性，也只有Mozilla能够通过其特殊的“-moz”选择符来呈现。对此，一个常见的变通方案是在元素的周围包装很多span标签，每个span中都包含圆角背景图像，以便完成元素的圆角效果。本书后面，我们会介绍一个不必使用图像就可以创建圆角的更简单的方法。

还有4种其他的伪类。第一个是 :lang，应用于带有特殊语言编码的元素。其他3个是:left、:right和:first，它们应用于媒体（如打印）而不是在浏览器中显示内容。这些伪类使用范围很窄并且也没有获得浏览器的一致支持，因此本书对它们不作介绍。

2.7 继承

如同你希望从富有的“狄克舅舅”那里继承财富一样，CSS 中的继承也涉及从祖先向后代传承某些东西——CSS 属性的值。还记得我们第 1 章关于文档层次的讨论吗，body 标签是文档中所有标签的祖先——CSS 要对准的所有标签都包含在 body 中。因此，由于 CSS 强大的继承机制，如果我们为 body 标签添加如下样式：

```
body {font-family: verdana, helvetica, sans-serif; color:
blue;}
```

那么，整个文档中所有文本元素的文本都会继承这些样式，并以蓝色的 Verdana 字体（或者在 Verdana 字体不存在时的其他字体）显示，无论该元素在文档层次中多么靠近底部。继承的效率是显而易见的——不用为每个标签分别指定期望的字体，像这样一次就可以为整个网站设置主字体。然后，只需为那些应该显示为不同字体的标签单独应用 font-family 即可。

现在，只需记住与文本及其颜色和大小相关的样式会被后代元素继承。通过为 div、段落等标签应用样式创建的与盒子的外观相关的样式，如边框、内边距、外边距和背景颜色等不会被继承。

许多 CSS 属性都会以这种方式被继承，其中最明显的就是文本属性。不过，也有很多 CSS 属性不会被继承，因为继承这些属性没有任何意义。不会被继承的属性主要涉及盒元素的定位和显示，例如边框、外边距和内边距。举例来说，假设你需要创建一个包含文本的侧边栏。你可能会为此使用一个 div（可以把它看成一个常规的盒子），其中包含一组列表。然后，你会为这个 div 添加边框效果，假设是两像素宽的红色边框。但是，让包含在 div 中的那些列表项都自动获得这样一个边框是没有意义的。事实上，它们不会取得边框——因为边框属性不能继承。在第 4 章讨论盒模型时，我们还会更详细地介绍继承机制。

此外，当使用百分比和 em 等相对尺寸时，也必须多加注意。如果你将一个标签的文本大小设置为 80%，而将其后代标签的文本大小也设置为 80%，那么这个后代标签中的文本大小就是 64%（80% 的 80%）。这恐怕不是你想得到的结果。在第 3 章中，我们会讨论绝对和相对文本大小的优缺点。

在之后的例子中，我们还会揭示继承对样式的影响，以及如何充分利用继承的样式来减少实现必要效果的 CSS 编码工作量。

2.8 层叠机制

现在，我们已经有了足够的信息对 CSS 中最难理解的问题——层叠，进行有意义的讨论了。如果你认为本节内容不好理解，可以跳到本章后面“Charlie 对层叠的简单概括”提示条部分。在你拥有实际的 CSS 编码经验并真正需要弄清楚细节之前，可以阅读这个提示条中包含的有关层叠的简要说明（可能不会那么精确）。

顾名思义，层叠样式表中的层叠意味着样式会从文档结构中的一个层次传递到另一个层次，其作用是让浏览器来决定在诸多来源中，为某个标签应用来自哪个来源的样式属性。

层叠是一种强大的机制。理解层叠有助于以最经济和最容易维护的方式来编写 CSS，也有助于根据你的意愿创建出理想的文档外观。与此同时，也会把涉及文档显示（例如全局字体大小）的控制，适当地留给有特殊需要的用户。

2.8.1 样式的来源

样式可能会来自很多地方。首先，不难理解的是，应该有一个隐藏在浏览器中的浏览器样式表（默认样式表），因为每个标签无需编写什么就会带有样式。比如，h1 标签中的文本会显示为较大的粗体，em 标签中的文本会显示为斜体，列表会缩进并且每个项目前面都会带项目符号，一切都是自动添加的。我们无需对生成这些默认的格式进行任何干预。

如果你的计算机中安装了 Firefox 浏览器，可以搜索文件 html.css——这就是 Firefox 默认的浏览器样式表。最好不要随意修改这个样式表。

其次，是用户样式表。尽管非常少见，但用户的确能够创建样式表，而且，这种能力也很方便。例如，对于视力残障用户来说，可以自己增大文本的基准大小，或者强制文本相互之间以容易辨认的颜色进行显示。在 Windows 平台的 IE（6 和 7）中，选择菜单“工具→ Internet 选项”，然后单击“辅助功能”按钮，就可以添加一个用户样式表。视力残障用户可以通过这种途径来添加类似下面的样式：

```
body {font-size:200%;}
```

以上规则会使字体大小翻倍——同样是由于继承的关系。而且，这也说明了为文本指定相对大小（例如 em）而不是固定大小（例如点）的重要性，因为相对大小不会覆盖用户样式表中的对文本大小的修改。关于这个有意思的主题，我们将在第 3 章中讨论。

最后，还有一个设计者样式表。不用多说，这个样式表就是由你——设计者编写的。前面，我们曾经讨论过样式的以下来源：链接样式表、在页面顶部嵌入样式和添加到标签中的内联样式。

下面是浏览器对待或者说层叠这些样式来源的顺序。

层叠机制规定了读取和更新文档时采用样式的顺序。

- 默认的浏览器样式表。
- 用户样式表。
- 设计者样式表。
- 设计者嵌入的样式。
- 设计者内联的样式。

当浏览器按照以上顺序查找相应位置的样式时，如果遇到了为某个标签定义的属性值，就会更新对相应标签的设置。浏览器在默认样式表中定义标签的样式，如果其他位置也为标签定义了样式，浏览器就会将设置更新为该位置定义的值。例如，如果设计者样式表中将 <p> 标签的 font-family 属性定义为 Helvetica，但嵌入（页面）样式中也为 <p> 标签定义了 Verdana，那么段落中的文本最终会以 Verdana 字体显示。因为，浏览器会在读取设计者样式表之后读取嵌入样式。但是，如果用户或设计者样式表中都没有为段落定义样式，那么段落文本的字体就是 Times，因为 Times 是所有浏览器的默认样式表中定义的字体。

这些就是层叠的基本工作原理。不过，也有一些控制层叠的规则。

可以在 W3C 网站（www.w3.org/TR/CSS2/cascade.html）上看到更多与层叠相关的信息。

2.8.2 层叠规则

除了上面应用样式的顺序之外，你还应该知道一些关于层叠如何工作的规则。

层叠规则 1：找到应用于每个元素和属性的全部声明。当每个页面加载时，浏览器会为页面中的每个标签查找匹配的样式规则。

层叠规则 2：按次序和重要性排序。浏览器会依次检查 5 种样式来源，并设置所有匹配的属性。如果后面的来源再次设置了某个匹配的属性，那么浏览器会在必要时更新该属性的值并重复这一过程。直到针对页面中每个标签的属性检查完全部 5 种可能的来源为止。在这一过程最后，浏览器为特定属性设置的值将决定该属性的显示效果。

在表 2-1 中，可以通过一个带有很多 p 标签的页面来演示这一过程。在此，我们假设页面中有两个 p 标签包含了将颜色定义为红色的内联样式。那么，除了这两个带有内联颜色样式的 p 标签会应用红色之外，其他段落的文本都是蓝色。

表 2-1 层叠的例子

来　源	标签	属性	值
默认样式表	p	color	black
用户样式表			
设计者样式表	p	color	blue
设计者嵌入的样式			
设计者内联的样式	p	color	red

当然，事情不会如此简单。规则的声明也有重要性之分。可以像下面这样，把一条规则中的声明定义为重要：

注意声明中的叹号

```
p {color:red !important; font-size:12pt;}
```

其中 !important 位于要标记为重要的声明后面的空格之后，但位于分号（;）分隔符之前。

由于这个样式将文本的红色定义为重要，因此，即使位于层叠顺序后面的样式来源中声明了其他颜色，文本也仍然会显示为红色。在通过 !important 规则强制定义与用户相关的属性时，一定要慎重行事并考虑周全，因为这样做可能会干扰用户样式表中原本很合适的设置。为此，要保证用真正重要的样式覆盖针对相应标签的其他可能的样式。

Charlie对层叠的简单概括

在这个概述层叠规则的简化版中，只需记住 3 条。事实上，这 3 条适用于任何情况。

规则 1：包含 ID 的选择符覆盖包含类的选择符，进而，包含类的选择符覆盖只包含标签的选择符。

规则 2：如果层叠中的不同位置为同一个标签的同一个属性多次定义了值，那么，内联样式覆盖嵌入样式，嵌入样式覆盖样式表中的样式。但是，当选择符更有针对性时，无论样式处于什么位置，都是优先适用的——也就是说，规则 1 优先于规则 2。

规则 3：无论针对性如何，定义的样式都覆盖继承的样式。对规则 3 而言，需要进一步解释。比如下面的标记：

```
<div id="cascadedemo">
<p id="inheritancefact">Inheritance is <em>weak</em> in the Cascade</p>
</div>
```

和下面具有高针对性的规则：

```
2-0-4 ─{ html body div#cascadedemo p#inheritancefact {color:blue:}
```

会导致全部文本——包括 weak 都显示为蓝色，因为 em 会继承其父元素（p 标签）的颜色。但是，如果我们为 em 添加了如下规则，即使它的针对性很低：

```
0-0-1 ─{ em {color:red}
```

也会使 em 中的文本变成红色。因为，无论父段落的规则针对性有多高，为 em 直接定义的样式都会覆盖继承的样式。

这就是 3 个简单的层叠规则。掌握起来很轻松，不是吗？

层叠规则 3：按照针对性排序。在很难作出判断的情况下，针对性可以决定规则的胜负。在讨论选择符时，我们曾经多次从层叠规则的意义上提到过针对性的概念。如果样式表中包含

```
p {font-size:12px;}
```

和

```
p.largetext {font-size:16px;}
```

这两条规则，那么下面标记中的文本将会显示为 16 像素高。

```
<p class="largetext">A bit of text</p>
```

由于第 2 条规则更具有针对性——因而它优先于第 1 条规则。这个结果从直觉上似乎是显而易见，但是，如果我们对这些标记使用下面的样式又会导致什么结果呢？

```
p {font-size:12px;}
.largetext {font-size:16px;}
```

这两条规则都匹配同一个标签，但是因为类选择符优先，所以文本会显示为 16 像素。具体原因是：标签选择符的数字针对性为 1，而类选择符的针对性为 1-0。这里，存在一个如何计算选择符针对性的问题。为此，针对每个样式都会有一个简单的计分系统，分值的表示形式为如下 3 个值：

A - B - C

其中的短划线是分隔符，而不是减号。具体的计分办法如下所述。

(1) 选择符中存在一个 ID，就要为 A 加上 1。

(2) 选择符中存在一个类，就要为 B 加上 1。

(3) 选择符中存在一个元素名（标签名），就要为 C 加上 1。

(4) 将最后的结果按照三位数来计算。（最后的结果不是真正的三位数，只不过在多数情况下，将其理解为一个三位数更容易判断。比如，0-1-12 与 0-2-0 相比，后者更具有针对性。）

我们来看一看下列例子的针对性：

0 - 0 - 1 针对性 =1	p
0 - 1 - 1 针对性 =11	p.largetext
1 - 0 - 1 针对性 =101	p#largetext
1 - 0 - 2 针对性 =102	body p#largetext
1 - 1 - 3 针对性 =113	body p#largetext ul.mylist
1 - 1 - 4 针对性 =114	body p#largetext ul.mylist li

以上每个例子的针对性都高于前一个。

针对性比顺序更重要，因此具有更高针对性的规则要优先于在层叠中更靠近底层但针对性更低的规则。

层叠规则 4：按顺序排序。如果两条规则具有完全相同的权重，那么在层叠顺序中位于最下层的规则优先。

以上，就是所谓的层叠。没错，这个概念的确不容易理解，特别是在你还没有丰富的 CSS 经验时尤其如此。不过，我整理的简化版的层叠规则（参见本章前面的侧边栏“Charlie 对层叠的简单概括”）适用于 98% 的情况。如果你在使用这个简化版时，发现了什么不符合规则的问题，请参考上面介绍的规则。

2.9 规则声明

到目前为止，我们的讨论一直围绕着使用CSS规则中的选择符对准标签而展开，还没有关注CSS规则中的另外一半——声明。虽然我们也曾利用各种声明来示范选择符的用法，但对声明的解释却很少。因此，下面我们就来详细地介绍声明。

本章前面展示CSS规则结构的图示（图2-2）表明，声明由两部分组成：属性和值。其中，属性定义的是要改变的元素的哪一方面（颜色、高度等），而值则定义了要把属性设置为多少或怎么样（12 px、绿色等）。

每个元素都有许多能够通过CSS设置的属性，这些属性根据元素而各不相同。例如，可以为文本设置font-size属性，但却不能为图像设置该属性。在本书后续的章节中，我们将围绕真实的例子来介绍各种元素的属性及可用的值。因为CSS规则中的值只有少数几种类型，所以我们先来介绍一下CSS中属性的值。

CSS属性的值主要可以分为如下3类。

- **单词**。例如在font-weight:bold中，单词bold就是值。
- **数字值**。数字值后面通常要跟着一个单位。例如，在font-size:12px中，12是数字值，而px是单位，即像素。
- **颜色值**。颜色值的形式是color:#336699，这里的颜色是使用十六进制值定义的。

关于单词这种类型的值，可以介绍的不多。因为它们与具体的元素息息相关，所以只有在使用时才能体会到它们的作用。而对于数字和颜色值，也只能以某些特定的方式进行描述。

2.9.1 数字值

数字值用于定义各种元素的长度（这里的“长度”也包括高度、宽度和粗细等）。数字值也可以分为两种主要的类型：绝对值和相对值。

绝对值（表 2-2）表示的是真实长度（例如 6 英寸[①]），而相对值表示的则是与其他度量值之间的相对关系（当我们说“两倍长”时，就是指的相对于其他值）。

表 2-2 绝对值

绝 对 值	简写单位	示 例
英寸	in	height:6in
厘米	cm	height:40cm
毫米	mm	height:500mm
点	pt	height:60pt
十二点活字（Pica）	pc	height:90pc
像素	px	height:72px

注：示例中的长度是不相等的

在编写 CSS 时，如果涉及与固定尺寸有关的元素（如图像），本书只使用像素。尽管使用哪种单位取决于你自己，但本书中的绝对单位只使用像素。唯一的例外是在打印样式表中——因为纸张的度量单位是英寸，所以使用相同的单位设计打印布局更加直观。

尽管绝对单位都不用解释什么，但相对单位（表 2-3）则需要更多地说明。

在相对单位中，em 和 ex 都是与字体大小有关的度量单位。其中，em 的计算依据是一种字体中字符的宽度，也就是说，em 的大小根据使用字体的不同会有所不同。而 ex 则等于给定字体中字母 x 的高度（之所以叫 ex，就是因为它是指小写 x 的高度——换句话说，ex 代表的高度不包括 p 和 d 这样的字母上下方多出的部分，只包括中间的主体部分）。

表 2-3 相对值

相 对 值	简写单位	示 例
em	em	height:1.2em
ex	ex	height:6ex
百分比	%	height:120%

使用百分比可以为包含元素（例如 div）设置相对于浏览器宽度的宽度值，而这正是创建“流动”布局的一种方式。所谓

① 1 英寸 =2.54 厘米。——编者注

“流动”布局，是指页面布局能够随着浏览器窗口的大小调整而平滑地发生变化。百分比也是创建成比例变化的行间距的正确方式。这里的行间距是指在段落包含的多行文本中，两行文本基线之间的距离。第 3 章会更详细地讨论有关行间距的内容。

为什么应该使用em指定字体大小

使用像 em 这样的相对度量单位来定义字体大小，主要有以下两个好处。

- 可以充分利用 CSS 中的继承性。通过为 body 元素声明 1 em 的字体大小，可以把这个大小作为整个文档的基准，因为文档中所有元素的文本大小都相对于 body 元素。由于内容文本总是会包含在相应的元素中（例如 p 或 h4），我们可以针对这个基准来编写 CSS 规则。比如，将 p 标签指定为 0.8 em，将文本中的链接指定为 0.7 em。这样，就可以在页面的所有文本元素之间构建起一种均衡的比例关系。

  这里要注意的是，在 IE 6 中，如果为 body 设置了 em 大小，段落中的文本大小会自动按照比例变化，但 h1 ～ h6 则不会。为此，必须要将 h1 ～ h6 明确地设置为相对大小（例如将 h1 和 h2 分别设置为 1.1 em 和 0.9 em 等），否则，标题中的文本会保持其默认大小。

  如果将来需要从整体上增大网站中的文本大小，可以只修改 body 标签，比如将字体大小设置为 1.2 em。随后，全部文本都会魔术般地按照相同的比例增大（在这里会增大 20%）。这是因为其他标签都会从 body 标签继承字体大小的值。

- 如果在定义字体大小时不使用相对单位，就等同于禁用了 IE 的“查看”菜单中缩放文字大小的功能（尽管其他浏览器能够缩放绝对的字体大小）。因而，会剥夺视力残障用户利用这一功能将内容调整到适当大小的权利。不过，在开发过程中，你也必须为此而频繁地进行测试，以确保放大后的字体不会破坏页面的结构。

因为以上两个理由，我建议所有对字体大小的设置都以 em 为单位，不要使用绝对单位（如像素）。当在固定的水平空间中设计一行选项卡时，布局结构可能会因为文本大小的变化而遭到破坏。但是，如果你在设计中时刻考虑到这种可能性，那么也可以开发出这样的界面组件，使它能够随着用户对字体大小的调整而容纳更大的字体。

2.9.2 颜色值

颜色值也包含几种不同的形式。可以使用下列任何一种你认为合适的形式来定义颜色。

大多数十六进制颜色都无法轻易猜出来。例如，#7CA9BE 是暗竹绿色。不过，假如你只注意每对 RGB 值中的第一个数字——即这里的 7、A 和 B，那么你会发现红色值要比最大值 16 的一半稍低一些，而绿和蓝色的值则较高，并且比较接近。学习识别这些信息，有助于猜测十六进制值表示的颜色。

十六进制值（#RRGGBB 和 #RGB）。如果你了解 C++、PHP 或 JavaScript 等语言，那么应该熟悉颜色的十六进制（hex）表示法。其格式如下：

```
#RRGGBB
```

在这个由 6 个字符组成的值中，前两个字符定义红色，中间两个字符定义绿色，最后两个字符定义蓝色。计算机使用二进制来计数，而不是人类使用的十进制方式，这也是十六进制（2 的 4 次幂）数使用 16 个数字（数字 0 ～ 9 和字母 A ～ F，其中 A ～ F 表示 10 到 15）的原因。由于每种颜色通过一对十六进制数表示，因此每种颜色就有 256（16×16）种可能的值，进而就会有 16 777 216（256×256×256）种颜色的组合。通过使用十六进制表示法，我们能够得到用不完的颜色，因为实际用到的颜色远远没有那么多。对于相邻的两个十六进制颜色，人类的眼睛很难分辨它们之间的差别（更不必说显示器了）。最后，不要忘记颜色值前面的 #（散列）符号。

举例来说，十六进制值表示的纯红色是 #FF0000、纯绿色是 #00FF00、纯蓝色是 #0000FF。

另外，也可以在适当的情况下使用以下面简写的十六进制格式：

```
#RGB
```

如果在我们要使用的颜色中，每对十六进制数都相同，例如 #FF3322（大红色），那么就可以将它简写为 #F32。

百分比 RGB（R%, G%, B%）。在这种表示法中，每种颜色使用百分比值表示：

```
R%, G%, B%
```

每种颜色可以接受的值为 0% 到 100%。虽然这种表示法只能表示最多 100 万种（100 × 100 × 100）颜色，但对我们来说也绰绰有余了。而且，同十六进制值表示法相比，百分比表示法更容易估计 RGB 值的比例。

比如说，100%、0%、0% 表示纯红，0%、100%、0% 表示

纯绿，而 46%、76%、80% 则接近于上面以十六进制值表示法示范的暗竹绿色。

颜色名（如 red）。在前面讨论选择符的例子中我们已经看到，可以通过正式的名字或者说关键字来指定颜色。但是，这种表示法存在着比较大的局限性。W3C 的规范没有明确规定浏览器如何呈现 olive（橄榄色）或 lime（浅绿色），一般来说，每个浏览器制造商会为相应的颜色关键字指定自己认为适当的值（大概也是十六进制值）。此外，W3C 的规范中只规定了 16 种颜色关键字，因此只能保证在每种浏览器中确实存在 16 种可用的关键字。以下是按字母顺序列出的这 16 种颜色关键字：

aqua、black、blue、fuchsia、gray、green、lime、maroon、navy、olive、purple、red、silver、teal、white、yellow

多数现代的浏览器都会提供更多的颜色关键字（通常能达到 140 种之多）。但是，如果你想用名字来指定颜色，那么有把握的只有这 16 种。

由于编程的缘故，我通常使用十六进制颜色表示法，而且这也是编程环境下的首选方式。为了节省自己调配颜色的时间，可以参考 http://www.bookmarkbliss.com/tools/bookmark-bliss-10-tools-to-help-you-select-a-web-20-color-palette/，其中包含一些有助于选择颜色的实用工具。另外，也请参见提示条“没有必要将自己局限在 Web 安全色中”。

既然我们已经理解了 CSS 的工作原理，那下一章就来探讨如何为文本添加样式。

没有必要将自己局限在Web安全色中

如果你使用 Adobe Dreamweaver 或其他 Web 开发工具，那么可能使用过 Web 安全色面板。所谓 Web 安全色，是指由一名工程师提出的 216 种颜色，其中包含了亮度和饱和度最高的颜色，同时排除了晦暗的和低对比度的颜色。这 216 种颜色全部由三对十六进制数字组成，比如 #3399CC 或 #FF99CC，而且只使用了 0、3、6、9、C 和 F 这几个数字。这样就确保了组合而成的任何颜色都符合 Web 安全色的标准。这些颜色是 8 位 VGA 显卡能够显示的 256 种颜色的一个大子集。（对 8 位显示器的时代还有印象吗？）几年之前，我就提醒人们不要再局限于这些颜色。到 2007 年 7 月为止，全世界只有不到 0.01% 的上网者（http://www.thecounter.com/stats/2007/July/colors.php）还在使用 8 位颜色的显示器。因此，你可以放心地使用本章介绍的方法在设计中应用任何颜色。

第 3 章

字体和文本样式

Web 设计中很大一部分工作都与处理文本有关，这些文本可能位于段落、标题、列表、菜单和表单当中。为此，本章介绍的 CSS 属性，主要用于构建从视觉上看具有专业水准而不是随意堆砌的页面文本。除了其他因素之外，字体应该是对网站内容品质的一种最直观的表现。图像基本上属于锦上添花，而排布才是良好设计的起点。

如果你对本章的标题感到奇怪——“难道字体和文本不是一回事吗？”，答案是“不是一回事”，理由如下。

字体是指一种具体的字形[①]。每种字体都包含一组具有独特外观的字母、数字和符号。外观具有共性的一组字体构成一个更大的集合，例如 serif（衬线）、sans-serif（非衬线）和 monospace（等宽）等。字体集合是由字体系列组成的，常见的字体系列有 Times 和 Helvetica 等。一种字体系列又包含多种字体（具体的字形），每种字体都是基本的字体系列的变体（如 Times Roman、Times Bold、Helvetica Condensed、Bodoni Italic 等）。

文本是由单个字符组成的字符块，就像这个句子或者本章的标题一样，和为它设置的字体无关。

在 CSS 中，字体和文本分别具有不同的属性。字体的属性主要涉及文字的大小和外观。具体来说，就是使用哪个字体系列（例如，Times 还是 Helvetica）？多大的尺寸？是粗体还是斜体？文本的属性主要涉及对字体的处理。比如行高和字母间距是多少？是否加下划线或者缩进？等等。

对于字体和文本的微妙差别，我是这样区分的：可以将字体样式，例如粗体或斜体，应用于一个字符；但文本属性，例如行高和文本缩进，则只对文本块（例如标题或段落）才有意义。

下面，我们就从字体开始。

3.1 在CSS中指定字体

在本节中，我们讨论如何使用 CSS 来指定字体。可以使用任何适当的长度单位（绝对值和相对值）来指定字体大小，但最好是使用相对的度量手段，比如 em。相对的字体大小可以使用户通过浏览器的相关选项（如“查看”菜单中的“文字大小”选项）很容易地将字体设置为其他尺寸值，或者通过用户样式表来自定义字体大小。

① 由于英文字体同中文字体概念不同，所以这里采取折中的译法。——译者注

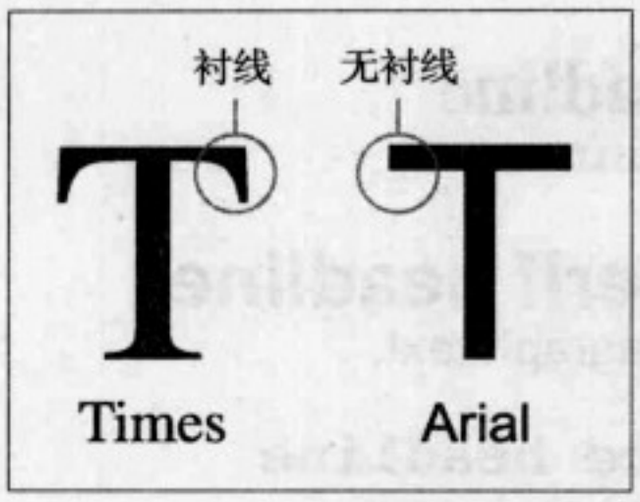


图 3-1 serif 字体的笔画末端有明显的细节。sans-serif 字体则没有这些细节

字体集合

例子：body {font-family: sans-serif;}

值：serif, sans-serif, monospace, fantasy, cursive

在 CSS 中指定字体的最简单方式，就是使用以下 5 种通用的集合名称：serif、sans-serif、monospace、fantasy（梦幻）和 cursive（草体）。这些通用的名称允许用户代理（浏览器、智能电话、手机等）从相应集合中选择一款字体。通用集合名称一般用来指定对字体样式的最低限度的支持，稍后我们会介绍到，CSS 也提供了更好的指定字体的方式。

serif 字体（也就是图 3-1 左侧的字体）之所以如此命名，是因为该字体在字符笔画末端有叫做衬线（serif）的一些小细节。这些细节在大写字母中特别明显。serif 字体包括 Times New Roman、Georgia 和 Palatino 等。本段中的英文使用的就是 serif 字体。

sans-serif 字体的字符笔画末端没有任何细节。与 serif 字体相比，它们的外形更简单。属于 sans-serif 集合的字体有 Trebuchet MS、Arial 和 Verdana。本书标题中的英文使用的都是 sans-serif 字体。

serif 和 sans-serif 字体在空间上都是成比例的，也就是说每个字符只会占据必要的空间。因此，字母“i”占据的空间要小于字母“m”。

monospace 字体，比如 Courier 和 Monotype，每个字母的宽度相等（即“i”与“m”具有相同的宽度）。这种字体通常用于在计算机相关的书籍中排版代码块，或者模仿打字机打出的文字效果。

cursive 字体看起来就像是手写笔迹，但比真正的手写效果要整洁得多。cursive 字体的例子有 Comic Sans MS 和 Brush Script。cursive 字体的笔画一般较细，不适合屏幕显示，所以在网页上使用得不多。如果你想使用 cursive 字体，一定要在各种浏览器中进行测试，因为每种浏览器都会使用不同的 cursive 字体。

一般来说，我建议只使用 serif、sans-serif 和 monospace 这 3 种字体集合。如果你想使用 cursive 或 fantasy，那么必须小心谨慎并不厌其烦地进行测试。

fantasy 字体是指不能归入其他种类的字体。这种字体的主要目的是为指定字体提供一种方案。但是，由于很难预知不同的浏览器会把什么字体归类为 fantasy 字体，所以最好是避免使用 fantasy 字体。而且，“fantasy”并不真的是同 cursive 或 serif 一样的字体集合名称。我只在 CSS 规范中看到过 fantasy 作为一种字体集合的名称存在，或许我的研究还不够深入。

图 3-2 Mac 平台的 Safari 2.0.4 中显示的通用字体系列

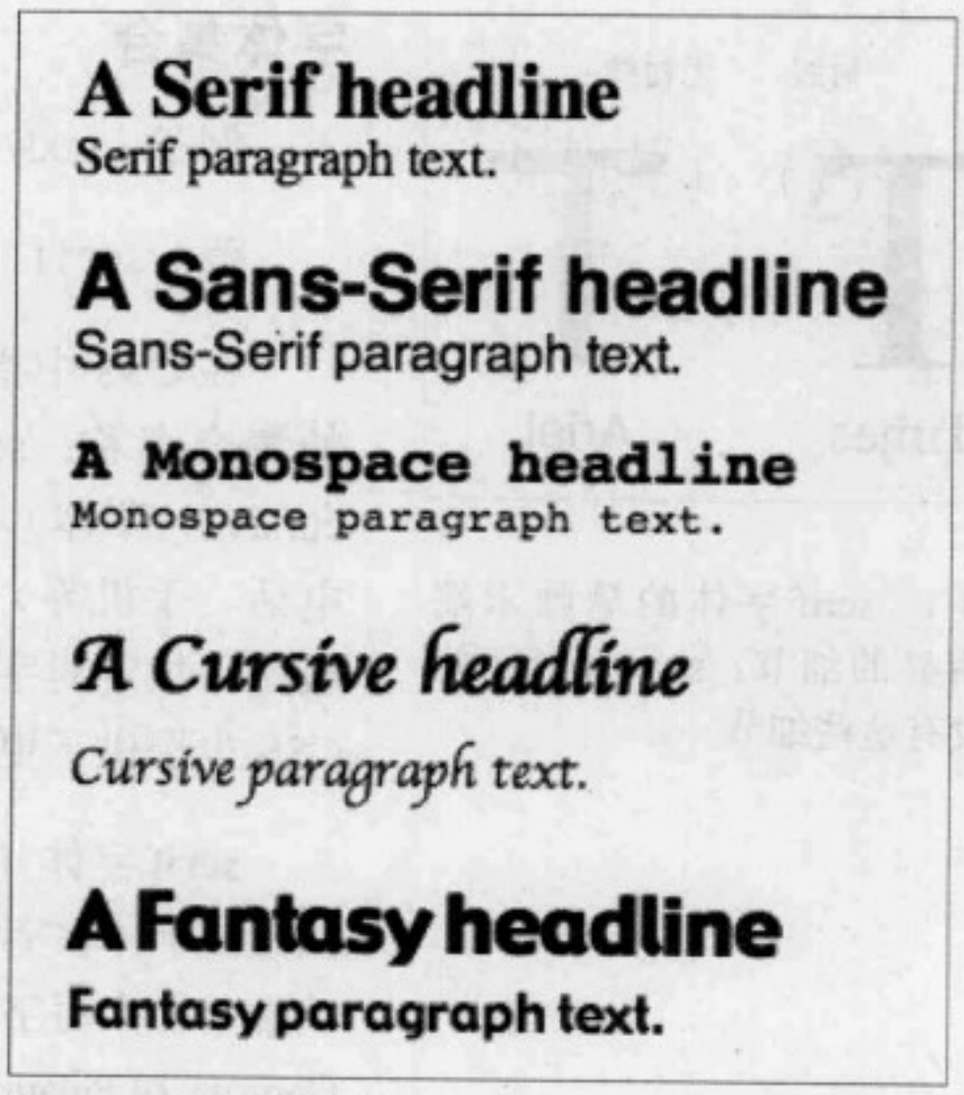


图 3-3 Mac 平台的 Firefox 1.0.4 中显示的通用字体系列

图 3-4 Windows 平台的 IE 6 中显示的通用字体系列，与 IE 7 中的效果相同

图 3-5 Windows 平台的 Firefox 2.0.4 中显示的通用字体系列

如果要指定一种通用字体，可以使用如下声明：

```
body {font-family:sans-serif;}
```

此时，浏览器会选择它自己默认的 Helvetica 或 Arial，或者其他 sans-serif 字体，这些也是用户的计算机中默认的（见图 3-2 ～图 3-5）。这是指定字体的最基本方式。不过，我们也可以在声明中使用一种更具体的字体系列的名称——通常，这也是定义字体样式的首选方式。

sans-serif适合Web设计

可以看一看那些大型的、以文字为主的网站，例如 CNN、MSNBC 或者 Amazon。不知你能否看出这些网站中广泛使用的 sans-serif 字体？在印刷品中，衬线字体的小细节有助于提供更多的视觉信息。但是，分辨率有限的屏幕却无法满足显示衬线的要求，尤其在字体较小时效果更差。

如果你刚开始涉足 Web 设计，我建议你使用 sans-serif 字体，因为它的字体看起来既明快又专业——除非你对使用 serif 字体有自己的经验。将所有浏览器默认的 Times 字体简单地修改为 sans-serif 字体，是把网站变得更具专业水准的一件最容易也最有实效的事。

3.2 探索字体系列

在印刷领域中，你几乎可以使用任何字体。如果缺少某种字体，就去购买并把它们安装到计算机中，之后就可以使用这些字体来设计文档。在设计结束后，你可以把文档输出为 PDF 格式，以便将字体转换成矢量图（轮廓线），然后设计就完成了。在印刷时，也有数以千计的字体可供选择，因为你对印刷机可以进行完全的掌握。

在 Web 设计领域，则没有这种随意选择字体的自由。对于从印刷设计转到 Web 设计的人来说，这是最令人沮丧的事情。你必须寄希望于页面访问者的计算机中安装了你想用来显示文档的字体。但是，字体并不属于浏览器，而是由浏览器所在的系统软件向计算机中的所有应用程序提供的。并且，根本无法预知用户的计算机中安装了给定字体的哪些子集（例如 Times、Times Regular 或 Times Roman 等）。

WEB FONT SELECTION CHART	
Serif fonts	Georgia
	Palatino
	Times New Roman
Sans-serif fonts	Arial
	Arial Black
	Arial Narrow
	Century Gothic
	Franklin Gothic
	Impact
	Tahoma
	Trebuchet
	Verdana
Monospace fonts	Courier New
	Lucida Console
Cursive fonts	*Comic Sans MS*

图 3-6 Mac 和 Windows 系统中通常会预先安装这些字体

要了解 Windows 和 Mac 系统中更多常见的字体，请访问：http://www.kathymarks.com/archives/2006/11/best_fonts_for_the_web_1.html 或者 http://www.ampsoft.net/webdesign-l/WindowsMacFonts.html。

由于 Trebuchet MS 字体名称包含两个单词，所以必须使用引号。如果是在使用了双引号的内联样式中声明这种字体，应该使用单引号，比如 <p style="font-family:'trebuchet ms', helvetica, arial, sans-serif;">。

假设你想以 Univers 87 Oblique 字体来显示网页中的标题，但用户安装了这种特殊字体的可能性却（借用歌手 Elvis Costello 的话说）“小于零（less than zero）”。即使对于现在最普及的 sans-serif 字体 Helvetica 来说，也没有包含在 Windows 系统中，尽管 Windows 中包含了它自己的几乎相同的 Microsoft Sans Serif 字体。不过，可以确定的是每台计算机中至少会安装有 Times New Roman、Arial、Verdana 和 Courier 字体。而且，所有较新的计算机中几乎肯定会包含图 3-6 中所列的字体。

“要是让用户根据需要从我的服务器上自动下载字体，不就可以解决问题了吗？”你可能会这样问——问得好。尽管 CSS3 中规定了浏览器可以根据显示文档的需要从服务器上下载字体的方式，但当前的浏览器都不支持这种功能。不过，这无疑是一个很好的想法，说不定哪一天这个想法就会变成现实（即使这一天到来，浏览器也不会在用户的计算机中安装字体，而只会用它显示网页）。

在字体能够按需使用的美好明天到来之前，为了通过 CSS 来指定字体，必须按照显示文档的优先次序列出字体。这个字体列表中只能包含系列名称，也就是说必须使用 Helvetica 或 Times，而不能使用 Helvetica Condensed 或 Times Expanded。

为此，可以在 CSS 声明中指定多种 serif 或 sans-serif 字体，并且以首选字体开头，以通用的字体名称（例如 serif 或 sans-serif）结束。以下的例子就使用了 sans-serif 字体：

```
body {font-family:"trebuchet ms", helvetica, arial, sans-serif;}
```

在这个例子中，我们使用了 font-family 属性。整个声明相当于告诉浏览器：“用 Trebuchet MS 字体显示这个文档，如果没有这种字体，改用 Helvetica。如果也没有 Helvetica，也可以使用 Arial。要是连 Arial 也没有，那么随便使用现有的任何 sans-serif 字体”。此处，将通用的字体集合名称（serif 或 sans-serif）作为最后的备用项目非常重要。因为，这样至少可以保证浏览器能够以正确的字体来显示文档。

再看一个使用 serif 字体的例子：

```
body{ font-family:"hoefler text", times, serif;}
```

这个例子中同样使用了 font-family 属性，其中首选的字体是 Hoefler Text。但是，因为 Windows 用户通常没有 Hoefler Text，所以他们会看到以第 2 种字体（Times）显示的文档。假如在这个例子中把 Times 放到 Hoefler Text 前面，那么所有人都会看到 Times 字体显示的文档，因为 Mac 和 Windows 平台都包含 Times 字体。

此外，在微软的新操作系统 Vista 中也提供了一些新字体。我把这些字体称为“VistaC”，因为它们都是基于微软的 ClearFont 技术创建的：Calibri、Cambria、Candara、Consolas、Constantia 和 Corbel。可以在 http://www.modernlifeisrubbish.co.uk/article/new-vista-fonts-and-the-web 中看到这些字体的外观。通过把这些字体放到列表前面，Vista 用户就能够通过新字体来查看页面，而其他的操作系统则会退而选择第 2 或第 3 种字体。

准备为文档添加样式

掌握字体之间差别的最好方式，就是实际地为文档添加样式。因此，需要准备好一个 XHTML 编辑器（例如 Adobe Dreamweaver）和浏览器，以便为本书源文件中包含的示例文档（例如 chapter_1 文件夹中的 sample_XHTML_markup_ch1.html）应用样式。相应的步骤如下。

(1) 从本书网站（www.stylinwithcss.com）上把示例文档文件夹下载到你的硬盘中。

(2) 通过 XHTML 编辑器的“文件”菜单，打开 chapter_1 文件夹中的 sample_XHTML_markup_ch1.html。

(3) 在浏览器中打开同一个网页文件。

同时在浏览器和编辑器中打开一个网页文件很方便，因为当你在编辑器中编写样式后，可以通过浏览器实时地观察效果。

(4) 每次修改 XHTML 文档并保存后，切换到浏览器（在 Windows 中可以按快捷键 Alt-Tab，在 Mac 中可以按 Command-Tab），并按 F5 功能键（在 Safari 中按 Command-R）刷新页面。

然后，就能看到更新后的文档外观。

3.2.1 使用嵌入样式（仅现在）

为了简单起见，我们先在文档头部的 style 元素中示范如何编写 CSS 样式。这样一来，就免除了管理独立样式表文件的麻烦，不过我们编写的样式也只能对所在的文档有效。对于设

计单个页面（比如下面我们要使用的例子页面）来说，使用嵌入样式是一种理想的方法，后面，我们将创建一个单独的样式表，以便将这些样式应用到多个页面。如果你对应用样式的方式还没有完全理解，请复习第 2 章开始的部分。

下面，我们使用第 1 章中创建的 XHTML 文档，在它的头部添加 style 元素，如下面突出显示的代码所示：

```
<head>
<title>A Sample XHTML Document</title>
<meta http-equiv="Content-type" content="text/html;
charset=iso-8859-1" />
<meta http-equiv="Content-Language" content="en-us" />
<style type="text/css">

</style>
</head>
```

指定字体系列的 CSS 规则位于这个空行中

位于开始和结束的 style 标签之间的空行，就是添加 CSS 规则的地方。当浏览器遇到 style 元素的开始标签时，它会停止对 XHTML 代码的解析并开始解析 CSS 代码。当它遇到 style 元素的结束标签时，则又会把后面的代码当作 XHTML 来解析。因此，我们在 style 元素中编写的任何代码都必须符合 CSS 语法的要求。同样，如果 CSS 规则位于一个独立的样式表文件中，也是如此。也就是说，位于 style 元素中的代码必须符合如下所示的 CSS 规则：

```
selector {property1:value; property2:value;}
```

在开发项目期间，你必须时刻提醒自己是在编写 CSS 还是 XHTML 代码，并确保代码的语法正确。

3.2.2　为整个页面设置字体系列

为整个页面设置字体系列，需要为文档的主体添加相应的样式：

```
<style type="text/css">
body {font-family: verdana, arial, sans-serif;}
</style>
```

保存修改，切换到浏览器并刷新页面。应该看到如图 3-7 所示的结果。

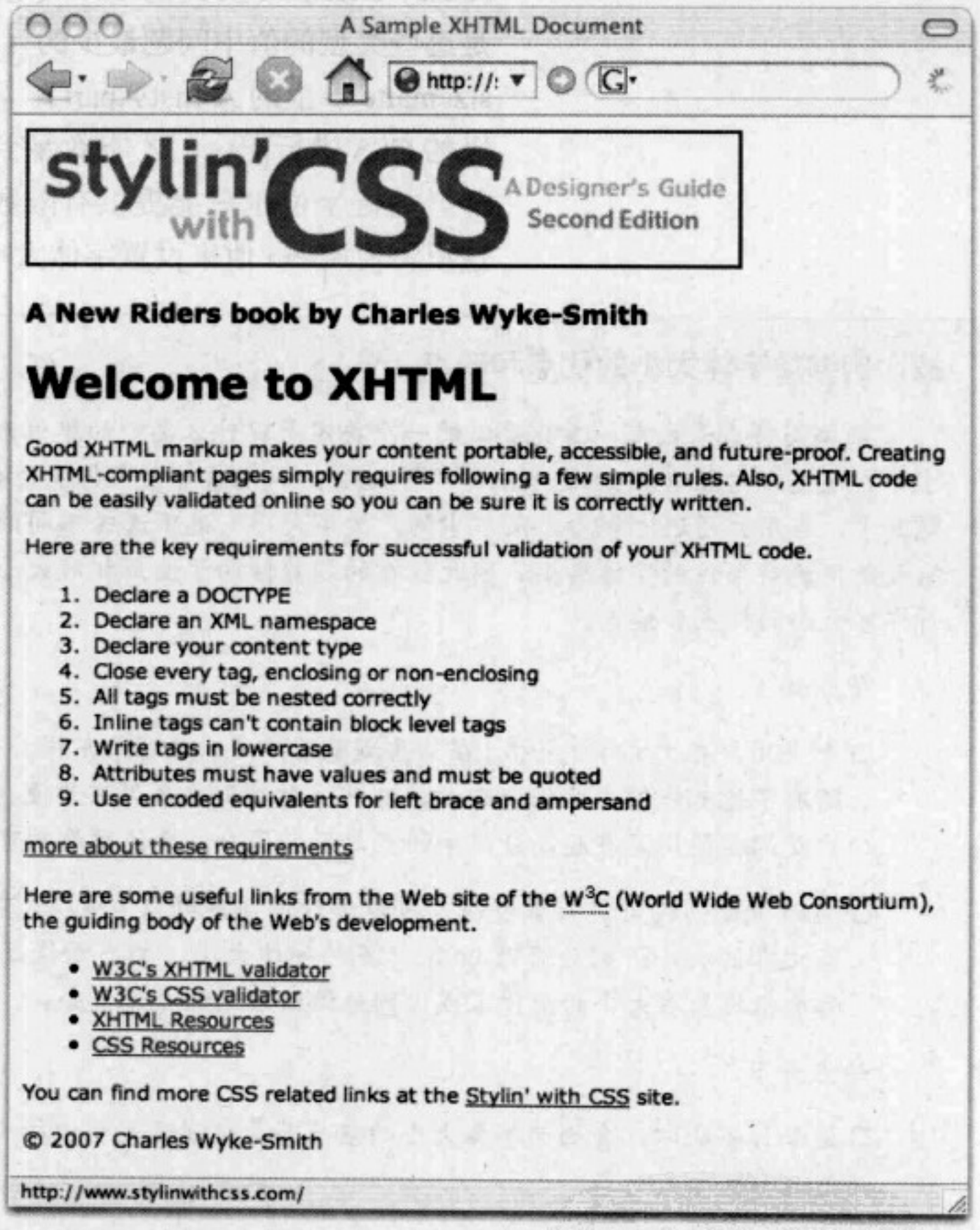


图 3-7 因为字体属性可以继承，所以为 body 标签指定字体会使整个文档都以指定字体显示

因为 font-family 是一个可以继承的属性，它的值会传递给所有后代元素，这里，由于 body 是顶层元素，也就是会传递给标记中的所有元素。这样，仅通过一行代码，就可以做到让所有元素都以期望的字体显示。现在，让我们在 CSS 魔力的光环之下沉浸片刻。好了，继续前进……

除 Safari 之外，所有浏览器都会在默认情况下为包含在 a 标签中的图像添加蓝色的边框，以表示该图像可以单击。不过，我更愿意通过工具提示条来给出这个提示并移除默认的边框，稍后我们就来实现这一点。

3.3 设置字体大小

在设置字体大小时，可以使用 3 种类型的值，分别是：绝对值（例如，像素或英寸）、相对值（例如，百分比或 em）和

我称之“运动衫尺码关键字”的值（例如，x-small、small、large 和 xx-large 等）。这 3 种类型的值都有各自的优缺点。Jeffrey Zeldman 及其他 CSS 专家提倡使用关键字，因为关键字是这些类型的值中问题最少的（参见 www.alistapart.com/articles/sizematters/ 上的 A List Apart）。不过，使用关键字值要求某些高级的 CSS 进行配合，才能确保字体在所有浏览器中的显示一致，并且关键字值也只能提供有限数量的字体大小。为此，我们在这里将使用 em 值来设置字体大小。

按比例缩放字体大小的优点和缺点

在编写样式表之前，首先要做的一个决定是以什么类型的单位来设置字体大小：绝对值（点、英寸等）还是相对值（百分比、em 等）？过去，人们喜欢使用像素，但是在 IE 及其他不符合标准的浏览器中，当用户通过“查看”菜单中的“文字大小”选项选择不同的字体大小时，这些浏览器都不能缩放使用绝对单位的字体大小。因此现在的趋势倾向于使用相对大小。以下我们分别来看一下使用相对字体大小的优点和缺点。

优点如下。

- 如果用户在“文字大小”（在某些浏览器中的叫法可能不同）选项中选择了更大或更小的选项，所有字体都能够成比例地缩放。这是一种对用户来说极为便捷的手段，特别是对视力残障的用户及那些使用具有超高分辨率的显示器的用户，会使网站的可访问性得到增强。
- 当对页面的设计进行调整时，可以通过简单地修改 body 标签的字体大小，就能按比例改变所有文本的大小；因为修改 body 标签的字体大小，相当于修改了基准大小，因而其他所有元素都会根据基准大小的变化来成比例地缩放它们的字体大小。

缺点如下。

- 假如你不小心，会因为字体大小的继承而导致嵌套元素以极小的字体显示（使用关键字则会避免这个问题）。
- 用户很容易“破坏”没有考虑到文本缩放因素的 CSS 页面布局。例如，当用户通过“查看”菜单设置了较大的字体时，“浮动栏”布局可能会遭到破坏——比如右栏会因为内容过大而无法适应原来的空间，被强制移到内容区的下方。在第 6 章讨论创建基于 CSS 的高级页面布局时，将会讨论这个问题，并且会详细介绍如何避免该问题。

多数浏览器都会为 1 em 设置默认的字体大小（一般是 16 像素高），如果我们把文本设置为 1 em，那么就相当于将字体大小设置为浏览器的默认值。所以，假如我们想让文本显示为该大小的 3/4，可以将其设置为 0.75 em。同理，如果想显示为一半，可以设置为 0.5 em。

Richard Rutter 就调整字体大小这一话题发表了一篇精彩的博客，地址为 http://clagnut.com/blog/348/。因此，我就不在这里浪费时间了，如果你想让用户能够缩放网站中的字体，建议你看一看这篇文章。另外，要更深入地了解与字体缩放有关的内容，请访问 CSS-Discuss 网站：http://css-discuss.incutio.com/?page=FontSize。

首先，将 body 规则修改为如下所示：

```
<style type="text/css">
body {font-family: verdana, arial, sans-serif;
font-size:1em;}
</style>
```

尽管这一修改在视觉上不会产生任何影响，但我们却为字体设置了可以调整的基准大小。

你可能已经注意到了，像 <h1> 到 <h6> 以及 <p>、 和 <li> 这样的常见元素，都具有相当（即非常）大的默认字体。如果页面中的内容较多，那么使用默认大小意味着用户必须频繁地滚动页面。长期的可用性测试告诉我们，滚动是浏览网页时最不招人喜爱的一个方面。而且，我也发现这些过大的默认字体会导致页面设计显得笨拙、不美观。但是，当使用成比例的 em 字体单位时，则可以简单地做到让全部字体都变小一号。而希望看到更大字体的用户也可以通过在浏览器中选择“查看→文字大小→较大（或其他选项，取决于浏览器）”，来获得更大的字体。

我们假设新的基准大小是 12 像素（从视觉的角度上看是这么大，但在设置字体大小时实际使用的单位则是百分比或 em）。那么，需要将 body 标签的字体大小设置为 0.75 em，这样段落中的文本就会变成浏览器为该段落指定的默认字体大小 16 像素的 75%，也就是 12 像素。

记住，当为后代元素添加样式时，它们会继承新的基准大小以及复合的成比例效果，即它们中的 1 em 等于 12 像素，而 0.75 em 等于 9 像素。

这种做法的可取之处在于，当我们为个别的选择符使用成比例的值（例如 em）指定字体大小时，可以针对主要的用户使用更加美观的较小的字体。与此同时，对于视力残障的用户来说，他们仍然有机会选择成比例地放大字体。

在下面的例子中，我们使用浏览器默认的 1 em 的字体大小。后面，当我们构建网站时，可以通过调整这个基准值来适应相应设计的需要。

以 1 em 的字体大小为基准，我们来设置每个元素的字体大

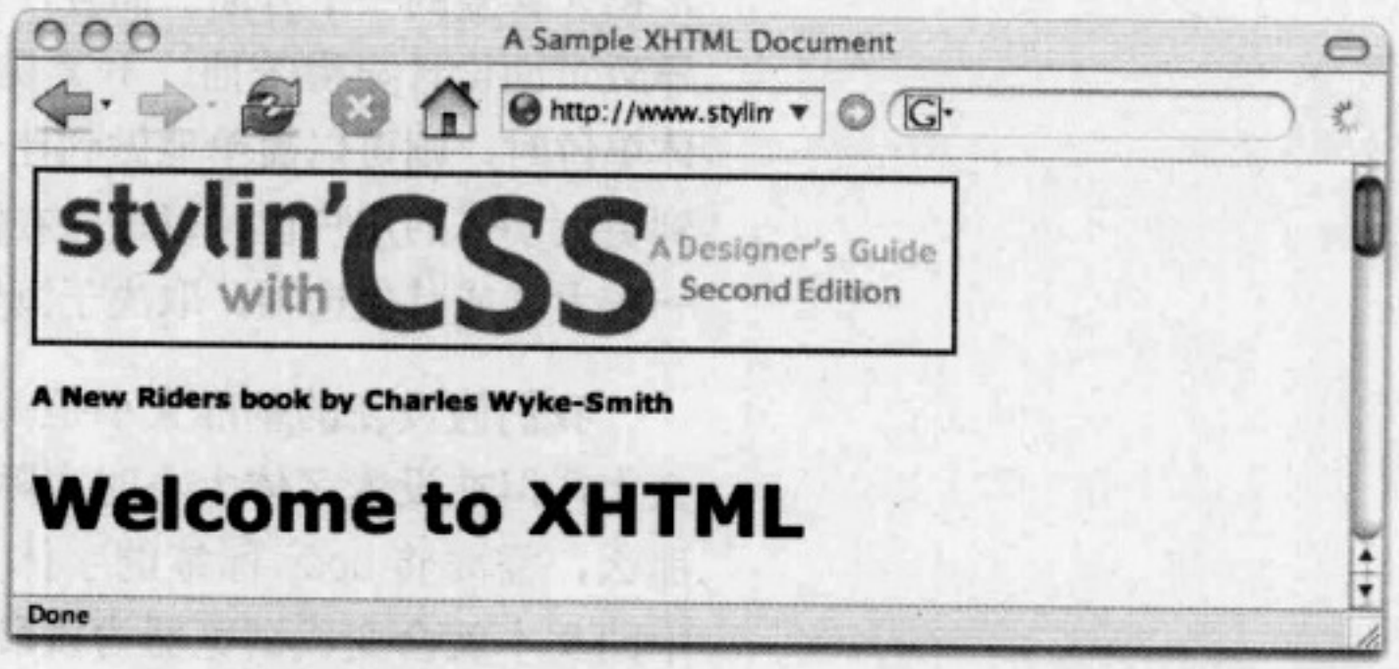
小，首先从位于标志下面的“a New Riders book...”这一行开始（图 3-8）。

这行文本位于 h3 元素中，这里我们把它设置为 0.8 em（之所以选择这个数值，是根据以前的经验，我知道这个值很合适）。下面就是修改后的规则：

```
<style type="text/css">
body {font-family: verdana, arial, sans-serif;
font-size:1em;}
h3 {font-size:.8em}
</style>
```

图 3-8 中显示了修改后的外观。

图 3-8　h3 标题中的文本变小了



我们注意到，这行标题文本相对于其原始的默认大小明显地缩小了（如果你感兴趣可以试验一下，我发现 h3 标题的默认大小是 1.2 em，或者说 16 × 1.2 = 19.2 像素）。

下面，我们继续为其他元素设置字体大小，结果如下所示：

```
<style type="text/css">
body {font-family: verdana, arial, sans-serif;
font-size:100%;}
h3 {font-size:.8em}
p {font-size:.8em}
ol {font-size:.75em}
ul {font-size:.75em}
</style>
```

修改后的页面外观如图 3-9 所示。

图 3-9 通过标签选择符，我们可以为文档中所有的标签指定字体大小。其中一些标签的样式是直接添加的，而另外一些则从父元素中继承了字体大小

与以上样式有关的两点注意事项：首先，我们没有为页面中两个列表的列表项（li）元素设置样式，但却为包含这些列表项的有序列表（ol）和无序列表（ul）元素设置了字体大小。如果直接为 li 元素添加样式，那么两个列表中的字体大小将保持相同，但由于这里是为 ol 和 ul 元素设置的样式，所以稍后我们还可以把列表项的字体大小设置为不同的值。

嵌套标签中的可继承样式

其次，尽管对于作为起点的未经样式化的布局来说，我们已经取得了不小的改进，但带项目符号的无序列表（ul）仍然显得太小，即使将它设置为同有序列表（ol）一般大也无济于事。

在IE 6中，相对字体大小的继承存在bug——参见http://css-discuss.incutio.com/?page=InternetExplorerWinBugs。

这个问题是因为将ul元素设置为0.75 em，而将嵌套在其中的a元素设置为0.7 em造成的。这样一来，位于嵌套的a元素中的文本的实际大小变成了0.525 em（0.7×0.75）。字体大小的继承可以为你所用，但正如这里所展示的，也可能为你带来问题。好在，解决这个问题也很简单——只需像下面这样设置一个上下文选择符即可：

```
<style type="text/css">
body {font-family: verdana, arial, sans-serif; font-
size:1em;}
h1 {font-size:1em}
h3 {font-size:.8em}
p {font-size:.8em}
ol {font-size:.75em}
ul {font-size:.75em}
ul a {font-size:inherit}
a {font-size:.7em}
</style>
```

这里，我们并没有为嵌套的a元素添加样式，只是让它从包含元素继承了字体大小，这里的包含元素也可能是p或者ol元素。对于ol元素中的链接，a会从li继承样式，而li会从ol继承样式。

通过以上两点注意事项，我们发现尽可能在最高层次的标签上设置字体大小，然后再沿着文档层次进行调整是最佳方式。换句话说，不要以那些深度嵌套的链接作为起点，而要从body标签开始向下设置字体大小。这样，不仅可以将需要编写的规则降至最少，也可以最大限度地发挥层叠机制的作用。

现在，我们可以测试一下这个基于em的布局所具有的可缩放性。

(1) 选择“页面→文字大小→最大”（IE 7）或“查看→文本大小→放大”（Firefox，多单击几次）。其他浏览器中也有类似的选项。结果是页面中的所有文本都能平滑地按比例放大，这对于视力残障的用户来说是非常方便的。

(2) 改变 body 选择符中 font-size 属性的值——例如，试试 80% 或 120%。保存修改并刷新页面。同样，所有元素中的文本大小会成比例缩放。如果你过去曾经因为字体过大或过小而在修改几十个页面中的数百个 FONT 标签的 size 属性时花费过数小时的时间，那么应该感激这种能力的强大和便利。当客户下次再提出意见说“你们这些设计者的问题就是总把字体设置得太小”时，你可以在 5 秒钟内把 body 标签也就是整个网站的字体大小修改为原来的 4 倍，然后客客气气地问“您觉得这样够大了吗？”

接下来，我们再讨论其他与字体相关 CSS 属性。

3.4 字体属性

调整字体大小的关键在于从视觉上表现文档中内容的层次关系。而通过理解各种字体属性，以及前面我们讨论的可以通过文档的结构层次被继承的字体属性，可以实现这一点。因此，下面我们分别介绍与字体相关的属性。

3.4.1 font-style 属性

例子：h2 {font-style:italic;}

其他值：normal、oblique

字体的样式用于设置字体是不是显示为斜体，就那么简单。如果想让一段文本显示为斜体，可以使用下面的规则：

```
p {font-style:italic;}
```

也可以用 oblique 代替 italic，但结果相同。

对于 font-style 属性来说，只有两个有用的设置：italic 可以使常规的文本变成斜体，而 normal 则可以使斜体类型的文本恢复为“直立”的文本。对这个例子来说，

```
p {font-style:italic;}
span {font-style:normal;}
<p>This is italicized type with <span>a piece of non-italic
text</span> in the middle.</p>
```

会得到如图 3-10 所示的结果。

图 3-10 为 font-style 设置 normal 值可以使位于斜体文本中的指定部分变成正常显示的文本

关于normal值的注意事项

normal 值会导致不应用某个属性时的效果。为什么要需要这样做呢？正如前面示范的在主文本中使用 font-style 的例子，通过设置 font-style:normal 能够使相关的文本保持常规状态，而不是斜体。提供这样一个选项的原因，就是让设计者能够有选择地覆盖默认的或在样式表中设置的全局属性。例如，标题 h1 到 h6 在默认情况下会显示为粗体，如果我们想让 h3 元素显示为常规文本，可以编写 h3 {font-weight:normal;}；如果为了让所有链接都显示为小型大写字母，而在样式表中声明了 a {font-variant:small-caps;}；而你想让一组特殊的链接显示为常规的大写或小写，那么就可以使用类似 a.speciallink {font-variant:normal;} 这样的声明。

3.4.2 font-weight 属性

例子：a {font-weight:bold;}

可能的值：100、200 直至 900，或者 lighter、normal、bold 和 bolder。

W3C 建议规范对这个属性的实现只规定了每个递增的值（无论是数字值还是“权重”值），都必须相对于前一个较低的值产生更高或相同的鲜明度。

bold 和 bolder 应该对字体给予不同程度的加粗。lighter 则在相反的方向上更进一步，它适用于在加粗的文本中“细”化某些文字。至少，意思没错。

图 3-11A ～ D 分别展示了在几种浏览器中的测试结果。

你能从这些截图中看到某个浏览器提供了两种以上的加粗效果吗？我也不能。我为此甚至试验了其他字体，结果也是无功而返。实际上，font-weight 的所有值只能产生两种不同的结果——加粗和正常。在展示各种类型的数据时，为文本应用不同的粗细变化，是一种有效地展示内容层次的方式。当我们通过中间层代码（例如 ASP 或 PHP）方便地生成不同的数字值，

并希望它们能够自动转换成不同的突出显示效果时尤其如此。然而，现实中的浏览器制造商们并不愿意实现更多的文本粗细效果，这是很令人失望的。通过测试结果，我们可以看出这里存在很大的改进空间。

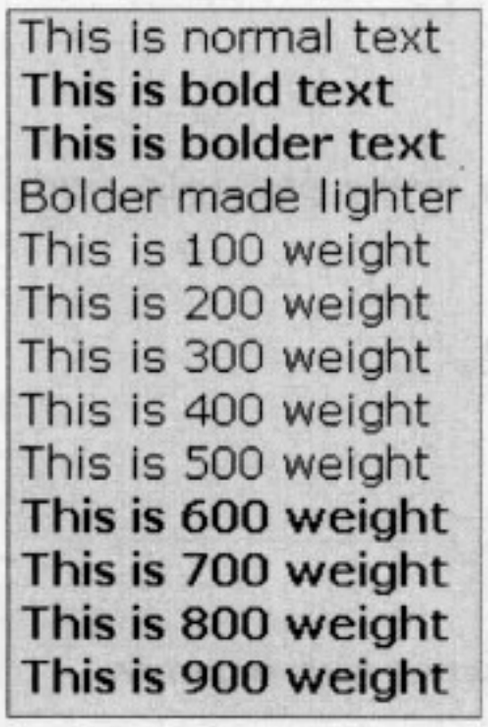


图 3-11A Windows 中的 Firefox

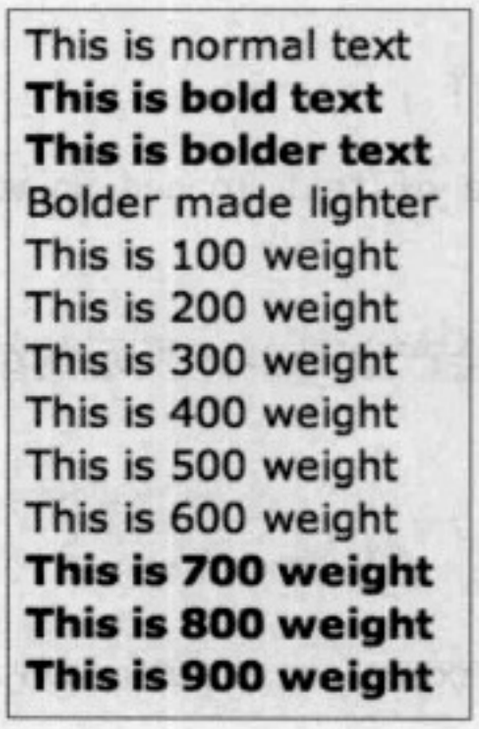


图 3-11B Mac 中的 Safari

图 3-11C Windows 中的 IE 6

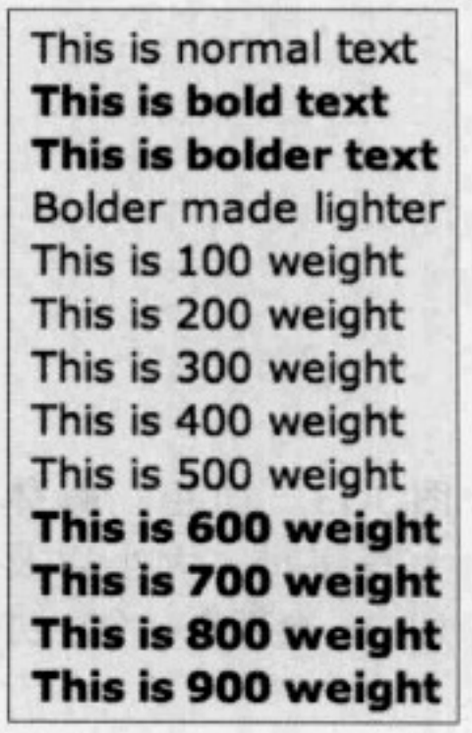


图 3-11D Mac 中的 Firefox

3.4.3 font-variant 属性

例子：blockquote {font-variant:small-caps;}

值：small-caps、normal

这个属性只接受一个值（除了 normal 之外），即 small-caps。这个值会使所有字母都转换成小型大写字母，比如：

```
h3 {font-variant:small-caps;}
```

会产生如图 3-12 所示的结果。

图 3-12 两个段落，其中一个显示为小型大写字母

我经常在 :first-line 伪元素（即使 IE 6 也支持它）中使用 small-caps，这样就可以将元素中的第一行文本指定为小型大写字母。一般来说，这种样式都会应用于段落上（参见第 2 章）。

不过，由于全部大写的文本不容易辨认（因为其中缺少小写字母的上下方笔画作为视觉提示），所以这种样式还是少用为好。

3.4.4　字体属性的简写方式

例子：p {font: bold italic small-caps .75em verdana, arial, sans-serif;}

```
<p>Here's a piece of text loaded up with every possible font property.</p>
```

上面的代码会产生图 3-13 所示的结果。

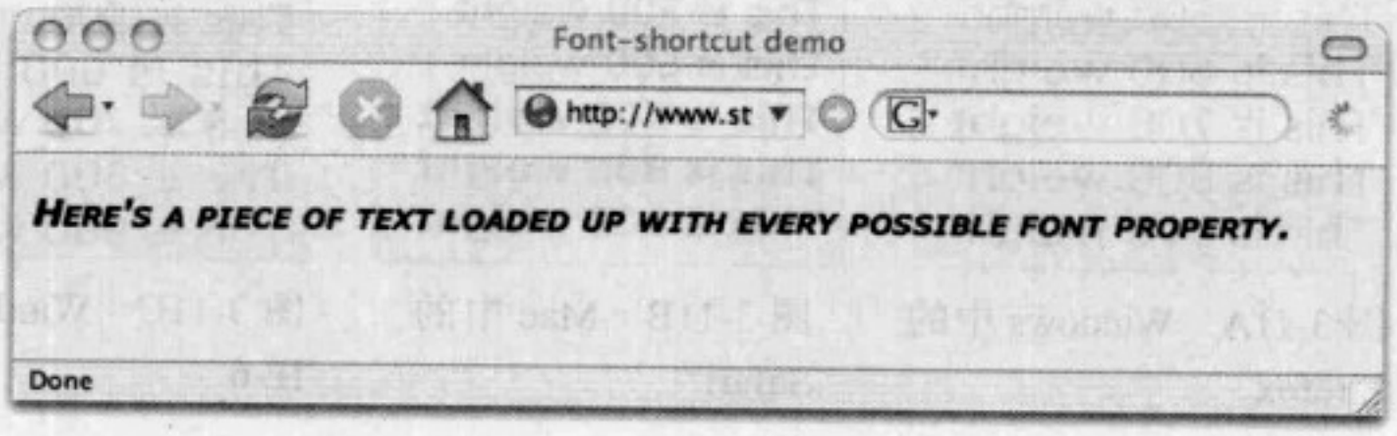


图 3-13　加粗、斜体、小型大写字母、大小以及字体系列——全部在一个声明中定义

提前介绍一下，我们也可以在 font-size 属性中包含 line-height 属性（这是文本属性，不是字体属性）。例如：12px/150%，按照印刷行业的说法，能够产生 12 像素的文本和 18 像素的行高。CSS 中的行高相当于印刷行业排版中的行间距。在下一节中，我们会介绍 line-height 属性。

通过使用 font 这个极好的简写方式，我们可以在一条声明中应用所有字体属性，这有助于减少完成目标样式所需的 CSS 代码量。但是，声明中属性值的顺序必须正确，以便浏览器正确地解释它们。

为此，请记住下面两条简单的规则。

规则 1：始终要保证声明 font-size 和 font-family 的值。

规则 2：这些值的先后顺序如下所述。

(1) 首先是 font-weight、font-style、font-variant——任意顺序；

(2) 然后是 font-size；

(3) 然后是 font-family。

3.5　文本属性

在学习了如何设置字体属性之后，我们再来看一看能够为文本添加哪些样式。假设要缩进一个段落，创建像 10^6 中的 6 —

样的上标，在标题的字母之间设置更大的间距，以及完成其他类型的格式化任务，都需要用到 CSS 中的文本属性。

在 CSS 中，有 8 种与文本相关的属性。

- text-indent。
- letter-spacing。
- word-spacing。
- text-decoration。
- text-align。
- line-height。
- text-transform。
- vertical-align。

认识“蛇”形文本

在 CSS 对文本的管理中，有一个非常重要的概念，即 CSS 会把一个盒子放到位于元素中的文本周围。例如，对于段落中的文本块而言，CSS 会把这些文本看成装在盒子中的细长的文本条，这个文本条可以适应容器的大小而跨越多行。为形象地表达这个概念，我们在图 3-14 中为包含元素（即容器段落）添加了红色边框，而为文本盒子添加了绿色边框。

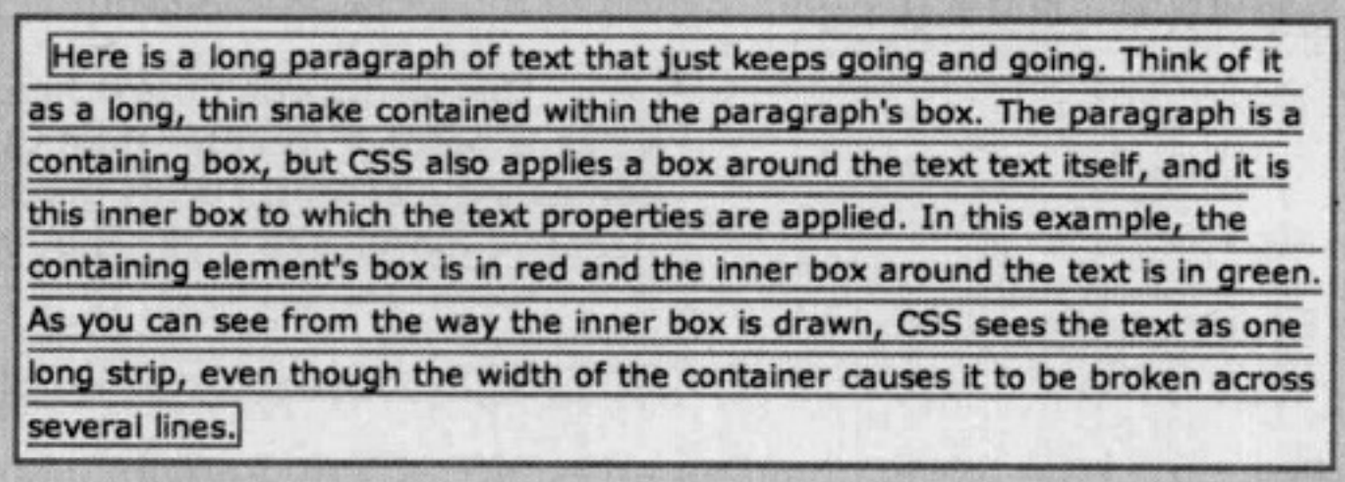


图 3-14 文本会被包含在一个细长的盒子中，而且通常会跨越多行（另见彩插）

在这个例子中，我为文本添加了如下标记：

```
<p><span>This is a long paragraph…(etc.)</span></p>
```

并应用了下列样式：

```
p {border:2px solid red;}
span {border:2px solid green;}
```

我们注意到，文本盒子跨过了多行，而且只在第一行的开头和最后一行的结尾是闭合的。

明白了这个概念可以确保更快地设置好想要的文本样式。例如，为了缩进段落第一行的文本（如图 3-14 的第一行所示），可以使用文本属性 text-indent，该属性可以移动文本盒子的开始位置。由于在 CSS 中，段落的后续文本只是一个长长的文本条，所以都不会缩进。

如果我们想缩进整个段落，则需要为段落设置 margin-left 属性；换句话说，需要把整个容器向右侧推动。对此，需要牢记的就是，所有文本属性都应用到一条长长的、细细的、像蛇一样的内部文本盒子上，而不是应用到包含元素的盒子上。

3.5.1 text-indent 属性

例子：p {text-indent:3em;}

值：任何长度值（正值或负值）

因为前面已经提到过了，所以我们就从 text-indent 属性开始。这个属性用于设置文本盒子相对于包含元素的开始位置。正常情况下，文本盒子的左边缘（对于多行文本而言，是第一行的开头位置）与容器的左边缘处于同一位置。

如果为 text-indent 设置正值，那么文本盒子会向右侧移动，从而创建首行缩进的段落效果（图 3-15）。

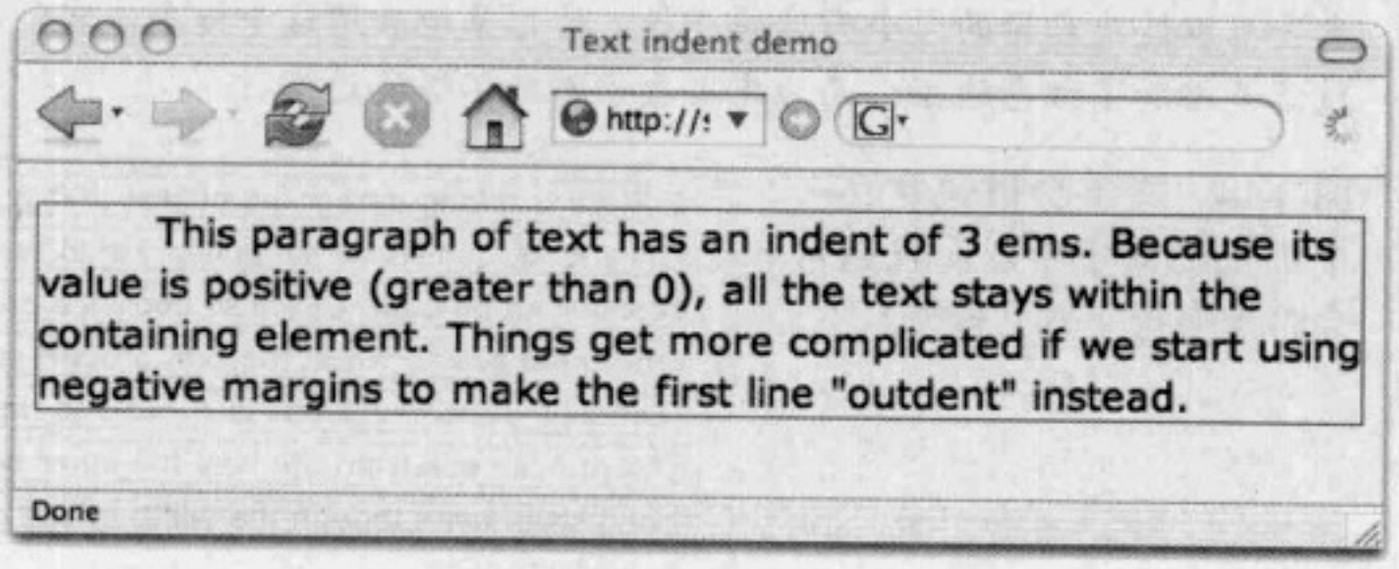


图 3-15 通过为 text-indent 属性设置正值可以为段落创建首行缩进效果（另见彩插）

在这些例子中，为了清晰起见我们为段落添加了红色的边框——通常，这些边框是不存在的。

但是，如果为 text-indent 设置负值，那么段落的第一行文本则会向左伸出包含元素之外。特别是在使用负外边距的情况下创建这种负缩进效果，一定要格外注意——在这种情况下，由于第一行文本实际上会伸出容器之外，所以必须确保有足够的空间容纳伸出的文本。如果包含元素的左侧紧挨着另一个元素，那么伸出的文本就会与后者重叠，或者如果它就位于浏览器的边缘，那么第一行文本会被截掉（图 3-16）。

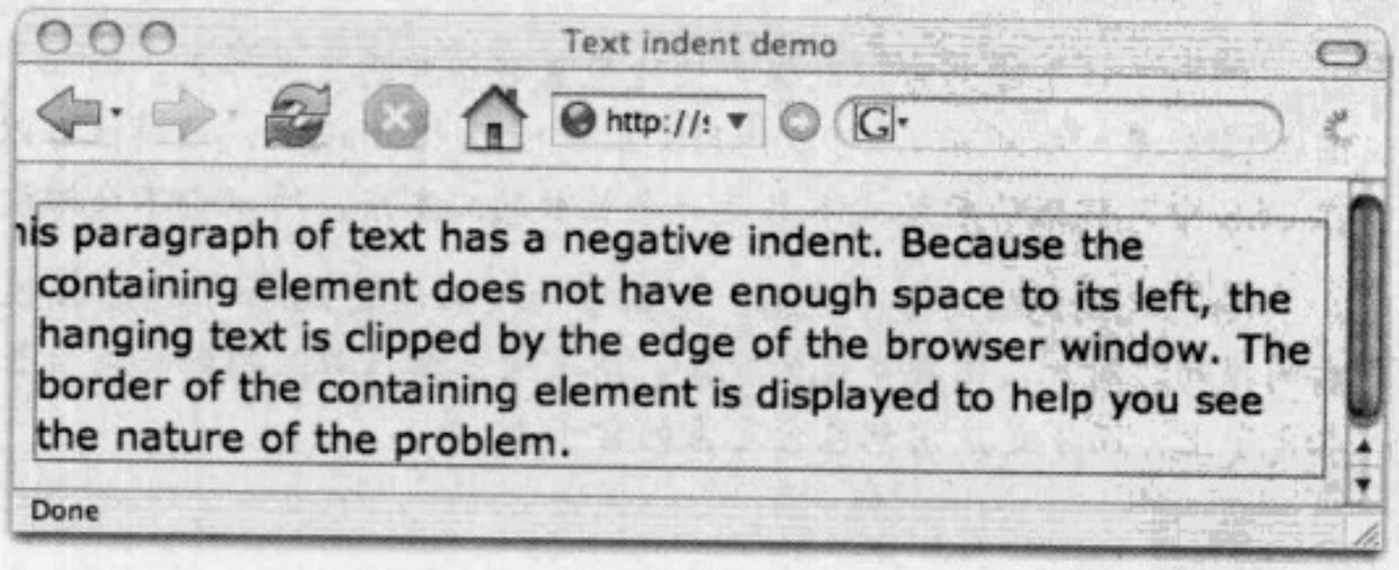


图 3-16 这个段落设置了首行的负文本缩进效果，但由于没有设置相应的左外边距值，导致了伸出的文本被截掉了（另见彩插）

要避免这个问题，需要为段落指定一个大于负缩进值的正的左外边距。在图 3-16 中，负的缩进值为 –1.5 em，但在图 3-17 中，也设置了 2 em 的左外边距。

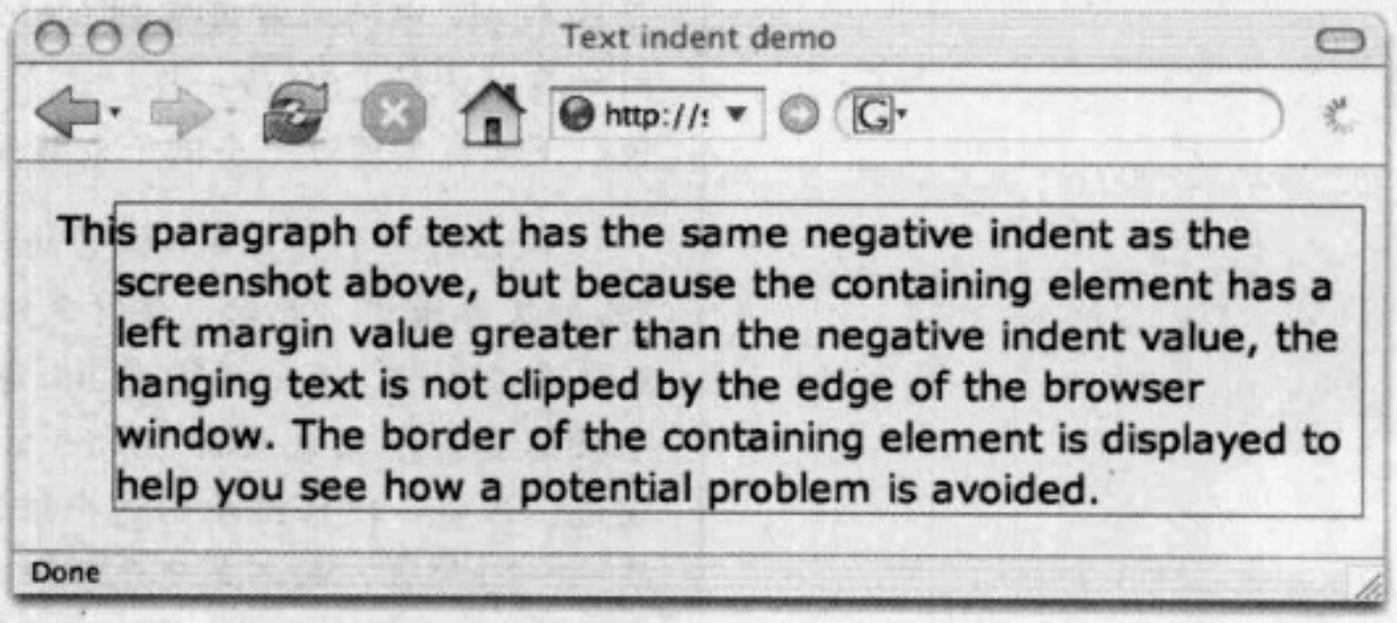


图 3-17 这个段落设置了负的文本缩进，同时也设置了相应的左外边距，从而为伸出的第一行文本创造了足够的空间（另见彩插）

生成图 3-17 所示结果的样式规则如下：

```
p {text-indent:-1.5em; margin-left:2em; border:1px solid red;}
```

悬挂段落（即首行伸出包含盒子之外的段落）会使文本产生经过专业设计的外观，同时还能为读者提供清晰的视觉入口点。

以 em 为单位设置缩进及相应的外边距是个好习惯，因为当用户（或者你）改变文本大小时，缩进能够与行的长度保持成比例的变化。对悬挂缩进来说，成比例变化能够确保为伸出的文本创造足够的空间，无论用户把字体放大到多少倍也没有问题。

在空间紧张但你又不想在段落间保持距离的情况下，可以把段落的上、下外边距的值设置为 0，然后通过缩进或者负缩进而不是垂直方向的间距来清晰地表示每个段落的起点位置。

继承的值和计算的值

这里需要格外注意的一点是：text-indent 属性会被子元素继承。例如，如果我们为一个 div 设置了 text-indent 属性，那么位于这个 div 中的所有段落都会继承这个 text-indent 的值。不过，同所有继承的 CSS 值一样，这个值也不是定义的值，而是计算的值。下面我们通过一个例子来解释什么是计算的值。

假设有一个宽为 400 像素的包含文本的 div，我们为它设置了 5% 的文本缩进。此时，实际的缩进值就是 20 像素（400 的 5%）。在这个 div 中，有一个宽为 200 像素的子元素 p。作为子元素，这个段落会继承父元素的 text-indent 值，因此段落中的文本也会缩进。但是，段落继承的这个缩进值是基于父元素计算之后的结果——20 像素，而不是父元素中定义的 5%。结果，即使这个段落的宽度只有父元素的一半，它的第一行也会缩进 20 像素。这样可以确保无论 div 中的段落有多宽，都会具有相同的缩进量。当然，通过为子元素明确设置不同的 text-indent 值可以覆盖这一行为。

3.5.2 letter-spacing 属性

例子：p {letter-spacing:.2em;}

值：任何长度值（正值或负值）

这个属性可以产生印刷品设计者称之为“缩排”（tracking）的效果，即调整字母间的距离。正值可增大字母间距，而负值则减少字母间距。对这个属性，我强烈推荐使用相对值，例如 em 或百分比，不要使用绝对值，比如像素，这样，就会在用户改变字体大小时保持字母间距会成比例地变化。图 3-18 给出了这个属性的实际效果。

缩排（tracking）在印刷术语中是指应用到文本块的字母间距。而紧排（kerning）则是调整两个字符之间距离的术语。

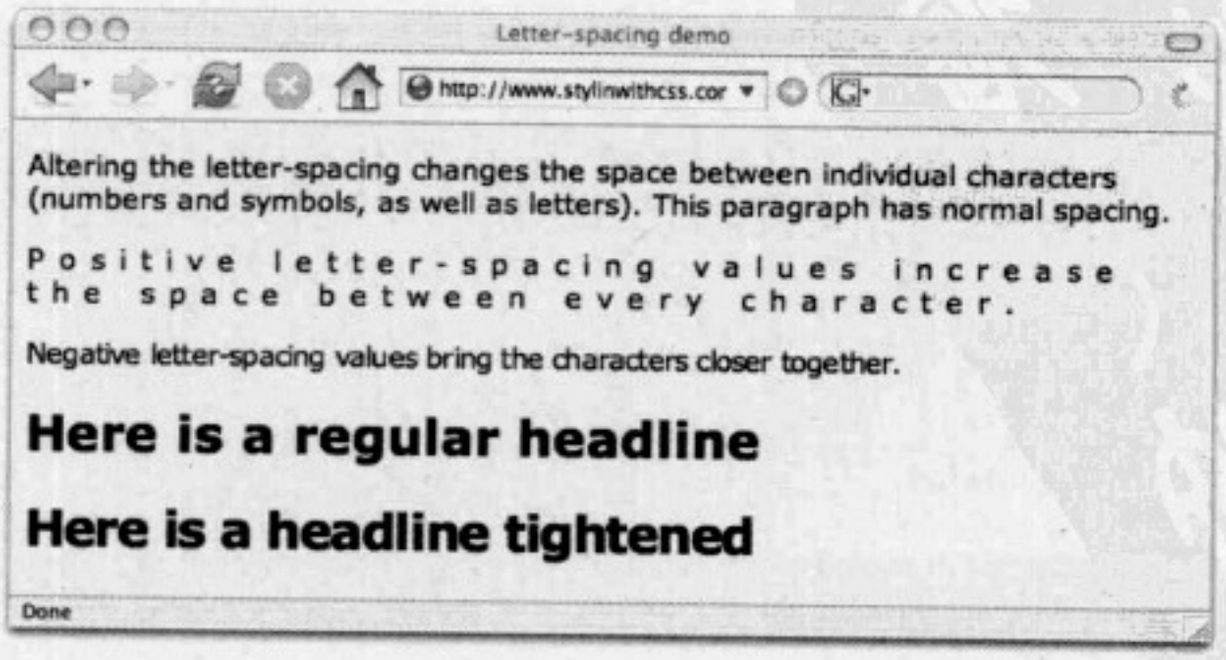


图 3-18 在这个例子中，可以看出修改字母间距属性会对文本的外观带来什么样的影响

默认的字母间距会随着文本的增大而越来越松散。如图3-18所示，通过缩排标题文本，可以给人更专业的感觉。

通常，无需修改页面中主体文本的字母间距——不过，对这个问题的看法可能会因人而异。比如，可以通过将页面文本设置得比常规形态更紧凑或更松散一些，使页面更有个性。不过要注意，过紧或过松都会导致文本难以阅读的问题。以图3-18所示的文本和标题为例，我只从这些字符之间移除了0.05 em（1 em的1/20）的空间，再大一些，恐怕这些字母就会相互重叠了。

3.5.3 word-spacing属性

例子：p {word-spacing:.2em;}

值：任何长度值（正值或负值）

当把字母间距调整得较宽时，会造成单词间距难以区分。此时，正是加大一点字间距的好机会。

字间距同字母间距非常类似，只不过（如你所料），它改变的是单词之间而不是字母之间的距离。首先，需要明确的一个问题是，CSS将空格分隔的任何字符或字符组都看成是一个单词。其次，字间距甚至比字母间距更容易过度使用，从而导致难以阅读的文本（图3-19）。

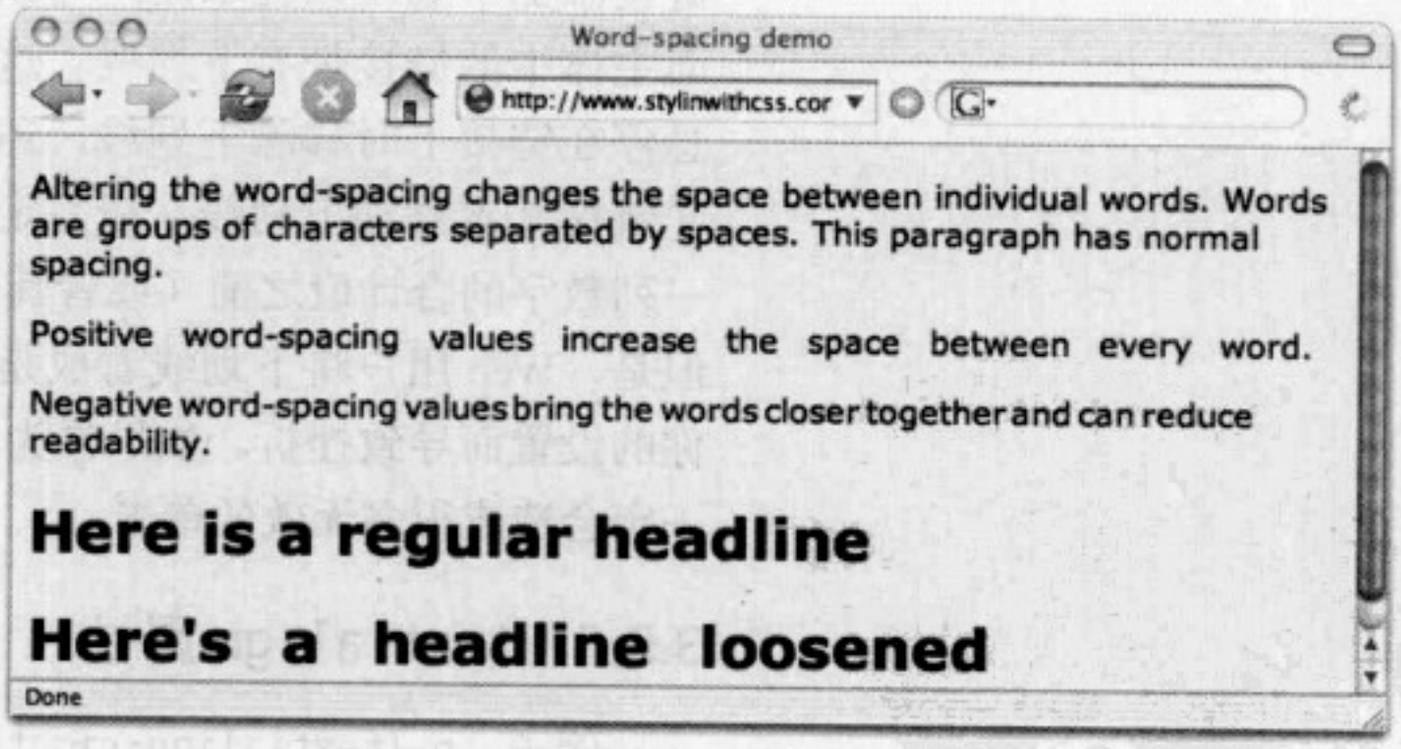


图3-19 字间距是容易使用过度的属性之一

3.5.4 text-decoration属性

例子：.retailprice {text-decoration:line-through;}

值：underline、overline、line-through、blink

在图 3-20 中可以看到这个属性的效果。不过，文本的装饰（decoration）并不是圣诞树上的小铃铛；而是指为文本添加的下划线、上划线、删除线及闪烁效果（不过，千万不要使用闪烁效果，因为这种效果实在太令人讨厌）。

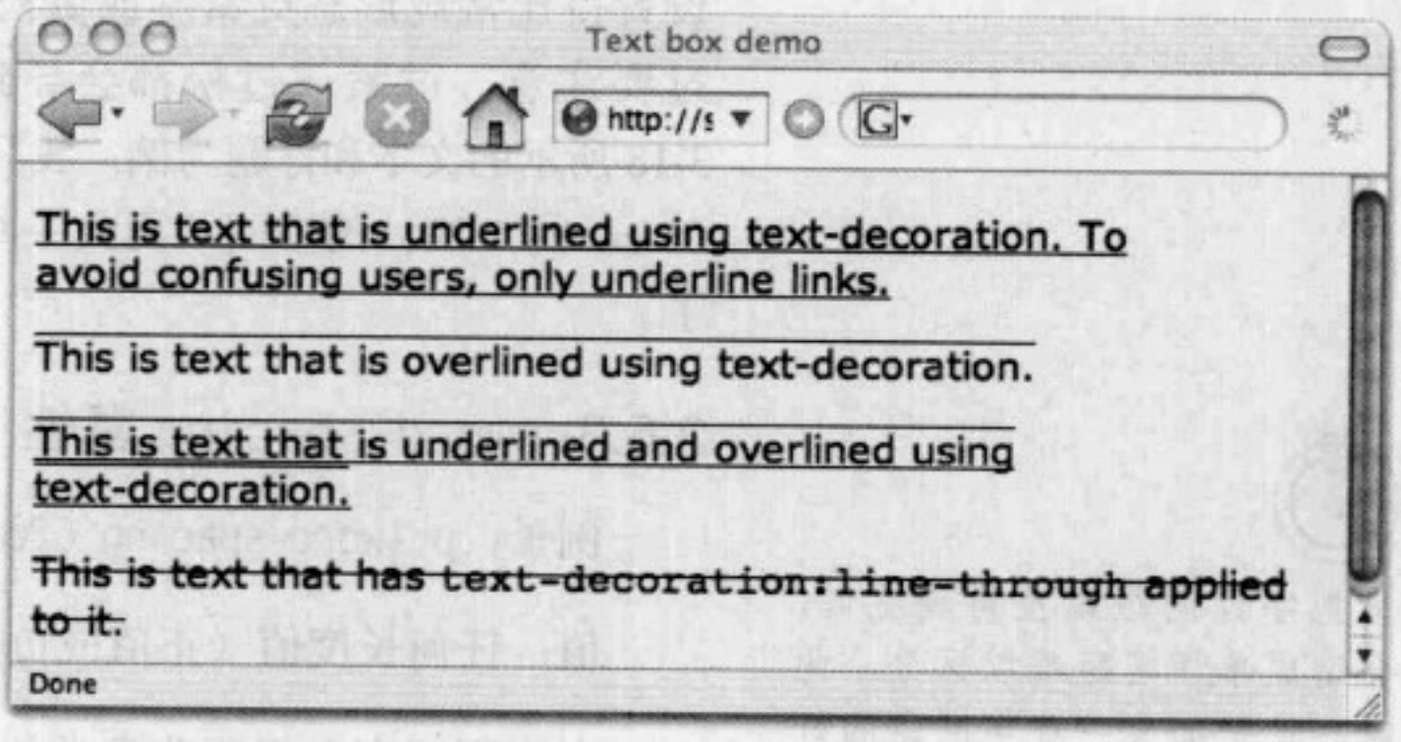


图 3-20　这幅图像展示了各种值的效果，但其中用处最大的文本装饰还是控制链接的下划线

文本装饰属性的主要用途是控制链接文本的下划线。例如，侧边栏中链接的位置和组织，使用户一看就知道它们是链接。在去掉默认的下划线的情况下，这些链接不仅更加容易阅读，而且看起来也更美观；当用户把鼠标放到链接上时，则可以为链接添加下划线以提示用户可以单击。相反，你可能只想让页面主体中的链接带有下划线。这种感觉不错，如果当用户鼠标悬停在链接上时移除下划线，还能够增强易读性。如果你想为非链接文本添加下划线，请一定要仔细考虑清楚。你也许会在一列数字的合计值之前（或者其他类似地方）添加一条下划线，但是，Web 用户将下划线看成是链接的普遍心理会使他们因为你的设置而导致挫折。如果你为不是链接的文本添加了下划线，一定会造成很多无效的单击。

3.5.5　text-align 属性

例子：p {text-align:right;}

值：left、right、center、justify

这个属性的值只有 4 个：left、center、right 和 justify。文本会在水平方向上与包含元素对齐，因此必须在包含元素上设置这个属性；换句话说，如果想让 div 中的 h1 标题居中对

齐，需要在 div 而不是 h1 中设置 text-align。图 3-21 展示了这 4 种 text-aling 值的效果。

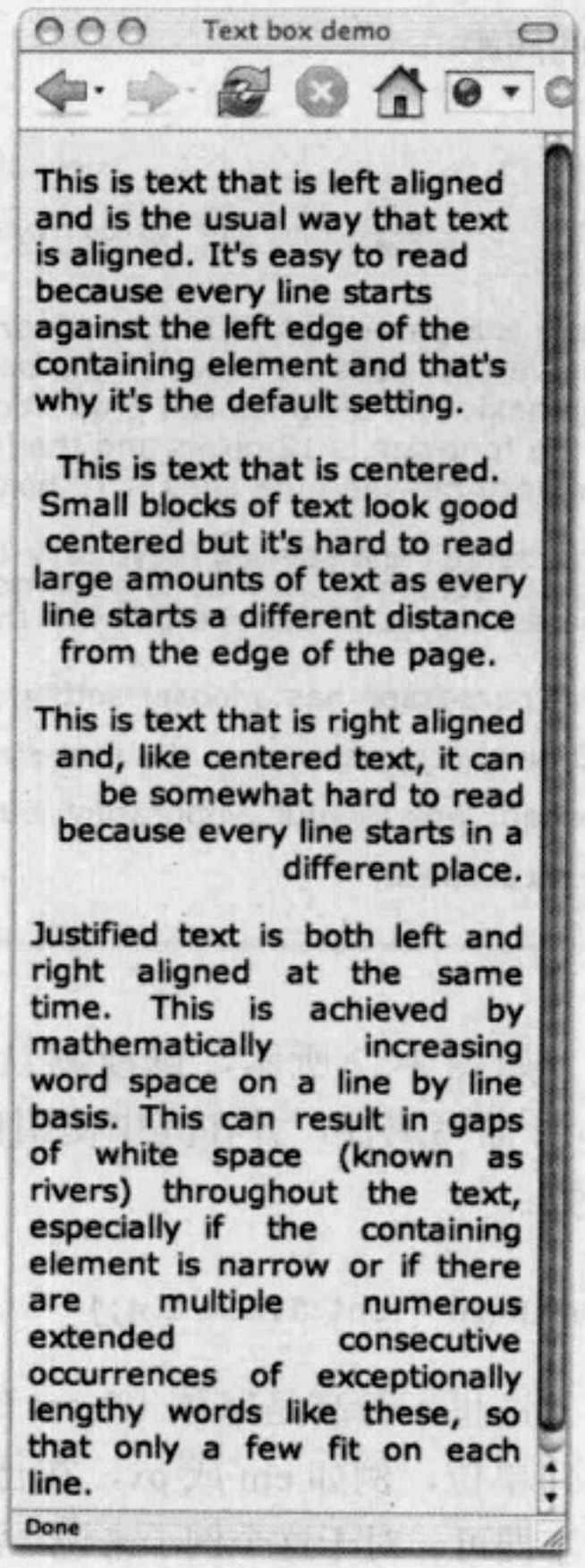


图 3-21 在 Mac 平台的 Firefox 中查看 4 种 text-align 值的效果

3.5.6 line-height 属性

例子：p {line-height:1.5;}

值：任何数字值（不需要指定单位）

CSS 中的 line-height 相当于印刷品设计中的行间空铅（金属铅）。行间空铅可以在文本块中创建行间距。行间空铅的高度不是指行与行之间空白的高度，而是指行与行基线之间的距离。为了保证文本的可读性，行间空铅要比铅字高一些，以便行与

行间空铅（leading）这个印刷术语，来源于用来隔离铅字构成的文本行的铅条。

行之间保持一定距离。在默认情况下，浏览器会将 line-height 设置为字体大小的一定比例——根据我的测试，通常是字体大小的 118%。因此，无论字体大小，行与行之间都会保持相同比例的间距。

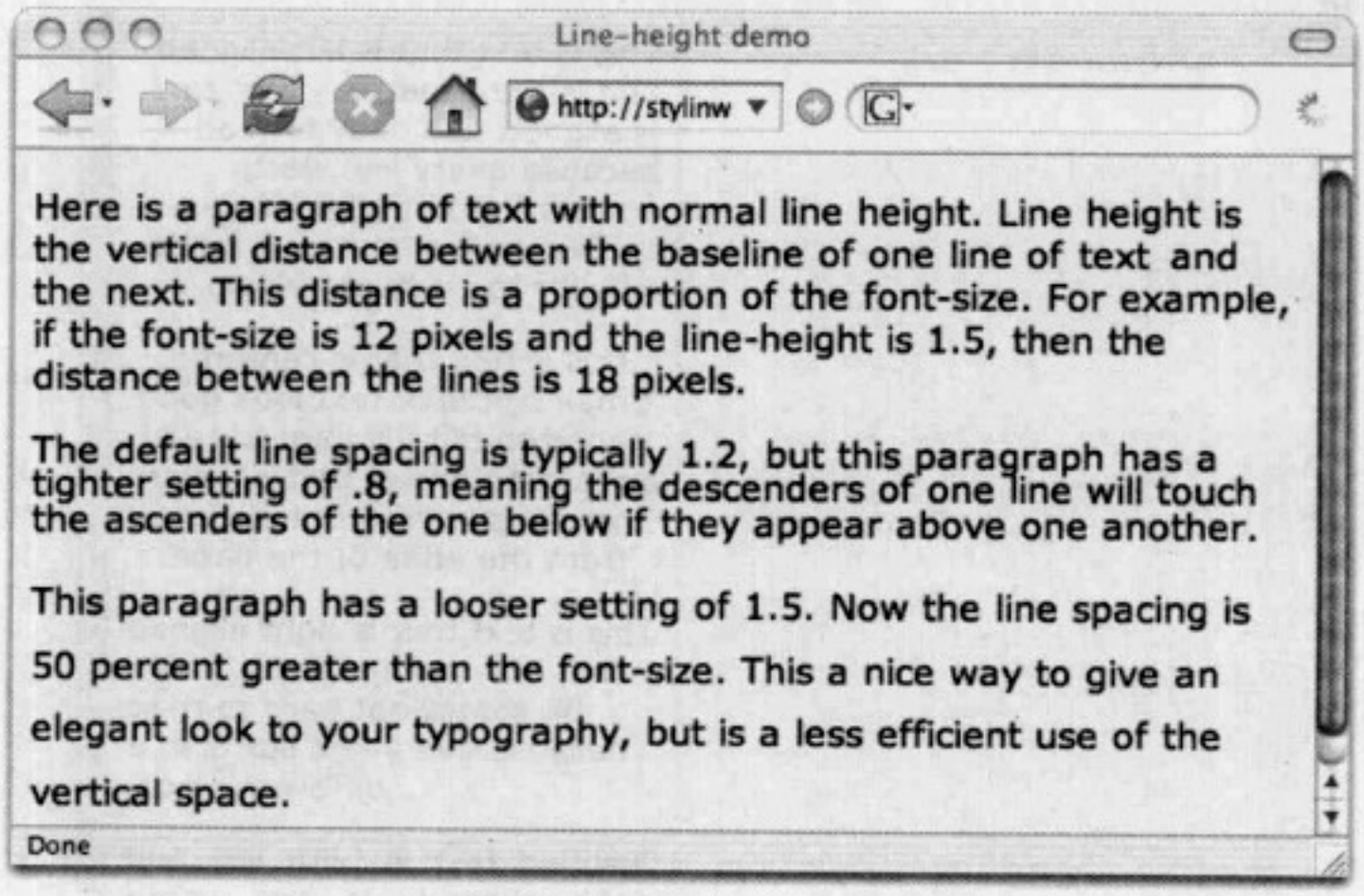


图 3-22 使用不同的行高值，是使网站更有特色的一种方式

如图 3-22 所示，修改默认行高的最简单方式就是使用 font: 简写属性，并在其中使用包括字体大小和行高的复合值。例如：

```
div#intro {font:1.2em/1.4;}
```

这里，行高是字体 1.2 em 的 1.4 倍。注意，行高值不需要任何单位，例如 em 或 px，在行高值的部分，只需要指定一个数字即可。对于这个例子来说，浏览器会根据计算得到的 1.2 em 代表的屏幕上的像素数，再乘以 1.4 得到行高的像素数。如果把字体大小修改为 1.5 em，那么行高就会是计算得到的 1.5 em 代表的像素数的 1.4 倍。如果为行高值指定了固定的单位（如 px），那么增大字体大小有可能会导致行与行相互重叠。

需要指出的是，行高比文本高出的部分，会在文本的上方和下方平均分配。下面，我们以像素为单位指定行高来简单地说明这个问题。尽管前面我曾多次提到像素不是指定行高的最佳方式，但在这个例子中使用像素会更容易理解。假设字体大

小为 12 像素，而行高为 20 像素，那么浏览器会在文字的上方和下方各加上 4 像素，因为 12 + 4 + 4 = 20。按照常规，由于在多行文本段落中，行与行之间会有 8 像素的空白，所以人们不容易注意到。但是，这个分配法则可能会对文本的第一行和最后一行比较有价值——在这两行的上方或下方分别只有 4 像素的空白。

单行文本的垂直居中

为了在包含元素内垂直居中元素，你可能会很自然地想到使用 vertical-align 属性，但是这个属性却会让你感到失望。不过，在包含元素内垂直居中单行文本则是可行的。在 CSS 中，可以通过把行高设置为等于包含元素的高度来实现（至少是对单行文本的）垂直居中效果。由于多出的行高会被平均分配到文本的上下方，所以文本最终会被垂直居中。

但是，在包含元素内垂直居中一个文本块（例如 p 元素），几乎是不可能的。即使能够做到，也需要使用额外的 div 元素。与其对此作出长篇大论的解释，不如建议大家看一篇名为“*Vertical Centering in CSS*”（CSS 中的垂直居中）的文章，地址为 www.jakpsatweb.cz/css/css-vertical-center-solution.html。虽然能够做到垂直居中文本块，但却没有那么轻松。有时候，我们真的会为 CSS 规范没有提及这个基本的功能而感到难以理解。

3.5.7 text-transform 属性

例子：p {text-transform: capitalize;}

值：uppercase、lowercase、capitalize、none

text-transform 会改变元素中文本的大小写形式。通过这个属性可以将一行文本中所有单词的首字母变成大写、所有文本变成大写及所有文本变成小写。图 3-23 展示了这些效果。

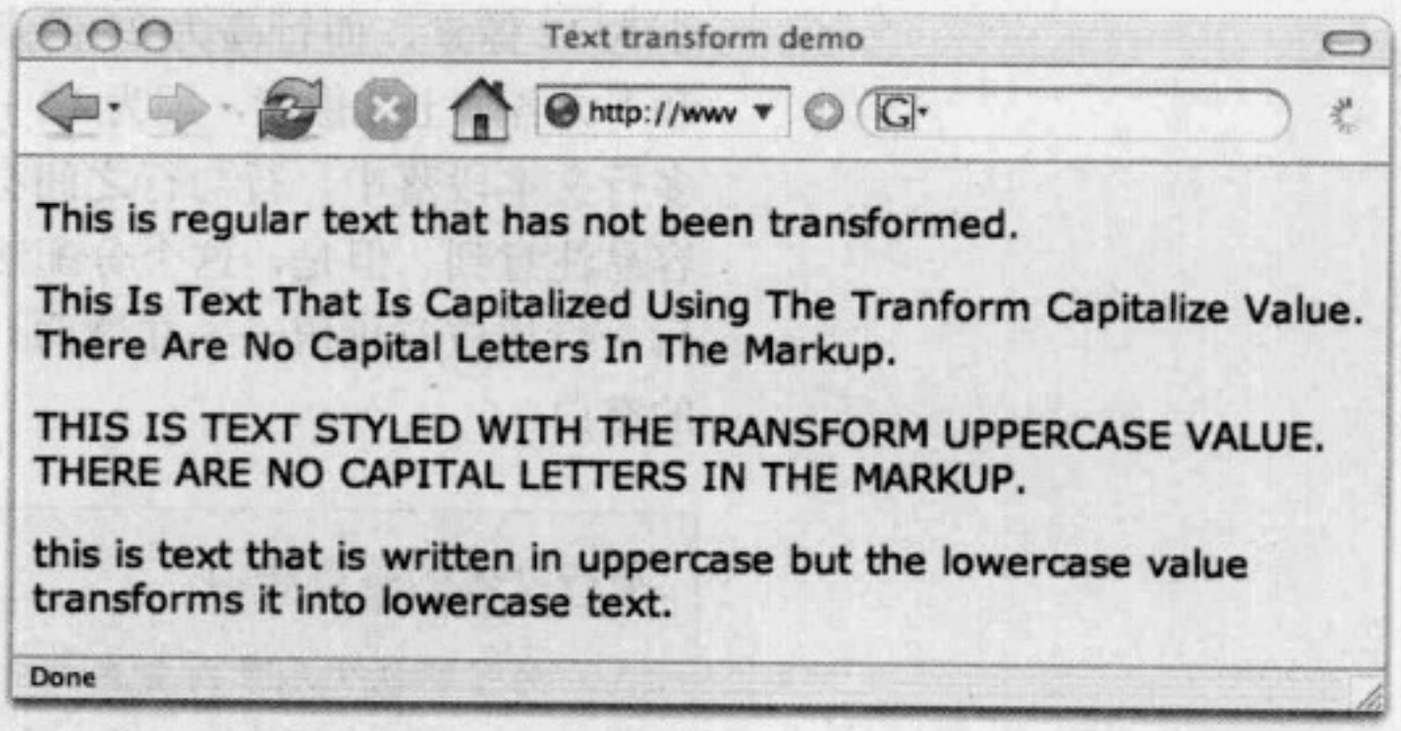


图 3-23 通过 text-transform 可以为网页添加类似报纸标题的效果

capitalize 会将每个单词的首字母变成大写。因此，可以用它来模仿广告、报纸或杂志中的标题。但是，人类一般不会对辅助性单词，如 of、as 及 and 的首字母给予大写。以“Tom and Jerry Go to Vegas.”为例，CSS 会将它转换成“Tom And Jerry Go To Vegas.”。不过，这称得上是一种不错的标题效果，而且，如果这些文本内容来自数据库或者 XML，那么通过 CSS 无需修改标记就可以实现这一效果。

要实现小型大写字母的效果，可以使用 font-variant 属性。另外，也可以考虑使用较小的负 letter-spacing 值使标题视觉上更加紧凑（参见 3.5.2 节）。

3.5.8 vertical-align 属性

例子：vertical-align:60%

值：任何长度值，及 sub、sup、top、middle、bottom

vertical-align 可以相对于基线将文本向上方或下方移动。举例来说，这个属性一个最常见的用途就是将公式或数学表达式中的数字转换为上标或下标，例如 x^4–y^{-5} 或 N_3O。另外，将文本中的星号或其他标记转换为脚注，也是这个属性的一种典型应用。我不太喜欢多数浏览器中默认的上标或下标样式——它们设置的字体过大、过高（或对于下标来说，是过低）。如图 3-24 所示，只需使用一点样式，就可以实现更具有专业水准的体验。

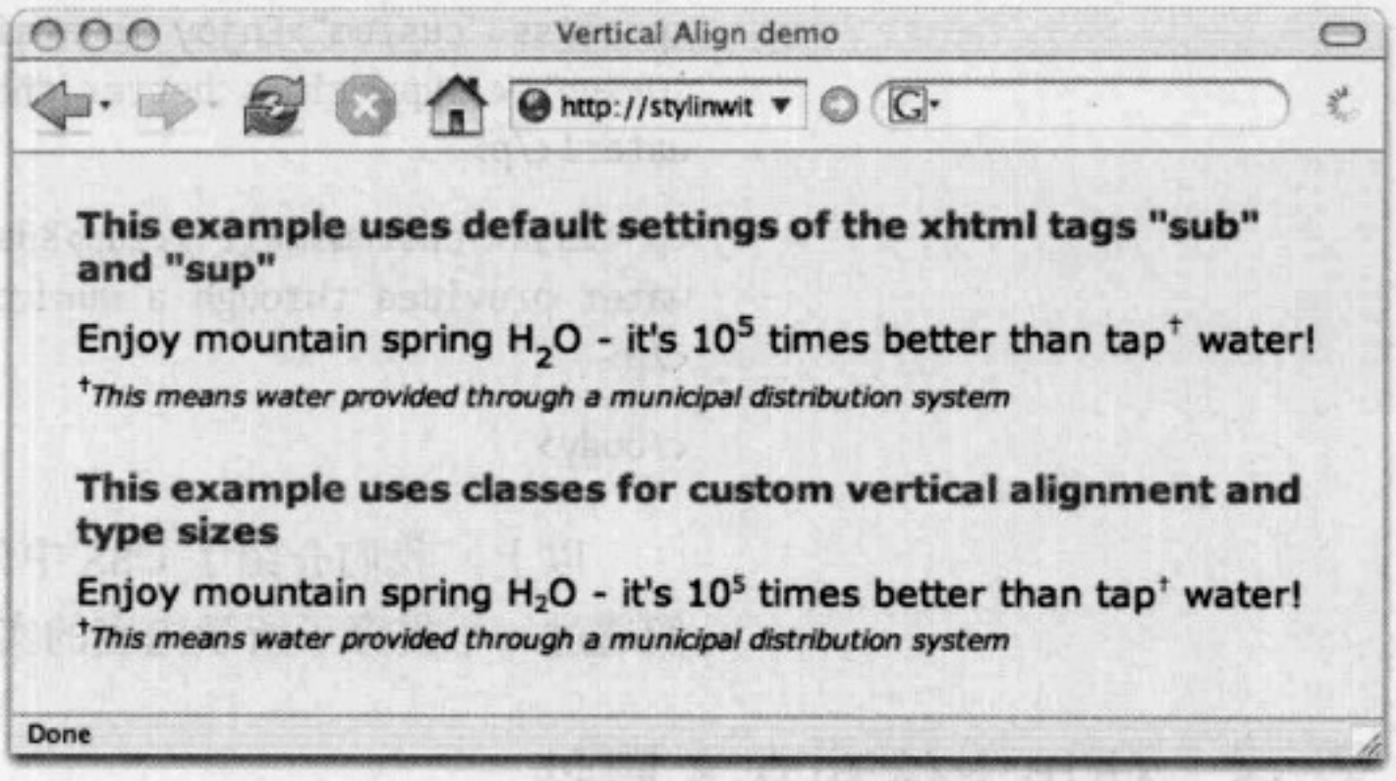


图 3-24 上标和下标可以相对于标准的基线变换文本的垂直位置。我在自己的样式表中改进了这些标签的默认样式，效果如图所示

尽管 XHTML 中的 sup 和 sub 标签能够自动创建上标和下标文本，但还是有必要使用 vertical-align 及 font-size[①]来定义更美观的结果。

IE 7 好像很难处理上标和下标样式，会在这些字符周围生成较大的空白。

下面就是相应的代码：

```
<style type="text/css">

body {font-family:verdana, arial, sans-serif; font-
size:100%;}

h4 {margin: 1.4em 20px .5em; color:#069;}

p {margin: 0 20px;}

p.custom sub {vertical-align:-.25em; font-size:65%;}

p.custom sup {vertical-align:.6em; font-size:65%;}

p.customsmall {font-size:.8em; vertical-align:1em}

</style>

</head>

<body>

<h4>This example uses default settings of the xhtml tags
"sub" and "sup"</h4>

<p>Enjoy mountain spring H<sub>2</sub>O - it's 10<sup>5</sup>
times better than tap<sup>&dagger;</sup> water!</p>

<p><sup>&dagger;</sup><em>This means water provided through a
municipal distribution system</em></p>

<h4>This example uses classes for custom vertical alignment
and type sizes</h4>
```

① 原文 text-size 有误。——译者注

```
<p class="custom">Enjoy mountain spring H<sub>2</sub>O - it's
10<sup>5</sup> times better than tap<sup>&dagger;</sup>
water!</p>

<p class="customsmall"><sup>&dagger;</sup><em>This means
water provided through a municipal distribution system</em>
</p>

</body>
```

以上，我们介绍了 CSS 中的字体和文本属性。接下来，我们通过一个贯穿了所学内容的实例来结束本章。

3.6 使用字体和文本属性

我们仍然以第 1 章的标记为例，通过应用本章学习的字体和文本属性，将一个普普通通的页面转换成更具专业水准的设计作品。图 3-25 中显示了未添加样式的页面外观。

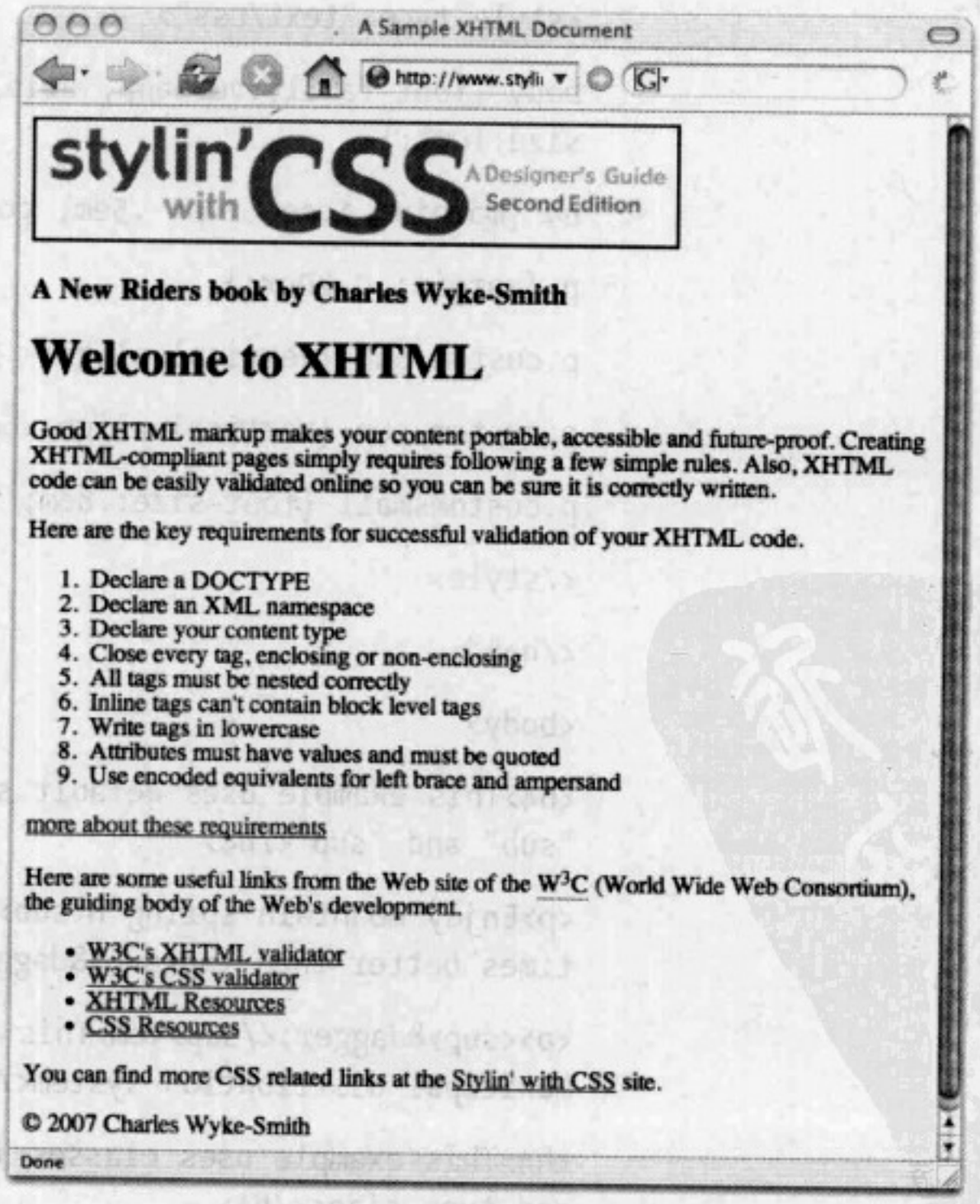


图 3-25 这是我们在第 1 章中看到的未添加任何样式的页面

通过为这个页面只应用本章学到样式，加上 margin 和 background-color 属性，页面在转眼之间就变成了一个精致的设计作品（图 3-26）。

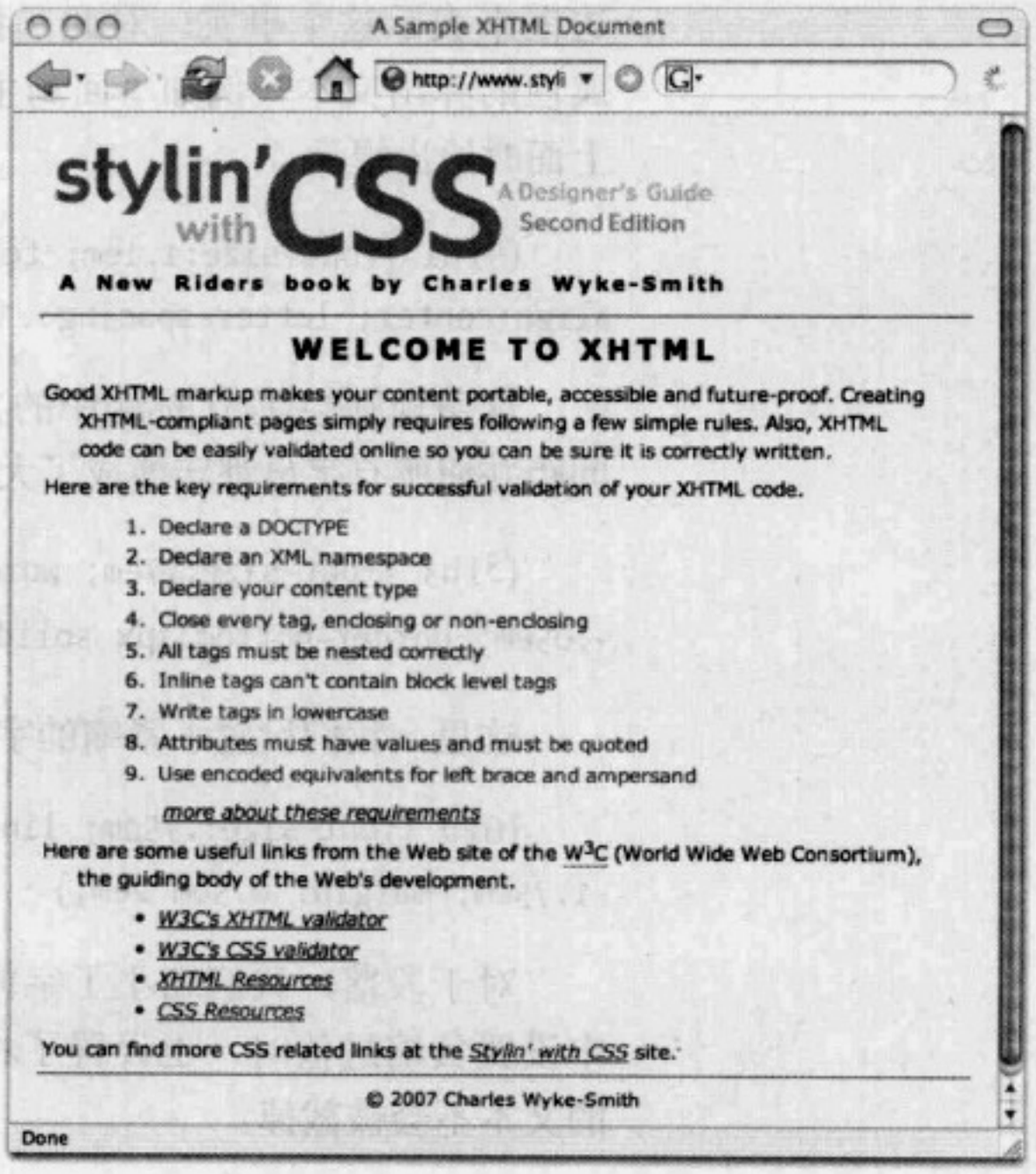


图 3-26 应用了样式的页面看起来更具有吸引力

以下是为页面应用的样式。

(1) `* {margin:0; padding:0;}`

首先，我们先“撤销”了所有元素上默认的外边距和内边距。这样，就去掉了未样式化的页面中很多垂直方向上的空间。通过移除所有默认的外边距和内边距，可以只让明确设置该属性的元素拥有它们。

(2) `body {font-family:verdana, arial, sans-serif; font-size:100%; margin:1em; background-color:#DFE;}`

这条规则设置了字体和页面外边距的基准样式。其中，font-family 和 font-size 属性会被所有元素继承，而 1 em 的外边距则会使所有元素都从页面边界向内移动相应距离。这样，页面中的元素就不会碰到浏览器窗口了。

(3) `img {border:0;}`

如果我们把一幅图像放到一个链接中使其可以单击，那么除了 Safari 之外的所有浏览器都会在图像四周添加一条难看的边框来表示这个事实。对此，我更喜欢移除边框并通过 title 属性的弹出文本（例如“回到主页”），在用户鼠标悬停在图像上面时给出提示。

(4) `h1 {font-size:1.1em; text-transform:uppercase; text-align:center; letter-spacing:.2em; margin:.5em 0;}`

这条规则会使 h1 标题中的文本水平居中，同时增大了字母间距并使所有字母都转换成了大写形式。

(5) `h3 {font-size:.7em; word-spacing:1em; letter-spacing:-.05em; border-bottom:1px solid #069; padding:0 0.5em 1em;}`

这里，我们设置了紧缩的字母间距和宽松的字间距。

(6) `p {font-size:.75em; line-height:1.4em; text-indent:-1.75em; margin: 0.5em 2em;}`

对于段落，我们缩小了字体并增大了行高。在为第一行文本设置负缩进值时，也设置了相应的外边距，以保证伸出左侧的文本不会被截掉。

(7) `ul, ol {font-size:.75em; margin-left:6em; line-height:1.25; color:#444;}`

较大的左外边距可以缩进列表。同时，为了强调列表也增大了行间距。

(8) `#contentarea a {margin-left:6em;}`

将“More about...”链接向右移到并与列表对齐，会使它们看起来更舒服。

(9) `a {color:#036; font-style:italic;}`

所有链接都将以深蓝色和斜体字显示。

(10) `a:hover {color:#069; text-decoration:none;}`

当处于悬停状态时，链接的颜色会变浅，同时会移除下划线。

(11) `acronym {border-bottom:1px dotted; cursor:default;}`

为了表示首字母缩写词在翻转状态时会出现工具提示条，我们为它添加了虚下划线（实心下划线表示链接）。此外，通过改变光标的指针，可以鼓励用户在相应文本上停留足够长的时间，以便弹出工具提示条。

(12) `#homepagefooter {border-top:1px solid #069;}`

在页脚上方添加一条蓝色的水平线。

(13) `#homepagefooter p {font-size:.7em; text-align:center; text-indent:0em; border-top:1px solid #069; padding- top:.5em;}`

最后，我们设置了页脚中文本的大小，将这些文本居中，为文本上方添加了一条边框，并为了将页脚与内容分开，为文本上方添加了内边距。值得注意的是，这里我们移除了前面样式中通过低针对性的 p 选择符声明的 text-indent 属性，以免继承该属性值。

尽管对于一个页面来说，完成这些样式好像需要很大工作量，但是不要忘了一旦把这些样式放到样式表中，就可以将它们应用到链接到该样式表的任何页面。因此，通过为一个页面设计样式，就可以使整个网站受益。

在理解了 CSS 原理和设计文本样式之后，我们就可以讨论与多栏页面布局有关的内容了。

3

第4章

定位元素

在采用 Web 标准的过程中，一个关键性步骤就是放弃将表格作为页面布局的手段。表格的用途从来就不是页面布局，而是通过网格展示数据——与 Excel 电子表格的方式类似。然而，在 CSS 开发之前，表格经常被用来创建页面网格，以便将页面中的元素组织成分栏。这样，就意味着在标记中使用令人讨厌的变通方法（例如添加空白 GIF 图、换行符以及非换行空格）才能创建出期望的布局。有了 CSS，就可以对 XHTML 元素进行更加精确的定位，并且不用在标记中添加表现元素。

通过使用外边距、内边距、边框等 CSS 属性，以及浮动和清除等 CSS 技巧，能够实现与过去相同（甚至更好）的设计。与此同时，不仅能够保持标记的简化和整齐，还能在相似的布局元素间共享编写的样式。最终得到的是轻量级的、容易看懂的代码。

利用这些技巧能够取得多大的成功，取决于对盒模型、position 属性以及 display 属性的理解程度。盒模型描述的是标记中存在的每个元素的定位控制。position 属性定义了这些元素在页面上的位置关系。display 属性定义了元素在页面上是堆叠还是并排，甚至是否出现。下面我们依次介绍这些概念。

4.1 理解盒模型

要了解与盒模型相关的更多信息，请参考 http://www.w3.org/TR/RECCSS2/box.html。

我们在标记中创建的每个元素都会在页面上生成一个盒子，因此 XHTML 页面实际上就是盒子排列的结果。

在默认情况下，每个元素盒子的边框是不可见的，而且盒子的背景也是透明的，所以我能够理解你看不到盒子的惊讶。使用 CSS，可以轻松地为页面中的盒子改变边框和背景颜色。这样，你就能够从一个全新的角度来观察页面的结构了。

比如，以上一章末尾添加了样式的部分页面为例，我们可以看到为元素盒子添加边框后的效果（图 4-1）。

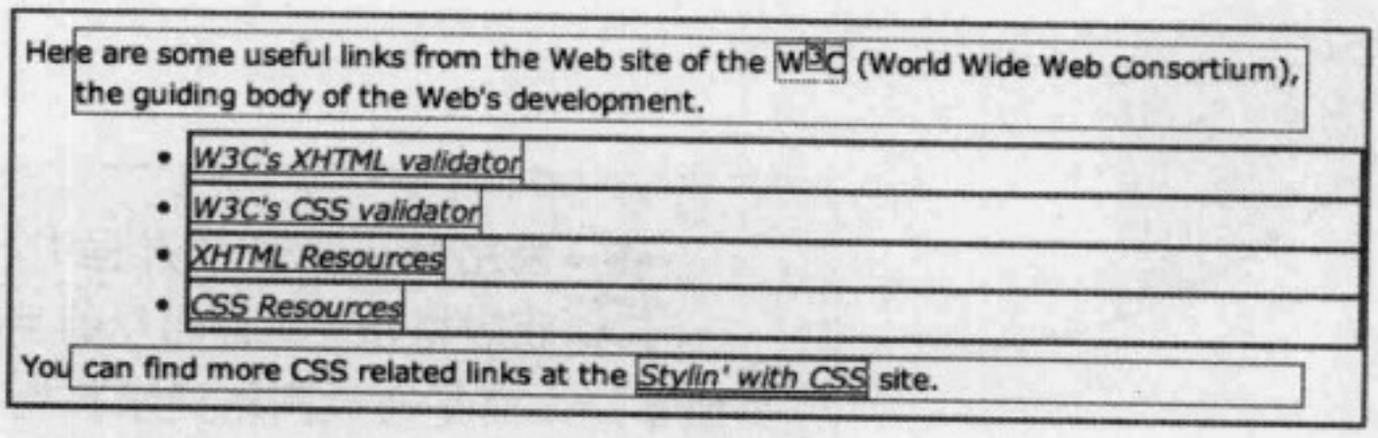


图 4-1 在添加了边框的情况下，我们可以看到行内链接元素的盒子“收缩包装”着它们的文本，而块级列表项的盒子则向右伸展直至填满它们的包含元素（除非它们的外边距把它们与包含元素强制隔开）。而且，段落上的负缩进值，使得首行文本向左伸出了元素盒子之外

在这个例子中，我们看到的是浏览器以默认方式自上而下排列块级和行内元素的效果。如果想创建比浏览器默认的 XHTML 布局更有意思的布局，必须理解如何控制元素盒子的外观和位置。第一步就是理解盒模型（图 4-2），其中定义了构成每个盒子的属性。

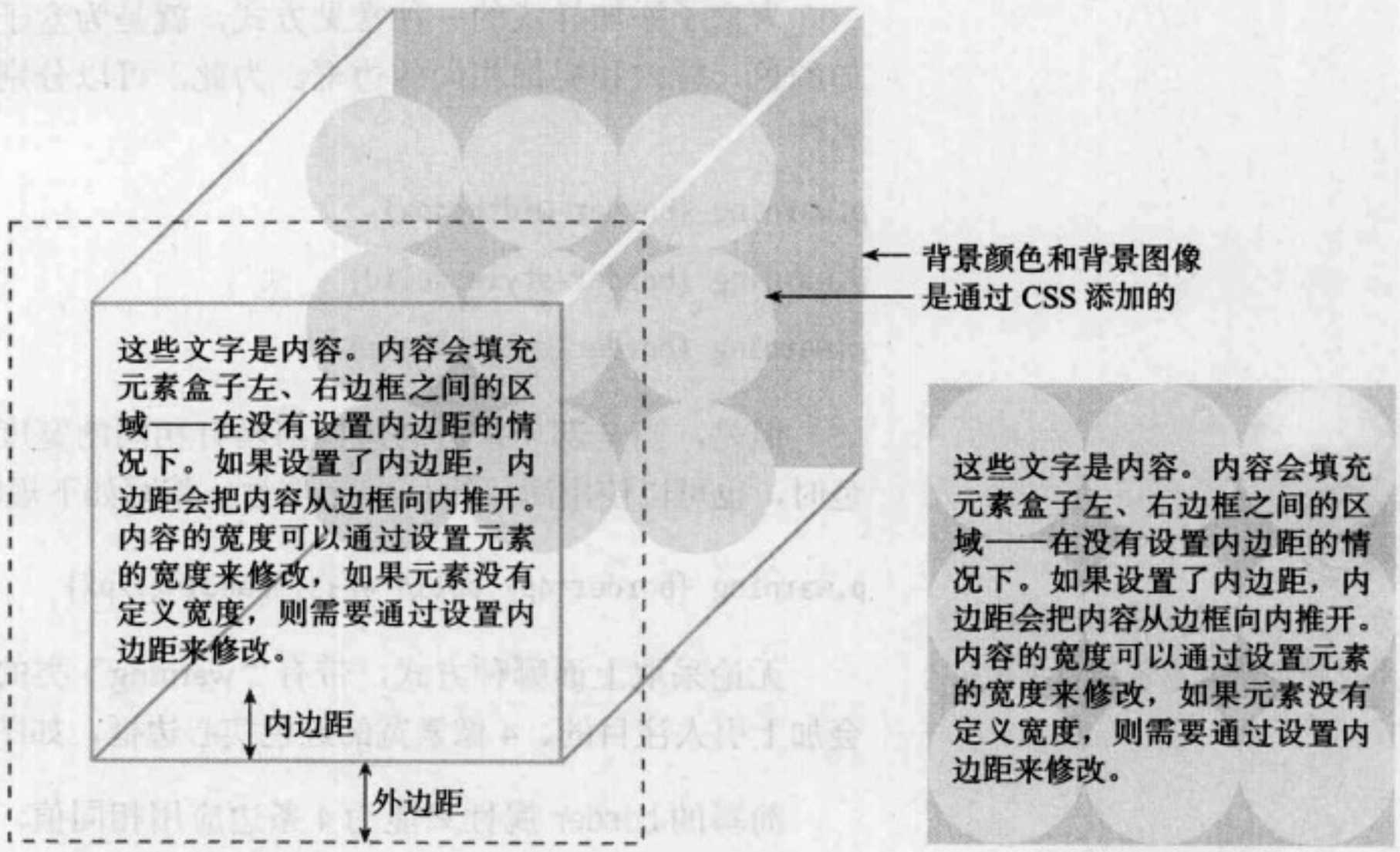


图 4-2 这幅盒模型示意图展示了一个 XHTML 元素的外边距、边框和内边距之间的关系。在盒模型中，前景（通常指文本或图像）在 XHTML 标记中定义，而背景的颜色或图像则只能通过 CSS 来添加（另见彩插）

使用 CSS 可以调整元素盒子的 3 个方面。

- 边框（border）：设置边框的粗细、样式和颜色。
- 外边距（margin）：设置盒子与相邻元素之间的距离。
- 内边距（padding）：设置盒子的内容与其边框之间的距离。

可以把这几个属性简单地理解为：外边距会从边框开始向外推开相邻的元素，内边距会从边框开始向内推开元素中的内容。因为盒子共有 4 条边，所以与外边距、边框及内边距相关的属性也分别有 4 个：top、right、bottom 和 left。

4.1.1 盒子的边框

与边框相关的属性有 3 个。

- 宽度（width）：相应的值包括 thin、medium、thick 或者任何长度单位（em、px、百分比等）。
- 样式（style）：相应的值包括 none、hidden、dotted、dashed、solid、double、groove、ridge、inset 和 outset。
- 颜色（color）：可以是任何颜色值（例如，RGB、十六进制或关键字）。

CSS 规范中没有具体规定 thin、medium 和 thick 应该是多宽，因此这几个关键字的实际宽度可能会因浏览器而异。另外，边框的样式（除了 solid 之外，因为它只是不间断的实线）也没有在 CSS 规范中明确定义，所以虚线（dashed）可能会在不同的浏览器中呈现不同的线条和间距。

为盒子添加样式的一种常见方式，就是为盒子的4条边添加颜色、样式和粗细相同的边框。为此，可以分别指定这3个属性：

```
p.warning {border-width:4px}
p.warning {border-style:solid}
p.warning {border-color:#F33;}
```

但是，当全部4条边的边框都具有相同的宽度、样式和颜色时，也可以使用简写的border属性，编写如下规则：

```
p.warning {border:4px solid #F33; padding:2px}
```

为了避免文本与边框接触，这里也添加了2像素的内边距。

无论采取上面哪种方式，带有“warning”类的所有段落都会加上引人注目的、4像素宽的红色实心边框，如图4-3所示。

简写的border属性只能为4条边应用相同值。不过，如果想为各个边指定不同的值，也很容易。假设我们想为4条边都添加红色实心边框，但想让右侧和下方的边框更细一些，以便产生视觉差异。那么，必须使用两条规则来完成这个效果：第1条规则使用border属性来为4条边指定公共的样式，第2条规则使用border-width属性来指定不同的边框宽度。

```
p.warning {border:solid #F33; padding:2px;}
p.warning {border-width:4px 2px 2px 4px;}
```

结果如图4-4所示。

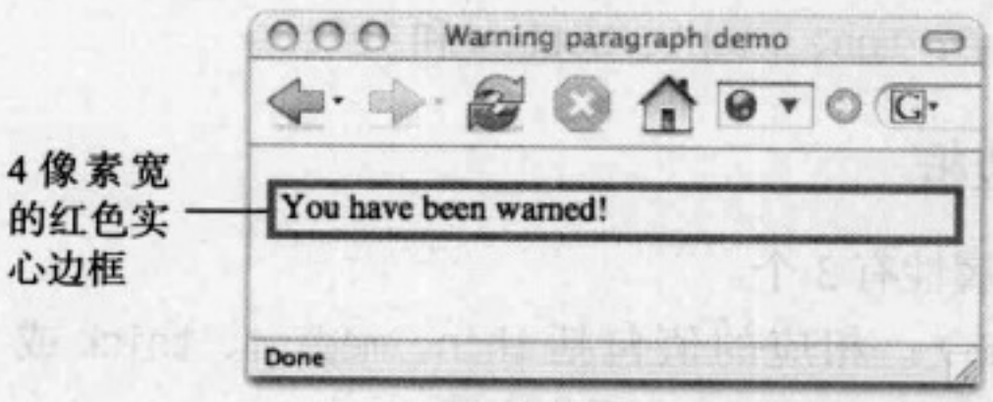


图4-3 这里，因为我们希望4条边的边框相同，所以可以使用简写的border属性。少量的内边距可以避免文本与边框接触

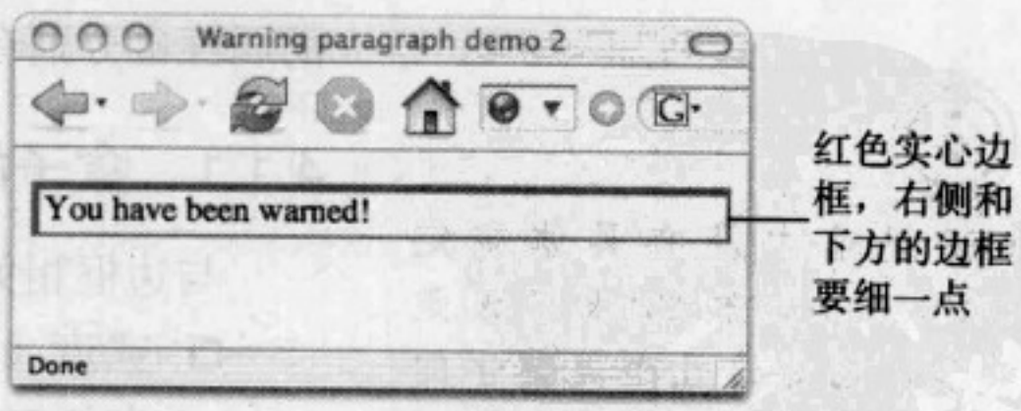


图4-4 通过把盒子的样式分成两条规则来定义，盒子的边框既具有公共（颜色和内边距）的样式也具有差异化的样式（线宽）

在开发过程中，临时性地显示盒子的边框非常有好处，因为这样我们就可以清楚地看到外边距或内边距等样式的效果。在默认情况下，元素盒子边框的默认值分别为：边框宽度

(border-width) 设置为 medium、边框样式 (border-style) 设置为 none、边框颜色 (border-color) 设置为 black。由于默认把边框样式设置为 none，所以盒子的边框不会显示。换句话说，显示段落盒子边框的最快捷方式就是使用以下规则：

```
p {border:solid;}
```

这条规则通过把边框样式设置为实线导致盒子的边框会显示出来，因为边框的颜色和宽度已经在默认样式中定义了。不过，需要注意的是，由于边框会增大元素的尺寸，因此添加边框会改变布局。根据相应元素在布局中的位置不同，这可能会也可能不会导致问题。显示盒子的另一种替代方案就是为盒子添加背景颜色，但盒子的尺寸不会因此而改变。

简写样式

为元素的 4 条边分别编写样式（无论是指定外边距、内边距，还是边框）都是非常令人讨厌的。CSS 为指定这些样式提供了简写方式，即在一条声明中逐个列出 4 条边的值。在简写方式的声明中，为盒子 4 条边指定值的顺序必须是上、右、下、左。可以将这个顺序记忆为 TRouBLe[①]，或者更形象化地按照逆时针方向来指定值。因此，如果我们想为一个元素指定外边距，就可以将

```
{margin-top:5px; margin-right:10px; margin-bottom:12px; margin-left:8px;}
```

简写为

```
{margin:5px 10px 12px 8px;}
```

注意，每个值之间只有一个空格，并且也不需要添加其他分隔符（比如逗号）。此外，也不是必须指定全部 4 个值——如果少指定一个，那么相应边就会取得对边的值。比如：

```
{margin:12px 10px 6px;}
```

在这个例子中，由于没有指定最后一个值（左边），因此会使用右边的值，即左侧也会添加 10px 的外边距。对于

```
{margin:12px 10px;}
```

这个只指定了前两个值（上边和右边）的规则来说，未指定的下和左边也会分别取得 12px 和 10px 的值。

最后，如果只指定了一个值：

```
{margin:12px;}
```

那么，所有边都会使用这个值。

使用这种简写方式，也可以只指定下边和左边的值，而不为上边和右边指定值（实际上是提供 0）。在下面的例子中，我们为上边和右边提供了不带单位的 0：

```
{border:0 0 2px 4px;}
```

① TRouBLe 是麻烦的意思（trouble），TR 和 BL 分别表示 Top、Right、Bottom、Left。——译者注

4.1.2 盒子的内边距

内边距是指盒子内容与盒子边框之间的距离。由于内边距位于盒子的内部，所以它也会带有为盒子的背景指定的颜色。图 4-5 中展示了两个段落，其中一个带内边距，另一个则不带内边距。

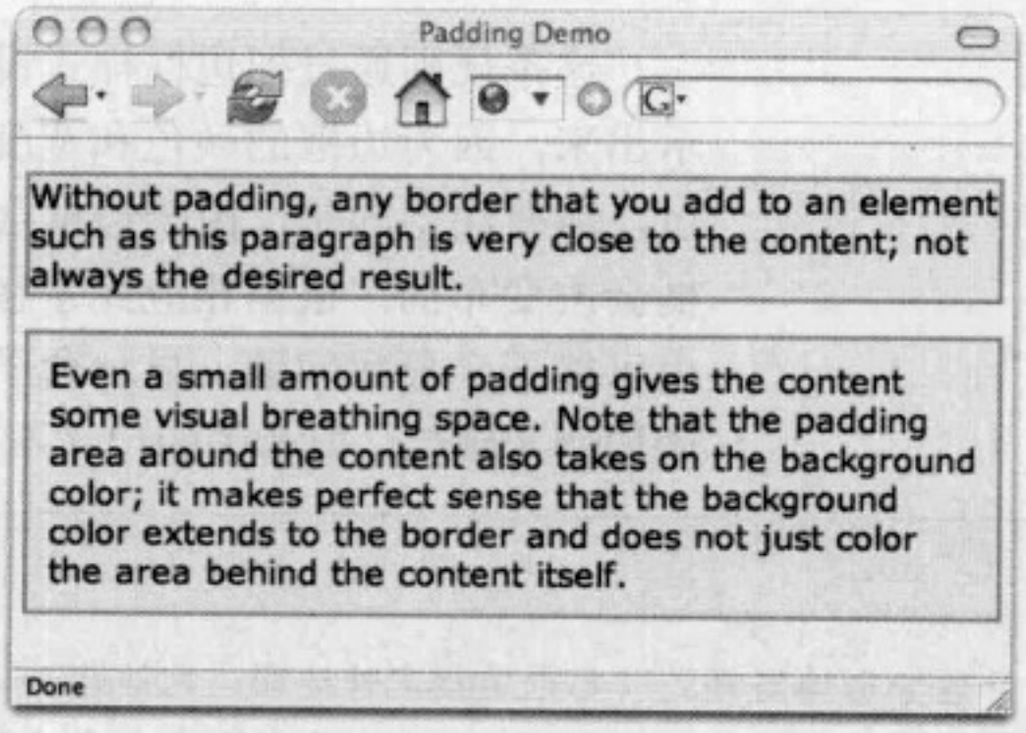


图 4-5 在为元素添加边框时，应该同时设置内边距以避免内容与边框接触

对于过去设计者们使用表格的单元格内边矩和空白 GIF 图的情况（会添加很多额外的表现标记），现在可以通过 CSS 的内边距样式更方便地实现相同的效果。

4.1.3 盒子的外边距

外边距比边框和内边距稍微复杂一些。首先，多数块级元素（段落、标题、列表等）都有默认的外边距（本章前面曾经看到过）。

在图 4-6 中，可以看到一个标题和两个段落分别显示了 3 次。第 1 个例子是标题和段落应用默认样式的外观。第 2 个例子也展示了一个标题和两个段落的相同排列方式，但这一次为它们添加了边框[①]，以便看清它们之间通过外边距创建的空白区域。第 3 个例子展示的是把标题和段落的外边距设置为 0 时的效果——元素紧挨在了一起。

把下面的规则放在样式表的顶部是一个好习惯：

```
* {margin:0; padding:0;}
```

① 原文 and background 有误。事实上并没有添加背景。——译者注

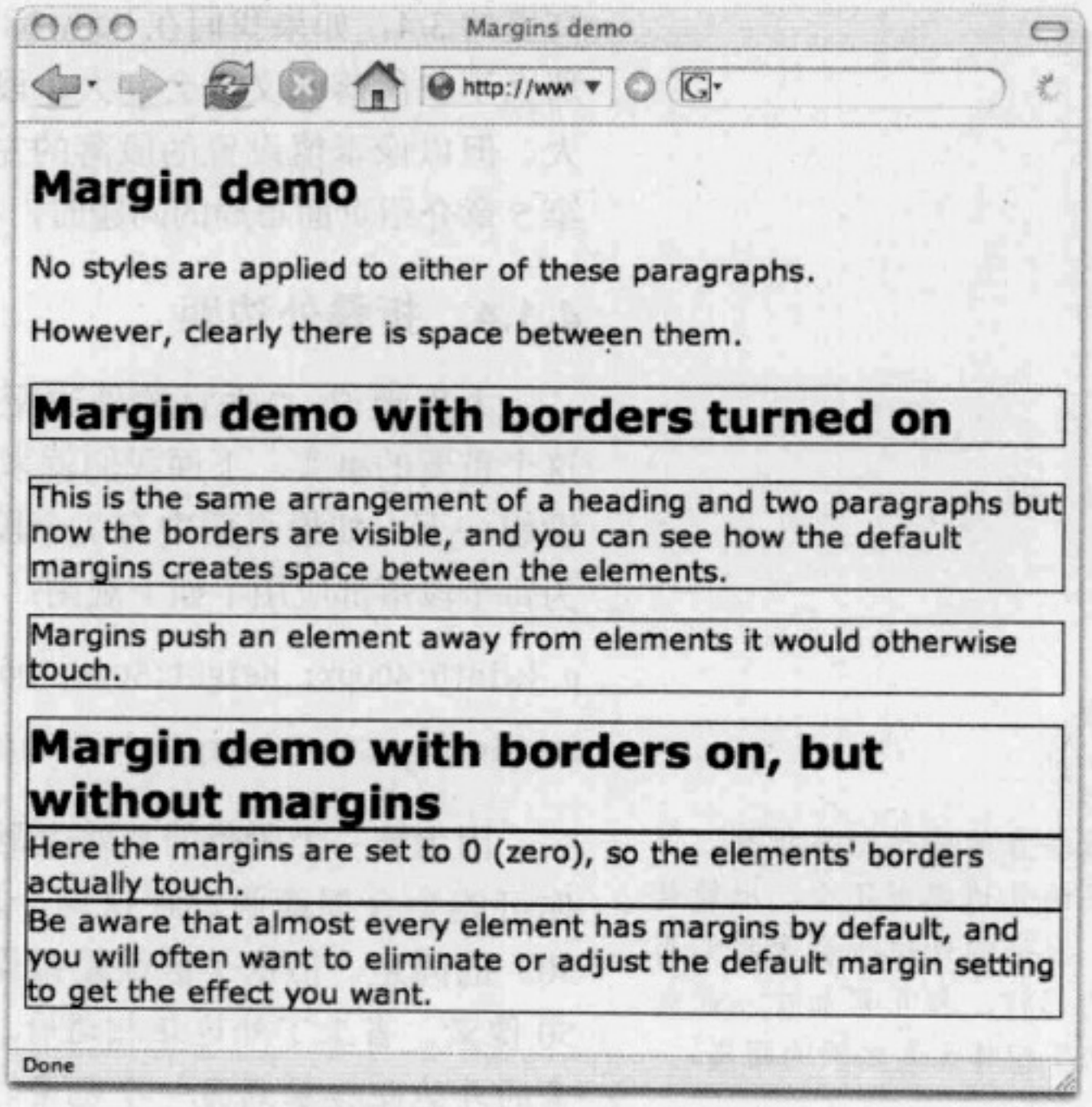


图 4-6 控制元素周围的外边距是一种关键性的技能——而且，知道每个元素几乎都有默认的外边距也很重要

这条规则会把所有元素的默认外边距和内边距设置为零，以便排除分不清浏览器和我们自己设置的内边距、外边距的问题。当把这条规则放到样式表中后，所有默认的外边距和内边距会统统消失。这样，就可以在为页面添加样式时，只为有必要带外（内）边距的元素应用外（内）边距。稍后我们会看到，不同的浏览器会为元素集合（例如表单和列表）应用不同的默认内边距和外边距，而通过“撤销”这些默认的设置并添加我们自己的定义，可以使用户得到更一致的跨浏览器体验。

通常，当为文本元素（如段落）设置外边距时，我们都希望混合使用不同的长度单位。例如，为了使段落文本与具有导航功能的侧边栏保持固定的距离，我们会用像素值来设置段落的左和右外边距。与此同时，为了使垂直方向上段落之间的距离与段落文本保持相对的比例关系，则需要为段落的上和下外边距设置 em 值。比如：

```
p {font-size:1em; margin:.75em 30px;}
```

在这个例子中，段落垂直方向上的间距始终会保持在文本

高度的 3/4，如果我们在 body 标签中增大整个页面的字体大小，那么不仅段落的文本会变大，段落之间的距离也会成比例地增大。但以像素值设置的段落的左和右外边距，则保持不变。在第 5 章介绍页面布局的问题时，我们还会深入讨论这个概念。

4.1.4 折叠外边距

大声地说："垂直的外边距折叠"——任何人都必须牢记这个重要的事实。下面我们就来解释这句话的含义及其重要性。设想一下，如果页面中有 3 个段落，一个接着一个，并且我们为每个段落都应用了如下规则：

```
p {width:400px; height:50px; border:1px solid #000;
margin-top:50px; margin-bottom:30px; background-color:#CCC;}
```

虽然垂直的外边距会折叠，但水平的外边距则不会。也就是说，水平的外边距会像我们想象的那样，相互累加在一起创建水平相邻元素之间的距离。

因为第一个段落的下外边距与第二个段落的上外边距相邻，你可能会合理地假设在这两个段落之间会出现 80 像素（50 + 30）的间距。但这个假设是错误的。因为这个距离实际上只有 50 像素。当上下外边距相遇时，它们会相互折叠，直至一个元素的外边距接触到另一个元素。在这个例子中，位于下方的段落具有较大的上外边距，因此它会首先与上方的段落接触，进而这个上外边距就决定了两个段落之间的距离为 50 像素（图 4-7）——这个过程称为折叠。

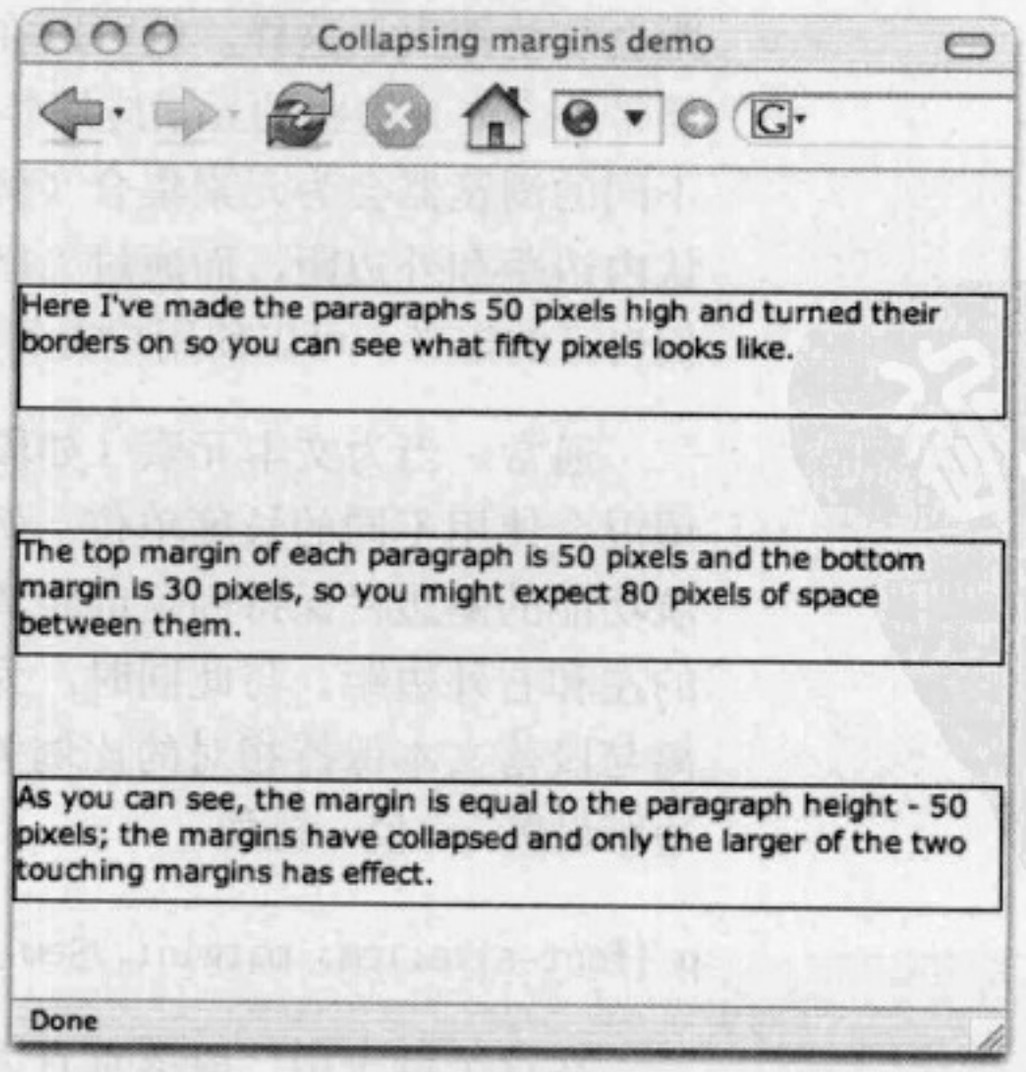


图 4-7 垂直的外边距折叠。折叠会在一个元素的外边距接触到另一个元素的边框时停止

这种折叠外边距的过程，可以保证一组标题、段落或列表元素中的第一个或最后一个元素，能够与页面或包含元素的上边或下边保持相应的距离；而当相应的元素出现在其他元素之间时，垂直方向上相邻的外边距会相互重叠，最终由较大的外边距决定间距。

4.2 盒子到底有多大

无论对初学者还是专家，盒模型的工作原理都是CSS中最令人头痛的地方。在接下来的讨论中，我们会以不同的方式来解释块级元素（例如标题、段落和列表）和行内元素。

下面我们就来逐步地、深入地观察盒模型。首先，我们讨论设置盒子的宽度，因为控制元素的宽度是创建多栏布局的关键所在。不过，同样的逻辑（或不合逻辑之处）也适用于元素的高度。

要设置元素盒子（以下简称为“盒子”）的宽度，可以使用width属性：

```
p {width:400px;}
```

然后，通过为盒子添加背景颜色可以在不影响其宽度的情况下显示盒子的边界：

```
p {width:400px; background-color:#EEE;}
```

图4-8展示了这个带背景颜色的400像素宽的元素。

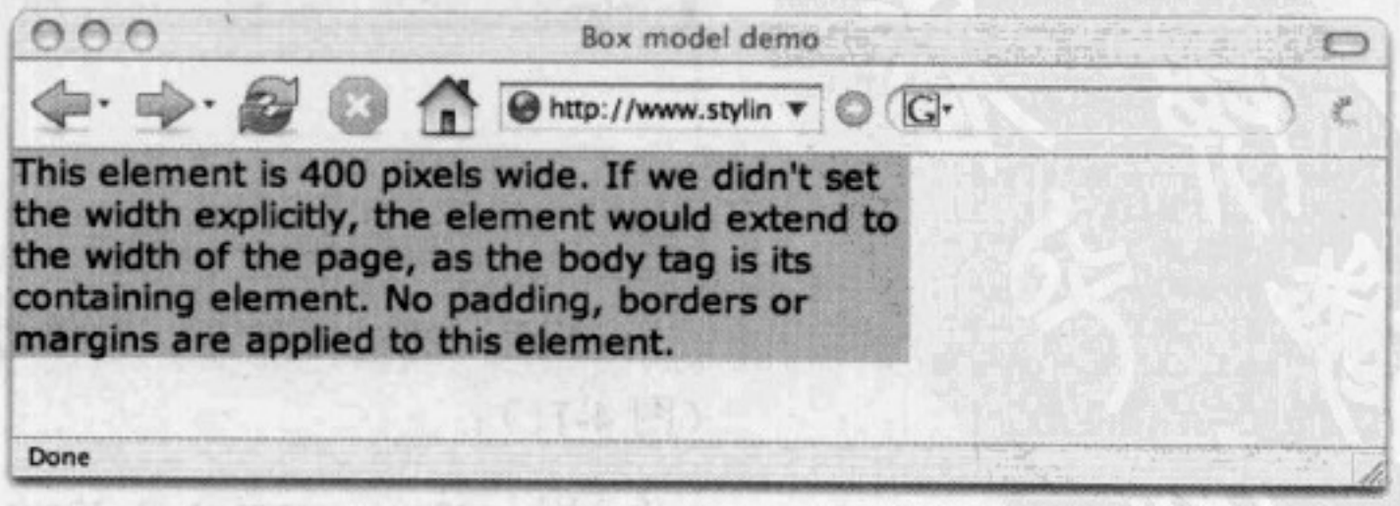


图4-8 通过设置width属性，元素不再保持与包含元素相同的默认宽度。这个例子中的包含元素是body元素，它的宽度默认与浏览器窗口同宽

在没有设置内边距的情况下，元素中的内容也是400像素宽，并且文本会与盒子的四边接触。这个结果很正常，但是当

我们为盒子添加内边距和边框后，结果就会发生变化。下面，我们为盒子的左右两边[①]都添加 20 像素的内边距：

```
p {width:400px; background-color:#EEE; padding:0 20px;}
```

你可能认为，当为 400 像素宽的盒子添加了 40 像素的内边距后，内容会压缩到 360 像素，但是你错了。在令人惊讶的、古怪的 CSS 中，盒子的宽度会增大 40 像素（图 4-9）。

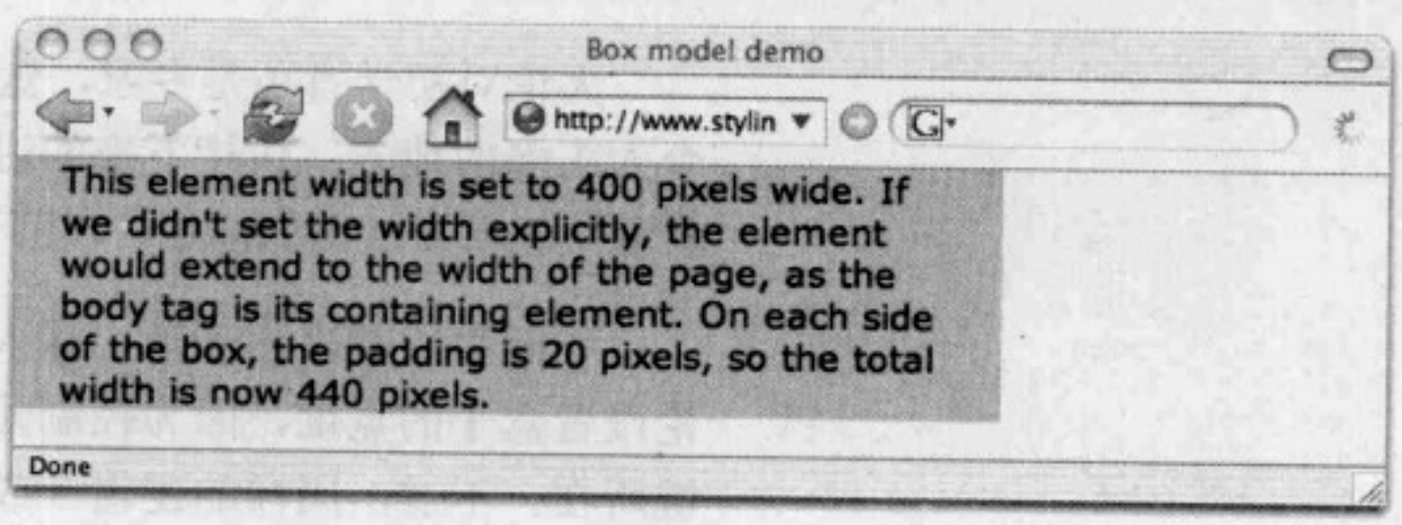


图 4-9 添加内边距会导致盒子变宽

如果再为盒子添加 6 像素宽的左、右边框：

```
p {width:400px; margin: 0; padding:0 20px; border:#000 solid;
border-width: 0 6px 0 6px; background-color:#CCC;}
```

那么盒子还会再加宽 12 像素（图 4-10）。现在，原来 400 像素宽的盒子变成了 452 像素宽（6 + 20 + 400 + 20 + 6 = 452）。

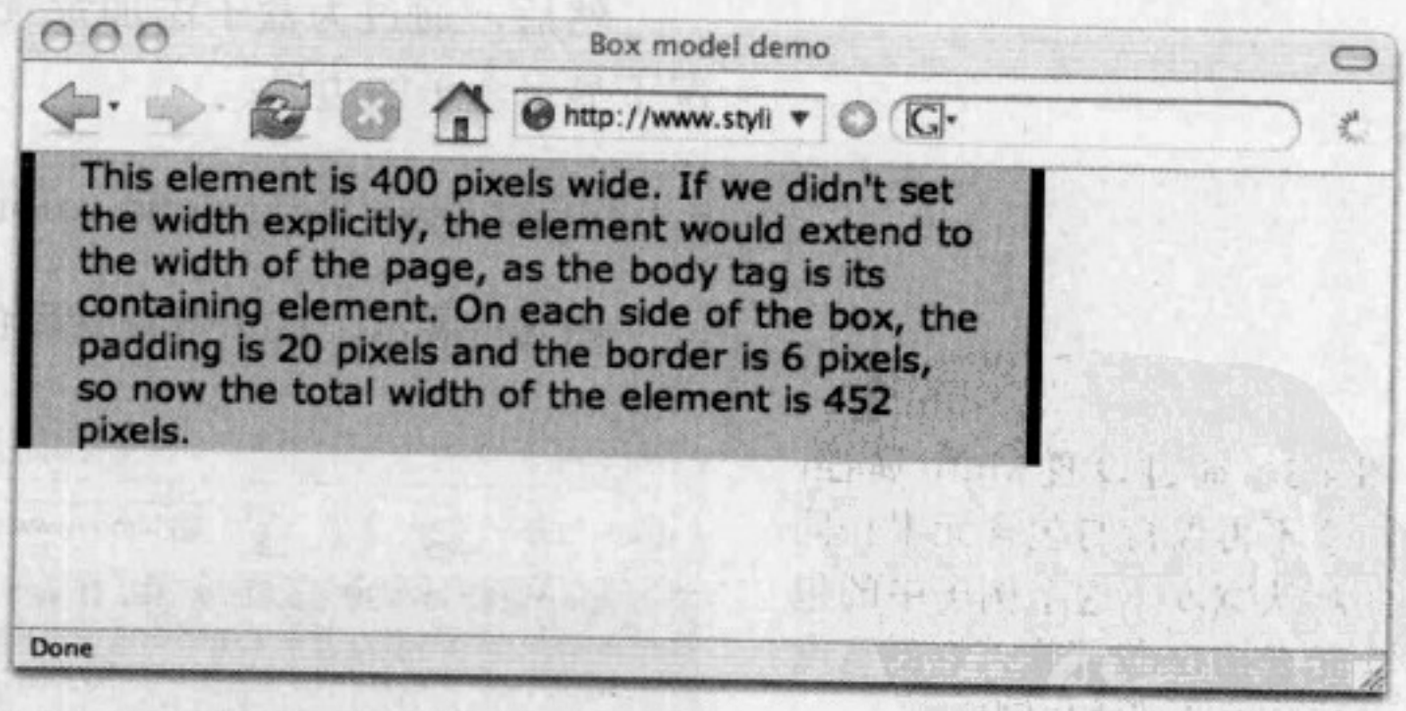


图 4-10 添加边框会导致盒子变得更宽

下面，通过添加左、右外边距再在元素两侧创建一些空白（图 4-11）：

```
p {width:400px; margin: 0 30px; padding:0 20px; border:#000
solid; border-width: 0 6px 0 6px; background-color:#CCC;}
```

① 原文 each sides 有误。——译者注

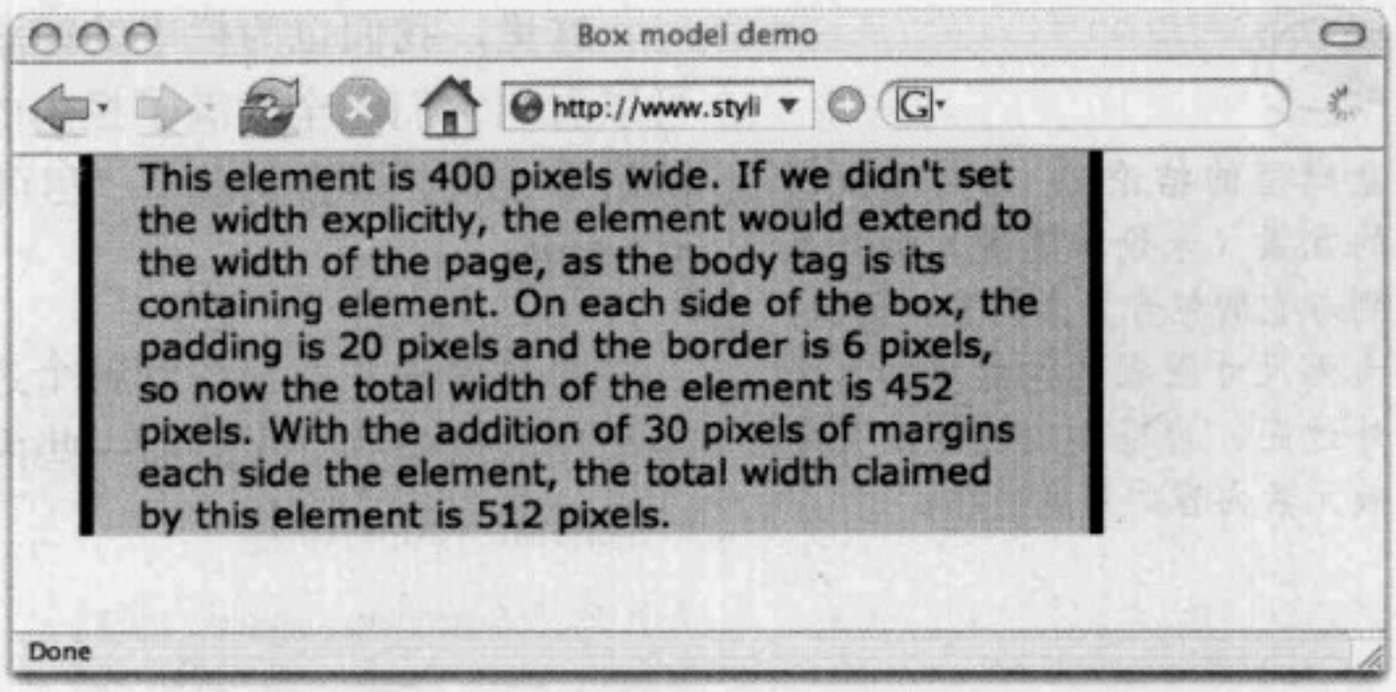


图 4-11 外边距会在元素周围创建空白区域

这里，通过为左右两边分别添加 30 像素的外边距，增大了元素占用的整体空间，因为外边距是位于盒子外部的空间。由此可见，尽管我们认为盒子的边框和内边距应该处于内部，因而不会增大盒子的宽度，但事实上边框和内边距会增大盒子的宽度。

盒模型的结论 1：特定尺寸的盒子（宽度已定）会随着内边距、边框及外边距的添加扩展，进而占据更多的水平空间。事实上，通过 width 属性设置的是盒子内容的宽度，而不是盒子本身的宽度。

当我们创建多栏布局，而每一栏必须维护适当的宽度时，理解盒子的上述行为具有很重要的意义。比如，在下一章我们要学习的“浮动布局”中，如果某一栏的宽度由于内边距、外边距或边框的变化而被意外修改，那么就会导致布局显示错误。一般来说，我们会使用具有一定尺寸（定义了宽度）的 div 来创建布局中的分栏，然后再将所有内容元素（标题、段落、导航列表等）嵌套在该分栏的内部。

创建包含内部元素的单栏

为了示范这一基本技术，以下面宽度为 170 像素的 div 为例，该 div 中包含一个标题和一个段落：

```
<div id="column">
    <h4>An h4 heading</h4>
    <p>The heading and this paragraph...</p>
  </div>
```

为这个 div 定义的 CSS 规则如下：

```
div#column {width:170px;}
```

添加到页面顶部的标尺图像有助于看清 CSS 规则改变时元素宽度的变化（图 4-12）。

盒模型的结论2：无尺寸限制的元素（未设置宽度）会扩展到与它的包含元素同宽。因此，为无尺寸限制的元素添加水平外边距、边框和内边距，会导致元素内容的宽度变化。

这里，我们也为栏中的标题和段落添加了背景颜色，因此可以看出它们完全填满了栏内的水平空间。块级元素宽度的默认值是auto，其含义就是“尽可能大”。这样我们就可以得出第2个结论。

面对这种文本与栏的各个边紧靠在一起的情形，我们的第一个反应就是为div添加一些内边距，以便在文字的四周创建一些间隙（图4-13）。

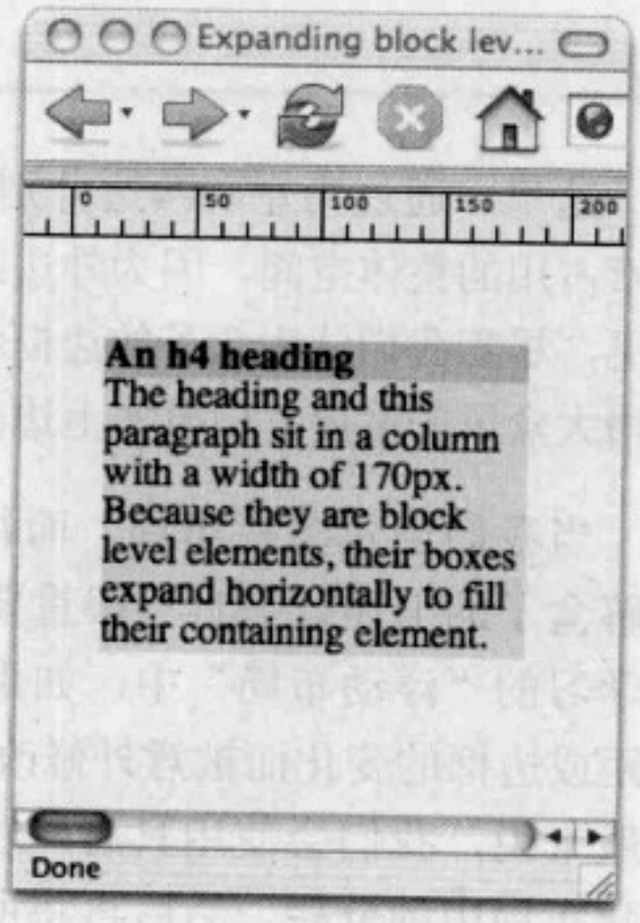


图4-12 在不给容器元素应用内边距时，块级的标题和段落会填满栏的宽度（另见彩插）

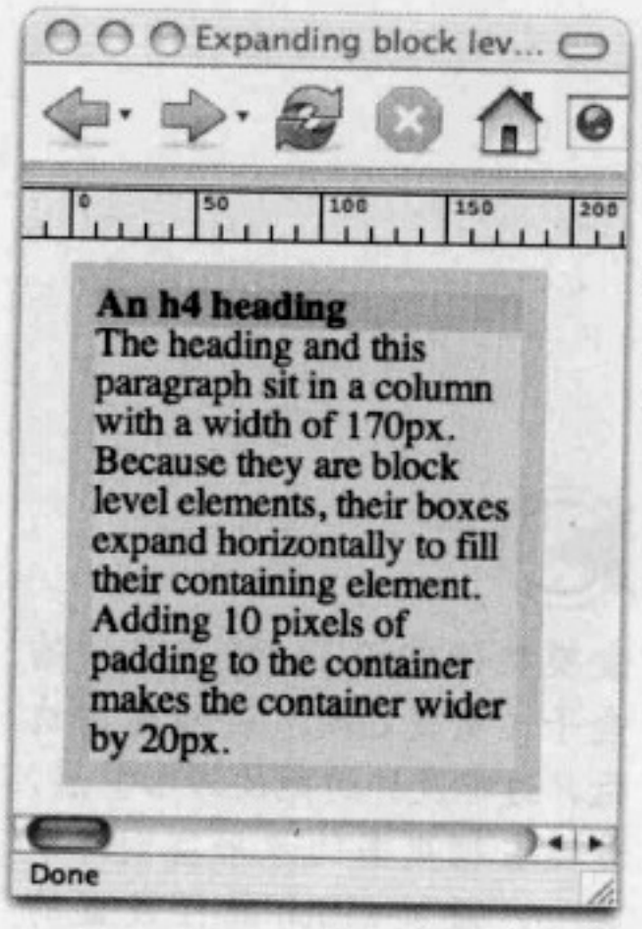


图4-13 为包含元素添加内边距也会增加它的宽度。现在，包含元素的宽度达到了190像素。为了看清楚，我们为div添加了粉红色的背景（另见彩插）

```
div#column {width:170px; padding:10px;}
```

通过屏幕截图（图4-13）中的标尺可以看出，为栏的每一边添加的10像素的内边距，使它的宽度增大到了190像素。虽然通过一条样式规则就为div中的所有元素都添加了整洁的边距，但为了保持栏的整体宽度等于170像素，接下来必须从当前盒子的宽度中减掉相应的数值（10+10=20像素），即将其宽度设置为150像素。每次在修改栏的内边距的同时都修改栏的宽度很令人讨厌，这种情况对于多栏布局尤其突出。

为此，一种替代方案就是为栏内元素应用等量的外边距。

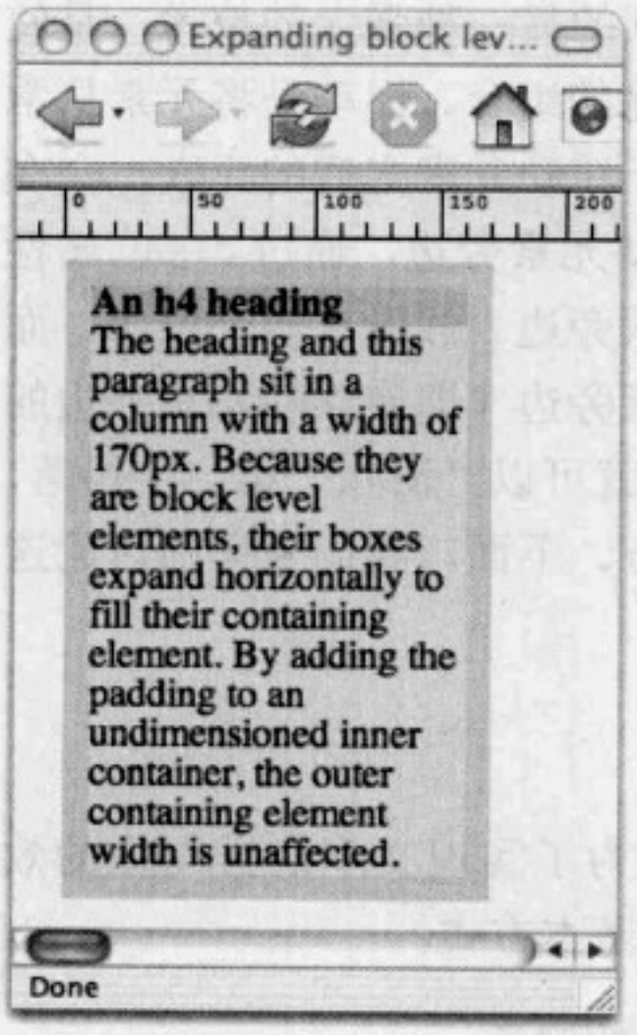


图 4-14 通过为内部 div 添加内边距，由外部 div 定义的栏的宽度能够保持不变（另见彩插）

前面我们曾说过，要避免使用表现标记。你可能会对我在这里为达到表现目的而使用额外的标记感到不理解。对此，我想说的是 div 与其他用于表现的标记（例如表格）不同，在添加样式之前它不会从视觉上影响布局。因此，我感觉这种交换是值得的，尤其对于刚开始学习 CSS 人，如果不需要在每次修改元素的边框或内边距时都求助于计算器，那么就足够了。

但是，这种方案同样也存在问题，即当我们决定调整栏的各边与内容之间的距离时，需要关注和修改多个元素。

比较简单的解决方案，是在作为栏的 div 中直接添加另一个 div：

```
<div id="column">
  <div id="column_inner">
    <h4>An h4 heading</h4>
    <p>The heading and this paragraph...</p>
  </div>
</div>
```

并为这个内部 div 应用内边距：

```
div#column {width:170px; padding:10px;}
div#inner_column {padding:10px;}
```

这样，我们既可以通过一条样式来控制栏的内边距，同时也避免了修改栏宽度的问题（图 4-14）。

由于这个内部 div 是无尺寸限制的，因此适用于“盒模型的结论 2”，即它的内容会被压缩。而且，我们也可以通过修改这个内部 div 的外边距，将栏中的所有元素由边界向内压缩，同样也不会改变栏的宽度。下一章，我们会在很多页面布局的例子中，使用这种在栏的内部放置一个内部 div 的技术。因此，请你在继续阅读下面的内容之前，一定要确保理解最后 3 个屏幕截图中所反映出的思想。

通过对盒模型的讨论，我们关键是明确了一点，即在所有现代的符合标准的浏览器中，当设置元素的宽度时，实际上设置的是元素中内容的宽度。而为该元素设置的任何内边距、边框和外边距都会在指定的宽度值基础上，增加元素占据的整体空间。

下面，我们再来看一看创建基于 CSS 的布局必须要理解的另两个关键技术——浮动和清除。

4.3 浮动和清除

在创建页面布局时可能会用到的另一种强大的技术，是使用 float 和 clear 属性的浮动和清除的组合。浮动是将元素从常规文档流中移出的一种方式。位于浮动元素之后的元素，会在空间充足的情况下向上移动到浮动元素旁边。通过 clear 属性可以阻止元素向上移动到浮动元素旁边。假设有两个段落，而我们只想让第一个移动到浮动元素旁边（即使浮动元素旁边的空间能够放得下两个段落），那么就可以“清除”第二个段落，这样它就会定位在浮动元素的下方。下面，我们来详细讨论这两个属性的用法。

4.3.1 float 属性

虽然 float 属性主要的目的是为了实现文本绕排图像的效果，但它也是创建多栏布局的一种基本方式。

我们先来看一个文本绕排图像的例子：

```
img {float:left; margin:0 4px 4px 0;}
```

这条规则向左浮动了图像，因此文本会环绕在图像右侧（图 4-15）。

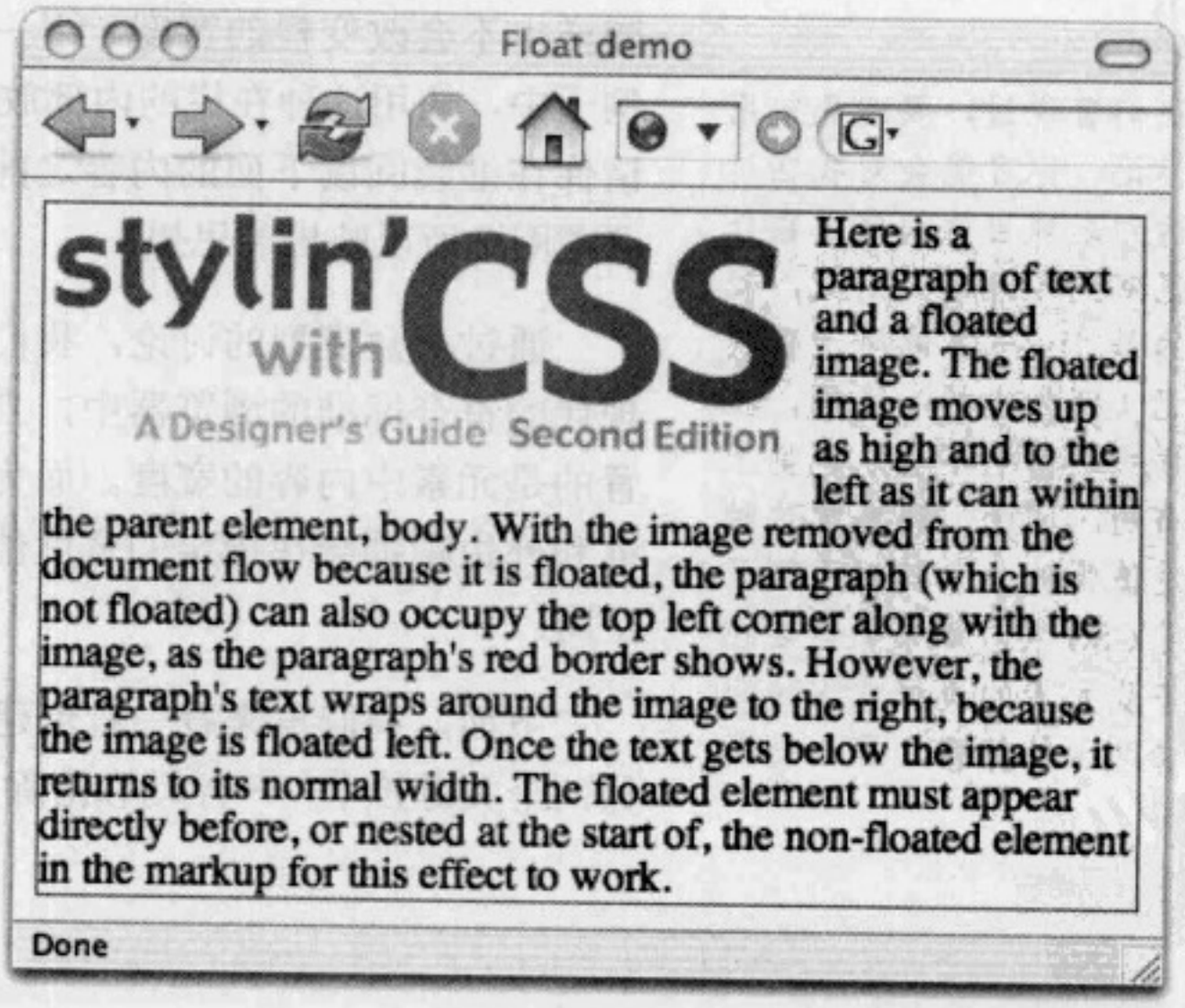


图 4-15 浮动的图像会从文档流中移出。如果在标记中的图像后面是文本元素，那么元素中的文本就会环绕图像（另见彩插）

为实现上述图像浮动的效果，相应的标记代码应该如下所示（图像位于前面）：

```
<img ……/>
<p>…the paragraph text…</p>
```

简而言之，当浮动图像或其他元素时，就是要求浏览器将它们尽可能地显示到父元素的左边（或者在 float:right 的情况下，是右边），在这个例子中，父元素就是 body。但是，段落（图 4-15 中带红色边框的元素）在文档流中看不到浮动元素（前面的图像），因此它仍然会占据父元素左上角的位置，而段落的内容，也就是文本，则会环绕在浮动的图像周围。

为此，通过浮动构建分栏只需简单的一步（图 4-16）：

```
p {float:left; width:200px; margin:0;}
img {float:left; margin:0 4px 4px 0;}
```

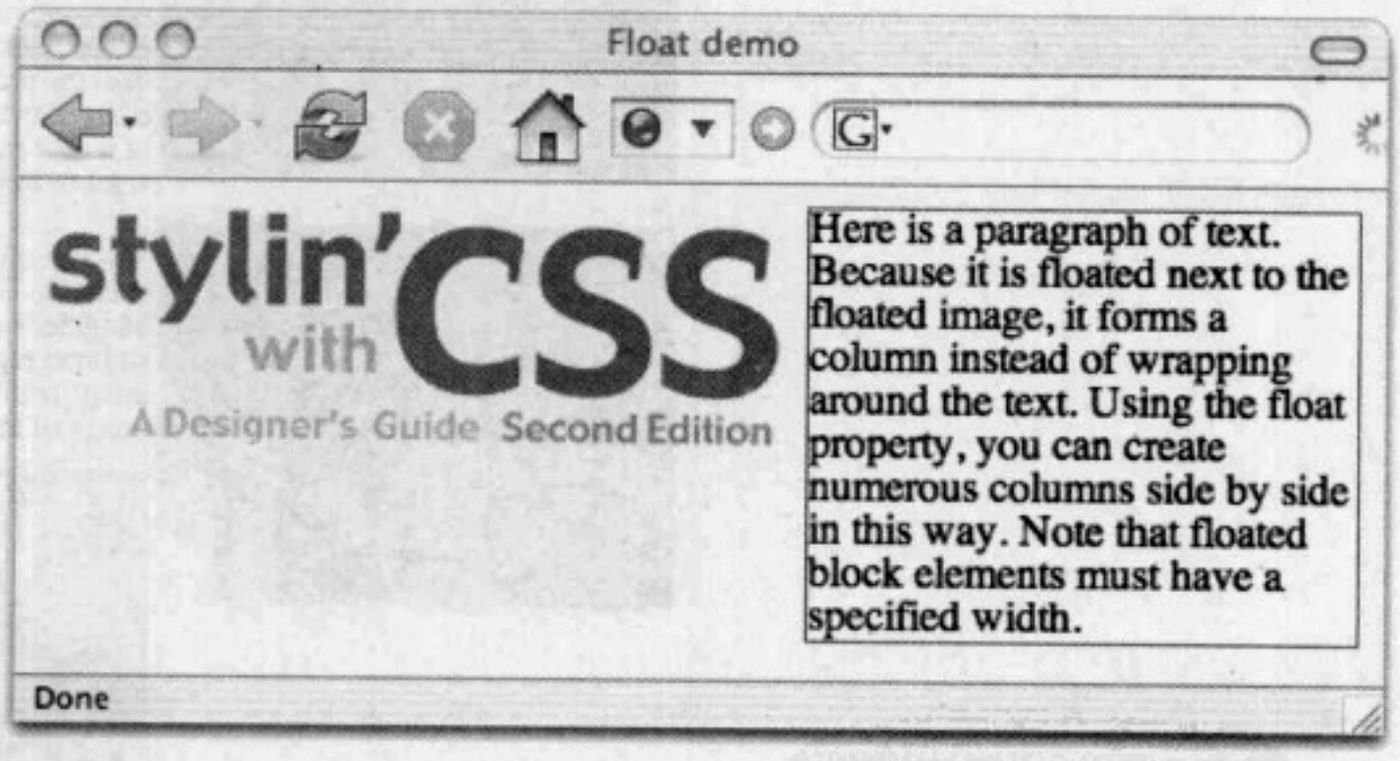


图 4-16 当具有固定宽度的段落浮动到浮动图像旁边时，会构成一个分栏而不再环绕图像

用来控制浮动的规则还有很多。在 Eric Meyer 的 *Cascading Style Sheets 2.0 Programmer's Reference*（2001 年，McGraw-Hill Osborne Media 出版社）一书中，详细介绍了这些规则。按照 Eric 介绍的规则简单来说，就是“当浮动某个元素时，这些（浮动）规则会说‘尽可能把这个元素放到一边的最高、最远处’”。虽然这是一本几年前出版的书，但它仍然是所有认真的 CSS 程序员的基本参考书。因为书中在一定程度上介绍了 CSS 的内部工作原理，这些细节在其他书中不可能看到。

当同时浮动图像和“具有指定宽度的”段落时，文本绕排效果就会停止，原来的文本会在图像旁边构成一个分栏。这就是通过浮动来创建多栏布局的基本原理。也就是说，只要浮动某个设置了宽度的元素，而且页面中也有足够的空间，那么相应的元素就会像栏一样排列在一起（当以这种方式浮动具有内置宽度的图像时，无需通过 CSS 指定它的宽度）。假如我们浮动 3 个具有固定宽度的 div，那么就会得到能够在其中放置其他元素（这些元素当然也可以浮动）的 3 个容器。在第 5 章中，我

们会看到与此有关的实例。

4.3.2 clear 属性

另一个与 float 同时频繁使用的属性是 clear。虽然在空间足够的情况下，任何元素都会向上移动到浮动元素旁边，但有时候，我们不想出现这种情况，我们想让它清除[①]——也就是说，让它保持在浮动元素下方（而不是旁边）。为了示范清除，图 4-17 展示了一个布局，其中每个项目都由一幅图像和旁边的文本组成，而且对每幅图像设置了浮动。这个例子与图 4-16 相似，只不过重复了 3 次。

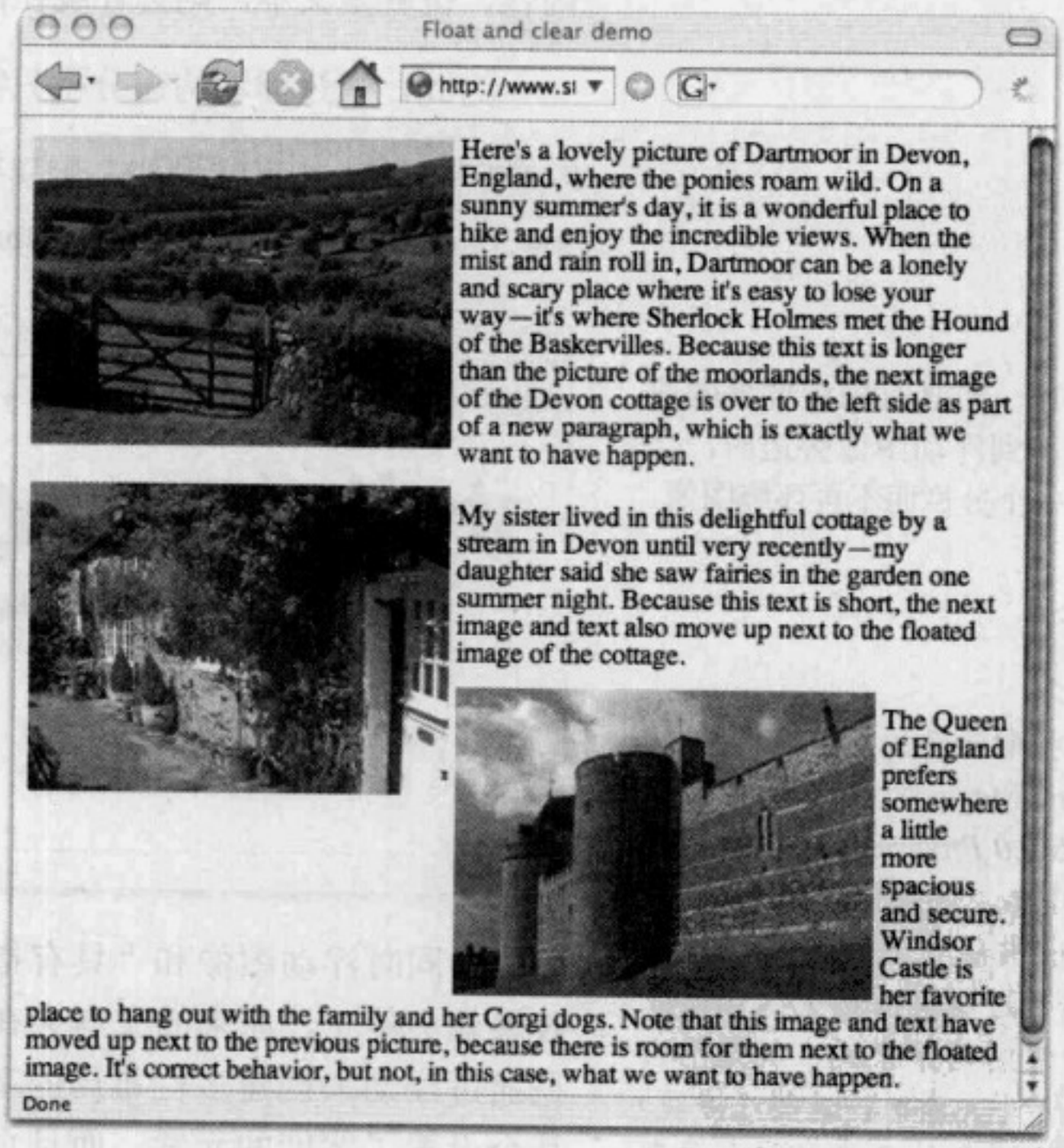


图 4-17 由于存在空间，第 3 幅图像及它的文本会向上浮动到第 2 幅图像的旁边——但这不是我们想要的结果

下面是这个例子的 XHTML 代码（为节省版面省略了部分内容）：

① 读者可以将 clear 属性理解为通过设置清除元素的上外边距来填满（或补足）浮动元素旁边的空间，以便清除元素从浮动元素的下方开始布局。事实上，clear 属性的原理，就是清除元素的上外边距能够被自动地重写并设置，从而使该元素只有可见部分会显示在浮动元素下方。因此，如果我们为清除元素设置了上外边距，这个上外边距也将被 clear 声明撤销。——译者注

```
<img src="../images/dartmoor-view.jpg" />
<p> Here's a lovely picture of Dartmoor... </p>
<img src="../images/english-cottage.jpg" />
<p> My sister lived in this delightful cottage ... </p>
<img src="../images/winsor-castle_walls.jpg" />
<p> The Queen of England...</p>
```

对这些标记应用的 CSS 如下：

```
p {margin:0 0 10px 0;}
img {float:left; margin:0 4px 4px 0;}
```

在这个页面中，每幅图像都应该浮动到与之关联的文本旁边。但是，当没有足够的文本清除到浮动的图像底部时（比如，图 4-17 中的第 2 段文本），后面的图像 / 段落就会向上移动到浮动图像的旁边。

对于这个例子来说，浏览器中显示的布局是正确的——有空间让第 3 个项目提高到前一个浮动元素的旁边，而且实际上也是如此。当然，这不是我们期望的布局效果。为此，我们的解决方案是在标记中添加一个应用了 clear 属性的非浮动元素，强制第 3 个项目从第 2 个项目下方开始。下面就是添加了额外 div 元素的标记及补充的相关样式：

```
<img src="../images/dartmoor-view.jpg" />
<p> Here's a lovely picture of Dartmoor... </p>
<img src="../images/english-cottage.jpg" />
<p> My sister lived in this delightful cottage ... </p>
<div class="clearthefloats"></div>
<img src="../images/winsor-castle_walls.jpg" />
<p> The Queen of England...</p>
```

接着，我们只需在 CSS 中添加一个清除类：

```
p {margin:0 0 10px 0;}
img {float:left; margin:0 4px 4px 0;}
.clearthefloats {clear:both;}
```

在添加标记及相应的清除类（这条规则能够清除左或右浮动的元素）后，页面会显示出正确的结果（图 4-18）。

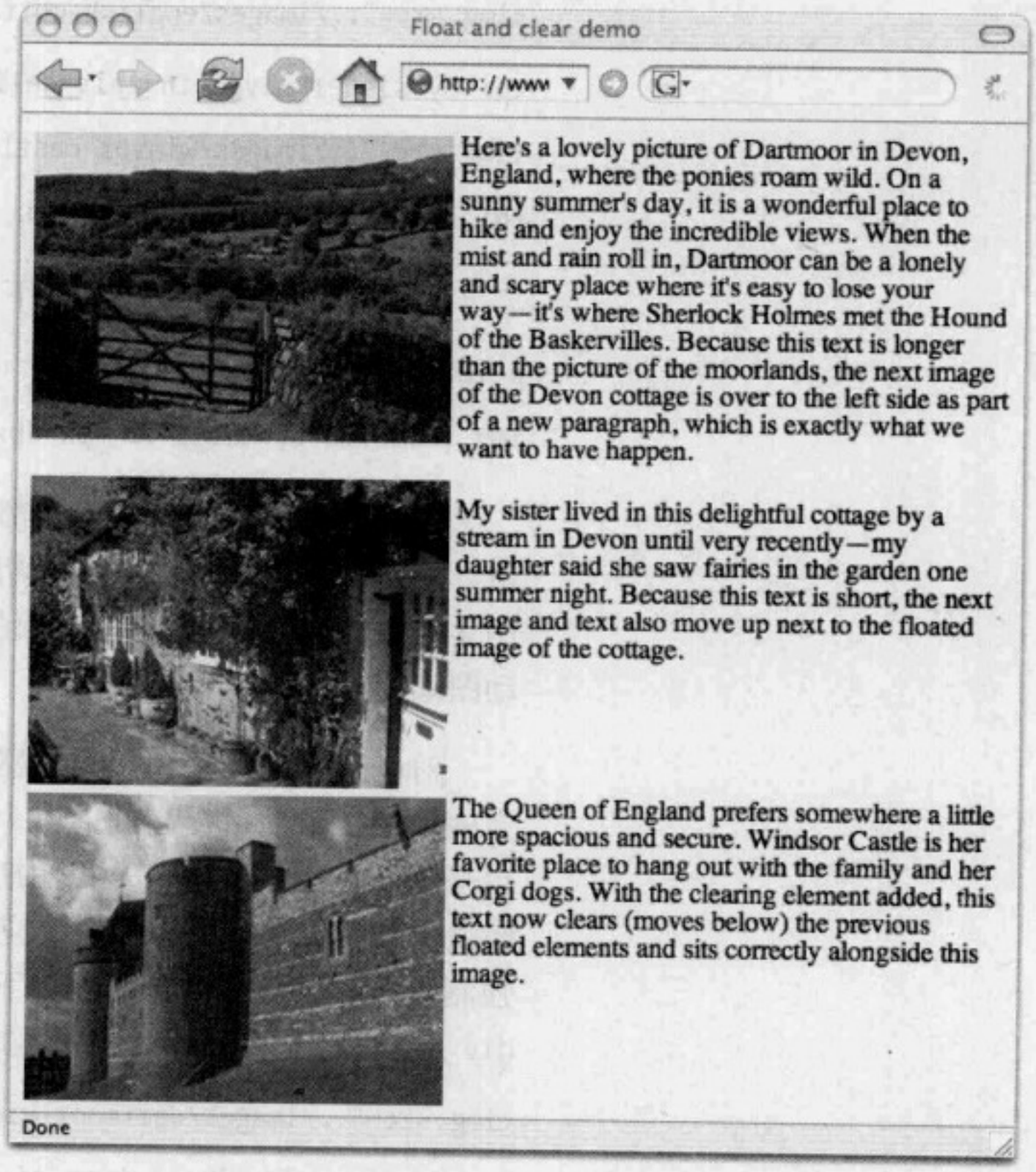


图 4-18 通过添加清除元素，布局显示正确了

我们在第 2 个项目和第 3 个项目[①]之间添加的新“清除的”元素，现在定位在了第 2 幅图像下方（由于不包含内容，所以不可见）。由于在标记中，第 3 幅图像及其段落位于这个清除元素之后，所以它们会在清除元素下方定位，于是完成了我们期望的布局效果。

在创建 CSS 布局的过程中，清除浮动是一个必须掌握的重要技术。提示条“Aslett 的清除方法”讨论了只通过 CSS 和标记中的一个类来清除浮动的实用技术。本书后面还要更深入地讨论浮动和清除，不过对于用作 CSS 页面布局基础的浮动，我们已经介绍得差不多了。接下来，我们介绍 position 属性。

为 clear 属性设置的 both 值的含义是让 div 同时清除左和右浮动元素（即定位在左和右浮动元素的下方）。在这个例子中，也可以使用 left 值，但使用 both 则可以保证将来向右浮动图像时，clear 仍然有效。

① 原文 paragraphs 有误。——译者注

Aslett的清除方法

以创建者 Tony Aslett（www.csscreator.com）命名的 Aslett 清除方法，要求使用一个容器（例如 div）将嵌套在其中的浮动内容封装起来，而这在平常是不需要的。这种方法使用 CSS 的 :after 伪元素在容器中其他内容之后插入一个隐藏的非浮动内容（零高度的句点）。而且，同样的代码也适用于清除非浮动的内容，但容器必须把内容封装起来。下面就是这种方法的代码：

```
.clearfix:after {
  content: ".";
  display: block;
  height: 0;
  clear: both;
  visibility: hidden;
}
.clearfix {display: inline-table;}
/* 反斜杠 hack，用于在 IE（Mac）中隐藏 CSS 规则 \*/
* html .clearfix {height: 1%;}
.clearfix {display: block;}
/* 反斜杠 hack 结束 */
```

把这些代码添加到样式表的末尾，就可以使它们对页面起作用（我已经把它们添加到了下载到的 texts_n_colors.css 文件中[①]）。然后，在需要通过容器封装浮动内容的情况下，请为该容器元素添加 clearfix 类，比如：

```
<div class="clearfix">
```

然后，容器元素会立即封装其中的任何浮动内容。以下是这种方法的两个代表性的应用。

(1) 强制页脚位于浮动的分栏下方（参见第 5 章中有关浮动分栏的例子）。通过为包含分栏的封装 div 添加 clearfix 类，可以使该容器始终在垂直方向上封装所有分栏，无论这些分栏最终会有多长。这样，在标记中位于该容器 div 结束标签之后的元素（例如页脚），总会位于最长的分栏下方。

(2) 为多个浮动元素的四周添加边框。将浮动的元素封装在一个 div 中，并为该 div 添加 clearfix 类，使其能够封装浮动的元素。然后，就可以为这个包含元素设计边框。

这只是在需要它们之前不知道它们为什么存在的诸多方法中的一种。不过，总有一天，你会知道。而且，这种方法可以避免向标记中添加过多的 div，同时也简单易用。

需要注意的是，在 IE 6 中这种方法会导致包含浮动元素的 div 不能正确地起到封装作用。不过，这也正是基于符合标准的浏览器开发并在后来的 IE 中测试的另一个原因。

要了解有关 Aslett 清除方法的更多内容，请访问 Position Is Everything（www.positioniseverything.net/easyclearing.html）。

① 位于 css_styles\text 目录下。——译者注

4.4 position属性

基于CSS布局的核心是position属性。position属性用于确定元素盒子在页面上定位的参照点。

下面我们看一看与定位相关的属性和概念。

position属性有4个值：static、absolute、fixed和relative。其中static是默认值。刚接触到这几个值时，通过字面可能看不出它们的实际作用。为了尽快帮你理解这几个值的含义，我们以一个包含4个段落的简单页面为例。在每种情况下，我们都会让第1、2和4段落保持默认的static定位状态，只修改第3个段落的position属性值。由于我们已经在标记中为第3个段落添加了specialpara类，所以改变它的position属性不会影响其他段落。

4.4.1 静态（static）定位

首先，我们看一看4个段落都是默认的static（静态）定位的情形（图4-19）。

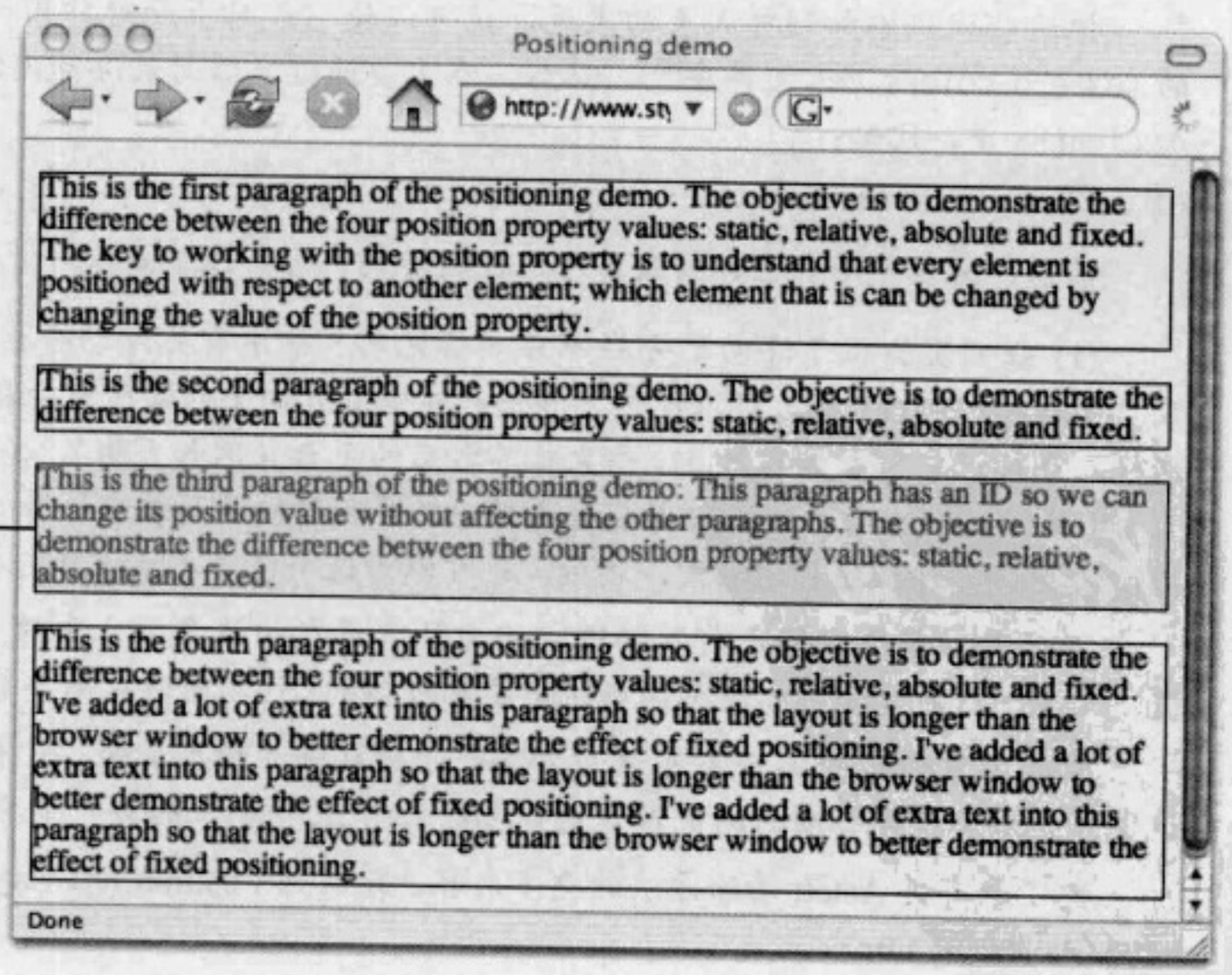


图4-19 在4个段落的position属性都是默认的static的情况下，它们与其他正常文档流中的元素一样，相互之间处于上下堆叠状态

在 static 定位的情况下，每个元素都简单地自上而下地相继排列，因此我们会看到 4 个段落上下堆叠，而它们的默认外边距在相互之间创建了空白区域。

要脱离由默认的 static 定位决定的这种前后相继的布局状态，必须把盒子的 position 属性修改为其他 3 个值中的一个。

4.4.2 相对（relative）定位

下面，我们将第 3 个段落设置为 relative（相对）定位。在这种情况下，可以通过 top、left、bottom 和 right 属性，相对于原来的默认位置移动该段落。一般来说，只设置 top 和 left 属性的值就能得到我们想要的结果。比如，下面的规则：

```
p#specialpara {position:relative; top:30px; left:20px;}
```

会产生如图 4-20 所示的结果。

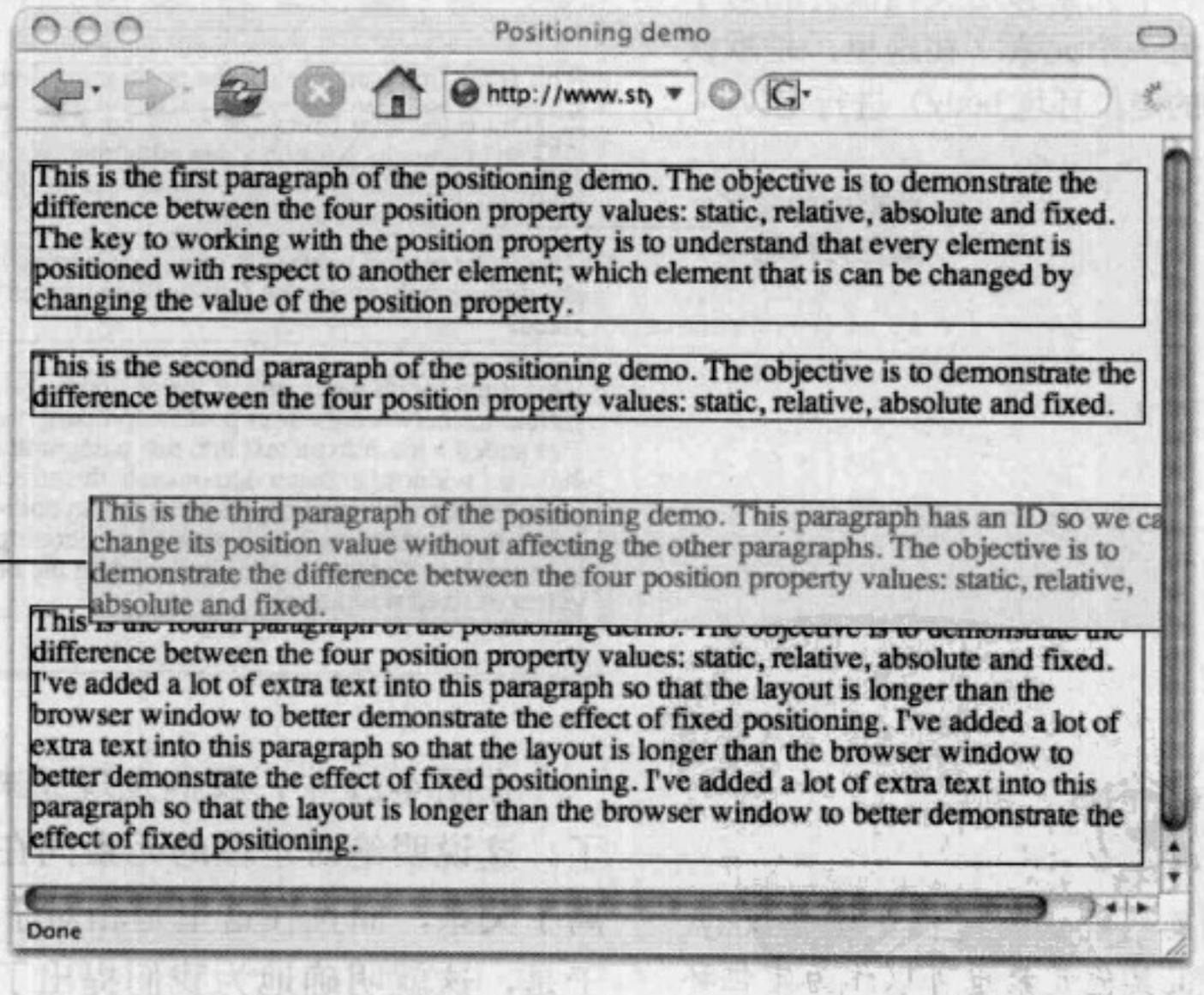


图 4-20 在相对定位元素的情况下，可以通过 left 和 right 属性相对于元素在文档流中的正常位置移动元素

这样，第 3 个段落会向下移动 30 像素，向右移动 20 像素。不过，我们注意到，尽管元素已经从其原始的位置上移开，但除此之外什么也没有发生。该元素作为静态元素时所占据的空间，以及其他元素的位置都保持不变。

在 top 和 left 属性中使用负值，可以向上和向左移动元素。

对此，我们得出的教训就是：如果通过这种方式来移动元素，那么必须为它准备足够的空间。对于图 4-19 所示的情形而言，就意味着需要为第 4 个段落添加 30 像素或更大的 margin-top 值，以便将它向下移动，从而避免与重新定位的第 3 个段落发生重叠。

4.4.3 绝对（absolute）定位

absolute（绝对）定位是完全不同于 static 和 relative 的一种定位方式。因为这种定位会把元素完全移出文档流。下面，我们把示范相对定位的代码中的 relative 修改为 absolute：

```
p#specialpara {position:absolute; top:30px; left:20px;}
```

结果如图 4-21 所示。

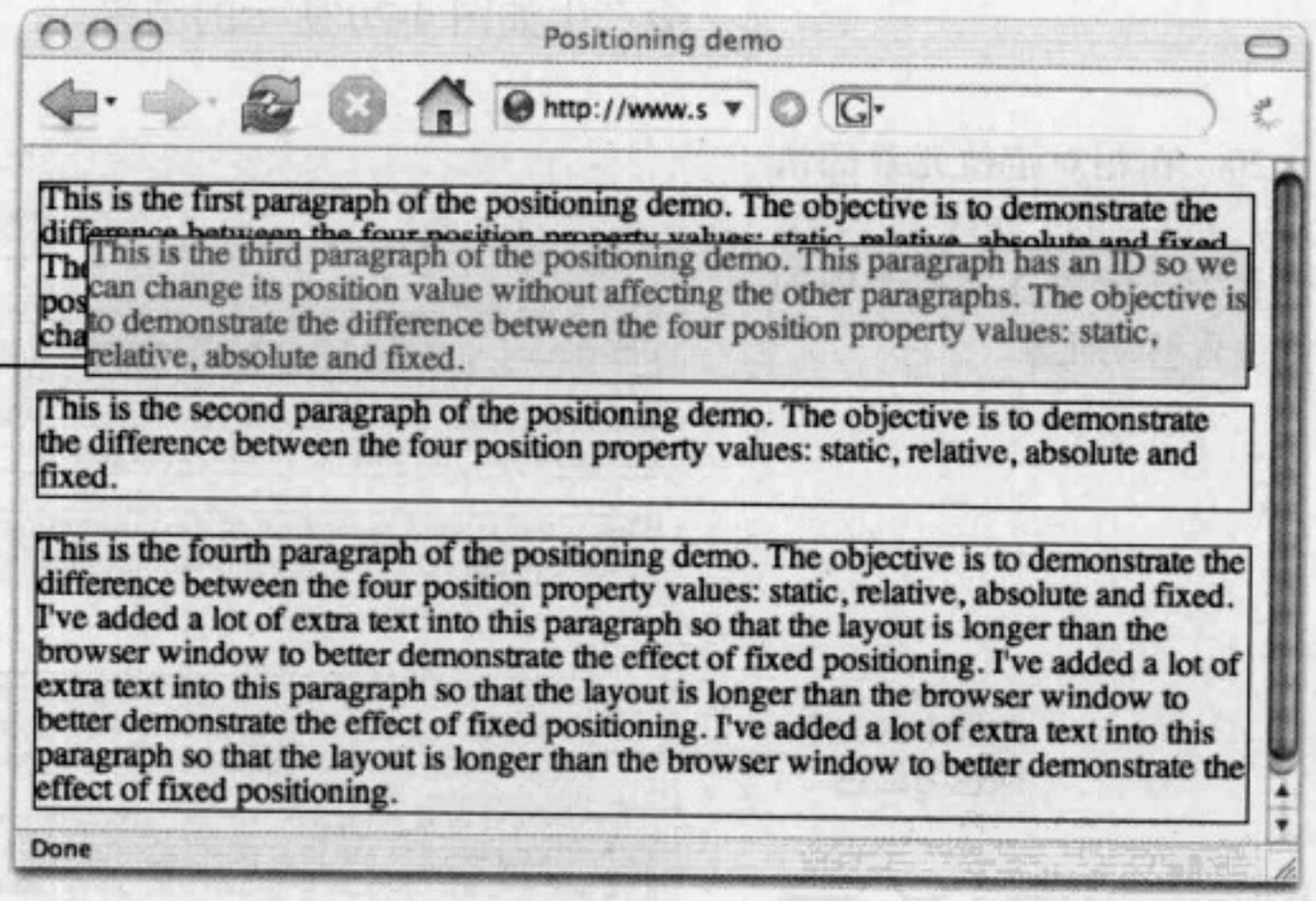


图 4-21 通过绝对定位可以将一个元素移出文档流并相对于另一个元素（在这里，是默认的定位环境 body）进行定位

此框中的文字颜色为红色

虽然默认的定位环境是 body，但其他元素也可以作为定位环境，稍后我们会讨论到。

在图 4-21 中，我们发现原来由第 3 个段落占据的空间不见了。这说明绝对定位的元素同在标记中与它相邻的元素完全脱离了关系，而且在这里是相对于顶级元素（body）进行了定位。于是，这就明确地为我们提出了定位环境这个重要思想，本章后面还会再提到定位环境。

不过，目前我们可以认为绝对定位元素默认的定位环境是 body 元素。如图 4-21 所示，由 top 和 left 值定义偏移量，会相对于 body 元素（标记层次中的顶级祖先元素）而不是其在文档

流中的默认位置（同使用 relative 定位的情况下那样）移动第 3 个段落。

由于绝对定位元素的定位环境是 body，所以绝对定位的元素会随着页面滚动而移动——它需要保持与 body 元素的相对关系，而 body 元素会随着页面滚动而移动。

在介绍怎样将除 body 元素之外的元素作为绝对定位元素的定位环境之前，我们再来看一看最后一个定位属性——fixed（固定）定位。

4.4.4 固定（fixed）定位

固定定位与绝对定位相似，只不过固定定位元素的定位环境是视口（即浏览器中显示网页的小窗口）或手持设备的屏幕，因此当页面滚动时固定定位的元素不会随之移动。图 4-22 和图 4-23 展示了固定定位的效果。

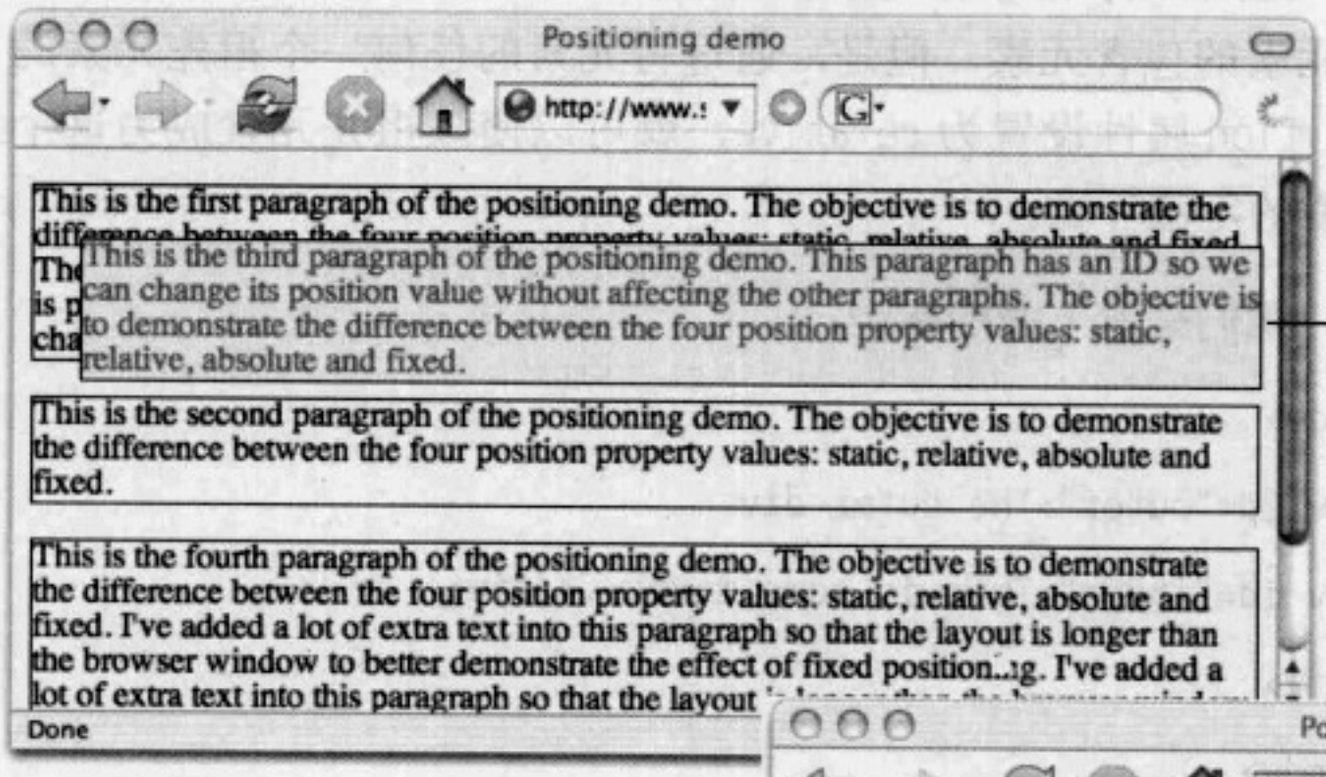


图 4-22 · 固定定位与绝对定位看起来很相似

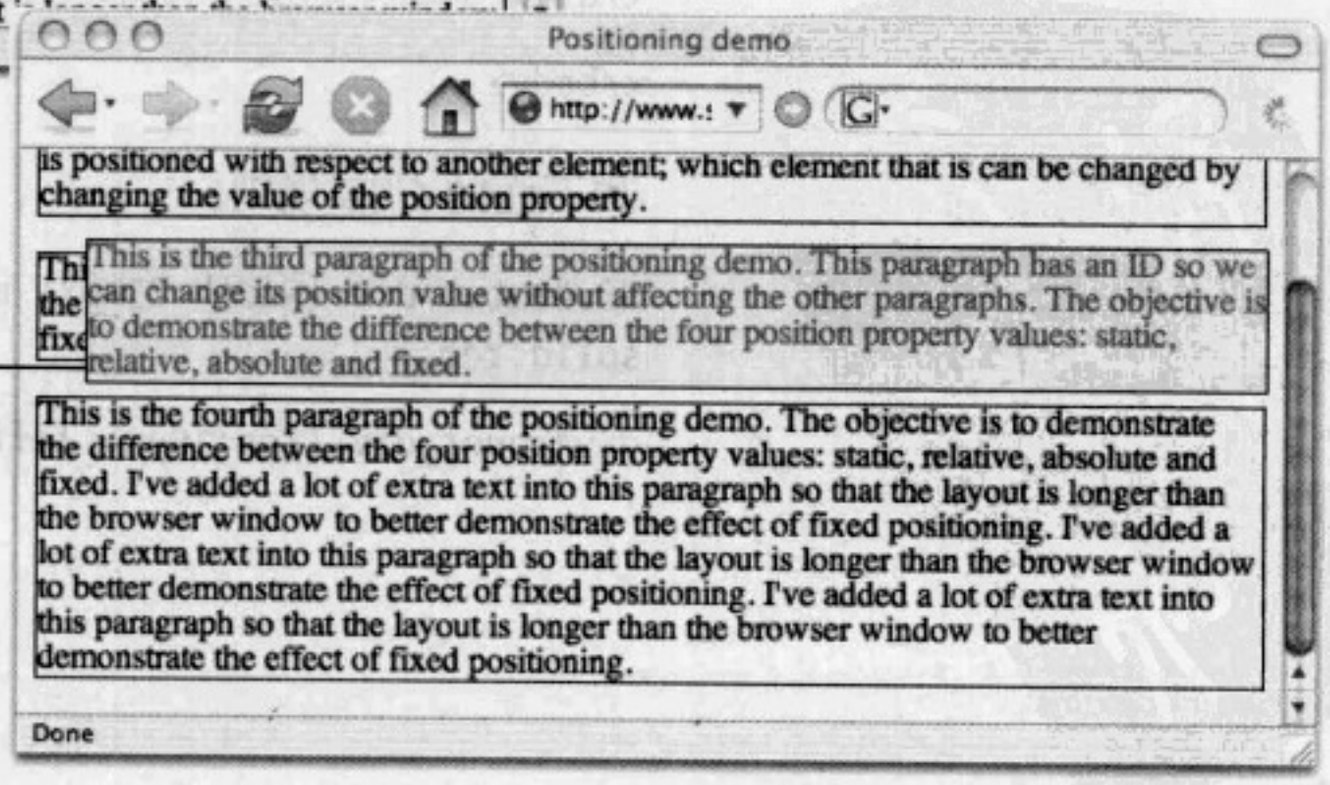


图 4-23 但在滚动页面时，固定定位的元素不会随之移动

这种“固定在浏览器窗口中”的效果可以用来模仿当前已经不推荐的框架。例如，可以用固定定位的元素来创建一个与页面滚动无关的导航条，而且能够省去在框架集中管理多个文档的麻烦。但是，IDWIMIE，position:fixed 在 IE 6 中是无效的，不过 IE 7 能够支持固定定位。在 TagSoup.com（http://devnull.tagsoup.com/fixed）上面可以找到让 IE 6 支持固定定位的的一个巧妙的变通方法。

4.4.5 定位环境

由于定位环境是摆脱表格布局所需掌握的一个极为重要的概念，所以下面我们就来详细地介绍这个概念。简单地说，定位环境就是当我们使用 top、left、right 或 bottom 属性移动元素时，我们会参照另外一个元素移动它，而这个作为参照的元素就称为定位环境。前面我们已经看到了，在绝对定位的情况下，绝对定位元素的定位环境是 body——除非我们明确地改变它。之所以将 body 作为默认的定位环境，是因为它是标记中所有元素的包含元素。但是，通过将元素的任何一个祖先元素的 position 属性设置为 relative，就可以使该祖先元素成为这个元素的定位环境。

我们来看下面的标记：

```
<body>
<div id="outer">The outer div
<div id="inner">This is some text...</div>
</div>
</body>
```

和下面的 CSS 规则：

```
div#outer_div {width:250px; margin:50px 40px; border-top:3px
solid red;}
div#inner_div {top:10px; left:20px; background:#AAA;}
```

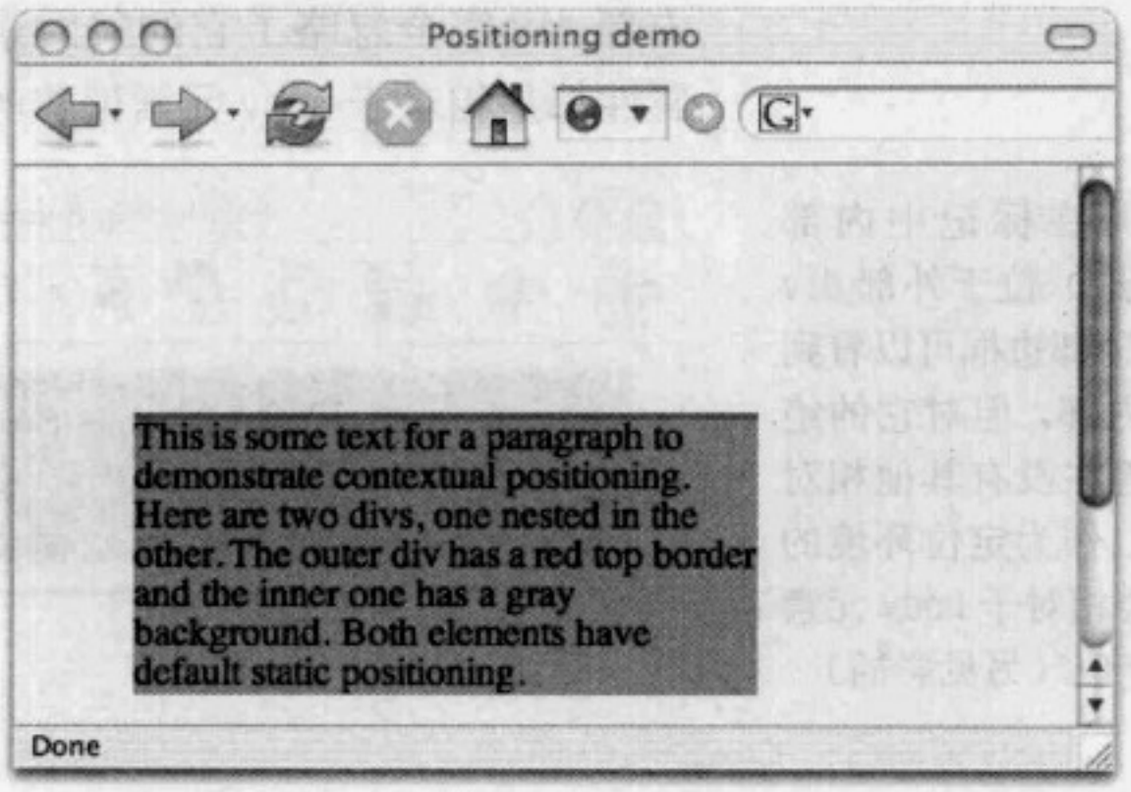


图 4-24 图中是两个嵌套的 div 元素[①]。其中，已经为外部的 div 添加了红色的顶部边框，为内部 div 添加了灰色的背景颜色。由于内部 div 处于静态（默认）定位状态，所以 top 和 left 属性的值被忽略（另见彩插）

任何人都会对这些代码的结果发出疑问：为什么内部 div 没有按照指定相对于外部 div 向下移动 10 像素、向右移动 20 像素呢？事实上这两个元素共享了同一个原点（左上角）。答案是内部（及无关的外部）div 都具有默认的（static）定位。这意味着它们都处于常规文档流中，而且由于外部 div 中没有内容，所以内部 div 与它具有相同的起点位置。换句话说，只有将一个元素设置为另外 3 个定位属性（relative、absolute 或 fixed）之一的情况下，top、left、right 和 bottom 属性才会起作用。下面，我们通过把内部 div 的 position 属性设置为 absolute 来验证这一点。

为了确保语义完整，应该将文本适当地包装在段落中。但在这个例子中出于简洁的考虑，我们把文本直接放在了内部 div 中。

```
div#outer_div {width:250px; margin:50px 40px; border-top:3px solid red;}

div#inner_div {position:absolute; top:10px; left:20px; background:#AAA;}
```

但是，相对于哪里进行绝对定位呢？由于没有其他相对定位的元素作为它的参照点，所以内部 div 会相对于默认的 body 元素进行定位。

因为外部 div 的顶部边框是红色的，所以我们可以清楚地看到它的位置。而且，外部 div 的外边距会使它处于分别上距和左距浏览器窗口左上角 50 像素和 40 像素的位置上。这里，我们把内部 div 的 position 属性设置为了 absolute，意味着它会相对于默认的定位环境 body 进行定位。结果如图 4-25 所示，

① 原文 paragraph 有误。——译者注

内部 div 完全忽略了它的父元素（外部 div），它的 top 和 left 属性都是相对于 body 元素偏移的。

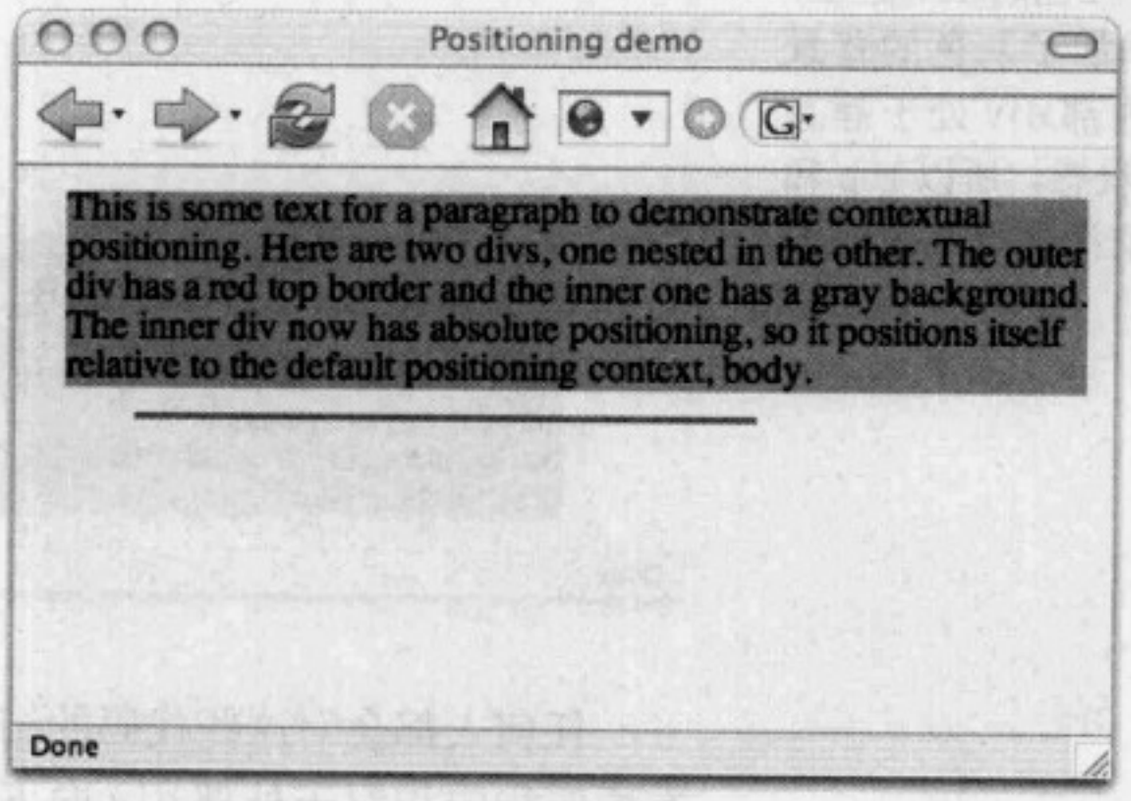


图 4-25 尽管在标记中内部 div（灰色背景）位于外部 div（通过红色的顶部边框可以看到它的位置）内部，但对它的绝对定位意味着在没有其他相对定位元素可以作为定位环境的情况下，它会相对于 body 元素对自身进行定位（另见彩插）

多数情况下，只要能够恰当地使用内边距和外边距，通过静态定位就可以实现所需的页面布局效果。许多刚开始接触 CSS 的设计者由于错误地为几乎所有的元素设置了 position 属性，最终发现对这些获得了自由的元素非常难以控制。因此，不要随意修改一个元素 position 属性的默认值 static，除非确实需要那么做。

如果将外部 div 的 position 属性设置为 relative，那么绝对定位的内部 div 的定位环境就会变成外部 div[①]，如图 4-26 所示。此时，top 和 left 属性设置的偏移量也将基于外部 div 进行计算。如果我们现在将外部 div 的 top 和 left 属性设置为不是 0 的值，那么内部 div 也会随之移动以维持它与外部 div（它的定位环境）之间的相对位置关系。明白了吗？

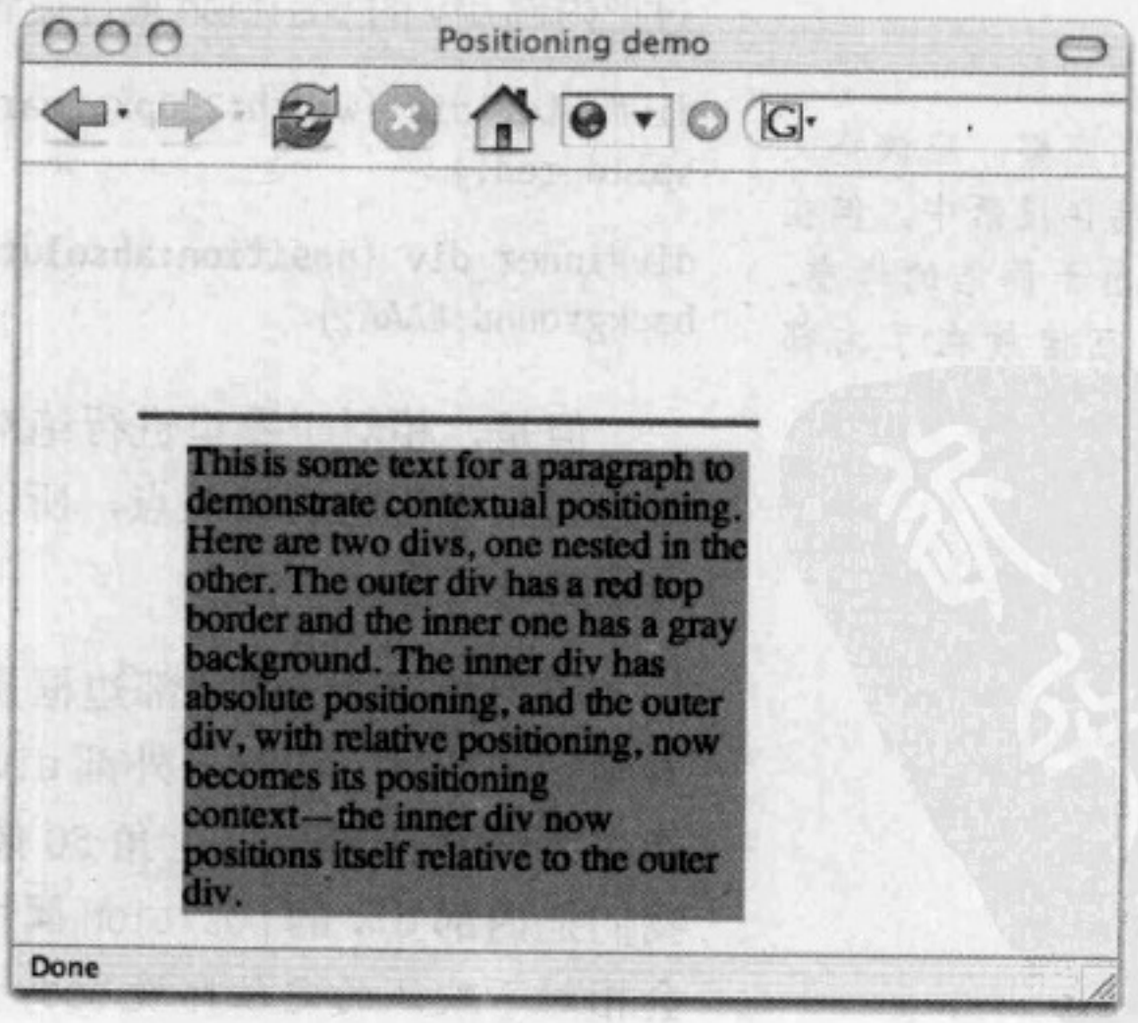


图 4-26 当外部 div 采取了相对定位之后，绝对定位的后代元素就会按照自己 top 和 left 属性的设置，相对于外部 div 进行定位（另见彩插）

① 事实上，将外部 div 的 position 属性设置为 absolute 同样也可以使它成为内部 div 的定位环境。但是这样一来，由于外部 div 也脱离了文档流，可能会失去一些可控性。——译者注

在下面学习完 display 属性之后，我们会介绍 一个使用 position 属性的例子。

4.5 display属性

同每个元素都有 position 属性一样，每个元素也都有一个 display 属性。尽管 display 的属性值相当多，但多数常用元素都有一个默认的 display 属性值——block 或者 inline。假如你在前面几章的课堂上“睡觉”了，那么我们再重复一下块级（block）元素和行内（inline）元素的区别。

- 块级元素（例如段落、标题和列表）在浏览器中显示时，一个位于另一个上方。
- 行内元素（例如 a、span 和 img）在浏览器中显示时，会并肩排列，只有当前一行没有足够的空间时才会显示到下一行。

将块级元素改变为行内元素（或相反）的能力，如

默认为块级元素
```
p {display:inline;}
```
默认为行内元素
```
a {display:block}
```

是非常强大的，通过这种能力，我们可以让一个行内元素填满它的包含元素。后面，当我们创建 CSS 下拉菜单时会在链接中使用这一功能。

另一个值得一提的 display 属性的值是 none。当把一个元素的 display 属性设置为 none 时，该元素以及嵌套在其中的任何元素，都不会再显示在页面中。而且，这个元素原先占据的任何空间都会被移除，就像相关的标记不存在一样（display 属性的这个值与 visibility 属性恰好相对，后者只有 visible 和 hidden 两个值。如果将一个元素的 visibility 属性设置为 hidden，会隐藏该元素，但这个元素占据的空间仍然会得到保留）。下面，我们来介绍如何在鼠标移动到元素上面时，将它的 display 属性在 none 和 block 之间进行切换，以便实现下拉菜单的显示和隐藏功能。JavaScript 也能够在指定的用户操作发生时，通过切换这两个属性使元素显示或消失。这里，我们来看一个组合使用前面所学的 position 和 display 属性的例子。

4.6 使用position/display属性的例子

2007年夏季，我正在为icyou.com（我工作的公司Benefit-focus.com创建的一个有关卫生保健的网站）编写CSS。这个网站中的几乎每个页面都会提供一个由小缩略图像组成的索引，用户通过单击这些缩略图可以播放相关的视频。为了节省屏幕空间，我们决定只在用户鼠标放到缩略图上时再显示相应视频的简介，其效果如图4-27所示。

图4-27 在icyou.com网站中，当用户鼠标放到缩略图上时会显示弹出信息

下面就是实现这种效果的过程。首先，构成缩略图的标记如下：

```
<div class="video_selection">
  <a href="#"><img src="../images/23_diabetes_testimonial-
  1.jpg" alt="Blood pressure video" /></a>
```

```
    <p> Bobby is a #2 Diabetic who weighed 274 pounds. After a
    change in diet he is no longer on medication. Runtime: 46
    sec.</p>

    <h2><a href="#">Living with Diabetes; Bobby's story</a></h2>

</div>
```

这些标记在未添加任何样式的情况下，如图 4-28 所示。

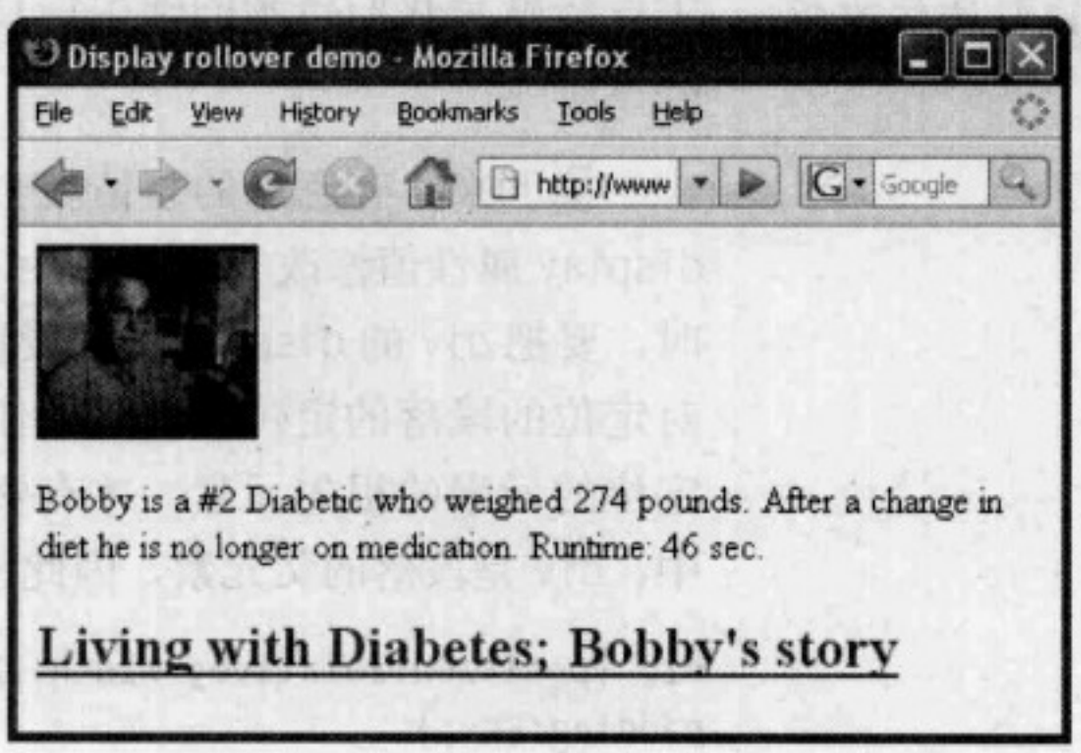


图 4-28 图中是我们用来创建缩略图区域及相关弹出信息的所有元素

下面，我们为这些标记应用一些基本的样式（图 4-29）。

```
div {width:92px; border:2px solid #069; padding:5px;}

h2, p {font-size:.7em; font-family:Arial, sans-serif;
margin:0;}

p {width:80px; border:1px solid gray; padding:.3em;
background-color:#FFD;}
```

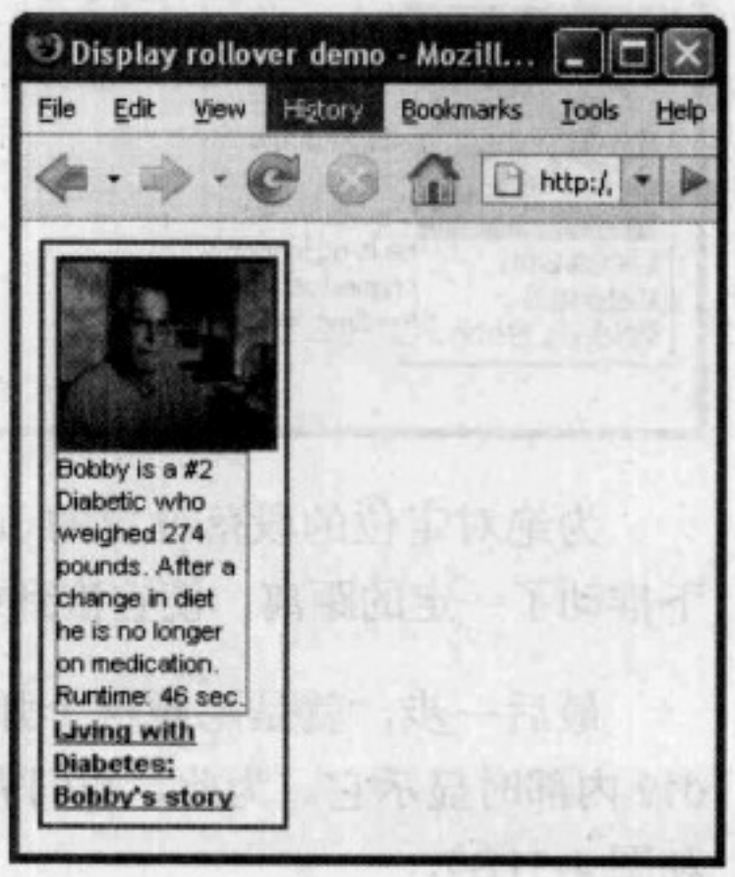


图 4-29 第一步是为元素的外观及宽度应用样式

这里的一系列屏幕截图是在我的 Windows 计算机中截取的。在这台计算机中我使用了 TechSmith 的 Snag-It 捕捉软件，这是一款优秀的屏幕截图工具，通过它能够捕捉到鼠标的指针——本例中的关键。

这里，我们为包含 div 添加了一条边框，也添加了一些内边距，以便其内容与边框之间保持一定距离。（当然，这也会增大声明的宽度！）因此，这个 div 的宽度就是 106 像素（92 + 2 + 2 + 5 + 5 = 106）。此外，我们还为 h2 及段落元素设置了字体，并移除了它们默认的外边距。

对于段落，我们也创建了一个类似的带内边距的盒子。这个段落就是我们将来的弹出信息——只不过现在还没有定位和隐藏而已。

接下来，有意思的事情就会发生了。我们要通过将段落的 display 属性值修改为 absolute，将它从文档流中移出。与此同时，要把 div 的 display 属性设置为 relative，以便使它成为绝对定位的段落的定位环境。记住，相对定位的元素必须是绝对定位的元素的祖先元素，才有资格成为定位环境。在这个例子中，div 是段落的父元素，因此没有问题（图 4-30）。

```
div {position:relative; width:92px; border:2px solid #069;
padding:5px;}

h2, p {font-size:.7em; font-family:Arial, sans-serif;}

p {position:absolute; left:96px; top:15px; width:80px;
border:1px solid gray; padding:.3em; background-color:#FFD;}
```

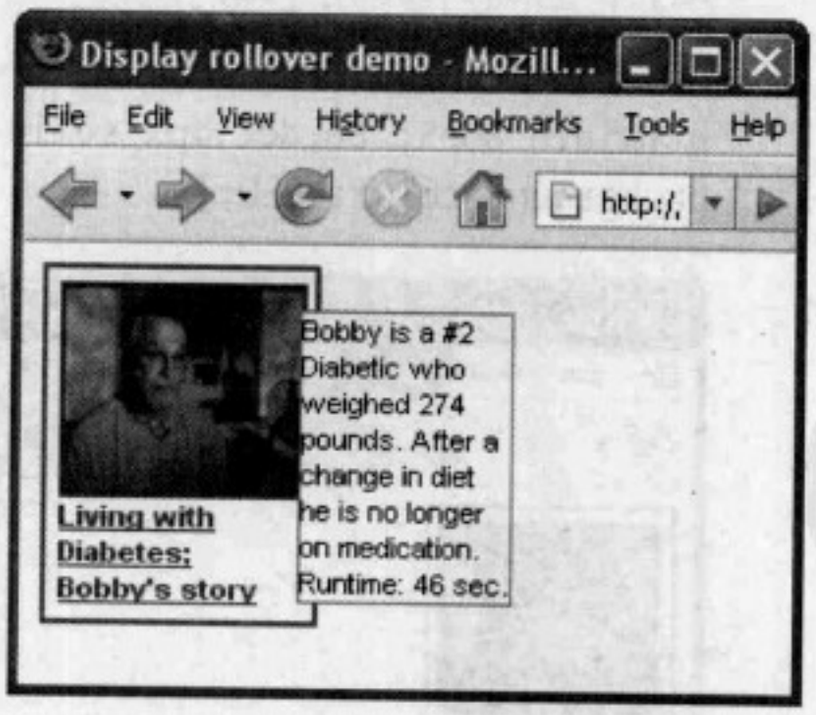


图 4-30 现在，绝对定位的弹出元素已经被定位到了我们希望它相对于包含 div 所出现的位置上

为绝对定位的段落设置的 left 和 top 属性，将它向右和向下推动了一定的距离，使它恰到好处地出现在了图像的右侧。

最后一步，就是隐藏这个弹出元素，并在用户鼠标移动到 div 内部时显示它。为此，我们需要用到 :hover 伪类（图 4-31A 和图 4-31B）。

```
div {position:relative; width:92px; border:2px solid #069;
padding:5px;}

h2, p {font-size:.7em; font-family:Arial, sans-serif;
margin:0;}

p {position:absolute; display:none; width:80px; left:96px;
top:15px; border:1px solid gray; padding:.3em; background-
color:#FFD;}

div:hover p, p:hover {display:block;}
```

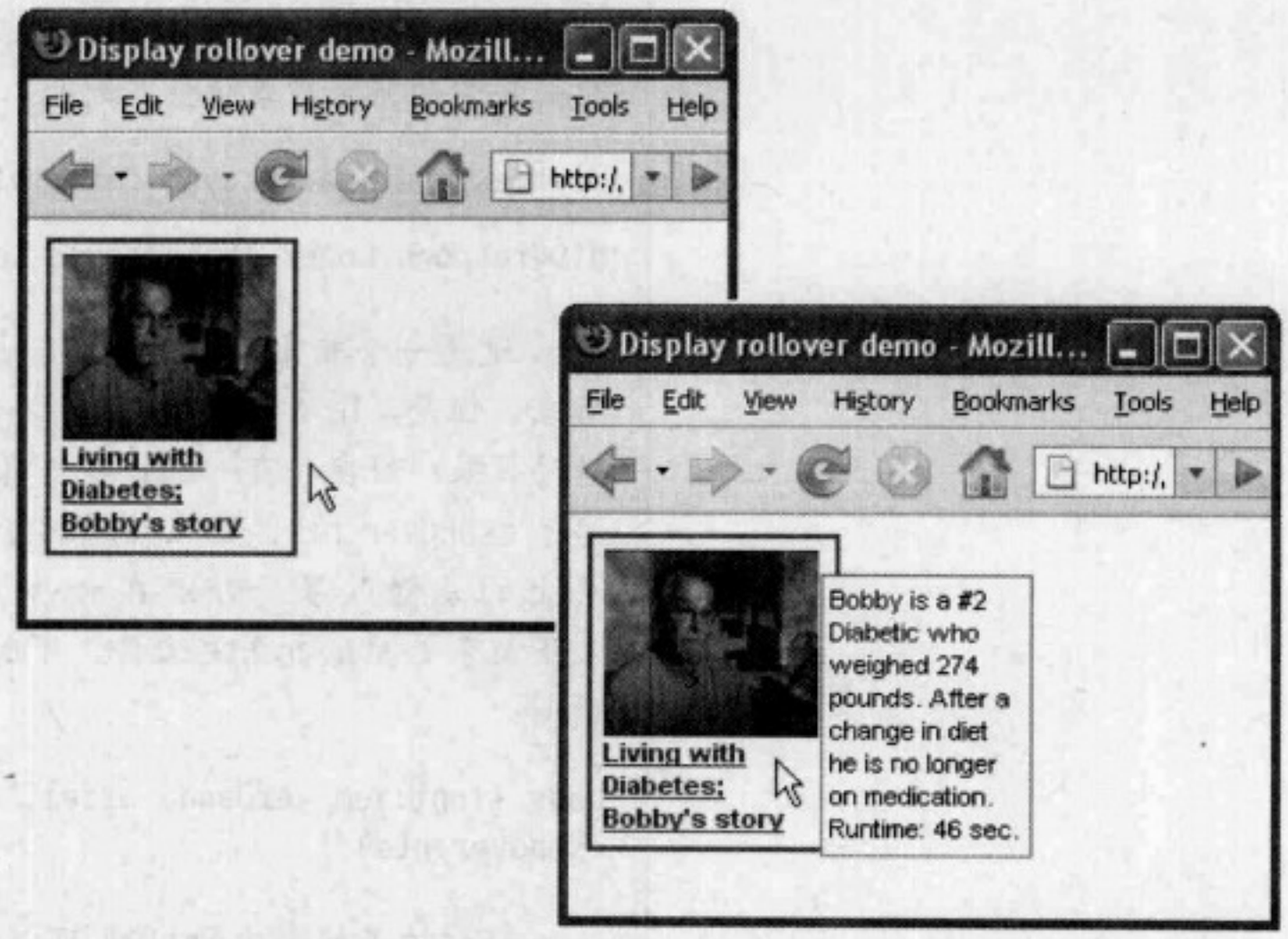


图 4-31A 和图 4-31B　通过组合使用 display 和 :hover 伪类，实现了在用户鼠标指针移动到 div 中时显示弹出信息

这样，正常情况下的段落会隐藏起来，因为我们将它的 display 属性设置为了 none。不过，最后一行突出显示的 CSS 规则说：如果鼠标出现在了段落上（这种情况发生在用户的鼠标从 div 上移开并放到了显示的段落上），让这个段落保持显示。当用户的鼠标移出 div 或者移出段落后，这条规则就不再适用了，因此段落会重新隐藏起来。

于是，仅通过 CSS 我们就实现了一个简单的弹出效果。实际上，这里也存在 IDWIMIE 的问题——IE 6 只支持 a（链接）元素上的 :hover 伪类。因此，我们还需要使用一个 JavaScript 文件（用于修改 IE 行为的名为 csshover.htc 的文件），以便 IE 能够同其他成熟的浏览器一样，响应任何元素上的悬停事件（参见提示条“IE 6 中的悬停行为”）。

```
body {behavior:url(csshover.htc);}
```

请注意，以上代码假设 csshover.htc 文件与 XHTML 文件位于同一个文件夹中。一般来说，你可能会把这个文件连同其他 JavaScript 文件一起，放到网站的 JavaScript 文件夹中，然后再通过相对路径来链接该文件。

IE 6中的悬停行为

在 CSS2 之前，只有链接能够响应鼠标悬停行为。但现在，通过在一个选择符中使用 :hover 伪类，能够为任何元素定义鼠标悬停时的样式变化。比如说，下面的例子会在鼠标悬停到 div 上时，将它的蓝色背景转换成红色：

```
div#respond {background-color:blue;}
div#respond:hover {background-color:red;}
```

这是一个非常有用的特性，也是创建基于 CSS 的菜单的关键所在。但是，IE 6 不支持在除了 a（链接）之外的其他元素上产生悬停效果。好在，有一位非常聪明的程序员——Peter Nederlof，编写了 csshover.htc，该文件通过修改 IE 的行为，使它支持所有元素上的悬停效果。可以在 www.xs4all.nl/~peterned/hovercraft.html 上下载到 csshover.htc 文件。下面是将这个文件添加到 CSS 中的示例：

```
body {font:1em verdana, arial, sans-serif; behavior:url(css/
csshover.htc);}
```

通过这条规则中的 URL 可以看出，我在本例文件的同一个文件夹中，创建了一个名为 css 的新文件夹，并把 csshover.htc 放入其中。如果你想把这个文件放到其他地方，则需要修改这条规则中的 URL。

在本书源文件的 chapter_4（本章）和 js_tools（Stylib CSS 库）文件夹中，都包含了 csshover.htc 文件。

通过以这种方式将 csshover.htc 与文件建立关联，IE 会在任何元素上响应悬停事件。这样，无论在我们选择的哪一种浏览器中，当 div 上发生悬停事件时，都会导致它的背景变成红色。

最后，请读者花点时间回顾并深入理解本章展示的 3 个例子。

- 创建一个带有内部 div 和嵌套元素的分栏。
- 清除浮动的元素。
- 使用 position 和 display 属性控制元素的定位和显示。

这些技术都是通过 CSS 来创建页面布局所必需掌握的，下一章我们就开始讨论页面布局。

第5章

基本的页面布局

在第3章末尾，我们展示了为包含文本的一个单独的长分栏添加样式的例子。虽然在实际开发中，偶尔也会用到该例子中的单栏布局，但为了最大限度地利用水平空间，使用户无需滚动页面就能够看到丰富的信息并与之交互，我们通常都需要构建包含不止一栏的页面布局。

如果仔细观察多数网站页眉之下的区域，就会发现这些网站的页面中至少包含两栏或三栏，即使从视觉上可能不容易分辨出页面采取了多栏布局。本章，我们就来讨论使用 XHTML 和 CSS 来创建这些布局的方法。可以把本章展示的各种页面布局想像成一辆汽车的底盘——虽然它对用户不可见，但却是优秀的网站设计这一闪光车体得以构建的底层框架。在以后的几章中，我们会介绍为一种布局框架添加视觉元素的技术。不过，在此之前我们需要先理解如何构建这些底层的页面布局。

在我们要讨论的这些布局中，有的只是固定宽度分栏的简单排列，有的则提供了一些高级特性，例如“受约束的流动性”（我的术语），即布局能够自动扩展并以最佳的宽度（不过，必须大于一个指定的值）适应各种浏览器窗口。你不必理解这些布局的工作原理也能使用它们。只需以一个 XHTML 模板及位于 Stylib 库中的相关 CSS 文件作为起点，就能立即开始页面的视觉设计并为其添加内容，从而摆脱创建跨浏览器的“底盘”的繁重工作。下面，我们就从观察一些现有的多栏网站及其重要特性开始。

可以通过 http://www.stylinwithcss.com/view_stylib.php[①] 下载 Stylib CSS 库。

5.1 有代表性的多栏布局

分栏的基本用途，就是将导航链接列表组织起来，放到页面主内容区的左侧或者右侧。我们举一个实际网站的例子。Jing 是业内标准的屏幕抓图软件 SnagIt 的创建者 TechSmith 公司开发的一种新的屏幕记录技术，Jing 的博客网站 http://blog.jingproject.com/ 就是流动式两栏布局的极好范例（图 5-1）。

在浏览器中打开这个网站，然后调整浏览器窗口的宽度，观察所谓流动性的含义——主内容区的宽度会随着浏览器窗口的调整而改变，而其中的文本也会随着主内容区宽度的变化而自动换行。但是，不要忽略一个细节——当布局到达不会再变小的最小宽度时，浏览器窗口的右边界会简单地把页面遮挡起来。本章后面，我们会分别讨论这两个特性：流动式布局和最小宽度。

大概最常见的布局还是三栏布局。在三栏布局中，通常左侧是导航栏、中间是内容区，而右侧区域我一般称之为宣传区——包括广告、友情链接、新闻标题及那些频繁出现在页面中的操作链接。图 5-2 是 Amazon.com 中的一个三栏的商品页面，该页面把导航区域整合到了页眉中，因此可以使用两个分栏来展示商品，使用第三个分栏来帮助客户选择购买操作。

① 原文 www.stylinwithcss.com/stylib 有误。——译者注

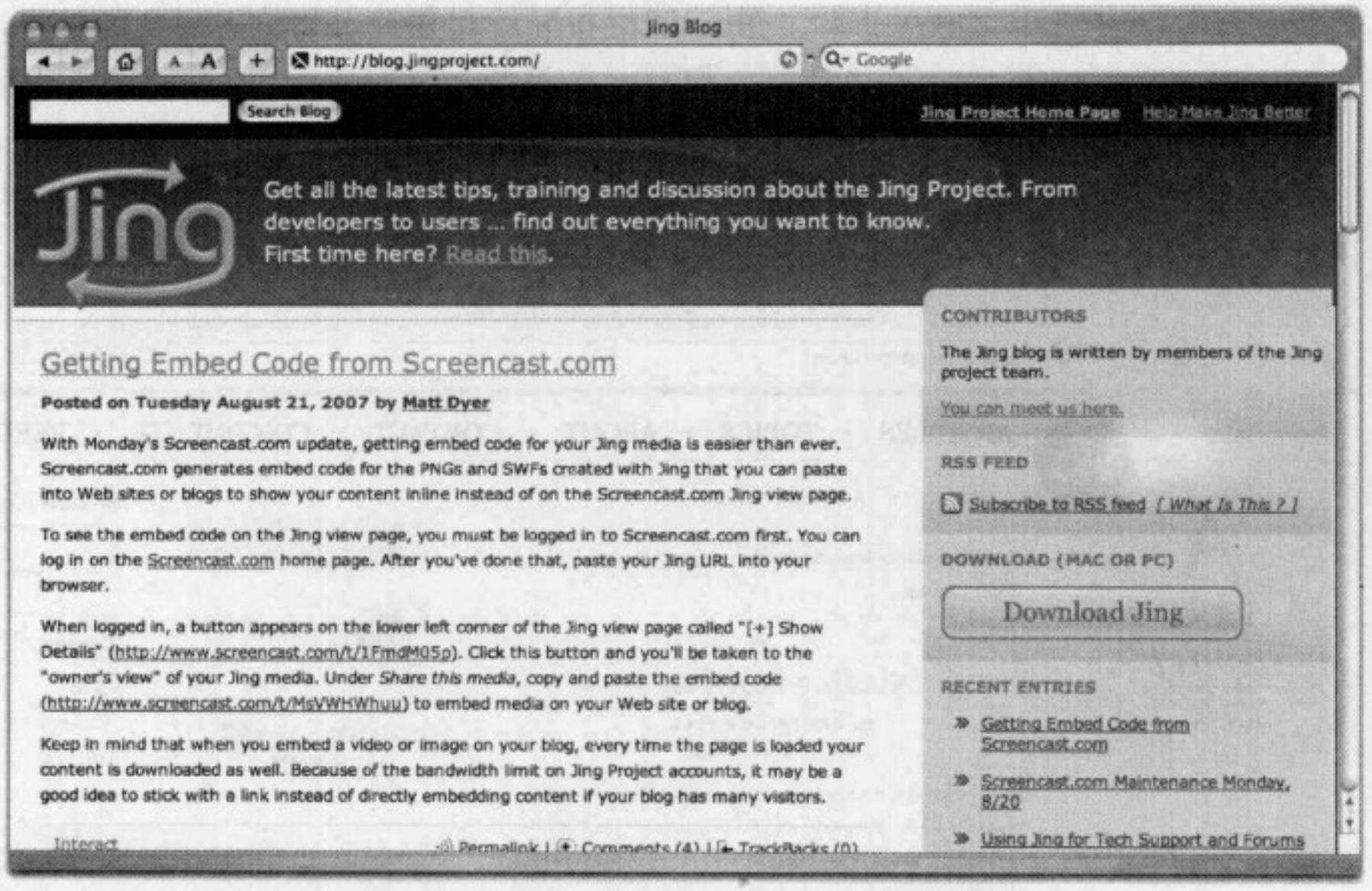


图 5-1 Jing 项目的博客网站使用了两栏的流动式布局

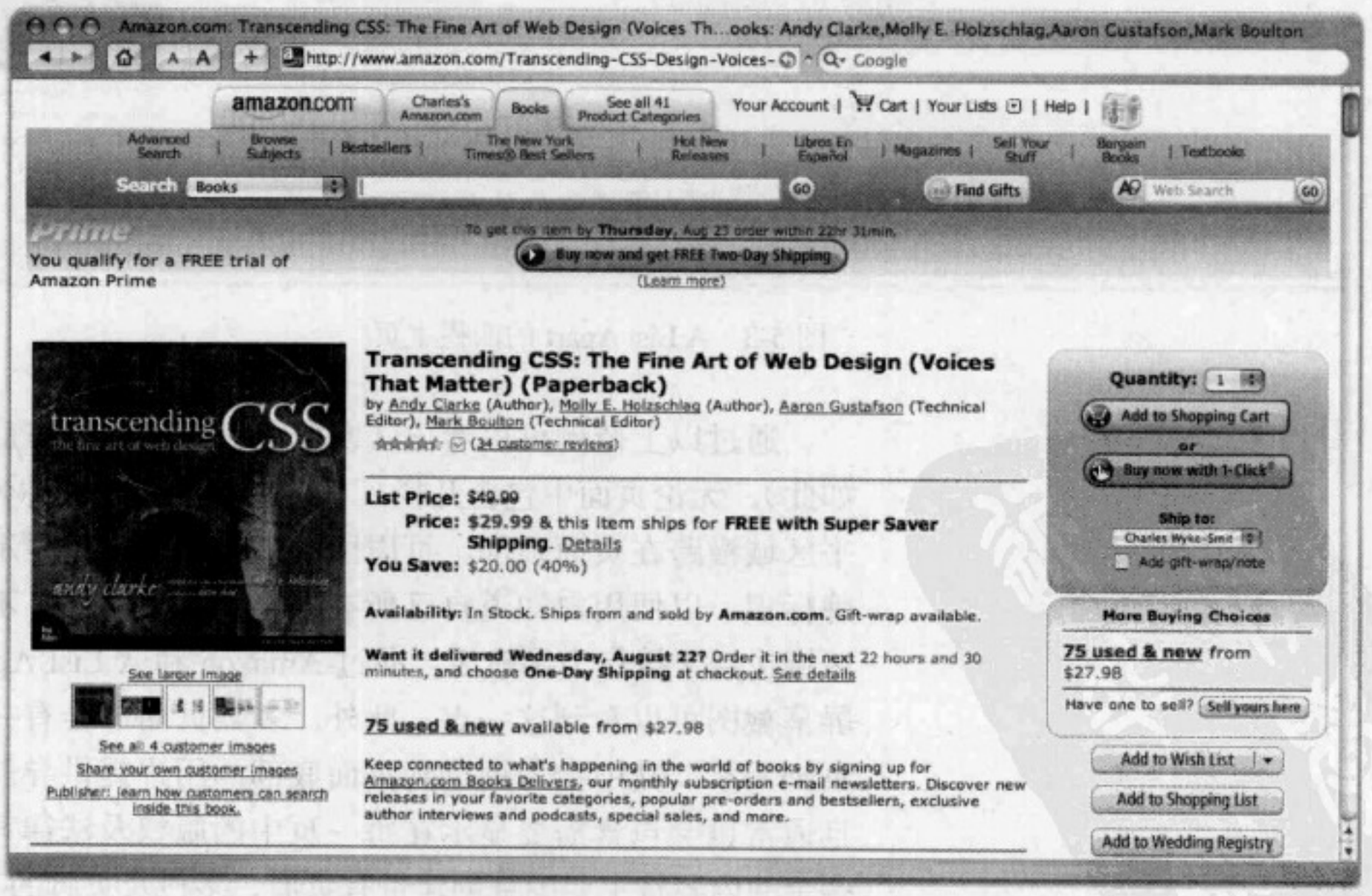


图 5-2 Amazon 的三栏商品页面中的中间栏具有流动性

由于四栏布局看起来很容易给人混乱的感觉，所以很难设计好。但是，在 Jeffrey Zeldman（Web 标准组织的创建人）及其团队的巧妙设计下，A List Apart（http://www.alistapart.com）朴素的四栏主页布局看起来既简洁又不失雅致（图 5-3）。

图 5-3　A List Apart 的四栏主页

通过以上范例可以看出（实际上每个网站的布局也都大致如此），无论页面中包含几栏，都会有一个通常被称为页眉的水平区域横跨在页面顶部。页眉区域的主要用途是展示网站的品牌标识，以便用户知道自己所在的是哪一个网站。不过，页眉区域也经常被作为导航区，通过 Amazon 和 A List Apart 网站的屏幕截图可以看到这一点。此外，多数页面都会有一个对应的页脚元素，其中会为浏览到页面底部的用户提供导航链接，而且通常也会包含需要显示在每一页中的版权及法律声明。本章剩余的内容将主要围绕创建带有页眉、多栏及页脚的布局展示。

5.2 本书CSS库——Stylib简介

在创建网站的过程中，我发现自己经常会重复编写同一段代码。多栏布局、导航链接、表单就是通过 XHTML 和 CSS 编程时，可能会多次遇到的一些组件。尽管不同网站中这些组件的潜在相似性由于受到颜色、字体及其他视觉属性差异的影响通常并不明显，但底层的 XHTML 标记和将这些标记组织到屏幕上的“例行的”CSS 方法则变化很小。最近，为了减少常见的重复性工作，我着手开发了一个名为 Stylib 的库。这个包含 XHTML、CSS 和 JavaScript 文件的库，不仅能够提供设计过程中常用的所有组件，同时还具备两个特点：第一，库中的所有组件需要协同配合，而不是像乐高（Lego）积木一样可以随意拼装；第二，库中需要根据不同网站修改的部分（例如字体大小和颜色）已经从底层的布局代码中分离了出来。这意味着，我能够在不影响菜单及其他已经设计好的组件的前提下，修改颜色和字体大小等视觉属性。

目前，Stylib 库还不是很完善，希望读者对这个库给出反馈和建议。使用这个库的唯一要求就是提供一个有效的 XHTML 页面，并在文档头部嵌入相关的 CSS 规则。而且，只需为包含 div 添加一个类就可以调用这些 CSS。任何为这个库的完善提供必要组件的人都能够获得一份荣誉。

作为本书的一部分，读者可以从本书网站（www.stylinwith-css.com）下载 Stylib CSS 库。在后面的章节中，我们将主要使用这个库。而且，你也可以把 Stylib 作为自己网站设计的基础。这里，我们不想详细地解释 Stylib 库的原理，而是要在实际使用的过程中，介绍它的各种特性。

5.3 宽度问题

在学习本章的过程中，你会发现所有布局都会根据页面中内容的多少，自动增加它们的垂直高度。也就是说，如果我们向页面中添加更多的内容，那么布局会增加其高度以便容纳新增内容。实际上，这也正是我们想要的结果。但是，实现这一点的关键，则是控制这些布局中的水平宽度。用户通常讨厌水平滚动页面，因此确保不出现水平滚动条非常重要。此外，其中多数布局都是通过使用 CSS 浮动的元素创建分栏，在无法维持关键宽度的情况下，会导致这些布局不能正确显示。以上这些都会在我们讨论实例时给出详尽的解释，不过请始终记住：我们要创建能够通过垂直扩展容纳任意数量内容的布局，但是

不会改变它们的宽度。因此，设置和控制布局的宽度也是我们要重点讲解的主要技巧。

5.4 浮动布局与绝对定位布局

在页面布局中创建分栏的基本手段有两种：一种是通过浮动使分栏并肩排列，同上一章我们在图像旁边创建的包含文本的分栏一样；另一种是使用绝对定位，在页面中以固定的宽度和位置排列分栏。

浮动的分栏实现起来简单快捷，但是要求我们必须确保不能意外地导致分栏的总宽度超过布局宽度（例如，由于向其中添加了一幅大图像而增加了分栏的宽度）。一旦发生这种问题，会导致右侧的分栏被挤下去，最终位于左侧分栏的下方——相当于破坏了布局。不过，通过组合使用 overflow 属性和前面展示的“内部 div”的方法（下面的例子中也将用到）能够避免这一问题。而且，我们希望展示的所有布局都会使用这种浮动分栏技术。

绝对定位的分栏有两个明显的好处。首先，可以通过调整标记的先后顺序把内容放在首位，或者提高内容在代码中的位置，这样据说能够改善对搜索引擎的可见性（个人认为，引用链接[①]、适当的 title 标签和对标题及正文中关键字的合理运用是被搜索引擎发现的关键。不过，这个话题还是有机会再谈吧）。其次，在任何情况下分栏都会停留在指定的位置上，也就是说布局不会像浮动分栏那样发生“断裂”。然而，由于分栏采取的绝对定位，它们会被移出文档流，因而分栏之间的关系也就无从谈起。这不仅意味着通过绝对定位难以创建流动式布局，而且也会导致位于所有分栏底部的页脚，不会在页面内容增多时自动地被推向下方——因为分栏完全独立，不会相互影响。虽然通过 JavaScript 能够弥补这个问题（例如本章后面介绍的 Nifty Corners 代码），但我还是倾向于使用浮动分栏进行布局。

① 所谓引用链接就是指一个网站中的网页被其他网站中的网页引用时使用的链接。某网站的引用链接越多，则说明该网站受其他网站的关注程度越高，因此搜索引擎将其排名提前的必要性就越大。对于 Google 而言，引用链接的多少是决定其 PageRank（网页排名）的一个关键指标，参见 http://www.google.com/intl/zh-CN/corporate/tech.html。这一主题涉及 SEO（Search Engine Optimization，搜索引擎优化）方面的内容，感兴趣的读者可以参考相关的文章或书籍。——译者注

这是我们展示的一种最简单的布局，但用来讲解该布局的版面可能是最长的。这种布局中涉及的许多技术也会应用到其他布局中，因此我会在这里详细地解释一遍，在后续的例子中就不再重复了。

本章最后，我们也会展示绝对定位布局的例子，读者可以自己比较这两种方法的优劣。

5.4.1 简单的两栏式固定宽度布局

在第一个例子中，我们介绍一种非常常见的布局形式。这种布局的左侧分栏较窄，用于包含导航链接；右侧分栏较宽，用于包含页面内容。在这个例子中，导航栏和内容栏的宽度都是固定的。而且，整个布局会在浏览器窗口的宽度超过布局宽度时在屏幕上居中（图 5-4）。

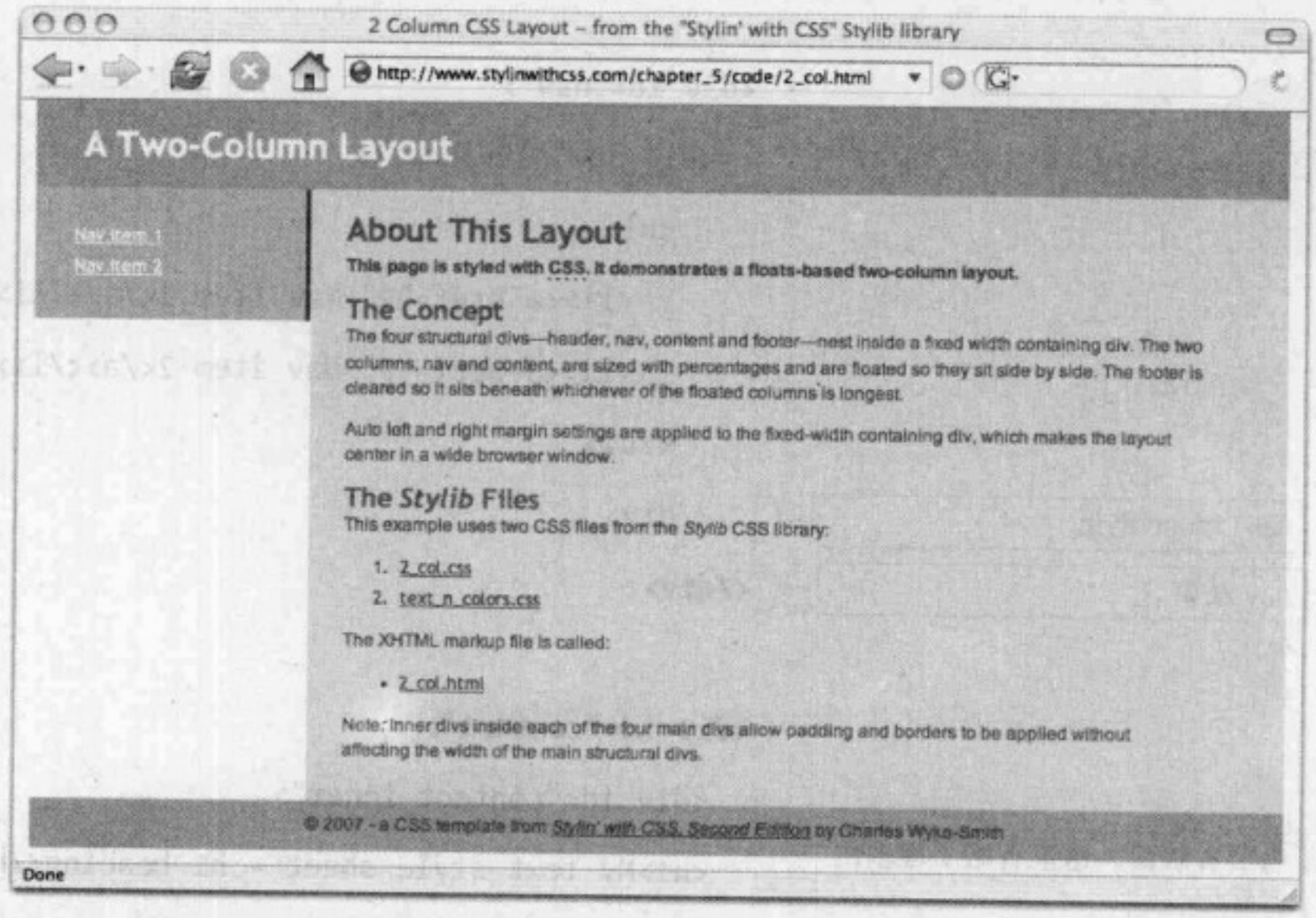


图 5-4 Stylib 库中包含的一个两栏式布局。其中，左侧的分栏只有适应其中内容的高度。本章后面会介绍如何从视觉上扩展这个分栏

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" />
<title>2 column layout</title>
<link href="../../lib/css_styles/layouts/2_col.css"
media="all" rel="stylesheet" />
<link href="../../lib/css_styles/text/text_n_colors.css"
media="all" rel="stylesheet" />
```

布局及文本样式表

```
</head>
<body>
<div id="main_wrapper">
<div id="header">
  <div id="header_inner">
    <h1>The header area</h1>
  </div>
</div>

<div id="nav">
  <div id="nav_inner">
    <ul>
      <li><a href="#">Nav item 1</a></li>
      <li><a href="#">Nav item 2</a></li>
    </ul>
  </div>
</div>

<div id="content">
  <div id="content_inner">
  <h1>My text style sheet - h1 heading</h1>
  <p> A brief paragraph under this heading</p>
  </div>
</div>

<div id="footer">
  <div id="footer_inner">
    <p>This is the footer.</p>
  </div>
</div>
```

header_inner 结束

header 结束

nav_inner 结束

nav 结束

为节省版面，这里省略了很多内容标记

content_inner 结束

content 结束

footer_inner 结束

footer 结束

```
main_wrapper 结束 ──┐  </div>
                    </body>
                    </html>
```

虽然这些代码看起来不少，但如果我们把它分解开来，就会发现关键的部分只有几块。

首先，在大约 1/3 的地方找到结束的 head 标签 </head>。在此之上，是浏览器显示页面所需的代码，而下面，在 body 标签之后，则是实际会显示的页面内容。

从代码的最顶部开始，在必需的 DOCTYPE 和 XHTML 标签之后，就是文档的头部，其中包含 title 标签和两个 link 标签。其中，link 标签从 Stylib 库中向这个页面链接了两个样式表，它们分别为页面布局和文本提供样式。稍后我们会详细地分析这两个样式表中的 CSS 规则。

文档的主体中包含 4 个 div，其 id 值分别为：header、nav、content 和 footer。而且，这 4 个 div 中都包含各自的内部 div。下面，我们就来看一看怎样对这些 div 进行布局，才能使 nav 和 content 这两个 div 并肩排列，同时让 header 和 footer 以全宽形式分别显示在它们的上方和下方。以下就是完成这一任务的 CSS 规则，它们位于链接的样式表 2_col.css 中（代码上方突出显示了链接这个样式表的代码）。

另外一个链接到这个页面中的样式表是 texst_n_colors.css，该样式表也位于 Stylib 库中。本章的每个例子都会用到 texst_n_colors.css。这个样式表用于设置分栏的背景颜色并为页面中的文本元素添加样式。通过为 body 标签添加一个类，例如“olive”或“lime”，可以在这个样式表中包含的不同页面配色方案中进行切换。如果 body 标签中不包含这样一个类名（例如上面的例子），那么就会使用样式表中的默认方案。由于这个样式表并不是本章的重点，所以我们不会在这里展示它。而且，在包含所有配色方案的情况下，这个样式表中的 CSS 规则代码差不多有 400 行。不管怎样，我们的重点是通过 CSS 来组织页面布局。因此，只要知道页面中的颜色和文本样式，都来自 text_n_colors.css 样式表并且与页面布局无关就可以了，我们会在第 7 章详细介绍这个样式表。

```
body {
    text-align:center;
    }
```

需要通过这条规则使布局在 IE 6 的浏览器窗口中居中

```
#main_wrapper {
    width:840px;
    margin-left:auto;
    margin-right:auto;
    text-align:left;
}
```

当这个宽度改变时，分栏的宽度也会随之成比例地改变

在浏览器中居中布局

重置 body 标签中为 IE 6 而设置的 hack

```
#header {

    }
#nav {
    width:22%;
    float:left;
    }
```

这里的宽度加上内容区的宽度必须等于 100%

浮动 nav 和 content 这两个 div，使它们并肩排列

```
#content {
    width:78%;
    float:left;
    top:0px;
    }
#footer {
    clear:both;
    }
#header_inner, #nav_inner, #content_inner, #promo_inner {
    overflow:hidden;
    }
#header_inner {
    padding:1em 2em;
    }
```

这里的宽度加上导航区的宽度必须等于 100%

浮动 nav 和 content 这两个 div，使它们并肩排列

确保页脚位于较长的分栏下方

裁剪（隐藏）过大的元素以免由于它们扩展而破坏布局

在盒子与内容之间创建间距

```
#nav_inner {
    padding:1em .8em;
    border-right:3px solid #B33;
    }
#content_inner {
    padding:0 1em 1em 1.5em;
    }
#footer_inner {
    padding:.5em 1em;
    text-align:center;
    }
```

在盒子与内容之间创建间距

在盒子与内容之间创建间距

在盒子与内容之间创建间距

如果想改变两栏的宽度，可以修改它们的百分比宽度值。但是，一定要确保两栏的宽度和等于100%。

在默认情况下，如果没有这些CSS规则，那么标记中的4个主div（XHTML中突出显示的部分）将会保持与浏览器窗口同宽，并且相互堆叠在一起。把这种默认的布局转换成两栏布局的CSS规则，只涉及上面CSS代码中突出显示的5行代码。其中，我们浮动了nav和content这两个div，以便它们并肩排列，而且为它们应用了总和为100%的百分比宽度，从而使它们能够与页眉和页脚保持同宽。

通过为页脚应用clear:both规则，确保了它总是位于两个分栏中由相应内容决定的最长的一个分栏下方。剩下的事情就由文档流替我们完成了——页眉和页脚在默认的情况下会与包含元素main_wrapper同宽。

顾名思义，main_wrapper的div封装了整个布局，并且我们随便为它指定了840像素的宽度。由于这个div是前面4个主div的父元素，因此通过设置这个宽度，就相当于设定了页眉、页脚以及两个分栏组合起来的宽度。这种布局的好处是双重的。首先，只通过调整main_wrapper的宽度，就可以修改整个布局的宽度（内部元素会随之成比例地变化，无需修改它们的代码）。其次，通过将main_wrapper的左和右外边距的值设置为auto（参见上面的代码），当用户将浏览器的宽度调整到超过main_wrapper的宽度时（当前的840像素），整个布局会在浏览器窗口中居中显示。

IDWIMIE 6——对于IE 6，我们需要多做一点工作才能实现布局的自动居中效果。另外，IE 6也不支持min-width和max-width属性，因此需要添加一个特殊的JavaScript文件来解决控制布局最小宽度的问题。在Stylib库的js_tools[①]文件夹中，可以找到一个名为minmax.js的文件。只要把该文件通过下面的代码链接到页面中即可：<script type="JavaScript" src="../../lib/js_tools/minmax.js"></script>。应该把以上代码放在页面的头部，而且通常是位于链接CSS文件的代码后面。另外，应该根据放置Stylib库的位置来修改其中的文件路径。

5.4.2 理解内部div

在我们介绍这个页面中的内容之前，还需要说明另外一件事。在样式表中，我们把针对内部div的CSS规则放在了末尾，也就是放在了用于页面布局的CSS规则之后。有关这些内部div的用途，我们在第4章已经介绍过了。虽然使用它们会多用一些标记，但却能够极大地简化为布局添加样式和修改布局的工作量。通过内部div可以解决在为分栏应用外边距、边框和内边距时，经常会遇到的盒模型的问题。

对于带有关键性宽度的元素，我倾向于为它们嵌套一个内部div。如果不这样做，那么即使为该元素周围添加1像素宽的边框，都必须记住从这个元素的宽度中减去两个像素（左和右），才能保持这个元素的宽度不变。这样，不仅会令人心烦意乱，而且最终会使CSS中声明的宽度无法反映该元素的实际宽度。然而，有些设计者不喜欢在XHTML中添加额外标记的做法。如果你也不认同这样做，那么对于后面展示的所有例子，你都可以从标记中删除内部div（不要忘记也删除相应的结束标签！），同时把为它们定义的样式转移到主div上——比如，从inner_nav到nav。如果你愿意这样做，那么请记住必须要从声明的div宽度中减去左、右边框及内边距的总宽度。下面，我们来看一看与这些内部div相关的CSS规则。

5.4.3 防止不必要的溢出

有时候，也可能需要移除在div上设置的overflow属性。例如，当这个div中包含着当用户指向它时会显示的带有下拉菜单项的元素时。在这种情况下，如果仍然将overflow设置为hidden，那么下拉菜单将不会出现在这个div区域的外部——此时就有必要移除overflow声明。也就是说，overflow声明只是为了防备由于过大的内容导致布局分裂而设置的。

在前面的CSS中，我们通过一个分组选择符，为所有内部div添加了一条规则，用于隐藏它们内部过大元素的溢出部分。

CSS中的overflow属性，用于控制元素如何处理它们包含的内容。这个属性的默认值是visible，即如果包含的内容过大，那么元素需要扩展以适应内容。例如，在向较窄的nav分栏中添加一幅大图像时，在正常情况下，该分栏应该在垂直和水平方向扩展，以便显示整幅图像。这种对分栏宽度的意外改变，是破坏浮动布局的罪魁祸首。这个问题会导致在向右推动最右侧的分栏而又没有足够的空间容纳该分栏时，最右侧的分栏突然被挤到下面，并停留在左侧分栏的下方。通过应用overflow:hidden规则，能够使较窄的分栏在上述情况下保持它的宽度不变，只显示图像中与分栏宽度适应的部分，因而不会影响到布局的宽度。当然，首先不添加过大的元素是最好的办法，但有时候想控制将来的内容是很困难的。

① 原文JavaScript有误。——译者注

如果我们为 nav 添加 3 像素宽的边框，那么布局的宽度将会是 100%（浏览器是根据父元素计算的，因此就是 840 像素）再加上 3 个像素。这就会导致布局宽度超过 main_wrapper 的宽度，因此浮动的内容分栏会移动到 nav 分栏下方。你可以暂时把这个边框样式从 nav_inner 转移到 nav 上试验一下，就会明白我的意思了。

5.4.4 按照需要为内部 div 添加样式

可以随意为内部 div 应用边框、外边距和内边距。由于没有明确的宽度，所以它们始终会填满各自所在的分栏 div。如果想为导航栏的底部加一条红线，那么可以通过添加到内部 div 上来实现。这样一来，既可以实现相应的效果，也不会因为影响关键的“外部”div 的宽度而导致破坏布局。也就是要牢记一点：别为那些构成主分栏的 div 直接添加视觉样式，要添加就给它们各自的“内部”div 添加。

5.4.5 为文本添加样式

本例中的页面还链接了一个名为 text_n_colors.css 的样式表，这个为文本元素添加样式的样式表也包含在 Stylib 库中。任何标记良好的文本在使用这个样式表时，都应该能够很好地显示。每个人对文本样式的偏好不同，这在我提供的样式表中也有所体现；比如，我喜欢让标题与它后面的文本间距更小一些。由于默认情况下，浏览器会为标题下方设置一个相当大的外边距，所以我在样式表中就将标题的 margin-bottom 设置为 0。

5.5 简单的两栏流动式布局

专为小屏幕设计的布局在大屏幕上看起来就像是一张邮票。因此，创建能够适应任意屏幕大小的流动式布局对用户而言会更加友好。为使现在的两栏布局具有流动性，需要移除 main_wrapper 上面的固定宽度设置，以便布局的宽度能够随着浏览器窗口宽度的变化而变化 。

在 main_wrapper 上，只要进行如下超级简单的修改：

```
width:840px;
```

固定宽度布局就会变成流动式布局。只要删除为 main_wrapper 设置的 width:840px; 声明，布局立即就会随着浏览器窗口宽度的变化而变化。此时，未限制尺寸的 main_wrapper 会扩展到与其父元素 body 同宽，而 body 的宽度在默认情况下就是浏览器窗口的宽度。由于两个分栏都使用了百分比宽度，因而它们也会随着浏览器窗口宽度的变化而成比例地变化，于是就

实现了“流动性”。而且，因为布局随时会填满浏览器的水平空间，所以自动的外边距也就形同虚设了[①]（图 5-5 和图 5-6）。

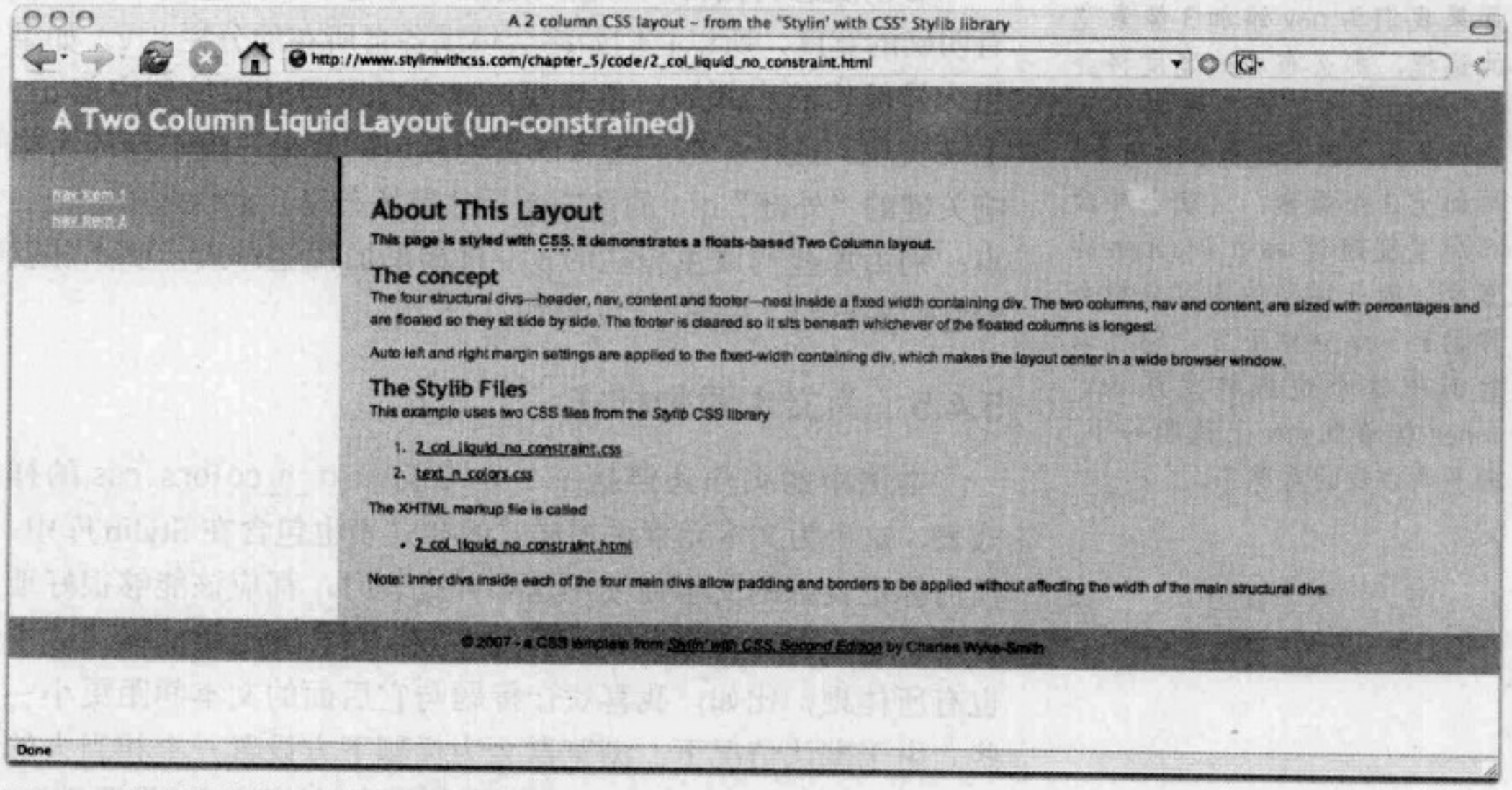


图 5-5 这是我们将固定布局转换为流动式布局的第一步。当浏览器的宽度改变时，分栏的宽度也会成比例地变化。不过，要注意的是，当浏览器窗口过宽时，文本行的长度也将因为过长而影响易读性

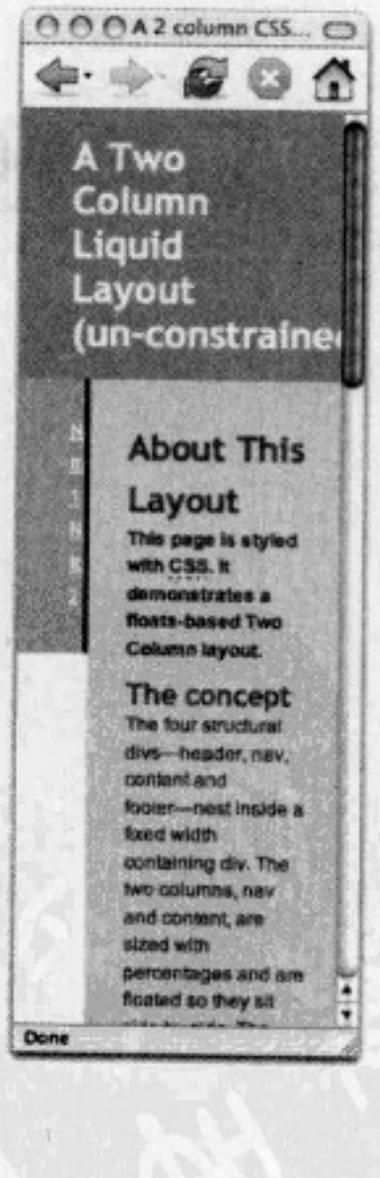


图 5-6 当同一个两栏流动式布局在非常窄的浏览器中显示时，内容会被向下挤压——行的长度变得过短，而导航链接几乎完全看不见了

① 对这个例子而言，自动外边距的工作原理是用 body 元素的宽度减去 main_wrapper 的宽度，将得到的差值除以 2，把这个结果值作为 main_wrapper 的左、右两外边距。因此，在 main_wrapper 的宽度小于 body 元素的宽度时，自动外边距会使固定布局在页面内居中。但当 main_wrapper 没有宽度限制后，以上的所说的差值就是 0，因此自动外边距也就没有什么意义了。——译者注

同第一个布局（及所有布局）一样，我们需要在页面头部使用 script 标签从 Stylib 库中链接 minmax.js 文件，以确保布局在 IE 6 中能够自动居中。

当前状态下的这个布局存在一个主要的用户体验问题，而且这也是许多不限制宽度的流动式布局的共性问题：在浏览器窗口决定布局宽度的情况下，布局的视觉效果可能会失控。如图 5-5 和图 5-6 所示，当流动式页面被挤压到一定程度时，一行文本中可能只有几个单词；另一方面，当通过大显示器浏览该页面时，由于一行文本过长，很可能会使用户在看到行尾时找不到下一行的开头。此外，我们通常只希望内容分栏而不是导航分栏的宽度改变。下面，我们就来讨论一下如何通过修复这些问题来改进我们新的流动式布局。

5.5.1 使用一点限制

CSS 属性 max-width 和 min-width 的作用是限制元素的最大和最小宽度。通过为 main_wrapper 应用这两个属性，可以将布局限制在一定的宽度范围内流动，比如：

```
width:840px;
max-width:960px;    /* 设置最大布局宽度 */
min-width:720px;    /* 设置最小布局宽度 */
```

我一般希望一行文本中最少包含 6 个、最多包含 15 个单词，你可以通过设置最小和最大布局宽度来限制自己的页面范围。如果你使用 em 来设置最小和最大宽度，那么即使在用户缩放整体字体大小的情况下，每一行的单词数量也会始终相同。

这里，我们声明了布局能够在原先 840 像素固定宽度的基础上扩展或收缩 120 像素，但不会再多了。

现在，当页面达到指定的最小或最大宽度时，就会像为它应用了固定的宽度一样。在达到最小宽度后，浏览器窗口的右边就会移动到已经保持固定宽度的布局上。相反，如果浏览器窗口扩展到超过了最大尺寸，那么布局也不会再加宽。此时，自动外边距又会发挥它的作用，结果会使“最大化”的布局像我们在前面固定宽度布局的例子中看到的一样，在浏览器窗口中居中。这种效果显然要好很多。

以下两处简单的修改会确保只有内容分栏会改变宽度：

```
#nav {
    width:22%;
    width:140px;    /* content 中的左外边距必须与这个值匹配 */
    float:left;
    }
```

```
#content {
  float:left;
  width:78%;
  margin-left:140px;
  }
```

左外边距必须等于 nav 分栏的宽度

由于已经为导航分栏设置了固定宽度，那么让内容分栏保持流动的唯一有用值就是 auto，也就是与包含元素——即 main_wrapper 一样宽（因为 width 的默认值就是 auto，所以我们不需要在 CSS 中声明它）。当然，如果内容分栏的宽度与父元素相同，那么也就没有导航分栏的空间了。因此，我们需要为内容分栏添加 140 像素的外边距，以便创建视觉上的空间。

5.5.2 浮动还是不要浮动

如果此时仍然浮动内容分栏[①]，那么内容分栏就会下滑到导航分栏的下方。由于它的宽度与父元素相同，而且还有导航分栏占据了左上角，所以该位置就是它能够向上浮动的最高点。

通过取消对内容分栏的浮动，可以使它返回文档流并从包含元素[②]的左上角开始排布。此时，导航分栏仍然向上浮动，因此也会占据 main_wrapper 的左上角并停留在由内容分栏左外边距产生的空白区域上面。

以上这些细节理解起来有点难度，但归结这里我们所完成的，就是将这两个元素的左上角与包含元素（main_wrapper[③]）的左上角对齐。由于浮动的 nav 不在文档流中，所以它会向左向上浮动并停留在包含元素中最高最远的位置上。而 content 现在则是包含元素文档流中的第一个元素，因此也会从包含元素的左上角位置开始排布。只是由于设置了它的左外边距，才使该分栏的左边缘位置远离了包含元素，从而避免了栖身于导航分栏之下。如果你想试验一下，可以临时移除内容区域的左外边距看一看结果。

在进行了以上修改之后，我们就获得了对用户友好的“流动但受限制”的布局。这个布局也可以用作我们网页设计的基础框架（图 5-7 和图 5-8）。

① 原文疑有遗漏，因为上面的代码中已经删除了内容分栏的浮动声明。——译者注
② 原文 top-left corner of the page 不准确，应该是 containing element 或 main_wrapper。——译者注
③ 原文 page 有误。——译者注

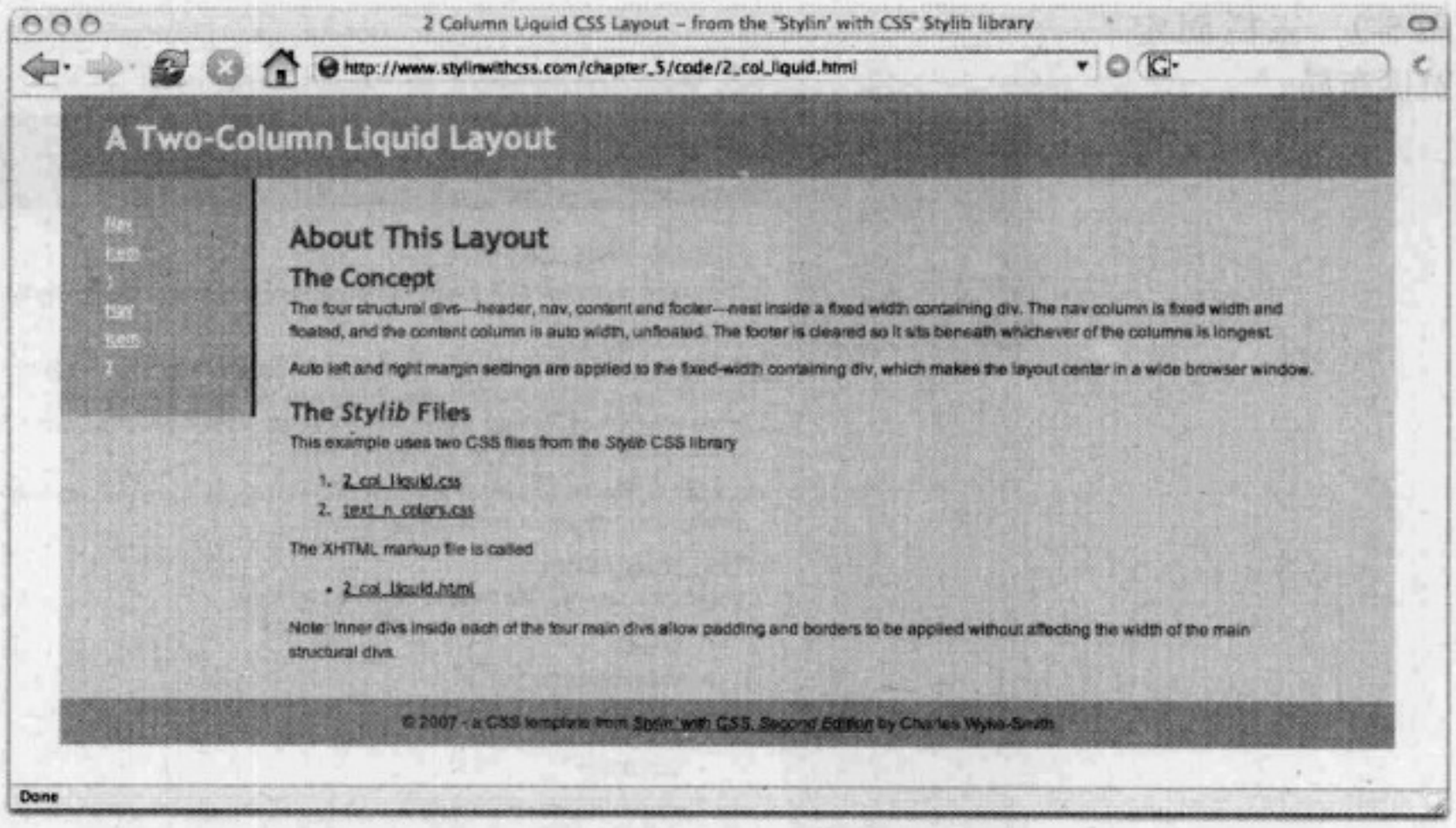


图 5-7 在设置了最小 / 最大宽度的情况下，当布局达到最大宽度后，它会在浏览器窗口中居中

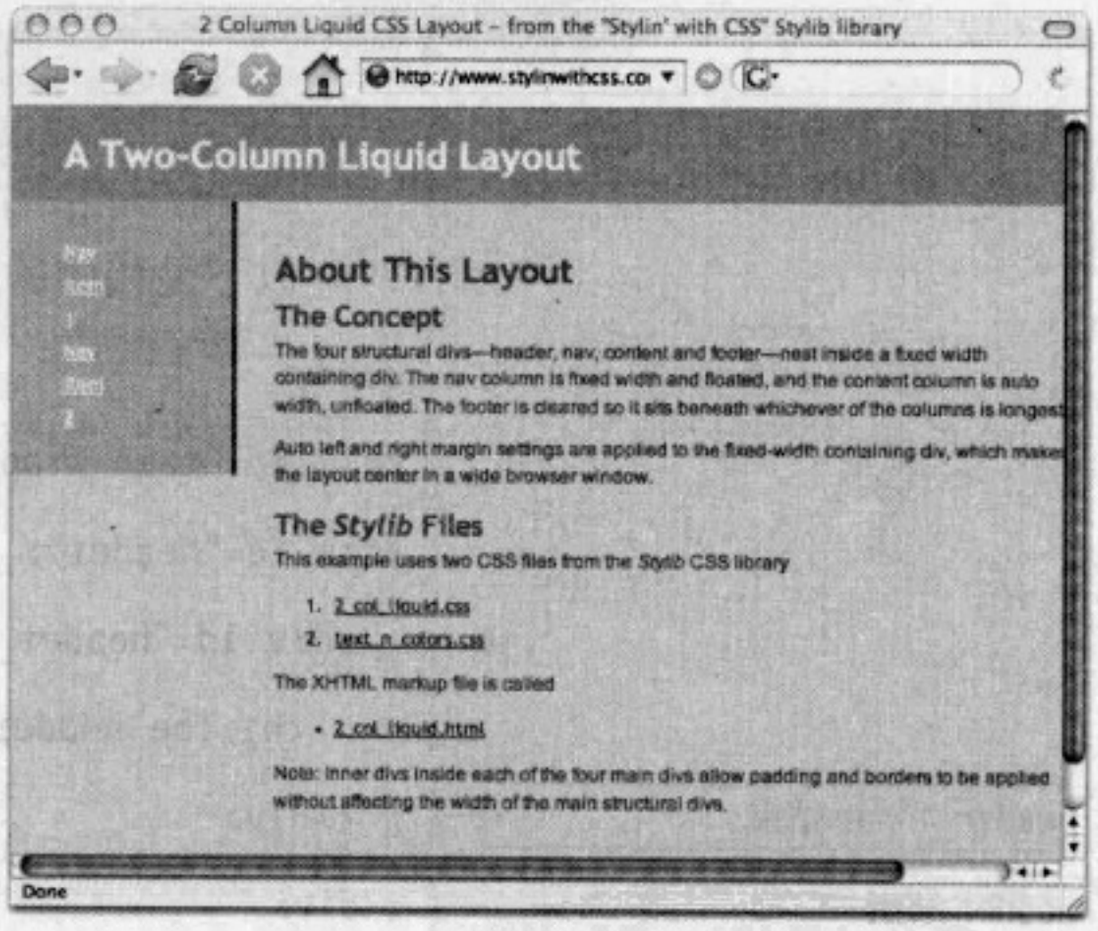
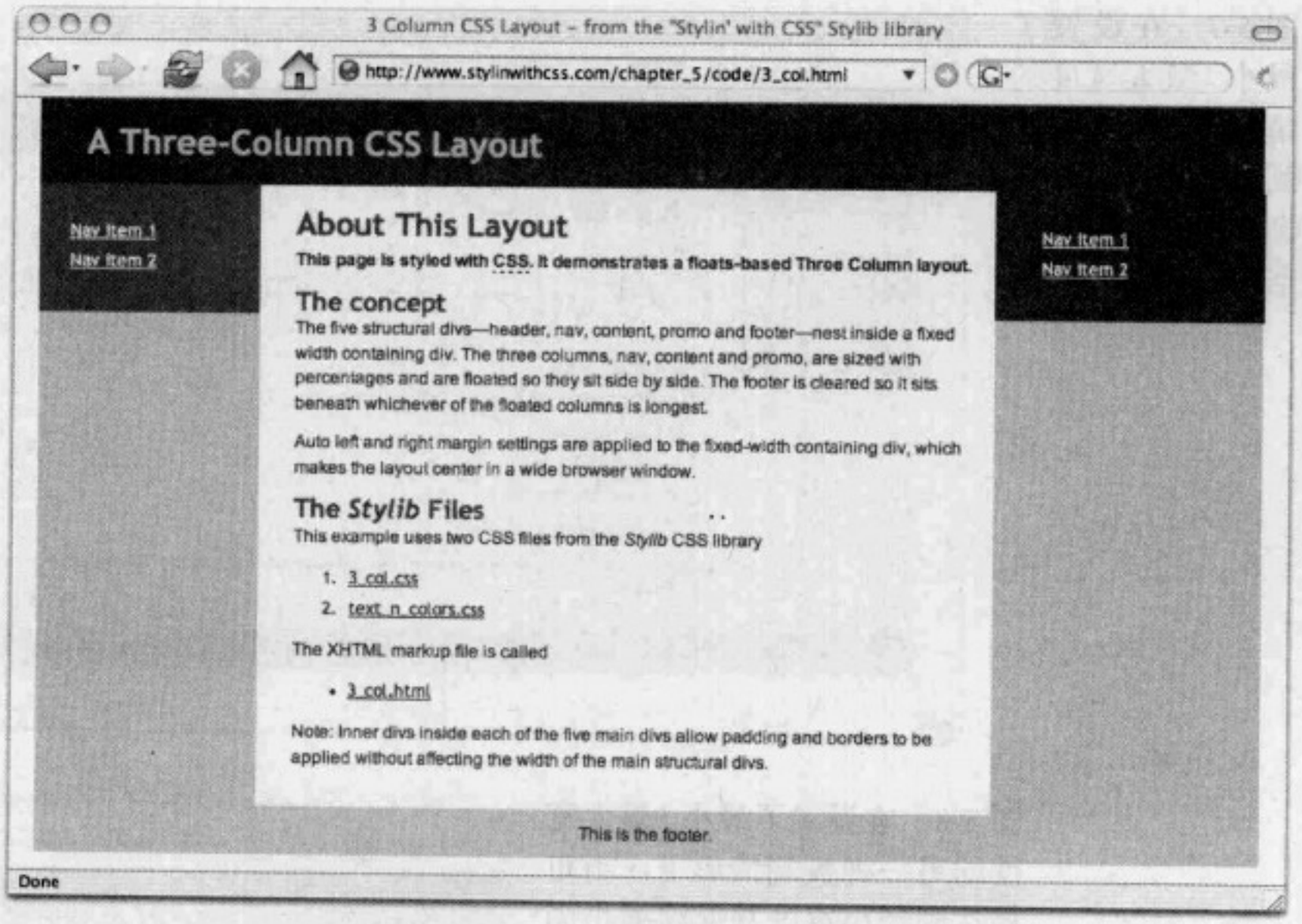


图 5-8 在设置了最小 / 最大宽度的另一种极端情形下，当布局达到最小宽度并继续缩小浏览器时，浏览器窗口会遮住布局的一部分

5.6 三栏式固定宽度布局

固定宽度的三栏布局同前面的两栏布局原理相同。唯一多出来的工作量，就是向标记中添加另外一个分栏 div，然后像浮动其他分栏一样浮动它，并且与其他两个分栏共享 100% 的宽度即可。下面就是新的标记代码。其中，与两栏标记的区别主要就是位于 content div 后面的新的 promo div（突出显示的部分），结果如图 5-9 所示。

图 5-9 三栏固定宽度布局



以下是相应的 XHTML 代码：

```
<body>
  <div id="main_wrapper" class="clearfix">
  <div id="header">
    <div id="header_inner">
      <h1>The header area</h1>
    </div>
  </div>

  <div id="nav">
    <div id="nav_inner">
      <ul>
        <li><a href="#">Nav item 1</a></li>
        <li><a href="#">Nav item 2</a></li>
      </ul>
    </div>
  </div>
```

header_inner 结束

header 结束

nav_inner 结束

nav 结束

```
<div id="content">
  <div id="content_inner">
    <h1>A 3 Column CSS Layout</h1>
    <p><strong>This page is styled...
```

为节省版面此处省略了内容

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

content_inner 结束

content 结束

```
<div id="promo">
  <div id="promo_inner">
    <ul>
      <li><a href="#">Nav item 1</a></li>
      <li><a href="#">Nav item 2</a></li>
    </ul>
  </div>
</div>
```

promo_inner 结束

promo 结束

```
<div id="footer">
  <div id="footer_inner">
    <p>This is the footer.</p>
  </div>
</div>
</div>
</body>
```

footer_inner 结束

footer 结束

main_wrapper 结束

为了节省版面这里省略了文档的头部标记，但该部分的代码与上面两栏固定布局是相同的，只不过其中前两行所链接的样式表是 3_col_layout.css。

这个样式表中包含如下 CSS 规则：

```
body {
  text-align:center;
  }
```

在 IE 6 中需要通过这条规则使布局在浏览器窗口内居中

```
#main_wrapper {
  width:840px;
  margin-left:auto;
  margin-right:auto;
  text_align:left:
  }
#header {
  }
#nav {
  width:18%;
  float:left;
  }
#content {
  width:60%;
  float:left;
  }
#promo {
  width:22%;
  float:left;
  }
#footer {
  clear:both;
  }
```

如果这个宽度值改变，那么各个分栏的宽度都会随之成比例变化

将最大化后的布局在浏览器内居中

将最大化后的布局在浏览器内居中

防止页面继承 body 上的 IE 居中 hack

没有为页眉设置样式

为节省版面，我们省略了下面针对内部 div 的代码（同两栏时一样）

以上 XHTML 和 CSS 中突出显示的代码是在两栏布局基础上需要做的唯一修改。如果你理解了两栏布局，那么对这个三栏布局也就没有什么好说的了。

下面，我们要介绍一些更高级的 CSS 应用，这涉及如何将两栏固定布局转换为流动式布局。不过，为了创建三栏流动式布局，我们必须拿出更多创意来。

5.7 三栏流动式布局

关于解决这个问题的挑战，请参考 http://www.mezzoblue.com/archives/ 2004/01/23/friday_chall/。

在向三栏布局中添加流动居中特性时，事情就变得复杂起来。因为此时，内容区仍然要随着浏览器窗口的调整而改变宽度，但两侧的分栏则始终要保持固定宽度。前面我们已经看到过了，实现流动的内容区和固定的左侧分栏很简单。这是因为左侧的分栏从来不会相对于它的参照点（包含元素的左上角）移动。当多出一个右侧分栏时，问题的复杂性就提高了——右侧的分栏必须在中间的分栏改变宽度的同时，不断地重新定位自己的位置，而通过 CSS 来实现这一点是很困难的。事实上，以 Zen Garden 闻名的 Dave Shea 在 2004 年就因遇到这个难题，而征集过解决方案。

CSS 专家 Ryan Brill 很快就找到了解决问题的方法（使用负外边距）而且，他的解决方案现在已经成为一种 CSS 经典技巧，被广泛应用于各种流动布局的网站中。拜 Ryan 所赐，以下就是实现过程（结果如图 5-10 和图 5-11 所示）。

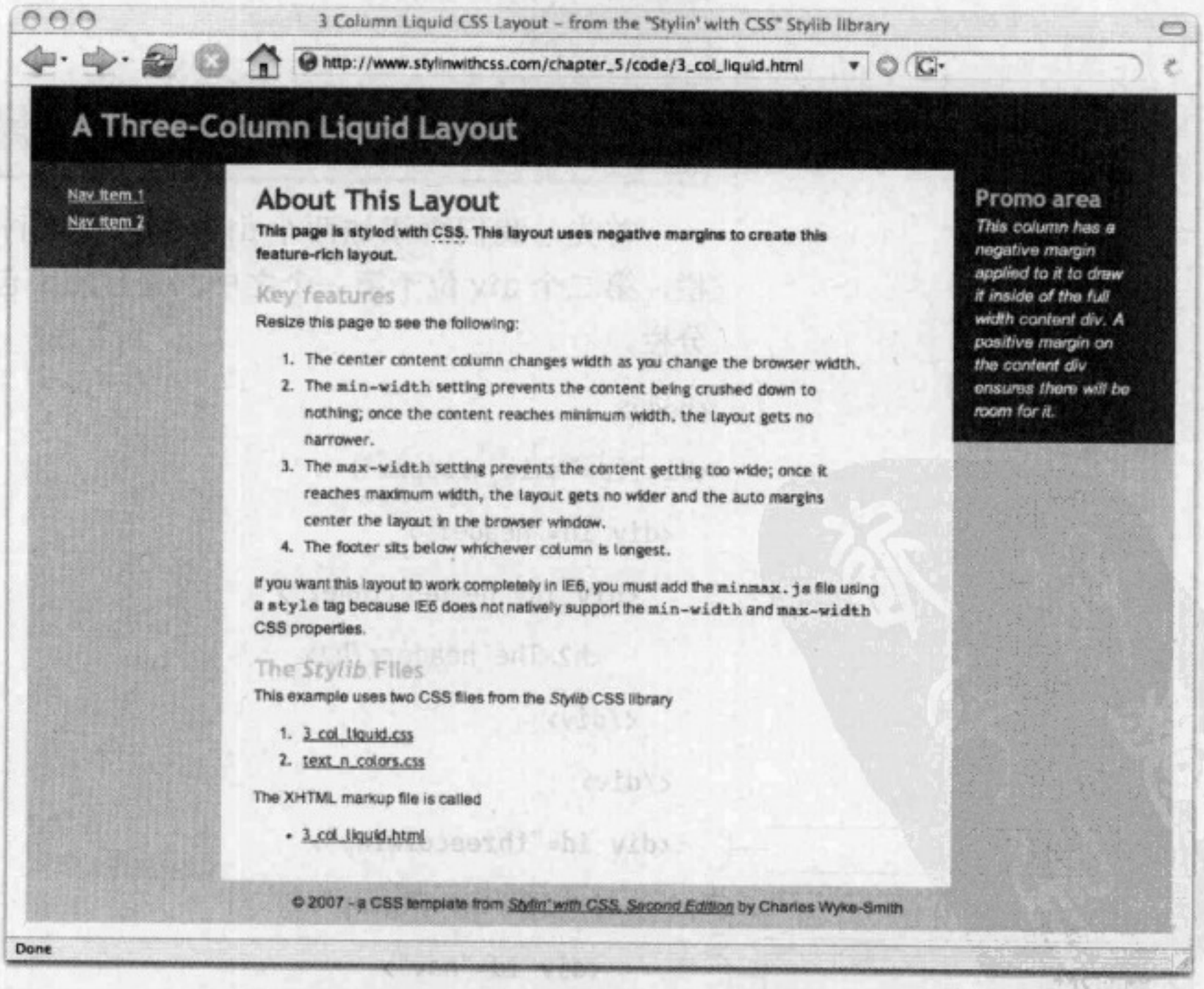


图 5-10 此时的三栏流动式布局已经达到了最大宽度，并将自己居中在了浏览器窗口中

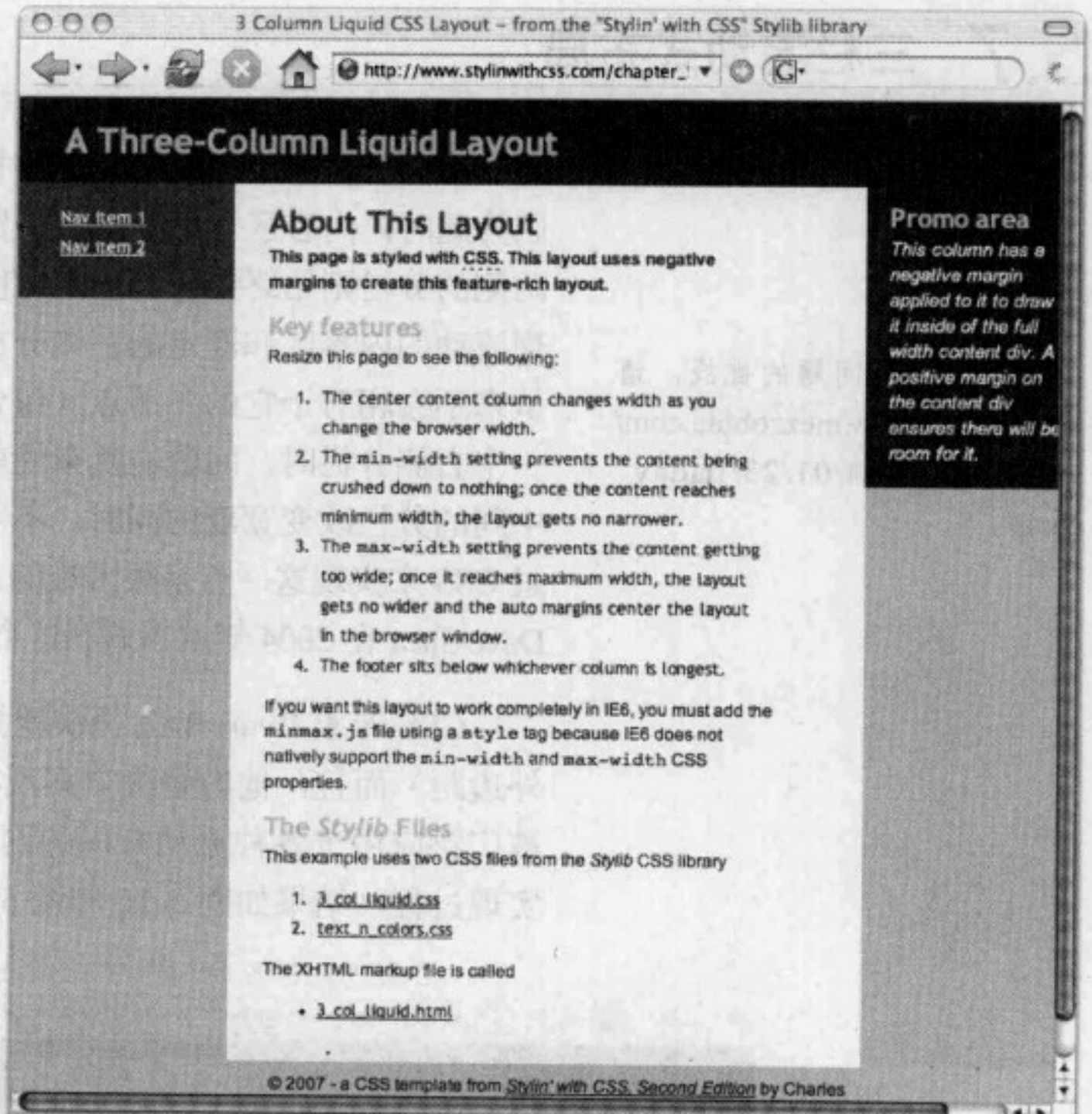


图 5-11 此时的三栏流动式布局已经达到了最小宽度，浏览器的右边缘已经覆盖了布局

首先，我们再添加两个 div。第一个 div 用于包含 3 个分栏，第二个 div 位于第一个之中，并且用于包含左侧和中间的分栏。

```
<body>
<div id="main_wrapper">
  <div id="header">
    <div id="header_inner">
      <h2>The header</h2>
    </div>
  </div>
  <div id="threecolwrap">
    <div id="twocolwrap">
      <div id="nav">
```

包含三个分栏

包含左侧和中间的分栏

左侧的分栏

```
<div id="nav_inner">
  <h3>Left nav</h3>
  <ul>
    <li><a href="#">Link 1</a></li>
  </ul>
</div>
</div>
<div id="content">
  <div id="content_inner">
    <h1>A Three Column Liquid layout</h1>


  </div>
</div>
</div>
<div id="promo">
  <div id="promo_inner">
    <h3>Promo area</h3>
    <p>
    </p>
  </div>
</div>
</div>
<div id="footer">
  <div id="footer_inner">
    <p>&copy; 2007 - a CSS template from <a href="http://
    www.stylinwithcss.com">Stylin' with CSS, Second
    Edition</a> by Charles Wyke-Smith</p>
  </div>
</div>
</body>
```

省略了链接代码以节省版面

nav 结束

中间的分栏

页面内容开始

为节省版面省略了内容

content_inner 结束

content 结束

twocolwrap 结束

这里的内容省略

promo_inner 结束

promo 结束

threecolwrap 结束

footer_inner 结束

footer 结束

main_wrapper 结束

以上是在前一个例子的标记基础上，添加了突出显示的两个包含 div 后构成的。与这些标记相关的 CSS 如下：

```
body {font: .8em verdana, arial, sans-serif;}
* {margin:0; padding:0;}
```

当超过最大宽度时居中布局

```
div#main_wrapper {min-width:760px; max-width:880px;
margin-left:auto; margin-right:auto;}
```

```
div#header {
  width:auto;
  }
#threecolwrap {
  float:left;
  width:100%;
  }
#twocolwrap {
  float:left;
  width:100%;
```

消除 IE 6 为浮动元素添加双外边距的 bug

```
  display:inline;
```

设置负外边距

```
  margin-right:-170px;
  }
#nav {
  float:left;
  width:150px;
```

消除 IE 6 为浮动元素添加双外边距的 bug

```
  display:inline;
  }
#content {
  width:auto;
  margin-left:150px;
  margin-right:170px;
  }
```

```
#promo {
  float:left;
  width:170px;
  }
 #footer {
  width:100%;
  clear:both;
  float:left;
 }
```

在这个布局中，中间的分栏同样也没有指定的宽度。如果为中间的分栏设置了宽度，那么它的宽度就不能随着浏览器窗口宽度的调整而改变了，因此，它固有的倾向就是扩展为布局的宽度。当然，如果真是那样，两侧的分栏就没有空间了。为此，我们为中间的分栏应用了等于两侧分栏宽度的外边距，以便将中间分栏的每一边都向内推动相应的距离。但问题在于，即使如此，如果不采取其他手段，右侧分栏的左边仍然会与包含元素的左边[①]对齐，因而处于错误的位置上，或者说当浏览器窗口变窄时，会移出屏幕之外。

Ryan Brill 的高明之外就在于为包含两栏的 div 设置负外边距，以便将右侧分栏吸回页面中，并让它的右边与包含元素的右边对齐[②]。这是一种令人叫绝的方案。每当创建这种布局，我都会感到自己对使用负外边距的原理又有了更深入的理解！据我所知，这可能是所有布局形式中最有用的一种布局了——正如英格兰人常说的“绝对令人瞠目！”。

5.8 设计长度相同的分栏

同基于表格的布局不同，基于 div 的布局不会导致所有分栏的长度都相同。因为 div 只会保持与它包含的内容同高，并随着内容的增加而在垂直方向上扩展。从视觉角度上说，如果

① 原文 right side of the container 有误。——译者注

② 事实上，对于这个方案作者并没有透彻地进行解释。为此，读者还可以试试另一种方案，即删除 twocolwrap 上的 margin-right:-170px;，然后在 promo 上添加一条 margin-left:-170px;，这样也可以达到目的。而这两个异曲同工的方案，同样都利用了 CSS 的算法规则：前者是通过负右外边距欺骗浏览器 twocolwrap 右侧还有 170 px 的空间，因此宽度为 170 px 的 promo 恰好可以提升到与前两栏比肩的位置上；后者则通过负左外边距欺骗浏览器 promo 本身没有宽度（0 宽度），因此当然也可以出现在前两栏的右侧，从而构成第三个分栏。——译者注

所有分栏都是全高的，那么会更加舒服。特别是在较长的页面中，两侧的分栏可能会随着用户向下滚动页面而从屏幕上方消失，这样就会在内容区的两侧留下空白区域。虽然可以为这些div指定高度，但当它的高度或者内容变化时，这样做的意义也不大。而且，我们需要让页面长度保持灵活性，以便它能够容纳其中的内容。这里，实际上需要灵活的高度，同时所有分栏的高度看起来应该是相同的——注意这里我们说的是“看起来”。也就是说，我们要做的只是创造一种分栏同高的假象。下面就是两种最佳解决方案。

- 人造分栏——为布局的背景添加一幅与分栏具有相同颜色和宽度的图像，从而创造出分栏沿页面向下伸展的假象。
- 以编程方式扩展div——当页面加载后，通过JavaScript找到其中最长分栏的高度，然后立即将其他分栏也设置为相同高度。

下面，我们就来看一看这两种方法各自的优缺点。

5.8.1 人造分栏

人造分栏方法涉及为页面中的包含div应用一幅背景图像，这幅图像应该与我们要从视觉上进行扩展的分栏具有相同的宽度和颜色。可以将背景图像设置为垂直重复，以便随着页面高度的增加，图像能够重复足够多的次数，从而填充必要的空间。

1. 使用人造分栏的两栏流动式布局

接下来，我们以本章前面介绍的两栏流动式[①]布局为例，从视觉上将其中的分栏扩展为完整的高度。在这个布局中，导航分栏是140像素宽，该宽度中包含位于内部div中的红色右边框。因此，相应的背景图像如图5-12所示。

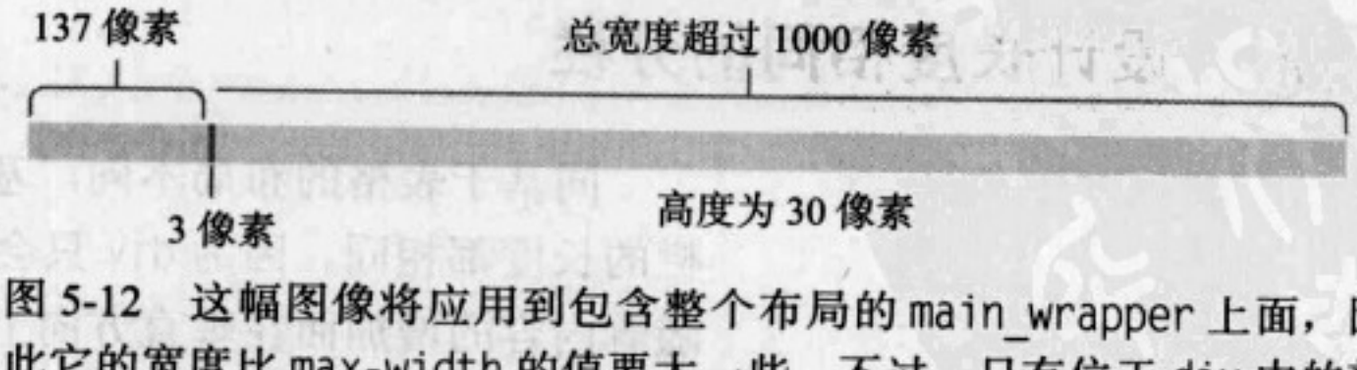


图5-12 这幅图像将应用到包含整个布局的main_wrapper上面，因此它的宽度比max-width的值要大一些，不过，只有位于div中的部分才是可见的。在CSS中，通过将背景属性设置为repeat-y可以使它在页面垂直方向上重复（另见彩插）

下面，我们通过以下规则将这幅图像添加为main_wrapper的背景：

① 原文fixed-width有误，因为本章前面无此布局。——译者注

```
#main_wrapper {
    max-width:960px;
    min-width:720px;
    margin-left:auto;
    margin-right:auto;
    text-align:left;
    background:url(../../../chapter_5/code/images/2_col_faux_
    art.gif) repeat-y;
}
```

设置最大布局宽度

设置最小布局宽度

在浏览器中居中布局

在浏览器中居中布局

重置在 body 标签上为 IE 6 设置的 hack

background 属性的 repeat-y 值能够使图像重复（“平铺”）足够多的次数，以便填满所在的包含 div。由于 main_wrapper 中包含着整个布局，所以图像会垂直平铺整个背景。采用这种方式，图像会在瞬间填充数千像素高的区域，而且由于它只有 30[①]像素高，因此下载也非常快。不过要注意的是，因为图像会填充整个背景，所以我们还需要为页眉和页脚添加一种背景颜色（即使是白色），从而使“人造”图像不会透过它们显示出来。相反，对于分栏来说，我们需要让它们保持透明，以便人造分栏背景图像能够透过它们显示出来（图 5-13）。

文件路径中 ../ 的含义是“上一级文件夹”。以我当前的文件夹结构为例，chapter_5 是位于当前 CSS 文件上面 3 级的文件夹。在你的文件夹结构中，从 CSS 文件到图像的路径可能会有所不同。

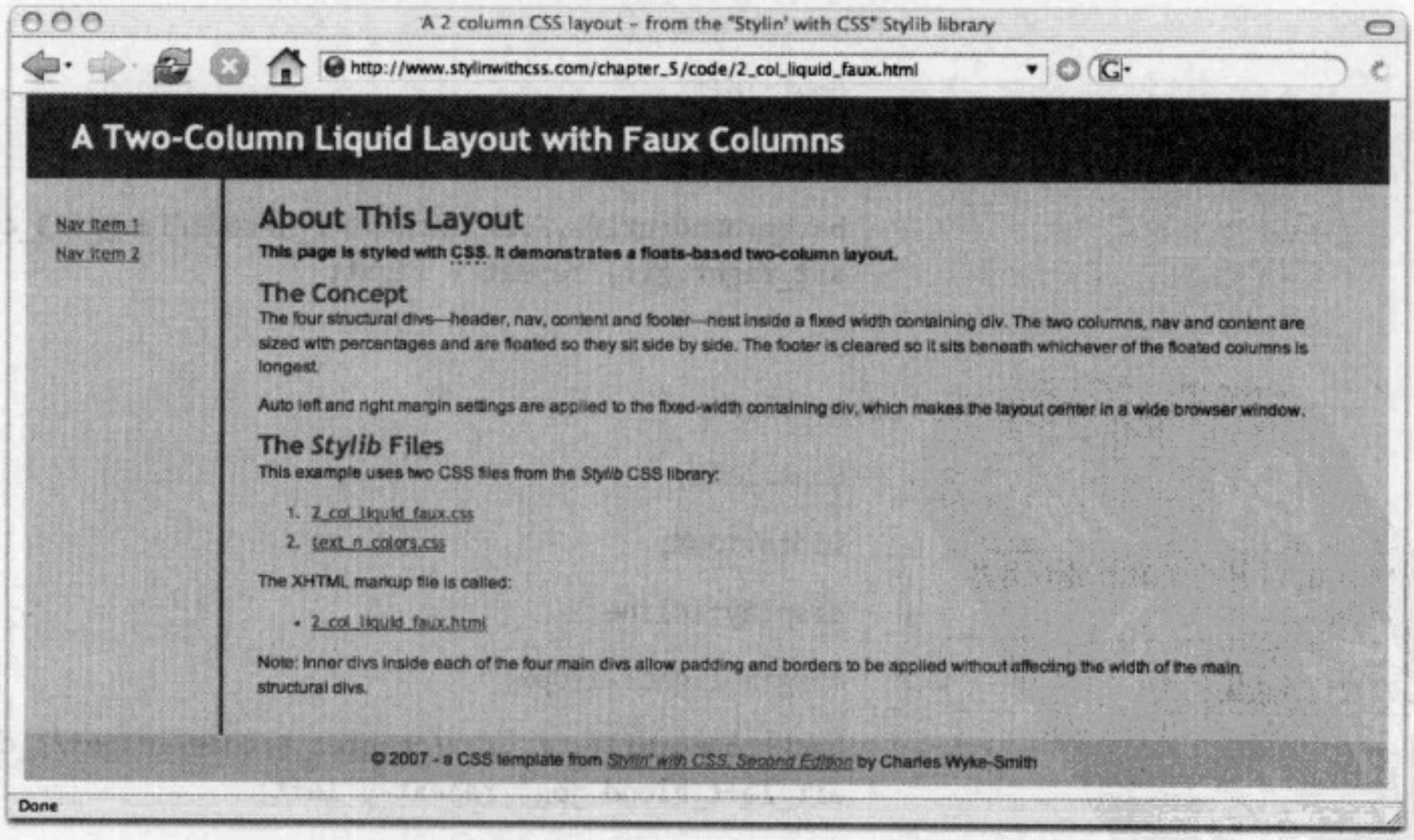


图 5-13 Stylib 库中的带人造分栏的两栏流动式布局（另见彩插）

① 原文 20 与前文不符。——译者注

现在，即使导航分栏的长度同本章前面两栏流动布局中一样短，但从视觉上看的结果则截然不同——页面中的两个分栏的高度看起来已经扩展到了页面的底部[①]。这种方法中需要注意的地方，就是要保证背景图像中元素的尺寸与分栏的宽度要完全一致。

2. 使用人造分栏的三栏流动式布局

如果要为前面的三栏流动布局也添加人造分栏，那么需要使用单独的图像来从视觉上扩展左侧和右侧的分栏，比如（图5-14）：

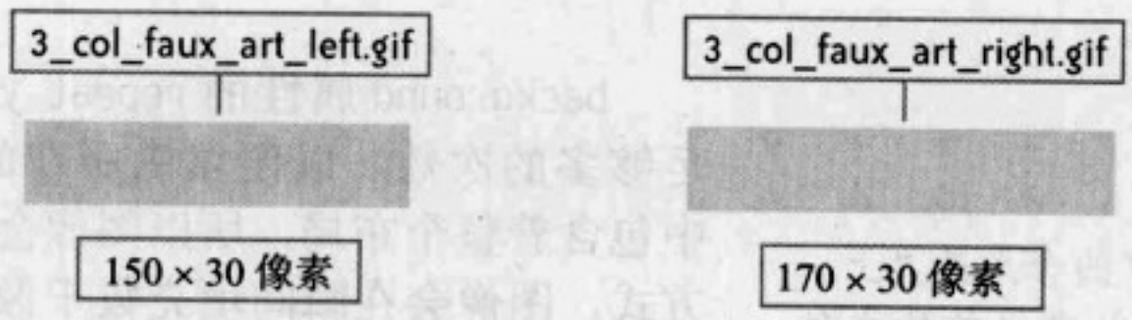


图5-14 要为三栏流动布局创建人造分栏，可以将这两幅图像分别作为twocolwrap和threecolwrap的背景（另见彩插）

左边的图像与左侧分栏的宽度相同（150像素），右边的图像与右侧分栏的宽度相同（170像素）。下面，我们将这两幅图像分别应用到不同的包含div上——将左边的图像应用到twocolwrap，将右边的图像应用到threecolwrap。

```
#threecolwrap {
  float:left;
  width:100%;
  background:url(../../../chapter_5/code/images/3_col_faux_
  art_right.gif) repeat-y right;
  }
#twocolwrap {
  float:left;
  width:100%;
  display:inline;                    /* 避免IE在浮动元素上的双外边距bug */
  margin-right:-170px;               /* 设置负外边距 */
  background:url(../../../chapter_5/code/images/3_col_faux_
  art_left_blend.jpg) repeat-y left;
  }
```

① 原文 each side of the content area 是针对三栏布局而言的，所以有误。——译者注

由于 threecolwrap 与布局的宽度相同，因此我们可以通过它来为右侧的分栏添加人造背景图像。不过，这里必须将 background 属性的位置部分设置为 right，从而将背景图像设置为自包含元素的右上角而不是默认的左上角开始平铺。

此外，这两幅背景图像也不一定是纯色的——可以使用任何经过设计的能够垂直重复显示的图像（如本例所示）。而且，我将 nav 和 promo 分栏自身的 background-color 设置为了 transparent，以便看清全高的人造分栏图像（图 5-15）。

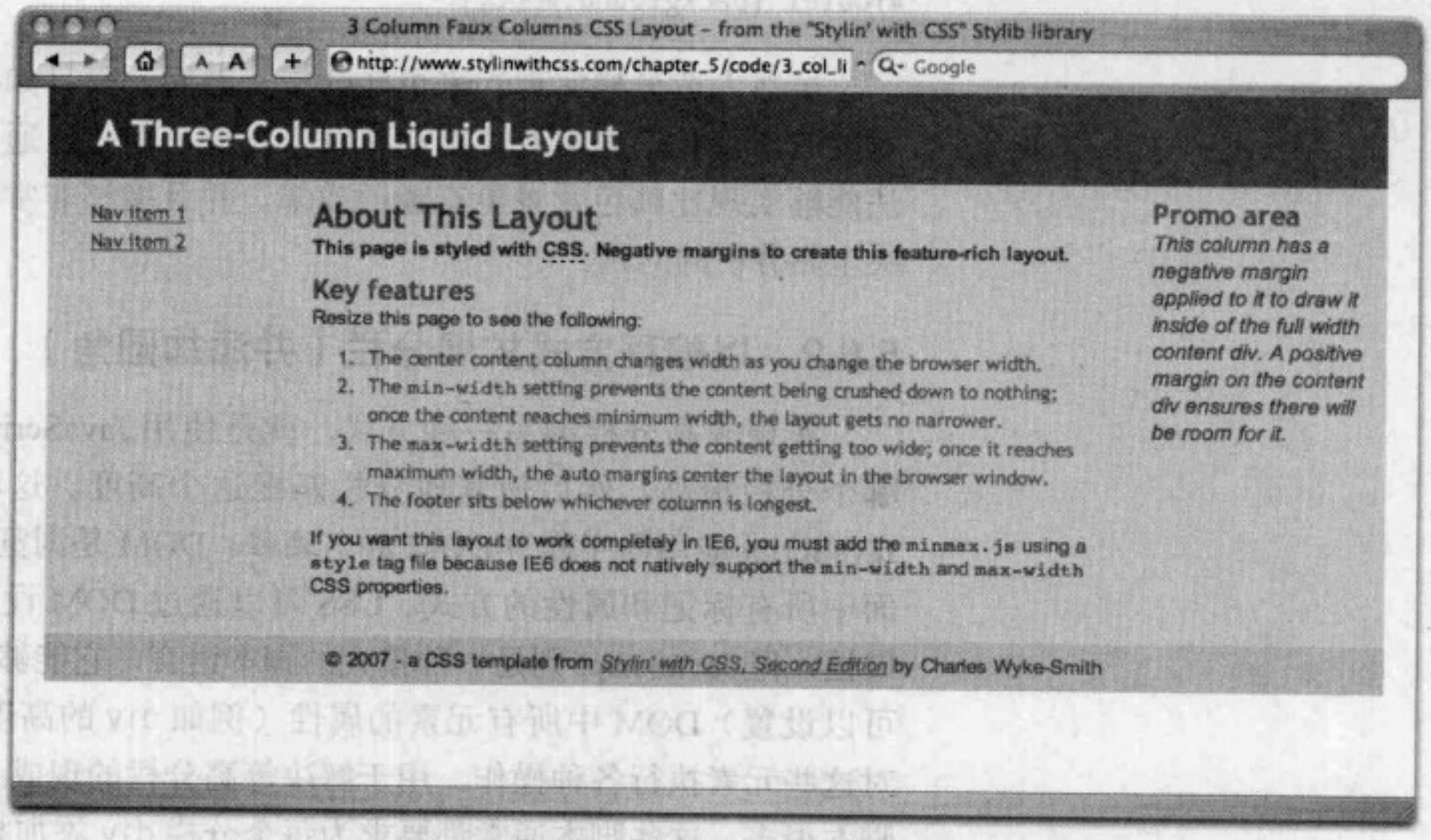


图 5-15 Stylib 库中带有人造分栏的三栏流动式布局（另见彩插）

最后，也可考虑到，如果页面中的内容很少，两侧分栏的高度有可能超过中间的内容分栏。由于内容分栏没有人造的背景，这样就会导致内容分栏底部出现空白（同前面例子中看到的两侧分栏较短时导致的空白类似）。为解决这个问题，我们去掉了内容分栏的背景颜色，并为 main_wrapper 设置了相应的背景颜色，后者的高度总是与布局相同。 在两侧分栏通过背景图像实现了全高的基础上，main_wrapper 中唯一可见的地方就是内容区域——这正是我们所需要的。（还有一种这里没有示范的方法，即也可以为 main_wrapper 应用背景图像，从而在三个分栏中都实现人造分栏的效果——彻底的样式效果！）

为此，需要改变的代码如下：

```
body {background:#BB9;}
#main_wrapper {background:#FFF;}
#header {background:#DDC;}
#nav {background:#CCB;}
#content {background:#EED;}
#promo {background:#CCB;}
#footer {background:#DDC;}
```

设置内容区的背景颜色

删除背景以便人造分栏显示出来

删除背景以便显示 main_wrapper 的背景颜色

虽然人造分栏实现起来可能会费点时间，而且如果分栏的宽度有所变化，也需要重新制作背景图像，但是，通过这种方法能够实现比纯色背景更有趣的效果，并且能够非常有效地解决全高分栏的问题。

5.8.2 以编程方式扩展分栏（并添加圆角）

实现等高分栏的另一种方式，就是使用 JavaScript 来确定哪个分栏最高，然后使其他分栏匹配这个高度。这种类型的 JavaScript 应用程序称为 DOM 脚本编程。DOM 是浏览器看待页面中所有标记和属性的方式。CSS 可以通过 DOM 设置标签的属性，但 JavaScript 则是一种强大的脚本语言，它能够取得（也可以设置）DOM 中所有元素的属性（例如 div 的高度），并且对这些元素执行各种操作。用于解决等高分栏的现成 JavaScript 脚本很多，这些脚本通常都要求为每个分栏 div 添加相同的类，以便 JavaScript 识别这些分栏并进行比较。

我最喜欢的完成这一任务的代码是 Alessandro Fulciniti 编写的 NiftyCorners。这个脚本很方便，通过它不仅能够实现等高分栏，而且正如它的名字所反映出的，还能够为元素添加圆角（圆角是 Web 2.0 网站中一种标志性的设计方法）。下面我们就来看一看通过 NiftyCorners 为本章前面的三栏固定宽度布局创建等高、圆角分栏效果的过程。

对NiftyCorners脚本所作的修改[①]

请注意，我对 niftycorners.js 文件作了一点模块化的改动，以便它不仅能够用在 XHTML 页面的同一文件夹中，也能够用在任何位置，还能够被更多的页面引用。修改涉及的内容，就是把这个文件中的一个函数由

```
function AddCss(){
niftyCss=true;
var l=CreateEl("link");
l.setAttribute("type","text/css");
l.setAttribute("rel","stylesheet");
l.setAttribute("href","niftyCorners.css");
l.setAttribute("media","screen");
document.getElementsByTagName("head")[0].appendChild(l);
}
```

修改为了

```
function AddCss(path){
niftyCss=true;
var l=CreateEl("link");
l.setAttribute("type","text/css");
l.setAttribute("rel","stylesheet");
l.setAttribute("href",path);
l.setAttribute("media","screen");
document.getElementsByTagName("head")[0].appendChild(l);
}
```

经过修改，就可以在每个 XHTML 页面中传递文件的相对路径了，这一点在三栏圆角布局的代码中可以看到。如果你使用的是 Stylib 库中的 NiftyCorners，那么上述修改已经完成了；但是，如果你从 Nifty Cube 网站上（http://www.html.it/articoli/niftycube/index.html）下载了新的版本，则需要进行上述修改才能使其与 Stylib 中的文件协同工作。

对于这个例子来说，我们使用前面看到过的三栏固定布局，并作了以下修改（图 5-16）。

① 原文 Cube 放在这里容易令人误解。——译者注

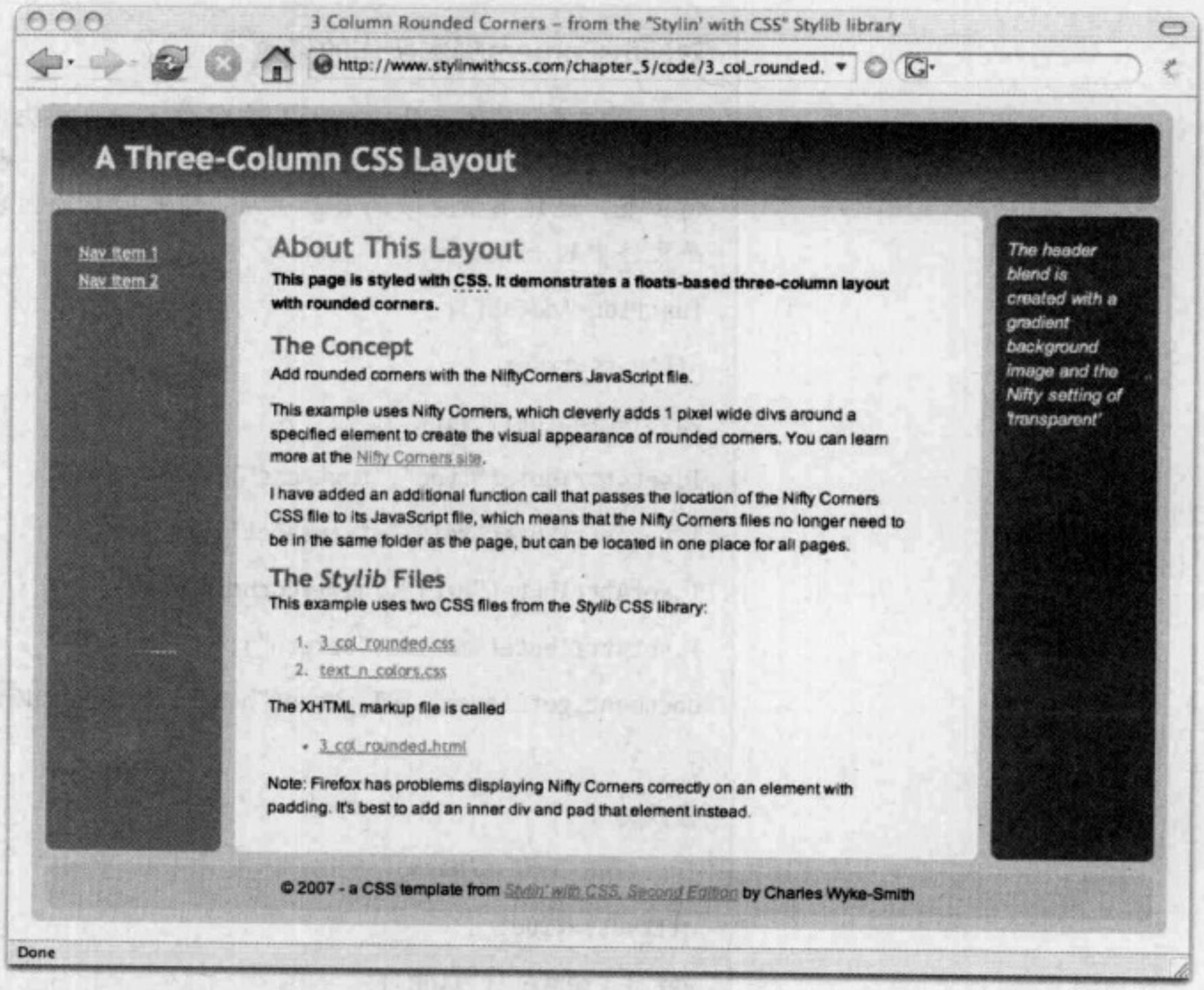


图 5-16 使用 NiftyCorners 脚本不仅可以实现分栏等高，而且可以为分栏添加圆角效果

需要像下面一样在页面的头部将 NiftyCorners 脚本链接到页面中：

```
<script type="text/javascript"  src="../../lib/nifty_corners/
javascript/niftycube.js"></script>
```

然后，再通过需要扩展的元素来调用 NiftyCorners 的代码：

```
<script type="text/javascript">
window.onload=function(){
Nifty("div#nav,div#content,div#promo","medium same-height");
Nifty("div#header,div#footer","medium");
AddCss ("../../lib/nifty_corners/css/niftyCorners.css");
}
</script>
```

将 nav、content 和 promo 分栏设置为等高并应用中等半径的圆角

为页眉（header）和页脚（footer）应用中等半径的圆角

传递指向 NiftyCorners.css 文件的相对路径

在为分栏应用了适当的设置之后，接下来需要为它们创建一些间距，以便分栏之间保持一些距离。如果能够将圆角效果应用给内部 div 会节省很多工作量，但是这样做会导致盒子的显示出现问题——有的圆角会出现奇怪的直角。为此，我们还要为分栏 div 添加外边距，并从这些盒子的宽度中减去相同数量的值，以保持布局的原始宽度不变，布局变高则没有关系。

```
#main_wrapper {
  width:840px;
  margin-left:auto;
  margin-right:auto;
  text-align:left;
  padding:10px 0;
  margin-top:10px;
  }
#header {
  margin:0px 10px 0px 10px;
  }
#nav {
  width:150px;
  width:130px;
  margin:10px 10px 10px 10px;
  float:left;
  }
#content {
  width:550px;
  margin:10px 10px 10px 0px;
  float:left;
  }
#promo {
  width:140px;
```

当这个宽度值改变时，分栏的宽度会随之成比例变化

在浏览器中居中最大化之后的布局

防止页面继承在 body 上面为 IE 6 设置的居中 hack

补偿水平外边距

补偿应用到 content 中的水平外边距

```
width:120px;
margin:10px 0 10px 0;
float:left;
}
#footer {
margin:0 10px;
clear:both;
}
```

当然，用户必须在浏览器中启用 JavaScript 才能使 NiftyCorners 脚本发挥作用。据 thecounter.com 在 2007 年 10 份的统计，有 95% 的用户浏览器都支持 JavaScript。在这种情况下，是否启用了 JavaScript 就不算是太大的问题[①]。

添加的水平外边距总和是 40 像素，因此我从左和右侧的分栏中分别减去 20 像素，从而维持 840 像素的布局宽度。

最终，NiftyCorners 为我们创建了可调整大小的圆角盒子。由于 XHTML 元素只能应用一幅背景图像，因此 NiftyCorners 脚本通过在 div 中添加另外 3 个 span 包含元素，为它们分别应用了 4 幅背景图像。使用 NiftyCorners 的缺点是无法添加弧形边框，每个圆角只能是一个单色块。虽然 CSS3 规范中提出了弧形角的特性，但目前只有 Firefox 能够实现该特性。

5.9 绝对定位的布局

我们曾经说过，要在本章结尾时介绍一种绝对定位布局。下面，就是使用绝对定位技术创建的一种三栏布局（图 5-17）。

首先来看一看标记代码：

这里的 olive 类，用于切换成 text_n_colors.css 中的“olive”配色方案

```
<body class="olive">
<div id="main_wrapper">
<div id="header">
<div id="header_inner">
<h2>Header</h2>
</div>
</div>
```

① 后半句 as the only difference is that the boxes are not rounded but have regular square corners. 是错误的。因为如果用户禁用了 JavaScript，那么脚本将无效。故该句未予翻译。——译者注

```
<div id="content">
  <div id="content_inner">
  <h1>A Three-Column CSS Layout</h1>
    <p><strong>This page is styled with <abbr
    title="Cascading Style Sheets">CSS</abbr>...
```

为节省版面省略了内容

```
    </p>
  </div>
```

content_inner 结束

```
  <div id="footer">
  <div id="footer_inner"><p>The footer</p></div>
 </div>
</div>
```

content 结束

```
<div id="nav">
  <div id="nav_inner">
    <ul>
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
    </ul>
  </div>
</div>

<div id="promo">
  <div id="promo_inner">
  <p>the promo area</p>
  </div>
```

promo 结束

```
</div>
```

main_wrapper 结束

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

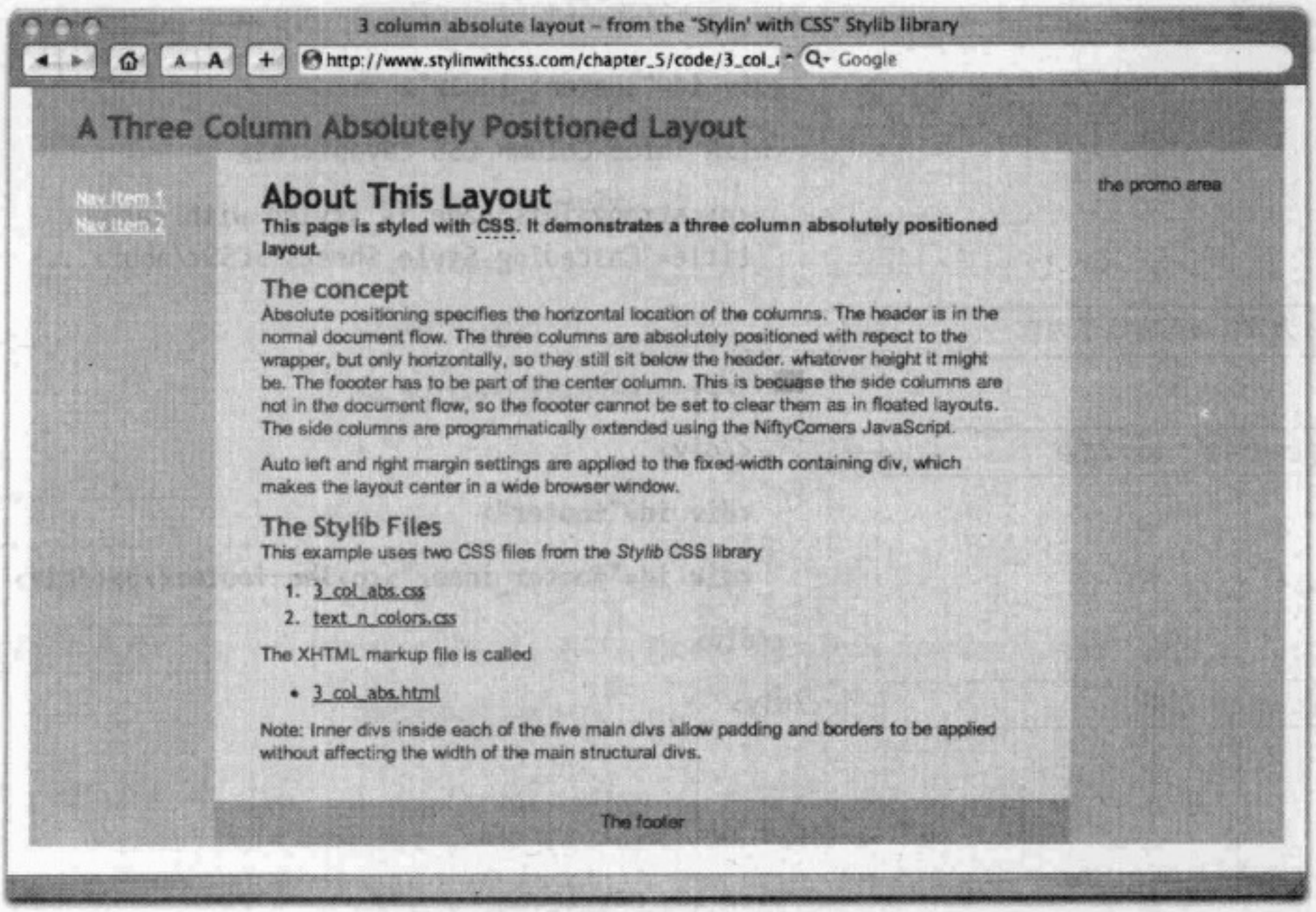


图 5-17 在绝对定位的布局中，我们将页脚包含在了中间的分栏中，因为页脚不能清除绝对定位的分栏。如果你能够容忍这种限制，那么这也是一种很有效率的[①]布局方式

同我们看到的浮动布局相比，这些标记中最值得关注的地方，就是标记中 div 的先后顺序并没有反映元素在页面中出现的顺序。这里，内容区直接位于页眉之后，导航和宣传分栏则位于最末尾（你也可以不这样做，但把内容在标记中放在尽可能靠前的位置，通常被认为是一种有助于搜索引擎索引的做法）。而页脚则位于结束的内部内容 div 和结束的内容 div 之间。通过仔细的观察，可以清楚地理解这些代码的含义。

因为我们要通过绝对定位方式来构建 3 个分栏，所以需要在页面中声明它们的左或右位置（相对于它们的定位环境 main_wrapper，见下面的 CSS 规则），至于它们在标记中出现的顺序则不是问题。以绝对定位方式在页面中将分栏排布到指定的位置，就如同向公告板上张贴通知一样，先贴哪张后贴哪张都无所谓。

① 原文 industrial-strength 在此采取意译。——译者注

以下就是创建绝对定位布局的 CSS 规则，其中重要的定位规则都突出显示：

```
body {
  margin:0px;
  padding:0px;
  text-align:center;
  }
#main_wrapper {
  width:880px;
  margin-left:auto;
  margin-right:auto;
  text-align:left;
  position:relative;
  height:100%;
  background-color:#585;
  }
#header {
  }
#content {
  position:absolute;
  left:130px;
        padding:0;
  width:600px;

  }
#nav {
  position:absolute;
  left:0px;
  width:130px;
  background:transparent;
```

需要通过这条规则使布局在 IE 6 的浏览器窗口里居中

设置最大的布局宽度

在浏览器里居中布局

在浏览器里居中布局

重置在 body 标签上为 IE 6 设置的 hack

默认全宽

```
    margin-bottom:300px;
    }
#promo {
    position:absolute;
    right:0px;
    width:150px;
    background:transparent;
    }
#footer {
    }
#header_inner, #nav_inner, #content_inner, #promo_inner {
    overflow:hidden;
    }
#header_inner {
    padding:1em 2em;
    }
#nav_inner {
    margin:1em 1.2em;
    }
#content_inner {
    margin:2em 2.5em 0em 2em;
    padding:0;
    }
#promo_inner {
    margin:1em 1.2em;
    }
#footer_inner {
    text-align:center;
    }
```

默认全宽

防止过大的元素破坏布局

本章一开始我们就介绍过，绝对定位的元素（这个例子中的分栏）会移出文档流，因此它们不能像浮动布局中的分栏一样将页脚推向下方。除非我们借助于 JavaScript 并向标记中添加适当的类（参见 http://www.shauninman.com/archive/2004/05/14/clear_positioned_elements），否则就需要对页脚的布局采取一些折中的办法。如果我们想让它始终定位在布局的底部，那么它不能做到与布局同宽，只能与内容分栏同宽，如图 5-17 所示。我觉得这并不是一个大问题，特别是在使用 NiftyCorners 来调整分栏高度并实现漂亮、雅致的布局效果的情况下。为了把页脚放到内容区中，我们在标记里把页脚放到了主内容 div 中，但仍然位于内容分栏的内部 div 外面。这样，可以使页脚扩展到与内容区等宽，而且不会受到内容的内部 div 中内边距的影响。

在 CSS 规则中，关键的代码是为 main_wrapper 设置的 position:relative。这条规则为绝对定位的分栏提供了定位环境。如果不设置这条规则，布局中的分栏将会相对于 body 元素确定自己的位置，这意味着它们会紧贴到浏览器窗口的边缘。通过相对于宽度固定而且在浏览器窗口内居中的 main_wrapper 进行定位，分栏就成为了布局的一部分。这样，当浏览器窗口超过布局的宽度时，它们也能够将自己定位在页面的中间。

本章有关布局的内容至此就要结束了。在第 6 章中，我们要讨论的是界面组件，例如菜单、表格和表单。在本书最后一章，我们会介绍如何组织本章的布局模板和第 6 章的界面组件，从而创建完整的网页布局。

第 6 章

设计界面组件

这里的界面组件指的是页面内容中所有的支持性元素：列表、菜单、表单、表格等。在本章中，我们介绍如何以有效且具有可访问性的方式，在 XHTML 中编写这些界面组件。同时，讨论如何通过 CSS 以符合项目设计需要的各种方式来为这些组件添加样式。

让我们从表格开始。

6.1 为表格添加样式

表格的一个广为人知的错误用法，就是在过去用它创建由各种表现标记构成的页面布局。但是，这并不意味着我们不能按照它本来的用途使用表格——像在微软的 Excel 电子表格中一样构建由行和列组成的数据网格。

以下是一个在 Firefox 中显示的默认的表格（见图 6-1）。

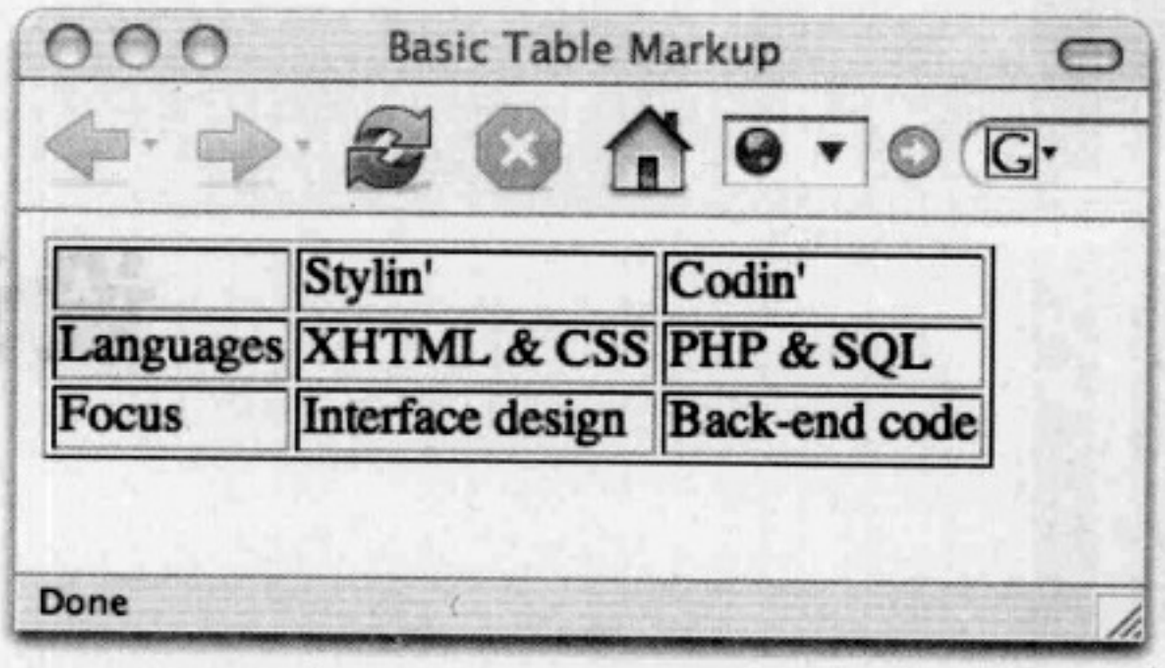


图 6-1　在默认情况下，通过 XHTML 的 border 属性设置了边框后显示的表格

相应的标记代码如下所示：

```
<table border="1">
  <tr>
    <td> </td>
    <td>Stylin'</td>
    <td>Codin'</td>
  </tr>
  <tr>
    <td>Languages</td>
    <td>XHTML &amp; CSS</td>
    <td>PHP &amp; SQL</td>
  </tr>
  <tr>
    <td>Focus</td>
    <td>Interface design</td>
```

```
        <td>Back-end code</td>
      </tr>
</table>
```

上面的屏幕截图就是我们通常看到的表格在浏览器中的显示效果。注意，我们通过表格标签中突出显示的表现性的边框样式，设置了网格的可见性；否则，表格中的数据只会浮在页面上，因而难以分辨行和列之间的关系。稍后我们就移除这些边框。

每一名设计者都应该学习Edward Tufte的研究成果，在此，我们推荐他的必读入门书*The Visual Display of Quantitative Information, 2nd Edition*（英国柴郡，CT: Graphics出版社，ISBN 0961392142）。这本书虽然开始读起来有些枯燥，但绝对会令人着迷。

在这种默认的显示效果中，存在很多被Edward Tufte[①]称为“图表垃圾（chart-junk）”的部分。比如，表格及每条数据四周的盒子比数据本身还要吸引人的注意力。

而且，表格的标记基本上是最小化的——只包含3个表格行tr（table row），每行中也只包含3个表格数据元素td（table data）。

下面我们看一看改进这个表格视觉外观的几种方式。为此，我们首先要对标记进行一些改进，以便为CSS提供足够的挂钩（hook）[②]，而且，更重要的是使其更清晰地表现出数据元素之间的相互关系。

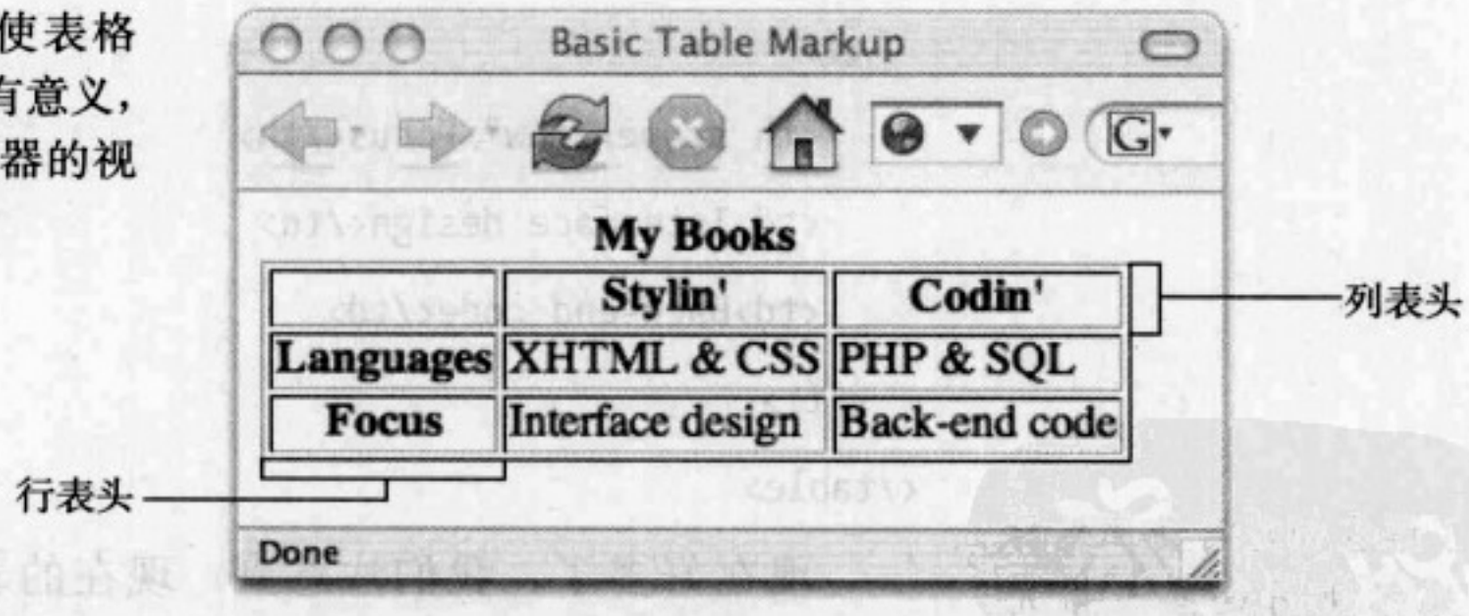


图6-2 改进后的标记使表格不仅对视力健全的人更有意义，而且对依赖于屏幕阅读器的视力不好的用户也更清晰

在没有额外帮助的情况下，屏幕阅读器很难处理好表格。因为它们会先从表格的首行读起，然后再一行一行地读取数据行，这种方式对于依靠听觉的视力不好的用户并不理想。不信可以将上面表格中的信息大声朗读给周围的人听，试一试他们能理解多少。实际上，在朗读每行（列）数据之前，需要先读出表格的

① 耶鲁大学名誉教授，数据分析专家，擅长数据的可视化表现设计，被《纽约时报》誉为“数据领域的达芬奇”（据http://encyclopedia.thefreedictionary.com/Edward+Tufte）。——译者注

② 所谓的挂钩，即ID和类等可以构成CSS选择符的内容。——译者注

（行、列）表头单元格[③]，而适合屏幕阅读器的良好标记能够做到这一点。为此，我们需要对现有的标记进行一些加强语义的调整（见图 6-2）。

```
<table border="1" summary="Summary of my books">
  <caption>
    <strong>My Books</strong>
  </caption>
  <tr>
    <th scope="col"> </th>
    <th scope="col">Stylin'</th>
    <th scope="col">Codin'</th>
  </tr>
  <tr>
    <th scope="row">Languages</th>
    <td>XHTML &amp; CSS</td>
    <td>PHP &amp; SQL</td>
  </tr>
  <tr>
    <th scope="row">Focus</th>
    <td>Interface design</td>
    <td>Back-end code</td>
  </tr>
</table>
```

要深入了解在创建可访问性标记时，如何使用这些表格元素，推荐阅读 Zoe Gillianwater 的文章“Using Tables Appropriately”，网址为：http://www.communitymx.com/content/article.cfm?cid=0BEA6。

现在好多了。我们注意到，现在的表格标记中多了一个 summary 属性、一个 caption 标签，而且更重要的是，通过表头单元格 th（table heading）标签将表头单元格从数据单元格中区分了出来，这样表头单元格在默认情况下就会以粗体显示。每一个表头单元格都有 scope 属性，该属性表示相应的表头单元格与列还是与行相关。

③ 为区分列的表头（th）和行的表头（th），我们把表格的第一行称为列表头（即 th[scope="col"]），把表格的第一列称为行表头（即 th[scope="row"]）。而这里“（行、列）表头的单元格”，指的就是行表头和列表头包含的所有单元格，即第一列和第一行中的单元格。——译者注

在这些改进后的标记基础上，我们可以删除 table 标签中的 border="1" 属性，并将其替换成一个类，以便在样式表中用它来对准表格中的所有元素。

```
<table class="basic_lines" border="1" summary="Summary of my
books">
```

现在，我们只能看到数据（见图 6-3）。

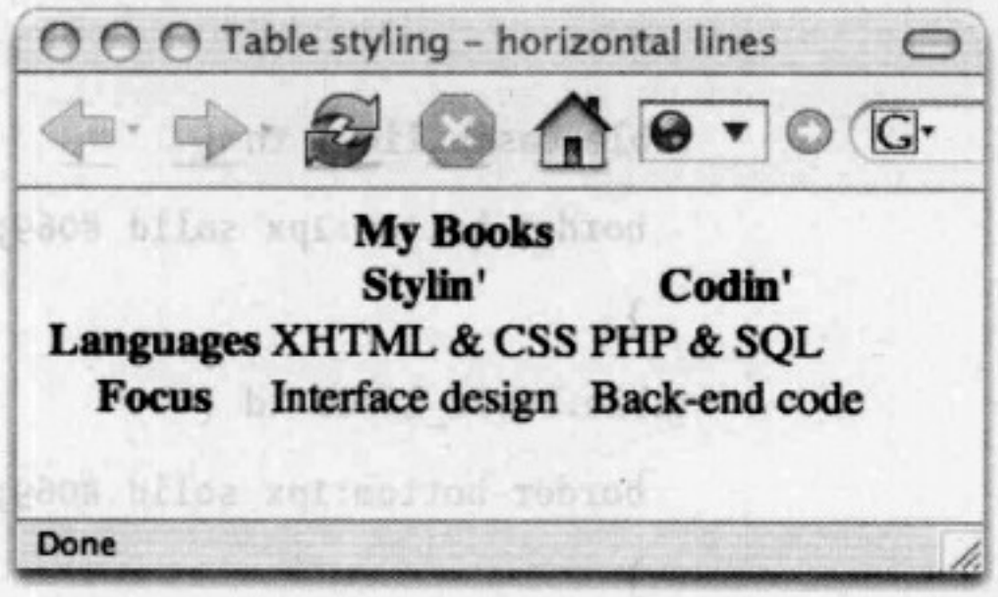


图 6-3 在没有单元格边框的情况下，不容易看出数据间的关系

我们对表格添加样式的目标，就是尽量添加最少的视觉因素，以保证用户对数据的理解。例如，可以试一试能否省略垂直的边框，比如（见图 6-4）：

```
table.basic_lines {
  width:300px;
  border-top:3px solid #069;
  }
```

设置布局的宽度

表格的顶部边框

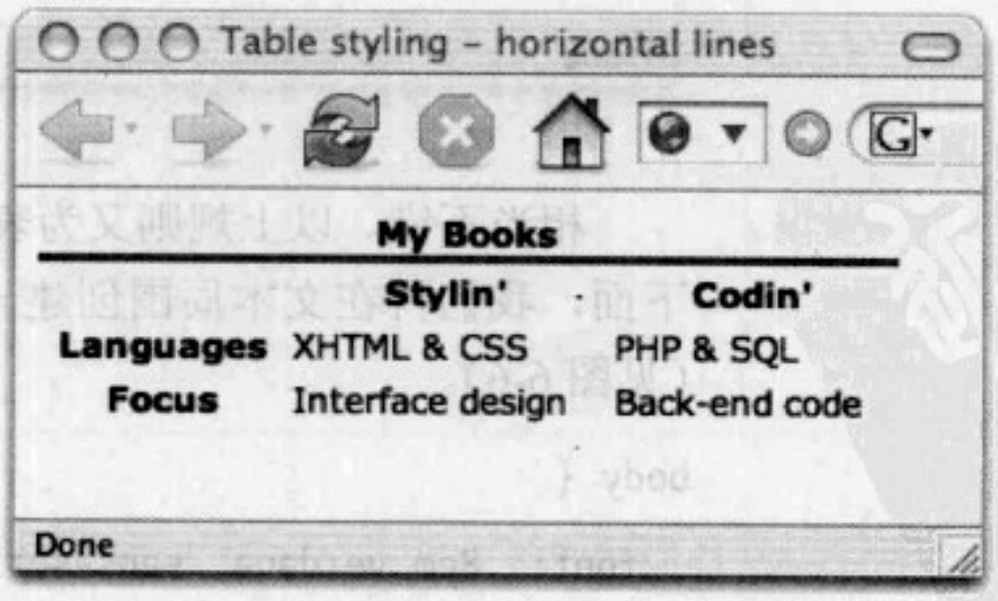


图 6-4 我们首先为表格的顶部添加了一条水平边框。注意，即使 caption 标签位于表格内部，但它默认也会显示在表格外部

sans-serif 字体系列和表题（即使 caption 在标记中位于表格内部，但它仍然会出现在表格上方）下方的一条清爽的蓝色线条看起来是个不错的起点。接下来，我们再为表头和数据添加一些线条，如图 6-5 所示。

```
body {
  font: .8em verdana, sans-serif;
  }
table.basic_lines {
  width:300px;
  border-top:3px solid #069;
  }
table.basic_lines th {
  border-bottom:2px solid #069;
  }
table.basic_lines td {
  border-bottom:1px solid #069;
  }
```

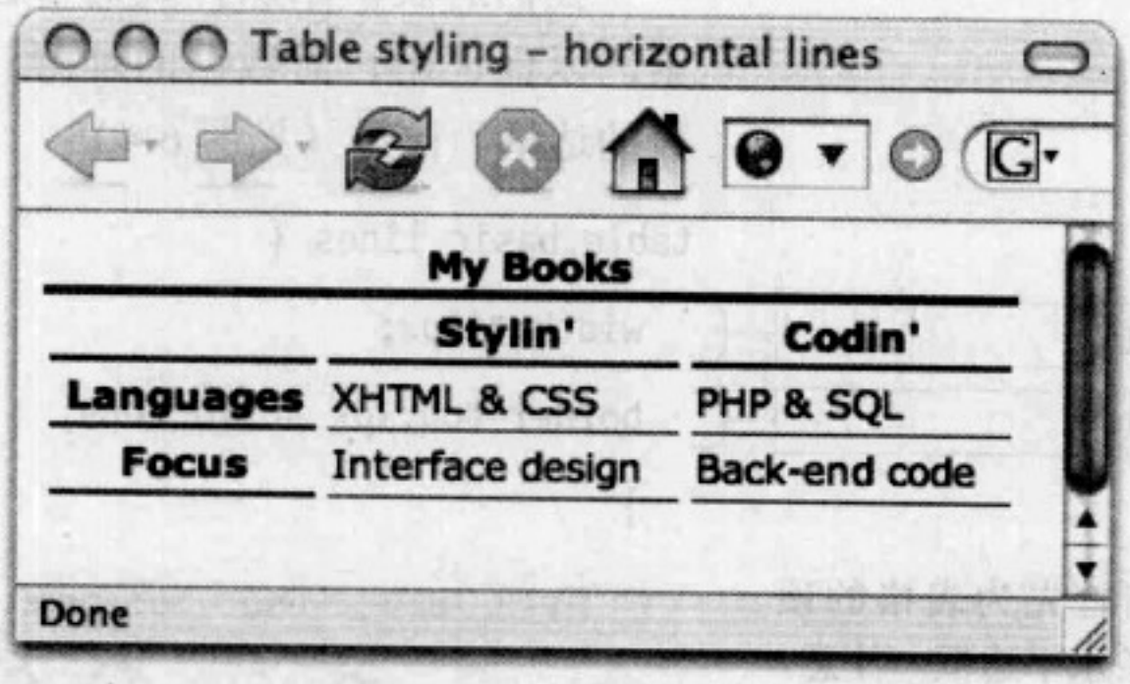


图 6-5 粗细线条的交替运用进一步将数据从表头中区分出来。但线条间的小间隙怎么办呢（另见彩插）

相当不错。以上规则又为表格增添了几分清晰的视觉效果。下面，我们再在文本周围创建一些空间并闭合线条之间的间隙（见图 6-6）。

```
body {
  font: .8em verdana, sans-serif;
  }
table.basic_lines {
  width:300px;
```

移除单元格之间的空白

```
border-collapse:collapse;
border-top:3px solid #069;
}
table.basic_lines caption {
```

设置表格与表题之间的空白

```
margin-bottom:6px;
}
table.basic_lines th {
  border-bottom:2px solid #069;
  }
table.basic_lines td {
  border-bottom:1px solid #069;
  }
table.basic_lines td, table.basic_lines th {
```

为每个单元格中文本的四周加空

```
padding:5px 3px;
}
```

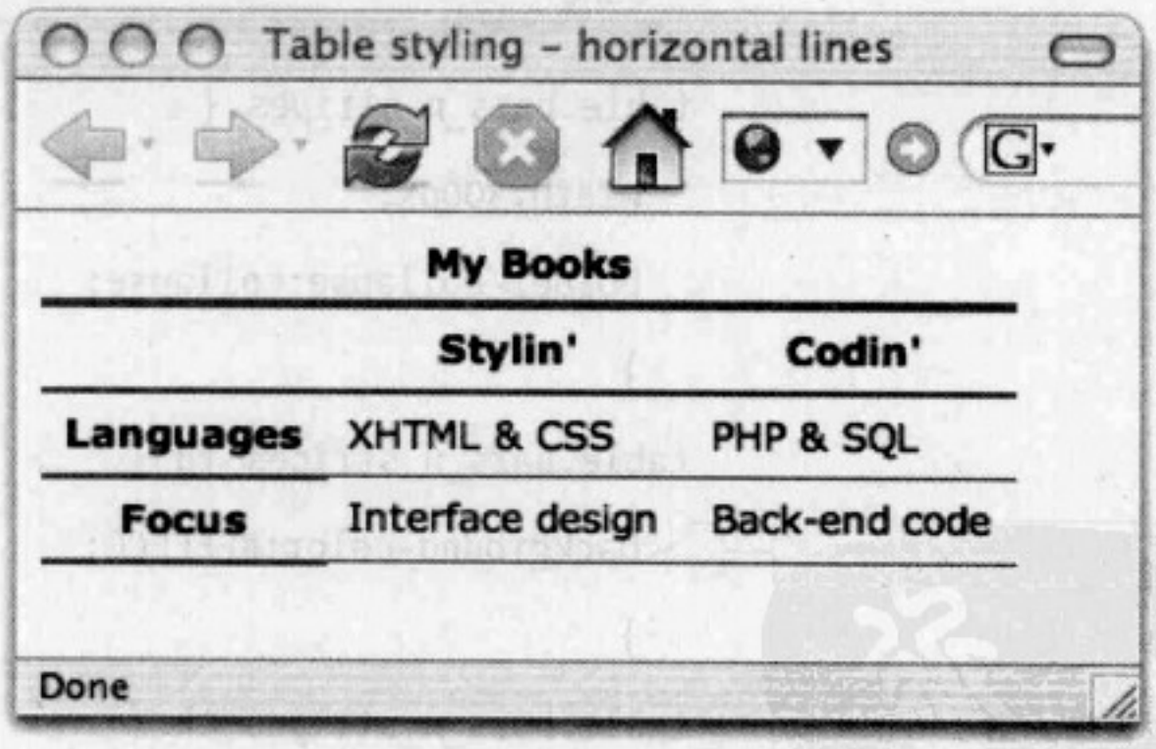


图 6-6 最终的样式为我们提供了一个清晰的、最低限度的、能够辅助对数据的理解而不是分散注意力的表格外观

注意一下我们在 table 标签中使用的 border-collapse 属性。通过边框折叠（collapsing）可以将默认的两个单元格间的双边框减少为单边框。这里我们没有为表格设置垂直的线条，如果设置了，那么在默认情况下，这些线条将会出现在每个单元格的四周，相邻的两条垂直边框之间会存在一定的间隙，就像我们前面看到的应用了默认样式的表格一样。由于我们没有设置这些垂直的线条，因此我们只会看到这些间隙。但是，通

过折叠边框，这些间隙会被清除。我们还为每个单元格内部添加了一点内边距，以便在文本周围创建一些空白。另外，还在表题和表格之间添加了几像素的空白。最后，我们得到了一个比默认的带有 10 个盒子（即 10 个带默认边框的单元格）的版本更简洁、更清晰的表格外观。

既然我们理解了基本的表格标记，那么接下来就看一些在同样的标记中加入变化效果的代码，如图 6-7 所示，我把这种效果称之为“横竖条纹”（Bars-n-Stripes）。

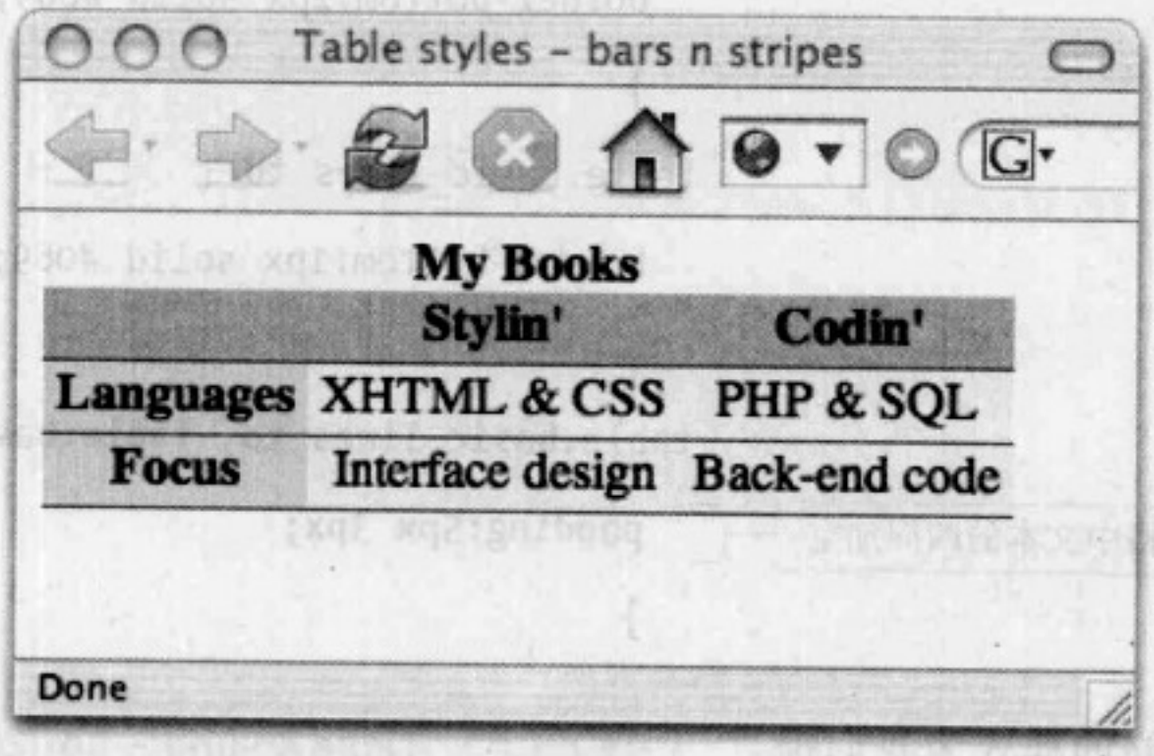


图 6-7　同样的标记，不一样的外观——添加了彩色背景（另见彩插）

```
table.bars_n_stripes {
  width:300px;
  border-collapse:collapse;
  }
table.bars_n_stripes td {
  background-color:#FFFFCC;
  }
table.bars_n_stripes th {
  background-color:#CCFFCC;
  }
table.bars_n_stripes th[scope="col"] {
  background-color:#99CCCC;
  }
```

表格数据区

（行、列）表头单元格

列表头单元格

为表头各单元格添加背景颜色——IDWIMIE（上一条规则适用于 IE 6）

```
table.bars_n_stripes td, table.bars_n_stripes th  {
  padding:3px 3px;
  border-bottom:1px solid #069;
  }
```

在每个单元格的文本与单元格边界之间添加空白

每个表格行下方的细线

这个表格具有300像素的固定宽度。如果删除这个设置，那么表格会随着添加文本而变宽。

在理解了前一个例子的基础上，这个带有一条提示的例子也很容易理解。我们注意到，在上面的CSS代码中，使用了一个属性选择符——相应的规则只适用于带有scope="col"属性的元素。这条规则可以使我们对准表格中的第一行，并为该行添加深蓝绿色的背景颜色。当然，差劲的IE 6不会理解这个属性选择符，因此这个例子在IE 6中的效果是：表格中的第一行与Languages和Focus单元格具有相同的背景颜色——不过，这个结果也是可以接受的。这就是对我们设计者来说永远应该坚持的所谓的“平稳退化”（graceful degradation）或者“渐进增强”（progressive enhancement）原则。无论哪个原则，其含义都是某些人会因为他们对的浏览器设置而获得比其他人更好的体验。不过，我们更喜欢强调的是，只要每个人都获得了可以接受的体验，就是值得欣慰的。下面的提示条解释了我介绍后面例子的目的，虽然后面的例子只有Safari和Firefox支持，但在其他浏览器中也都能够得到可以接受的效果。

多数人每次只使用一种浏览器

我经常会遇到一种情况，与我共事的一些设计者会向我展示在某些浏览器（比如IE 6）中显示正确的页面，然后又说“可是，如果用户发现它们在Firefox中看起来更好，那么他们会怎么想呢？”。对这个问题，我通常会回答说“不会出现这个问题！”。因为使用IE 6的用户，不会同时使用IE 6和Firefox来查看同一个网站。只有像我们这样可笑的人才这样做。只要网站对IE 6的用户而言没有问题（没有明显的毛病），那用户就不会感觉有什么差异。我的观点是：不必执著于让所有一切在每个浏览器中看起来都完全一样——这是不可能的，只有你自己知道“完美”意味着什么。

在下面的表格例子中（我将其称之为tic-tac-toe [①]），表格四周单元格的外边都没有边框。无论表格中包含多少行或者列，外部单元格的外边都没有边框，但所有内部单元格的四边都有边框。为了更清楚地体验到这个例子中的效果，我在水平和垂直方

① 一种井字格游戏。——译者注

向上复制了表格中行和列的标记。最终的样式如图 6-8 和图 6-9 所示。

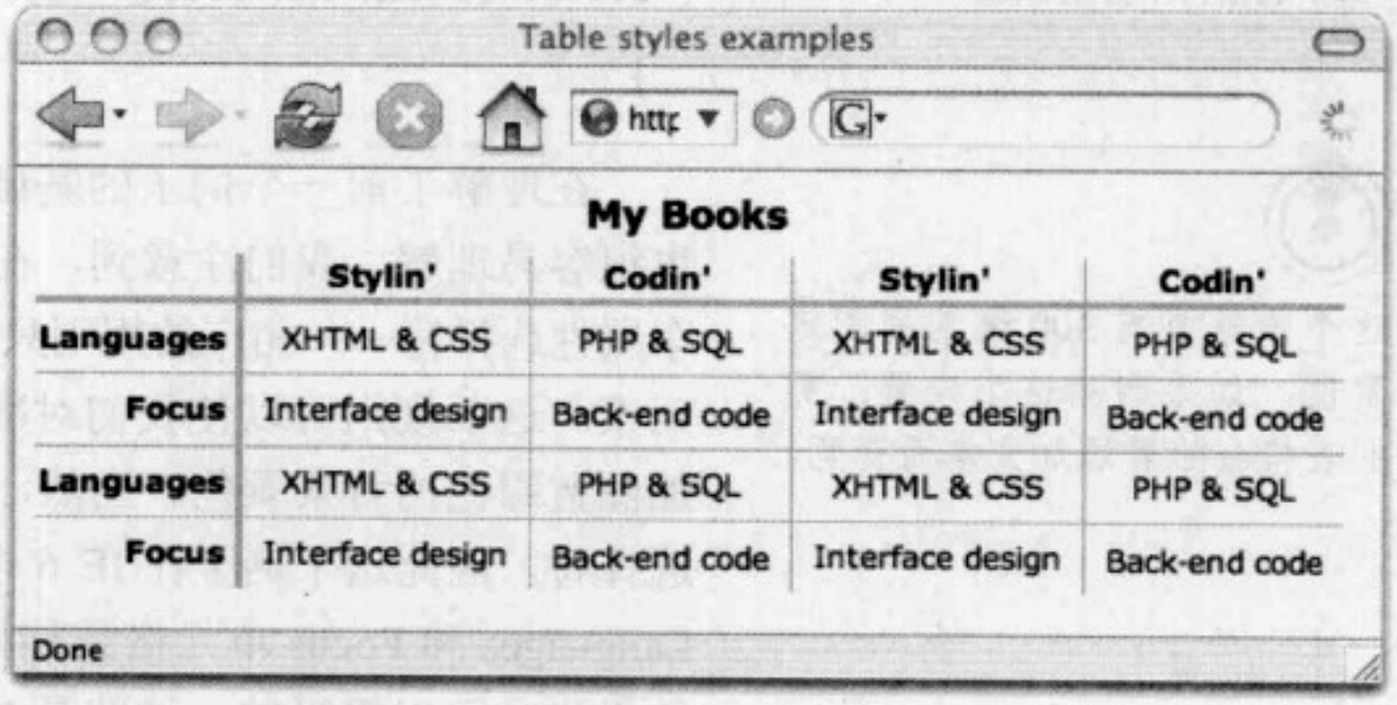


图 6-8 这个“tic-tac-toe”式表格外围的开放式边界是通过属性选择符和伪类实现的

完成这个效果需要依赖于伪类的运用，虽然 IE 6 不支持这种伪类，但设计结果在 IE 6 中平稳退化后仍然是可以接受的。

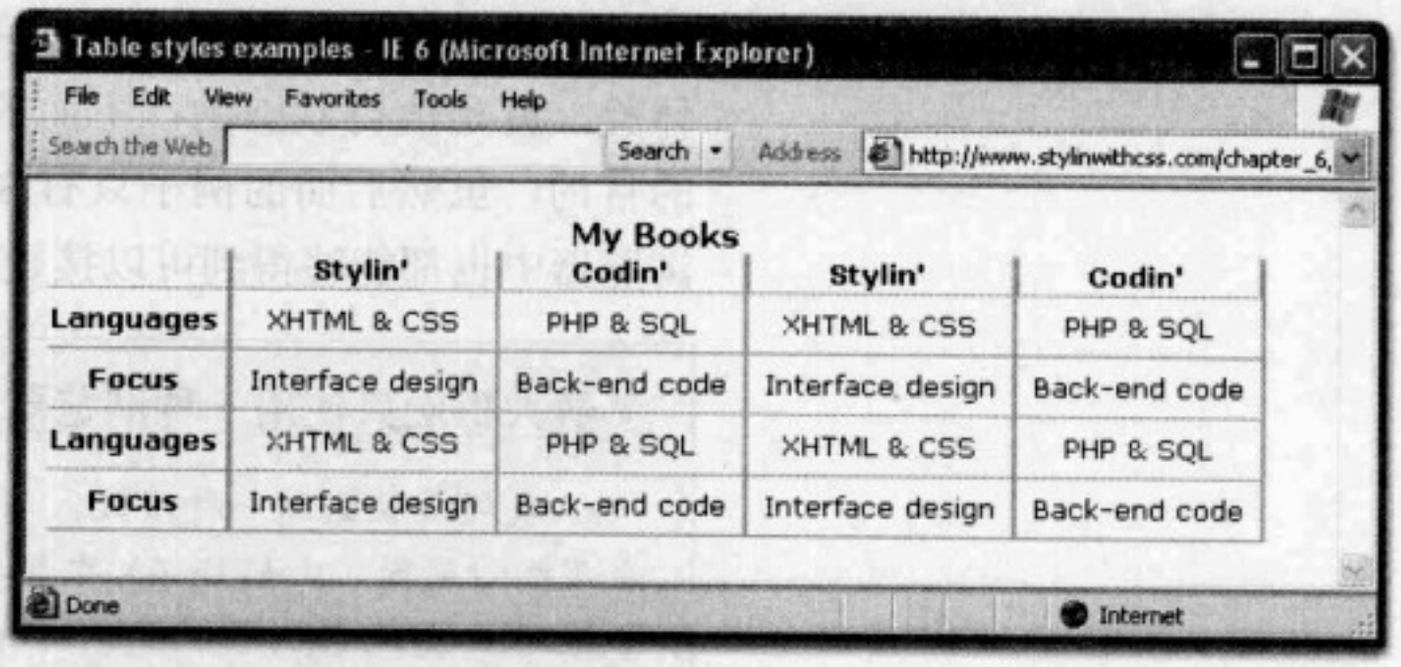


图 6-9 IE 6 不支持本例中使用的伪类，因此不能呈现通过相应伪类规则指定的细节。不过，最终的表现仍然很不错

下面，我们从头开始介绍一下创建这种效果的过程。

```
table {
  border-collapse:collapse;
  }
```

创建单元格的网格

```
table.tic_tac_toe td {
  border-right: 1px solid #99CCCC;
  border-bottom: 1px solid #99CCCC;
  }
```

（行、列）表头单元格（但下面会覆盖列表头单元格）

```
table.tic_tac_toe th {
border-right:3px solid #99CCCC;
border-bottom: 1px solid #99CCCC;
padding-right:.3em;
}
```

我们首先为 td 和 th 单元格添加边框。在折叠边框的情况下，表格单元格之间的边框是共享的。因为我们不想要外部单元格的上边框和左边框，所以只为每个单元格设置了下边框和右边框，结果如图 6-9 所示。

如前所述，这也是 IE 能够显示这个例子的最大限度，因为它不支持本例用到的伪类。

对于更符合标准的浏览器，我们需要继续处理表格的第一行。这里，需要加粗包含列表头单元格的行下方的线条，并让该行每个单元格的右边框（除了第一个单元格）变细。下面，我们再添加两条完成这一效果的选择符（见图 6-10）。

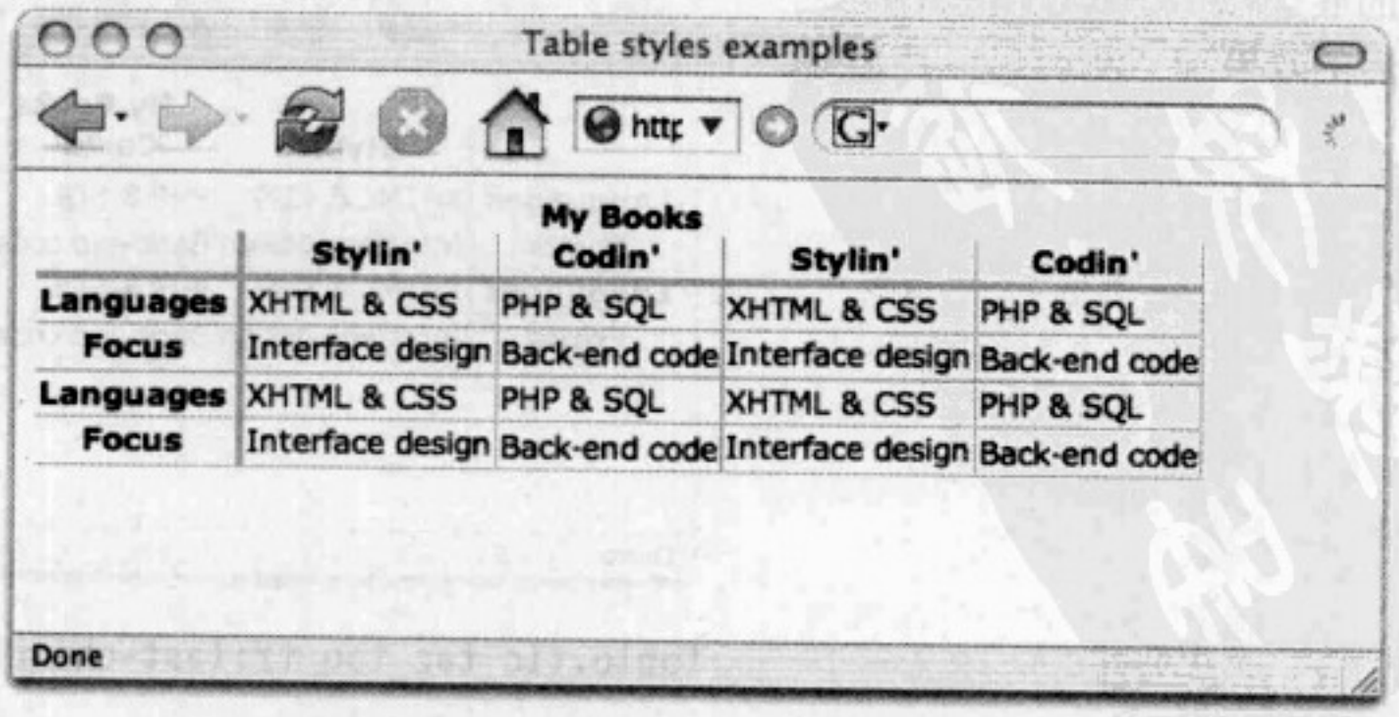
列表头单元格

```
table.tic_tac_toe th[scope="col"] {
border-right:1px solid #99CCCC;
border-bottom:3px solid #99CCCC;
}
```

列表头的第一个单元格

```
table.tic_tac_toe th[scope="col"]:first-child {
border-right:3px solid #99CCCC;
}
```

图 6-10 表头中的边框实现了期望的效果



这里使用了属性选择符来对准表格中的第一行，即带有 scope="col" 属性的 th，以便将这些单元格的右边框和下边框分别设置为 1 像素和 3 像素。然后，还需要把第一行中的第一个单元格的右边框再设置回 3 像素。为了对准这个单元格，我们组合使用了属性选择符和 first-child 伪类：th[scope="col"]:first-child，这个组合选择符的含义是“带有 scope="col" 属性的第一个 th”。在选准了该元素后，我们将它的右边框加粗到 3 像素。

现在要做的只剩分别去掉表格下方单元格的下边框和表格右侧单元格的右边框了。它们当前的样式都带有我们在第一步中为 th 和 td 添加的右和下边框。我们首先从 th 开始。

为此，需要对准行表头中的最后一个单元格（最后一个 scope="row" 的 th）和列表头中的最后一个单元格（最后一个 scope="col" 的 th），分别设置它们的下边框和右边框。

列表头中的最后一个单元格

```
table.tic_tac_toe th[scope="col"]:last-child {
  border-right:0;
  }
```

行表头中的最后一个单元格

```
table.tic_tac_toe tr:last-child th {
  border-bottom:0;
}
```

接下来，我们移除表格最后一行的底线（实际上是该行单元格 td 的下边框）和最后一列的右边框，结果如图 6-11 所示。

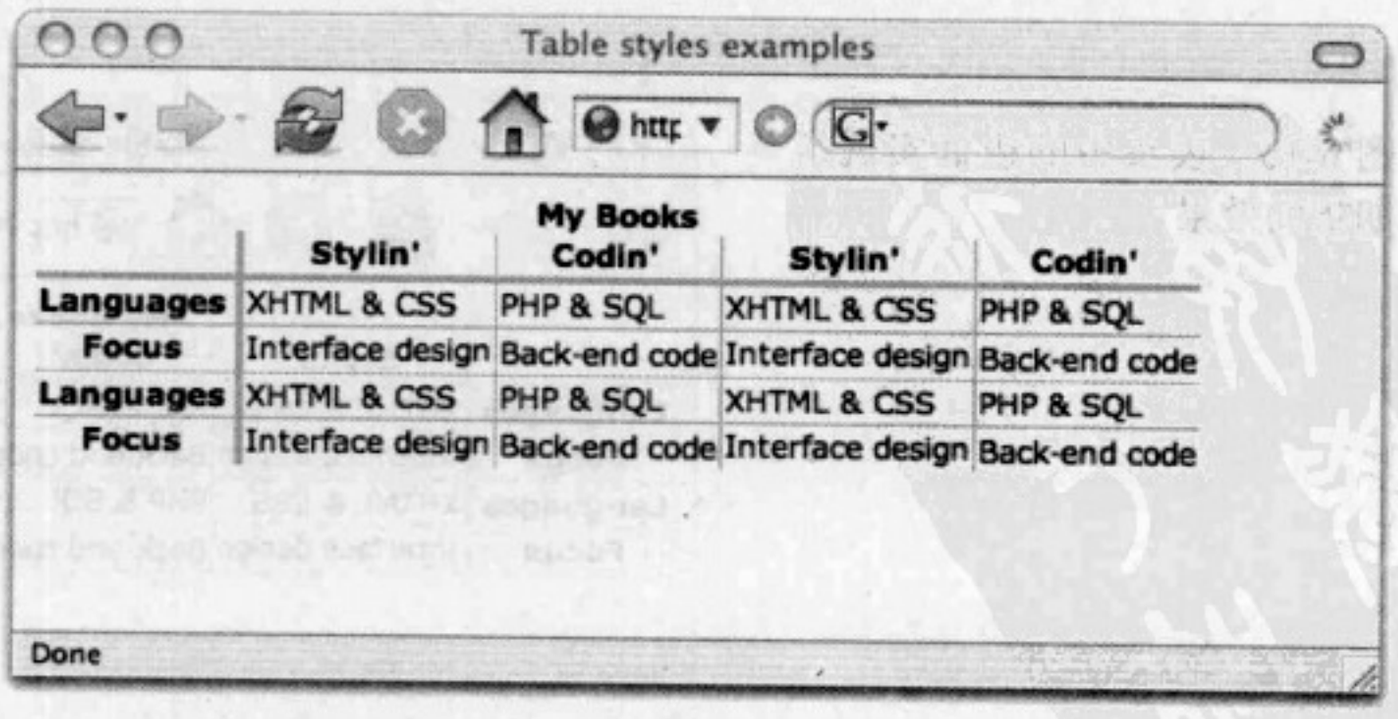


图 6-11 删除了表格最后一行的底线和最后一列的右边框之后的效果

```
table.tic_tac_toe tr:last-child td {
  border-bottom: 0;
```

```
}
table.tic_tac_toe td:last-child {
  border-right:0;
  }
```

第一个选择符对准了表格最后一行中的单元格。第二个选择符只需使用简单的 td:last-child，因为每行的最后一个单元格都是其最后一个子元素（last-child），即它的父元素是表格行。在下面完成后的代码中，选择符与前面逐步看到的稍微有些不同，这一方面是出于清晰的考虑，另一方面是因为其中有些规则需要覆盖前面的样式。

折叠所有表格边框——使单元格共享边框

```
table.tic_tac_toe {
  border-collapse:collapse;
  }
table.tic_tac_toe td {
```

创建表格单元格的网格

```
  border-right:1px solid #99CCCC;
```

创建表格单元格的网格

```
  border-bottom:1px solid #99CCCC;
```

居中所有 td 单元格中的文本

```
  text-align:center;
  }
```

去掉表格最后一行底部的下边框

```
table.tic_tac_toe tr:last-child td {
  border-bottom:0;
  }
```

去掉表格每一行最后的右边框

```
table.tic_tac_toe tr td:last-child {
  border-right:0;
  }
```

影响（行、列）表头单元格（但下面会覆盖列表头单元格）

```
table.tic_tac_toe th {
  border-right:3px solid #99CCCC;
  border-bottom:1px solid #99CCCC;
  text-align:center;
```

防止右对齐的文本接触单元格的右边框

```
  padding-right:.3em;
  }
```

去掉最后一个行表头单元格的下边框

```
table.tic_tac_toe tr:last-child th {
    border-bottom:0;
    }
```

表格第一行的边框，但是IDWIMIE6——前面的规则适用于IE 6

```
table.tic_tac_toe th[scope="col"] {
    border-right:1px solid #99CCCC;
    border-bottom:3px solid #99CCCC;
    }
```

加粗第一个列表头单元格的右边框

```
table.tic_tac_toe th[scope="col"]:first-child {
    border-right:3px solid #99CCCC;
    }
```

最后一个列表头单元格没有右边框

```
table.tic_tac_toe th[scope="col"]:last-child {
    border-right:0;
    }
```

行表头中的文本右对齐

```
table.tic_tac_toe th[scope="row"] {
    text-align:right;
    }
```

为所有单元格添加内边距

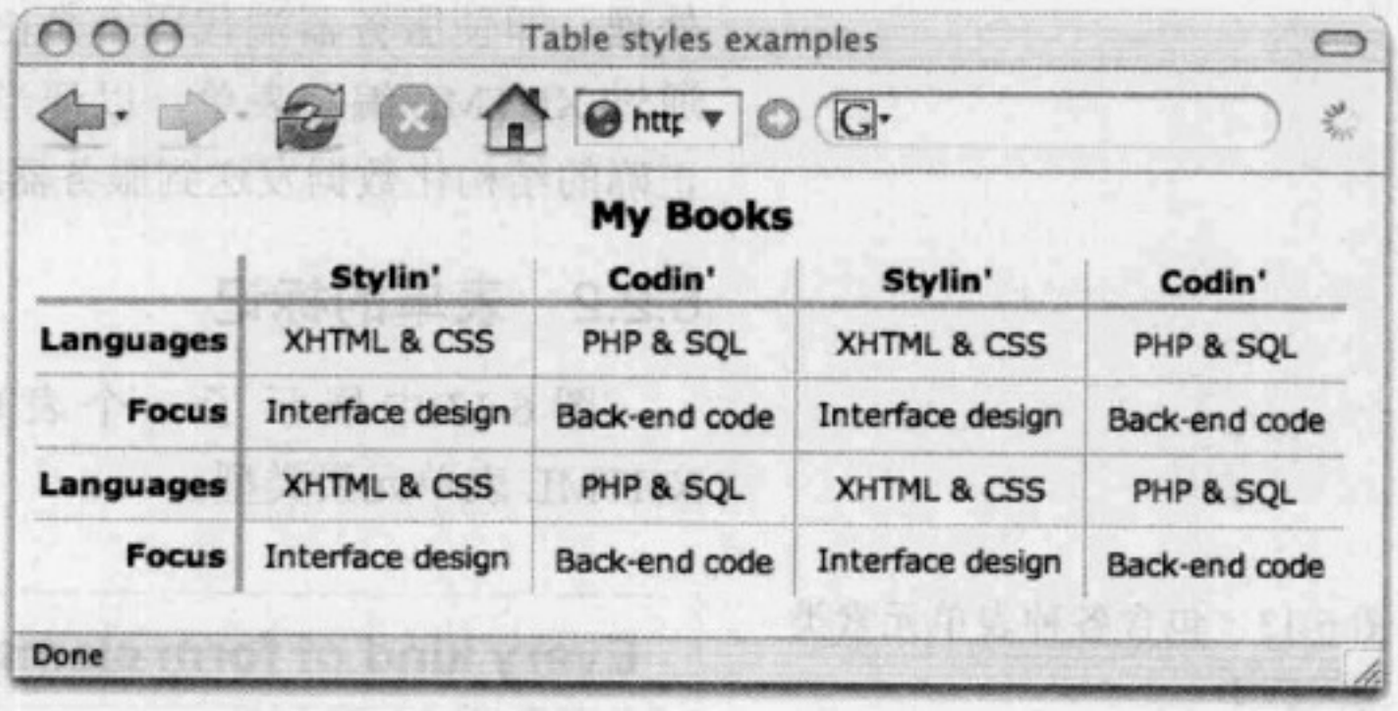
```
table.tic_tac_toe td, table.bars_n_stripes th  {
    padding:4px 8px;
    }
```

表题的样式

```
table.tic_tac_toe caption {
    margin-bottom:.5em;
    font-size:1.2em;
    }
```

我们注意到，在这些样式的最后，也添加了将行表头中的文本右对齐的规则，同时也在文本周围创造了一些空白，如图 6-12 所示。这里，通过使用属性选择符，在 IE 中隐藏了 text-align:right 声明，因为 IE 不支持属性选择符。如果在这里采取其他方式，那么行、列表头都将变成右对齐，从而导致表格列表头中的文本看起来不协调——它们必须居中才好看。IE 6 用户看到的行表头中的文本是居中的，如前一个屏幕截图所示——我认为，这个结果也是很优雅的。

图 6-12 完成后的“tic-tac-toe”式的表格



本章这一部分的用意是让你对样式化表格的可能性有一个认识，我们介绍了如何将表格变成页面中有感染力的区域，并将用户的视线吸引到表格中的数据上；换句话说，表格不一定由一堆令人讨厌的让用户不想多看一眼的单元格盒子组成。最后，我想说的是还有一个与表格相关的 XHTML 元素这里没有介绍。如果你想成为一名 XHTML 表格专家，建议学习 Roger Johansson 在他精彩的 456 Berea Street 网站中发表的一篇文章“Bring on the Table”（http://www.456bereastreet.com/archive/200410/bring_on_the_tables/）。

6.2 为表单添加样式

处理表单数据超出了本书的范围，不过，如果你想学习如何将表单数据发送到服务器，验证数据是否有效（例如，电子邮件地址中是否包含 @ 符号），为用户给出错误提示以便纠正错误，然后将验证后的数据写入文件或数据库，我推荐同样由 New Riders 出版社出版的我的另一本书 *Codin' for the Web*。这本书涵盖了在你掌握本书的技术之后，下一步应该掌握的与 Web 开发相关的技能。

表单对于多数网站而言都是必不可少的，因为它是把用户输入的数据通过因特网发送到网站所在 Web 服务器的手段。从简单的登录和注册，到多页面电子商务验证，表单可以说无处不在。理解如何创建表单也是每一个网站设计人员需要掌握的关键技能。鉴于表单的重要性，我们首先详细介绍一下表单的工作原理及相应的 XHTML 标记方式，然后再讨论如何为它们添加样式。

6.2.1 表单的工作原理

表单的用途是收集用户提供的各种数据。当表单被提交后（通常是单击一个按钮），表单中的数据会结构化成一个“名/值”对（例如 user_name=tracey）的集合，然后传递到服务器由服务器端的脚本（一般是用 PHP、Java 或 Perl 编写的）进行

处理。即使服务器端代码不是由你编写的，你也应该知道如何通过 XHTML 编写表单，以便当用户单击“提交”按钮时，将正确的结构化数据发送到服务器进行处理。

6.2.2 表单的标记

图 6-13 中展示了一个表单，其中包含了各种不同的 XHTML 表单元素类型。

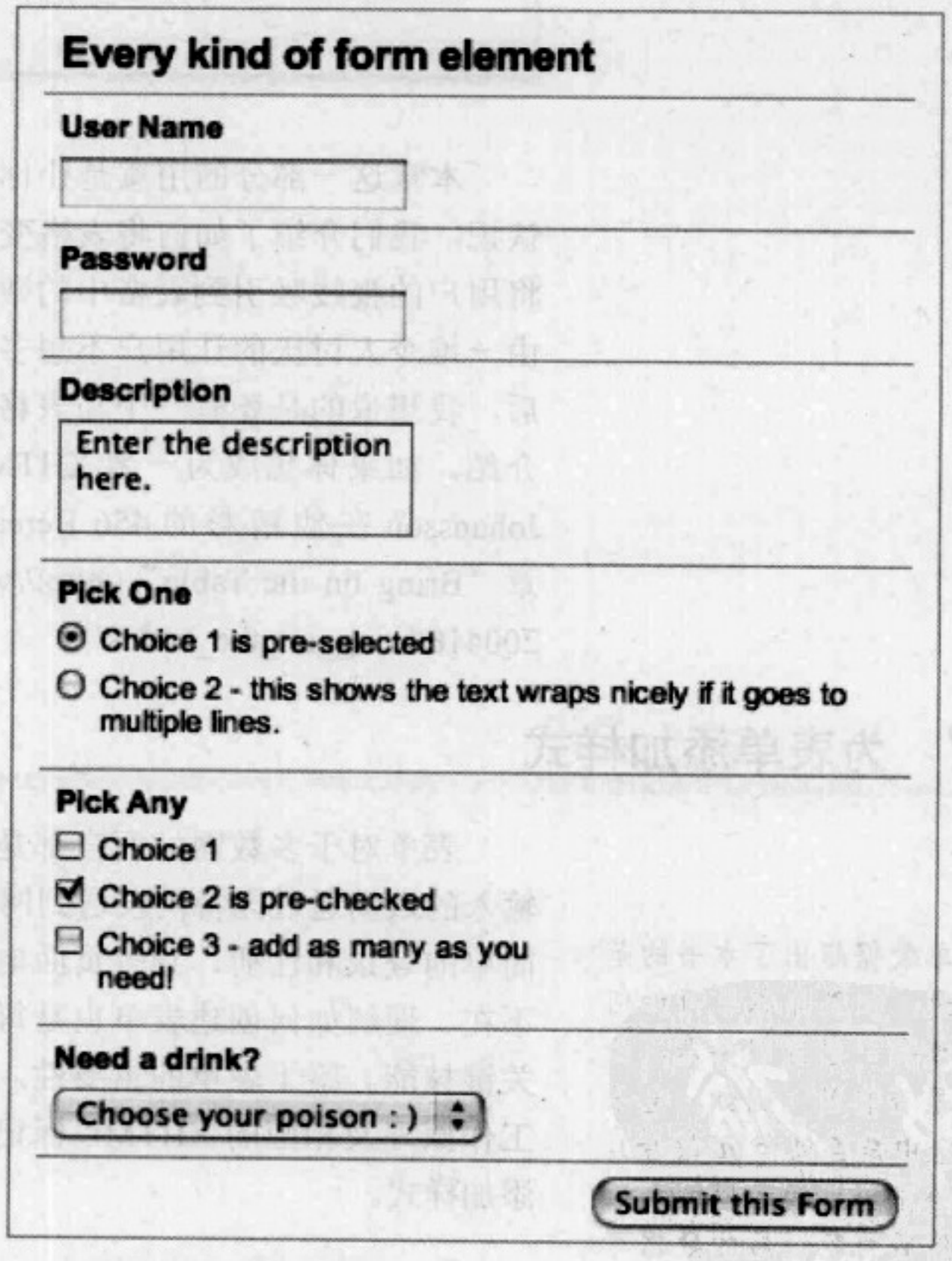


图 6-13 包含各种表单元素类型的例子

以下是构成该表单的代码：

```
<body>
<!-- FORM WIITH LABELS ABOVE INPUT FIELDS -->
<div class="two_col_form">
```

```
<form action="process_form.php" method="post">
<div class="formsection">
  <h3>Every kind of form element</h3>
```

添加 class="message" 可以显示该列表——用于显示错误信息

```
  <ul class="">
    <li>This list can be used to provide general info about
    filling the form</li>
    <li>Use this list also to display errors in the form.
    Adding an "error" class to the ul will turn the text
    red.</li>
    <li>You can cut and paste the form elements to create
    a form for your needs and it will automatically style
    itself to look like this one.</li>
  </ul>
</div>
```

单行文本字段

```
<div class="formsection">
  <label for="user_name">User Name</label>
  <input type="text" id="user_name" name="user_name"
  size="18" maxlength="36" tabindex="1" />
```

单行文本字段结束

```
</div>
```

密码字段

```
<div class="formsection">
  <label for="password">Password</label>
  <input type="password" id="password" name="password"
  size="18" maxlength="20" tabindex="2" />
```

密码字段结束

```
</div>
```

多行文本字段

```
<div class="formsection">
  <label for="description">Description</label>
  <textarea id="description" name="description" rows="3"
  cols="14" tabindex="3">Enter the description here.
  </textarea>
```

多行文本字段结束

```
</div>
```

单选按钮

```
<div class="formsection clearfix">
  <label for="radioset">Pick One</label>
    <div class="buttongroup" id="radioset">
```

```
<input checked="checked" id="radio1" name="radioset"
type="radio" value = "Choice_1" />

    <label for="radio1">Choice 1 is pre-selected</label>

<input id="radio2" name="radioset" type="radio"
value="Choice_2" />

<label for="radio2">Choice 2 - this shows the text wraps
nicely if it goes to multiple lines.</label>
```

按钮组结束

```
</div>
```

单选按钮结束

```
</div>
```

复选框

```
<div class="formsection clearfix">

<label for="checkset">Pick Any</label>

  <div class="buttongroup clearfix" id="checkset">

     <input type="checkbox" id="check1" name="checkset" value
     = "1" tabindex="4" />

  <label for="check1">Choice 1</label>

  <input type="checkbox" checked="checked" id="check2"
  name="checkset" value = "2" />

    <label for="check2">Choice 2 is pre-checked</label>

  <input type="checkbox"  id="check3" name="checkset" value =
  "3" />

    <label for="check3">Choice 3 - add as many as you
    need!</label>
```

按钮组结束

```
</div>
```

复选框结束

```
</div>
```

选择列表，也叫弹出菜单

```
<div class="formsection">

<label>Need a drink?</label>

    <select id="drink_choice" name="drink_choice">

    <option value="0">Choose your poison : )</option>

    <option value="tea">Tea</option>

    <option value="coffee">Coffee</option>

    <option value="water">Water</option>
```

```
            <option value="beer">Beer</option>
        </select>
选择列表结束 ──{ </div>
将输入字段包含在块级元素以便通过验证 ──{ <div>
提交按钮 ──{   <input type="submit" value="Submit this Form" />
    </div>
</form>
</div>
</body>
```

下面，我们依次来看一看这个表单的各个区域的标记。在屏幕截图中，每个部分都由一条细线分隔。在标记中，每个部分都包含在一个带有 formsection 类的 div 中。

第一个 formsection[①]中没有包含表单元素，只包含一个列表。可以通过这个列表向用户显示操作说明，不过它的主要用途还是显示错误信息。如图 6-13 所示，这个列表通常是隐藏的（它的 display 属性被设置为 none）。服务器端代码（例如 PHP）会动态生成一个包含错误信息的自定义列表，然后通过为它添加一个类使它显示在页面上。这样用户就会得知并纠正表单中的问题，然后重新提交表单。

也就是说，即使你不参与编写服务器端代码，也应该通过设置表单的 XHTML 和 CSS 使其具有显示错误信息的能力，然后，你可以同网站的技术团队合作来保证服务器端代码能够向代码中添加适当的文本和类，使其按照你的期望显示错误信息。

现在，我们深入分析一下前面表单的标记。

1. form 元素

每个表单都包含在一个 form 元素中。

```
<form class="2_col_form" action="process_form.php"
method="post">
```

其中，2_col_form 类用于关联表单和为表单提供样式的 CSS。表单的另外两个属性——action 和 method 都是必要的。

当用户提交表单并重载页面后，在表单顶部显示一个错误列表以便用户纠正错误是一个常见并非常基本的错误处理方式。在今天的众多网站中，通过使用“免刷新”的 Ajax 技术（如果你还不了解 Ajax，请参考 http://www.adaptivepath.com/publications/essays/archives/000385.php，Jesse James Garrett 在这篇文章中提出了 Ajax 的概念），可以在用户结束信息输入的同时就将错误信息显示在表单字段旁边。这种交互方式虽然对用户来说很友好，但实现起来却有点复杂。这里，我们只示范最基本的在表单顶部显示错误信息的方法。例如，我建议你体验一下 37signals.com 的应用程序中基于 Ajax 的实时表单验证的良好交互性。

① 即第一个带有 formsection 类的 div，下同。——译者注

其中，action 属性用于确定服务器上哪个页面中的代码用于处理这个表单（通常，处理代码与表单位于同一个页面中，这有点类似于 CSS 能够被嵌入到同一个页面的头部）。method 属性用于定义数据传递到服务器的方法：post 会使数据包含在 HTTP 头部信息中以不可见的方式传递，而 get 则会导致将数据作为 URL 字符串来传递。

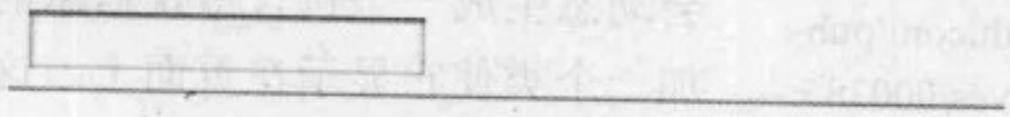
2. 表单控件

在开始和结束的 form 标签之间，可以包含任意数量的表单文本字段、单选按钮、复选框和选择列表等称为表单控件的元素（对于能够在其中输入文本的表单控件，通常又称作字段）。这些表单控件都应该带有一个 label 标签，以便为控件添加标注。而且，对于屏幕阅读器来说，label 标签也是辅助它们理解表单的重要手段，具体原因稍后解释。注意，不要使用段落或标题为表单控件添加标注。

表单中的前 3 部分示范了 3 种文本输入元素——text、password 和 textarea。

3. 文本输入控件

图 6-14　单行文本字段



text 是用于输入单行文本的输入元素，如图 6-14 中的 User Name 所示。文本字段的基本标记格式为：

```
<label for="user_name">User Name</label>

  <input type="text" id="user_name" name="user_name"
  size="18" maxlength="36" tabindex="1" />
```

表单中的大部分控件都基于 input 标签实现，input 标签中的 type 属性决定了该控件的外观（例如，是文本字段还是单选按钮）。由于这里输入（input）元素的类型是 text，因此会在页面上显示一个单行文本字段。

控件的 id 属性与对应的 label 标签关联——注意 label 的 for 属性与控件的 id 属性是完全相同的。控件的 name 属性用于在提交表单时，构成“名 / 值”对中的“名”部分；在这个例子中，如果用户输入的用户名是 css whiz，那么基于这个控件传递到服务器的“名 / 值”对就是 user_name=css whiz。

控件的 size 属性定义控件中可见的文本数量，在这里是 18 个字符。而 maxlength 属性则用于确定在控件中最多能输入多少字符，这里最多能输入 36 个字符。

4. 密码输入控件

```
<input id="password" type="password" size="18" tabindex="2" />
```

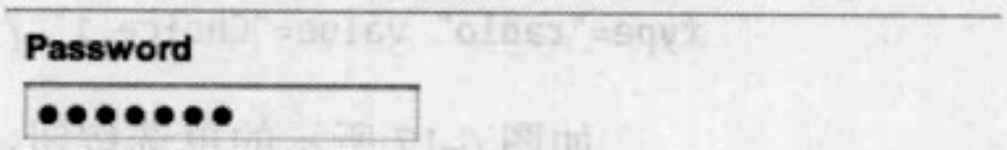


图 6-15 尽管看上去与单行文本字段相似，但在密码字段中输入的文本都会以圆点形式显示

类型为 password 的 input 元素也是一个单行字段，如图 6-15 所示。因此，前面的内容对于密码输入控件同样适用。但是，用户在密码输入字段中输入的文本在屏幕上将以点的形式显示，以防止别人看到你输入的密码。不过，密码字段中的内容是以“明文”的形式发送给服务器的；换句话说，在通过因特网发送的过程中，密码同其他文本内容一样都不会被加密。如果通过安全连接（即页面的 URL 以 https 开头的连接）提交文本，那么就会在浏览器与服务器间传递文本时对内容进行加密。一般电子商务网站中，都使用安全连接处理付款交易。

5. 文本区输入控件

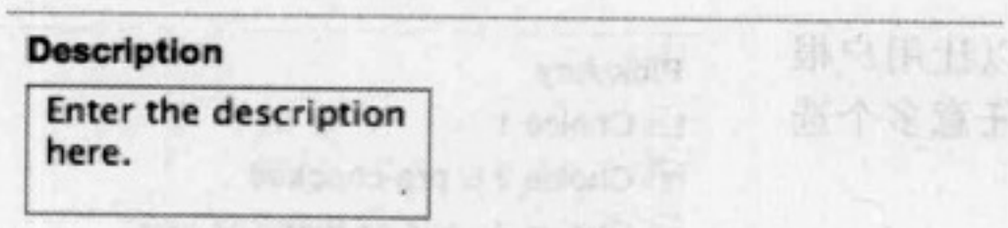


图 6-16 当需要用户输入超过一行的内容时，就要使用多行文本字段

```
<textarea id="description" name="description" rows="3"
cols="14" tabindex="3" >Enter the description here.
</textarea>
```

textarea 元素是一个多行文本字段，其外观如图 6-16 所示。与表单中的其他文本输入字段不同，文本区的标记带有结束标签，也就是可以在它的开始和结束标签之间输入当页面加载时显示的默认文本（如上面突出显示的代码所示）。代码中的 rows 和 cols 属性分别用于设置文本区的宽度和高度。如果用户输入的行数超过了 rows 属性的值，文本区右侧就会出现一个滚动条，拖动该滚动条就能看到所有输入的文本。

6. 单选按钮

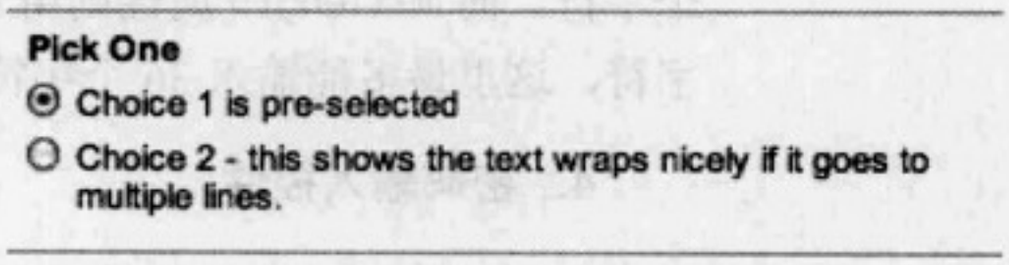


图 6-17 单选按钮可以让用户从多个选选项中选择一个选项

```
<input checked="checked" id="radio1" name="radioset"
type="radio" value="Choice_1" />
```

如图 6-17 所示的单选按钮，允许用户选择一个选项，这些选项是互斥的，也就是说，用户只能从中选择一个。当用户改变主意选择另一个选项时，当前选项就会取消选定（就像你汽车收音机中的按钮一样，故而得名[①]）。单选按钮用于在两个或多个选项中选择唯一可能的选项。通过为每个 input 元素中的 name 属性指定相同的值，就可以创建一组单选按钮，同一组中的单选按钮在任何时候都只能有一个处于被选定的状态。由于单选按钮没有用户输入的数据与之关联，所以当特定的单选按钮被选中时，它的 value 属性值会被传递到服务器。此外，通过为其中一个单选按钮添加 checked 属性，可以使它在页面加载后处于预先选定的状态。

7. 复选框

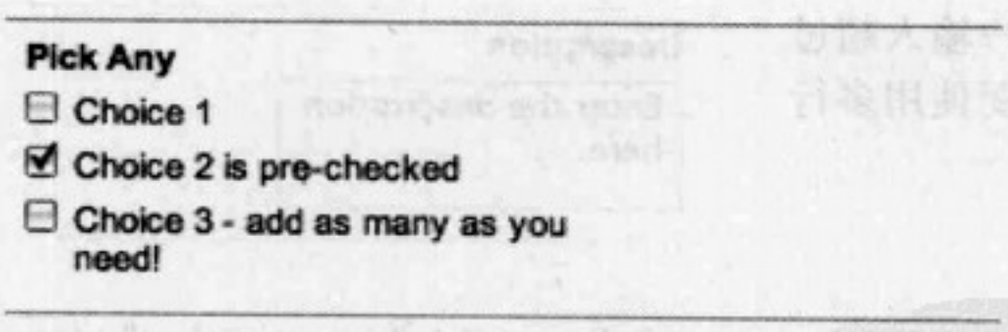


图 6-18 复选框可以让用户根据自己的想法选择任意多个选项

```
<input name="checkset" id="check1" type="checkbox" size="35"
value = "Choice_1" tabindex="4" />
```

复选框与上面介绍的单选按钮之间唯一的差别，就是（如图 6-18 所示）其中的选项不是互斥的，用户可以从中选择任意多个选项。复选框用于允许用户选择多个或全部选项的情况，例如通过网上购买三明治时，让用户选择西红柿、生菜、洋葱和泡菜等。

① 单选按钮的英文是 radio，意为收音机。——译者注

8. 选择列表

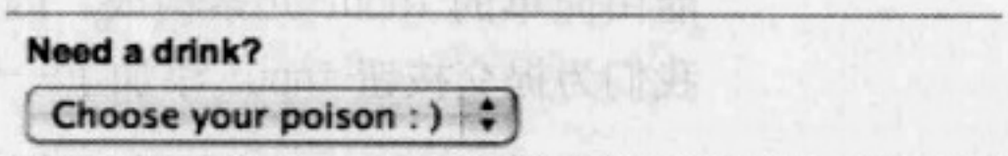


图 6-19 选择列表可以让用户通过弹出菜单来选择一定数量的选项

```
<select name="drink_choice">
  <option value="0">Choose your poison : )</option>
  <option value="tea">Tea</option>
  <option value="coffee">Coffee</option>
  <option value="water">Water</option>
  <option value="beer">Beer</option>
</select>
```

下一个表单元素是 select（其外观如图 6-19 所示），它的更广为人知的名字叫做弹出菜单。我想，通过上面的标记，你应该很清楚选择列表的原理了。其中唯一需要注意的是，传递给服务器的“名 / 值”对中的“名”是 select 元素的 name 属性值，而“值”则是选择的选项的 value 属性值（例如 drink_choice=tea）。

9. 提交按钮

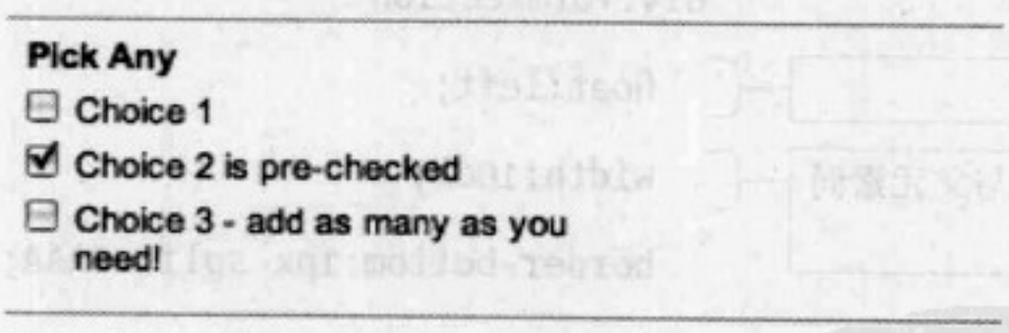


图 6-20 单击提交按钮会将所有表单数据提交给在表单元素的 action 属性中指定的 URL

为通过验证，必须将 input 包含在一个块级元素中

```
<div>
<input type="submit" value="Submit this Form" />
</div>
```

最后，是如图 6-20 所示的提交按钮。由于提交按钮的标记很像是文本字段，但是通过将同一个 input 元素的 type 属性设置为 submit，它就会魔术般地变成一个按钮。按钮中的 value 属性用于定义按钮上面显示的文本。注意，当使用 Strict 类型的 DOCTYPE 时，不能单独把行内元素放到页面中——这是 XML 中不可改变的规则，该规则要求必须把行内元素包含在适当的

块级元素中。而且，根据 W3C 验证程序的错误提示，表单不能由简单的 input 元素构成。因此，与页面中的其他控件一样，我们为提交按钮 input 添加了一个 div 父元素。

通过以上分析，我们理解了表单的标记。下面，我们讨论表单的样式。

6.2.3 表单的样式

以下是图 6-12 中所示表单的 CSS 规则，其中都是我们曾经学习过的 CSS 属性。多数浮动声明都用于使 div 包含内部浮动的元素。这样我们就不必使用第 4 章中介绍的 clearfix 代码了。

```
h3 {margin-top:.5em;}
form {
  float:left;
  width:24em;
  margin:20px 0 0 50px;
  padding:1em .75em .5em;
  border:1px solid #AAA;
  }
div.formsection {
  float:left;
  width:100%;
  border-bottom:1px solid #AAA;
  }
input {
  font-size:.8em;
  }
input:focus, textarea:focus, select:focus {
  border:2px solid #7AA;
  }
label {
```

包裹内部浮动的 formsection

这个例子使用的临时样式——用于将表单推离页面左上角

包裹表单控件及标注

使浮动的元素全宽（与父元素同宽）

当单击字段时突出显示

```
    display:block;
    clear:both;
    font-size:85%;
    font-weight:bold;
    margin:.5em 0 0;
    padding-bottom:.5em;
    }
```

IDWIMIE6，因此在 IE 6 中按钮位于左侧

```
input[type="submit"] {
    float:right;
    margin:.5em 0;
    }
```

为复选框和单选按钮集合添加样式

```
.buttongroup {
    float:left;
}
.buttongroup input {
    float:left;
    clear:both;
    margin:0;
    padding:0;
    line-height:0;
    }
.buttongroup label {
    background-color:none;
```

注意：这里是 24×85%（继承的值），因此实际宽度只有 20.4 em

```
    width:24em;
    float:left;
```

为相邻的复选框和复选框及标注之间创建空白

```
    margin:0 0 0 .5em;
```

重设继承的值

```
    font-weight:normal;
```

重设继承的值

```
    clear:none;
    }
```

我们将 form 元素（即表单中其他元素的顶级元素）的宽度设置为 24 em。将围绕每个标注和控件组的 formsection 设置为大小不限（100%），因此它们会扩展到与表单同宽。通过将 label 元素的 display 属性设置为 block，同时移除它的外边距（第一处突出显示的代码）——除了上面的一点空间外，会使 3 个文本输入字段及相应的标注在页面中有序地堆叠起来。只要不将文本字段的宽度（由 XHTML 中的 size 属性决定）设置为超过表单元素的 24 em，这就是一个相当可靠和简洁的布局。

在处理单选按钮和复选框时，问题会变得复杂一些。不仅需要为每个复选框或单选按钮组添加相应的标注，而且也要为组中的每个元素添加单独的“子标注”。因此，我们将每组单选按钮或复选框及其各自的标注包装在一个带有 buttongroup 类的 div 中。通过浮动这个 div，可以使它包裹同样浮动（突出显示的代码）的内部控件及标注。而为这些标注设置较小的宽度，一方面可以为小单选按钮或复选框留出空间，另一方面也可以使标注向上浮动到它们旁边并与之对齐。还记得第 4 章中通过浮动图像和文本块形成分栏的例子吗？这里实际上也是同样的原理。如果个别复选框的标注会折行，它也会与上面行的开头完美地对齐，而后面的所有元素都会向下移动为它留出空间。

最后，我们希望把提交按钮放到右侧，因为放到右侧会比较符合用户的操作习惯。由于只需对准提交按钮而不是表单中所有类型的 input，所以我们使用了属性选择符来选择类型为 submit 的 input（见上面突出显示的 float:right）。当然，IDWIMIE6，也就是说，在 IE 6 中，提交按钮仍然会处理左侧，但也没什么大不了的——只是位于右侧会具有更好的可用性。

以上就是对表单样式的分析。由于对表单中的控件需要正确设置很多重要的 XHTML 属性，所以标记表单是一件比较麻烦的事。不过，上面讨论的标记和样式能够帮你加快这一过程。当你要在自己的项目中布局表单时，通过遵循上面的标记格式（实际上，只需从下载的例子代码中将相应的 formsection 剪贴到你的页面中即可），就可以创建出所需的各种控件。然后，你可以按照自己的特殊需求修改个别元素的属性，而当在页面中链接了这个样式表时，表单的布局立即就会呈现前面所看到的效果。

微型登录表单

在设计表单的过程中，最具挑战性的就是将它们放在一个很小的空间中。网站侧边栏中的登录表单就是这样一个例子（见图 6-21）。

图 6-21 这个整洁的表单被挤进了只有 128 像素宽的分栏中（另见彩插）

在图 6-21 中，灰色的盒子是包含表单的 div，通过它的边框可以看出这个分栏很窄——只有 128 像素。

以下是这个微型表单的标记：

这个 div 是要放置微型表单的分栏

```
<div id="column">
```

微型的垂直登录表单

```
<form id="tiny_form_vert" action="#" method="post">
```

单行文本字段

```
    <div class="formsection">
        <label for="user_name">User Name</label>
        <input id="user_name" type="text" size="10" />
```

单行文本字段结束

```
    </div>
```

密码字段

```
    <div class="formsection">
        <label for="password">Password</label>
        <input id="password" type="password" size="10" />
```

密码字段结束

```
    </div>
```

提交按钮

```
    <div class="formsection">
```

必须把 input 包装在块级元素中才能通过验证

```
        <input type="submit" value="Sign-in" />
```

提交按钮结束

```
    </div>
```

用于显示错误和链接的列表

```
    <div class="formsection">
```

通过添加 "error" 类显示错误信息的段落

```
        <p class="">Problem with name or password</p>
```

```
            <ul>
              <li><a href="#">Sign Up</></a></li>
              <li><a href="#">Lost Password</a></li>
            </ul>
          </div>
微型登录表单结束 ——  </form>
分栏 div 结束 ——  </div>
```

首先，你可能会注意到，这个微型表单的标记是从前面较大的常规表单（该表单也包含在 Stylib 库中）中复制过来的，因此组合这些标记不需要很长时间。

其次，在这个标记的末尾，我们添加了一个段落（突出显示），该段落正常情况下会被隐藏起来，但当用户填写的表单出现错误（例如输入了无效的密码）并需要给出提示时，这个段落就会显示出来（见图 6-22）。当前，这个段落中的 class 属性中不包含值，如果把它的值设置为 error，那么段落就会显示在页面上。稍后我们会看到这一过程。

图 6-22 如果用户名或密码填写错误，表单下方会显示错误信息（另见彩插）

数据提交之后有效性的测试及为显示错误信息而添加的类都必须通过服务器端的中间件（例如 PHP）来实现。推荐通过我的书 *Codin' for the Web* 来学习相应的内容。

另外，在这种登录表单中经常会需要通过一个无序列表来包含两个链接。其中，Sign Up 链接会把未注册的用户带到一个页面（没有截图）上，创建用户名和密码。对于当前已经注册的用户，则可以通过 Lost Password 链接访问另一个页面（没有截图），在该页面中申请将密码发到自己注册时使用的电子邮箱中。

在为这个表单添加样式的 CSS 规则中，我们首先移除了每个元素上的外边距并设置了较小的文本大小。由于所有文本的大小都以 em 进行设置，所以当用户改变整体的字体大小时，这个布局也会相应地成比例变化。甚至，成比例变化的范围还会包括包含元素的宽度，当前它 8 em 的宽度等于 128 像素（16 乘以 8）。

以下是相应的 CSS 规则：

```
div#column {
    margin:20px;                          /* 为将表单推离左上角而在本例中添加的临时样式 */
    width:8em;                            /* 包含元素的宽度 */
    border:1px solid #AAA;                /* 包含元素的边框 */
    font-family:Arial, sans-serif;
    float:left;                           /* 使分栏包裹内容 */
    }
#tiny_form_vert * {                       /* 移除表单所有元素的内、外边距 */
    margin:0;
    padding:0;
    }
form#tiny_form_vert {
    padding:4%;                           /* 表单元素中的相对内边距 */
    float:left;                           /* 使表单包裹内容 */
    margin-bottom:0;
    }
#tiny_form_vert input[type="submit"] {
    float:right;                          /* 将提交按钮移到右侧 IDWIMIE */
    margin:.5em 0 .2em 0;                 /* 在按钮上下方创建空白 */
    }
#tiny_form_vert p {                       /* 为信息 / 错误段落添加样式 */
    display:none;                         /* 在不存在 error 类（见下面）的情况下隐藏段落 */
    margin-top:0em;
    width:100%;
    clear:both;                           /* 确保文本不会向上浮动到按钮旁边 */
    }
```

```
#tiny_form_vert ul {
  font-size:.85em;
  margin:0;
  padding:.2em;
  }
#tiny_form_vert li {
  list-style-type:none;
  }
#tiny_form_vert p.error {
  display:block;
  color:red;
  font-size:.8em;
  }
#tiny_form_vert a {
  color:#069;
  }
#tiny_form_vert a:hover {
  color:#336;
  text-decoration:none;
 }
```

减小链接的字体大小

移除列表中的项目符号

当添加了 error 类时显示段落

减小字体

未悬停时的链接颜色

悬停时的链接颜色

当翻转时移除链接的下划线

自己重新编写这些组件对于提升设计表单的技能非常有帮助。这个微型表单的标记和 CSS 规则相对较短而且也很简单，它示范了在误差要求很小的条件下如何组织嵌套的元素。通过以上设计，我们最终获得了一个简洁实用的界面组件。

下面，我们来看一看列表和菜单。

6.3 为列表和菜单添加样式

列表就是一组相关的文本项——例如某个过程中相关的对象或步骤的名字。列表可以是显示在一起的一系列简单的项，也可以是能够让用户导航到网站其他位置的链接。

菜单也是一种列表，它提供导航选项——文本可以被单击。本节，我们首先来介绍如何创建列表，然后再讨论可单击的菜单。最后，我们要创建一个当顶级选项处于悬停状态时，通过顶级选项展示子选项的多级菜单。首先，我们介绍列表。

6.3.1 列表

列表有 3 种类型：无序列表、有序列表和定义列表。虽然这 3 种列表的标记类似，但应该基于它们包含的内容来使用。

- 无序列表在默认时带有项目符号。可以把默认的项目符号修改成空心圆或方块，甚至可以用图像或实体（如腭化符号 ~）来代替默认的项目符号。
- 有序列表在默认时带有数字编号。可以把数字编号修改为字母或罗马数字。
- 定义列表（或嵌套列表）包含子项，可以将它用于标记术语词汇表。

列表的标记很简单。下面就是一个无序列表的代码：

```
<ul>
<li>Gibson</li>
<li>Fender</li>
<li>Rickenbacker</li>
<li>Ibanez</li>
</ul>
```

无序列表以一个无序列表（ul，unordered list）标签开始，其中包含一些列表项（li，list item），然后以另一个 ul 标签结束。

有序列表也差不多如此，只不过列表标签是 ol（ordered list）而不是 ul。

```
<ol>
<li>Gibson</li>
<li>Fender</li>
<li>Rickenbacker</li>
<li>Ibanez</li>
</ol>
```

在有序列表中，根据 list-style-type 属性的值，每个项目都使用数字、字母或罗马数字进行连续地编号。可以通过 list-style-position 属性来设置这些编号是位于列表的外部还是内部。

定义列表包含 3 个元素：

```
<dl>Web Terms
  <dt>XHTML</dt>
    <dd>A mechanism for indicating the structure of a
    document.</dd>
  <dt>CSS</dt>
    <dd>A mechanism for presentational styling of XHTML.</dd>
</dl>
```

定义列表以定义列表标签（dl，definition list）开始，然后可以包括任意数量的定义术语（dt，definition term）及相应的定义描述（dd，definitioin description）。

图 6-23 展示了在前面标记的基础上使用了几种 list-style-type 和 list-style-position 属性之后的无序列表、有序列表和定义列表。

1. 为列表添加样式

列表是导航和菜单的基础——毕竟，像侧边栏中的链接这样的导航元素，通常都由页面的链接组成。因此，菜单实际上就是一个选项列表。并且，将导航和菜单作为列表来看待并添加样式是一种最佳实践。这种思想的另一个优点在于，如果用户在一种不能应用 CSS 样式的用户代理（例如手机）中查看页面，那么 XHTML 的列表标记至少能够将导航或菜单表现为一组整洁嵌套的链接。下面，我们就从为一组导航链接添加样式开始，这些导航链接通常会出现在网站的左侧分栏中。

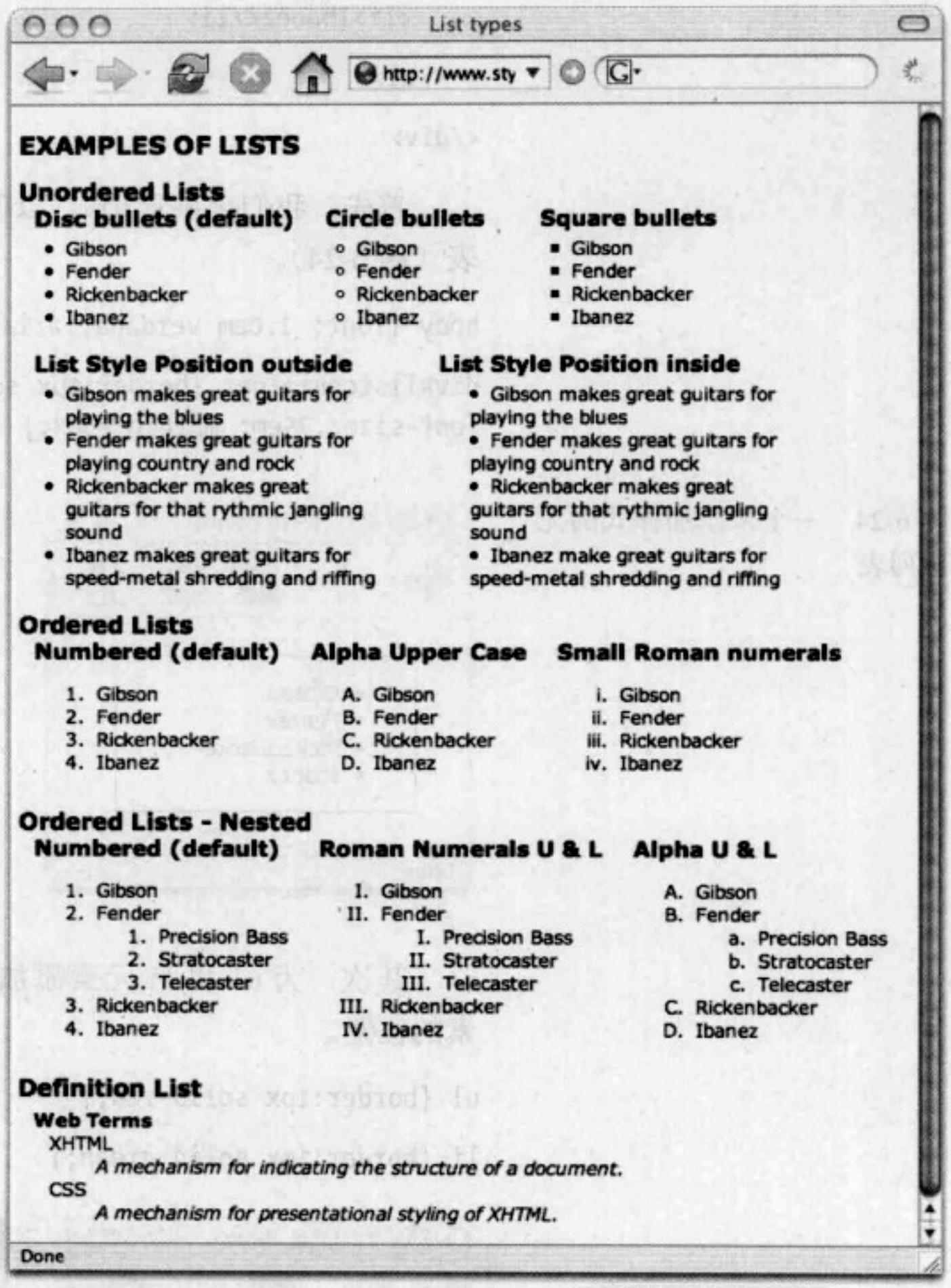


图 6-23 XHTML 为设置列表的格式提供了很多选项

以下是位于一个 div 中的无序列表的标记，因此我们可以相对于包含元素（在左侧导航的布局中，这个容器应该是左侧的分栏）来为该无序列表添加样式：

```
<div class="listcontainer">
  <ul>
    <li>Gibson</li>
    <li>Fender</li>
    <li>Rickenbacker</li>
```

```
    <li>Ibanez</li>
  </ul>
</div>
```

首先，我们来显示 div，以便能够看到其中未添加样式的列表（图 6-24）。

```
body {font: 1.0em verdana, arial, sans-serif;}
div#listcontainer {border:1px solid #000; width:160px;
font-size:.75em; margin:20px;}
```

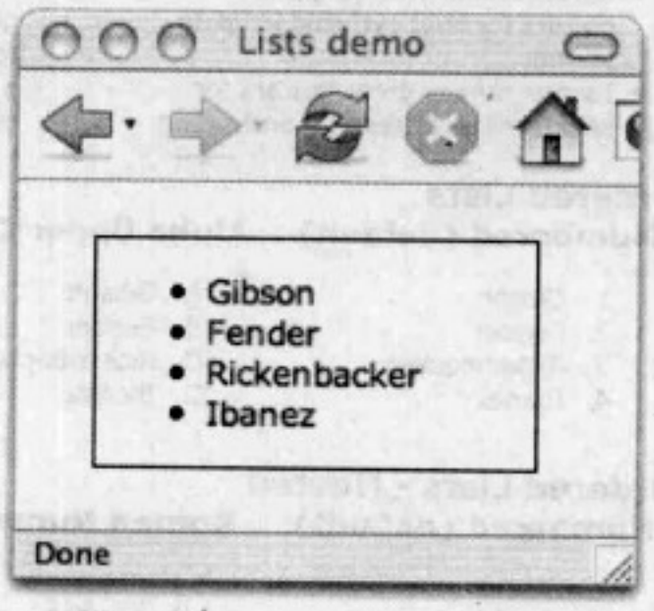


图 6-24 一个未添加样式的无序列表

其次，为 ul 和 li 元素添加边框，以便看清列表中每个元素的位置。

```
ul {border:1px solid red;}
li {border:1px solid green;}
```

图 6-25 在 Firefox 中显示的列表和列表元素的边框（另见彩插）

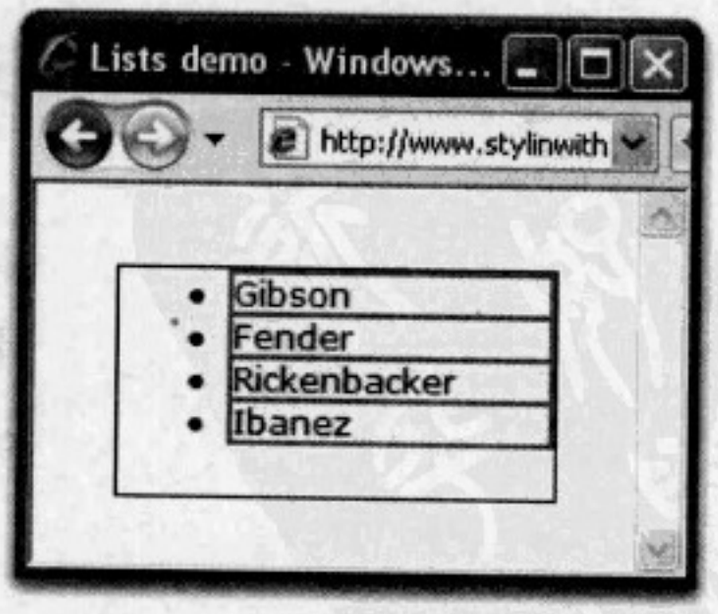


图 6-26 在 IE 7 中显示的列表和列表元素的边框。显然，在 IE 7 和 Firefox 中，默认的内边距和外边距有所不同（另见彩插）

在图 6-25 和图 6-26 中，ul 具有红色边框，而每个 li 具有绿色边框。我们注意到，Firefox（图 6-25）在 ul 上面使用内边距缩进列表（绿色的 li 元素被从红色的 ul 容器边缘向内推开），同时也为 ul 添加了少量的上和下外边距，以便列表与周围的元素保持间距。IE（图 6-26）则在 ul 上使用外边距来缩进整个列表（注意 ul 紧紧地包住了 li 元素，而且它们都同 div 保持了一定的距离）。而且，IE 只为 ul 添加了下外边距，没有添加上外边距。由于存在这些差异，默认情况下的列表很难在不同的浏览器中保持一致的外观。弥合这些差异的唯一方法，就是首先将列表的外边距和内边距都重设为 0，然后再重新为它们添加样式。

使用通用选择符在样式表的开始处将所有元素的外边距和内边距都设置为 0（使用 * {maring:0; padding:0}）也具有相同的效果。如果你已经做了这样的设置，那么就不需要在这里再为列表重复设置了。

因此，我们先将 ul 和 li 这两种列表元素的外边距和内边距设置为 0。

```
ul {border:1px solid red; margin:0; padding:0;}
li {border:1px solid green; margin:0; padding:0;}
```

现在，去掉了外边距和内边距之后的列表在 Firefox 和 IE 中看起来就相同了（图 6-27 和图 6-28）。

图 6-27 在去掉了内外边距的情况下，通过 Firefox 查看列表和列表元素。注意项目符号悬挂在无序列表元素的外部（另见彩插）

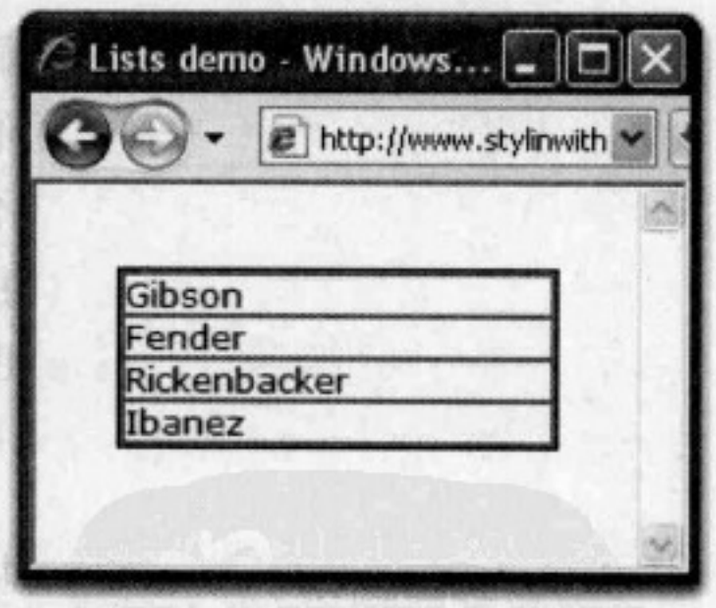


图 6-28 在去掉了内外边距的情况下，通过 IE 7 查看列表和列表项。虽然项目符号也悬挂在无序列表元素的外部，但与 Firefox 不一样，IE 并没有显示列表项（另见彩插）

注意列表项前面的项目符号，它们属于 li 元素，现在都悬挂在 div 的外部。如果 div 的左边正好与浏览器窗口的边缘重合，那么这些项目符号将消失。因此，必须保证为 li 元素应用最小的左外边距，或者为 ul 应用内边距。这样，才能确保项

目符号位于 div 内部，从而不会与页面中的其他元素发生重叠，而且 IE 6 和 IE 7 都能够正确显示它们。下面我们就来设置左外边距（图 6-29 和图 6-30）。

```
div#listcontainer {border:1px solid #000; width:160px;
font-size:.75em; margin:20px;}

ul {border:1px solid red; margin:0 0 0 1.25em; padding:0;}

li {border:1px solid green; margin:0;}
```

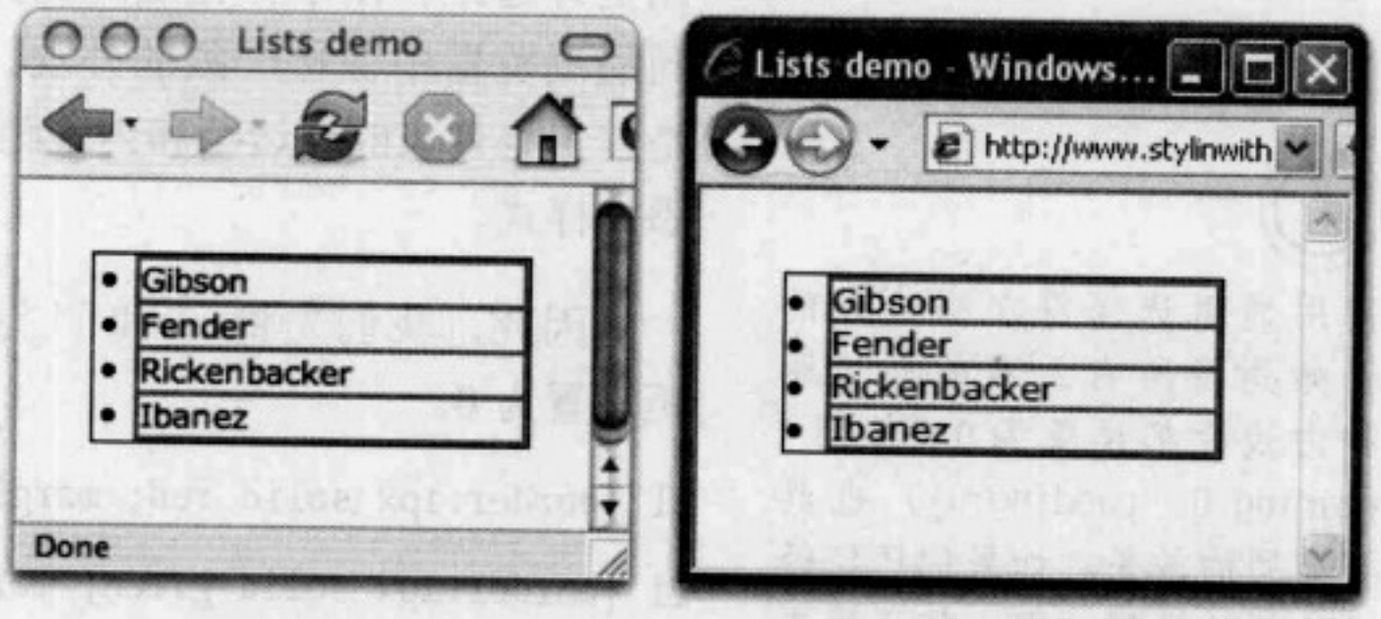


图 6-29 和图 6-30　现在，Firefox 和 IE 中显示的列表外观相同（另见彩插）

注意，我们使用了简写的样式设置了所有的外边距（突出显示的声明），而不是只设置左外边距。如果这里不明确地把其他外边距设置为 0，那么由于不同浏览器中的上和下外边距设置不同，最终还会导致列表外观的差异。并且，这样也相当于为我们准备好了其他 3 个值的占位符，如果将来要修改它们的值也会很方便。

由于列表项的间距过小，下面来为它们之间添加一些空白。

```
div#listcontainer {border:1px solid #000; width:160px;
font-size:.75em; margin:20px;}

ul {border:1px solid red; margin:0 0 0 1.25em; padding:0;}

li {border:1px solid green; border-bottom:2px solid #069;
margin:0; padding:.3em 0;}
```

设置列表项间距的最明显方式就是定义 li 元素的 margin-top 或 margin-bottom 属性，但是，我更倾向于为它们设置相同的上内边距和下内边距。这样以来，可以使 li 元素的边框保持接触而不是在相互之间造成空白，因而也为我们添加其他样式

提供了便利。为了进一步说明增加内边距要好于改变外边距，前面的样式已经把 li 元素周围的边框替换成了每个项目之间轻灵的水平线（图 6-31）。

图 6-31 通过添加垂直内边距，每个列表项的边框实际上都恰好位于文本行之间的中点上，这样当我们为列表项添加下边框时，就能够得到定位精致的分隔线

通过为 li 元素添加上内边距和下内边距来增加它的高度，而不是在它们之间创造空白，li 元素的上边和下边实际上恰好位于文本行之间的中点上。这样，无论是为 li 元素添加上还是下边框，都会得到位于它们中间的一条分隔线。

下面，再对列表进行一些清理（图 6-23）。

(1) 移除项目符号。

(2) 设置 ul 元素的外边距，以便列表在 div 内部的位置恰到好处。

(3) 缩进列表项，使它们不再与水平线前端平齐。

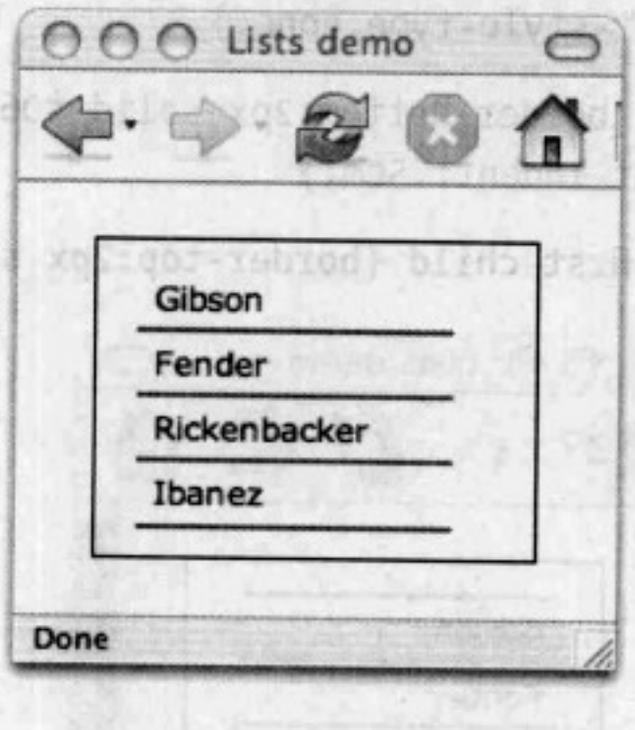


图 6-32 现在，我们撑开了列表元素，移除了项目符号并相对于分隔线缩进了文本

对 CSS 规则的修改如下：

```
body {font:1em verdana, arial, sans-serif;}

div#listcontainer {border:1px solid #000; width:160px;
font-size:.75em; margin:20px;}
```

```
ul {border:0; margin:10px 30px 10px 1.25em; padding:0;
list-style-type:none;}

li {border-bottom:2px solid #069; margin:0; padding:.3em 0;
text-indent:.5em}
```

这一步中进行的最显著的修改，就是将 list-style-type 属性的值设置为 none，从而移除项目符号。text-indent 属性将文本从分隔线的左端轻微向右移动了一定的距离，而为 ul 设置的新外边距使列表在容器 div 中获得了精致的定位。

假如我们为第一个列表项再添加一个上边框，列表看起来一定会更美观。理想的方案应该是只为第一个列表项设置 border-top 属性，而通过 CSS 的 :first-child 伪类实现这个目标非常简单。但是，IDWIMIE——IE 6 不支持这个伪类。

这就意味着，要么通过 :first-child 伪类来添加这条边框，然后以禅宗般的宽容来接受在 IE 中看不到这条边框的事实，要么就拿出一种变通的方法来。下面我们先使用伪类，然后再为我们不够标准但却很流行的朋友（IE 6）考虑一种变通方案。下面是使用伪类的代码：

```
body {font:1em verdana, arial, sans-serif;}

div#listcontainer {border:1px solid #000; width:150px;
font-size:.75em; margin:20px;}

ul {border:0; margin:12px 20px 10px 1.25em; padding:0;
list-style-type:none;}

li {border-bottom:2px solid #069; margin:0; padding:.3em 0;
text-indent:.5em;}

li:first-child {border-top:2px solid #069;}
```

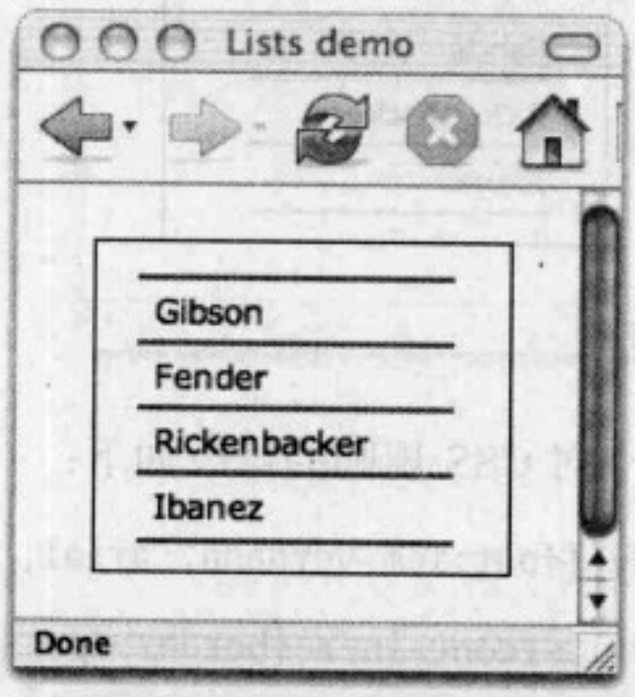


图 6-33 这里我们使用 :first-child 伪类（因为这正是它的用武之地），在第一个列表项和其他列表项之间添加一种必需的差别样式，即为 4 个列表项添加第 5 条分隔线

接下来，我们再看一看如何解决 IE 6[①]不支持 :first-child 伪类的问题。当我们把项目符号移除之后，ul 元素缩小为与列表项同宽。这样，我们可通过为包含所有列表项的 ul 添加上边框来创建最上端的分隔线。因此，用下面的代码来替换前面的代码，也可以在 IE 6 中创建出相同的上分隔线。

```
body {font:1em verdana, arial, sans-serif;}

div#listcontainer {border:1px solid #000; width:150px;
font-size:.75em; margin:20px;}

ul {border:0; margin:12px 20px 10px 1.25em; padding:0;
list-style-type:none; border-top: 2px solid #069}

li {border-bottom:2px solid #069; margin:0; padding:.3em 0;
text-indent:.5em}
```

这些代码生成的结果与图 6-33 相同。有时候，我们可以为 IE 找到像这样比较简单的变通方法，但有时则不得不接受并非所有人都能够获得相同体验的事实。只要所有人都能获得可以接受的体验，那就是算是可以了。

2. 将列表转换为菜单

现在，我们有了一组类似导航的链接。接下来，我们要做的就是将其中的文本行转换为菜单——一组可单击的链接，这样就能得到一个有吸引力的具有导航功能的组件（图 6-34）。通过下列代码实现这个组件非常简单：

```
<div id="listcontainer">
<ul>
<li><a href="gibson.htm">Gibson</a></li>
<li><a href="fender.htm">Fender</a></li>
<li><a href="rickenbacker.htm">Rickenbacker</a></li>
<li><a href="inbanez.htm">Ibanez</a></li>
</ul>
</div>
```

① 原文 IE 7 有误，IE 7 在严格型 DOCTYPE 下支持 :first-child 伪类。——译者注

图 6-34 为列表项中的文本添加锚标签后将它们转换成了可单击的链接

这里需要注意链接标签与文本内容及列表项的嵌套关系。

3. 基本的链接样式

下面，我们为这些链接添加样式。首先，移除“坐在那等待什么事件发生”的常规状态下的下划线，然后当用户鼠标悬停在链接上时，改变链接的颜色。

此外，因为这是完成导航组件的最后一步，所以还要进行一些清理工作——调整 ul 的下外边距并将上下文（包含 div 的 id）添加到选择符中，以便这些样式只会影响位于 listcontainer 中的元素。为了展示如何使用伪类（以便在 IE 能够理解的情况下发挥作用），也为了确保顶端分隔线的显示，我在代码中恢复了使用伪类的规则并添加了星号 hack——请参见提示条“星号 hack 和反斜杠 hack”——以便 IE 6 和 IE 7 中都能出现第一个列表项上方的边框。

如果你的网站需要长期使用，最好能够把针对 IE 6 的 hack 放到单独的样式表中。这样，当 IE 6 消亡时，可以简单地移除该“hack 样式表”。

下面就是完成后的列表导航组件的最终代码，通过图 6-35 可以看到完成后的效果。

```
body {font:1em verdana, arial, sans-serif;}

div#listcontainer {border:1px solid #000; width:150px;
font-size:.75em; margin:20px;}

div#listcontainer ul {border:0; margin:12px 20px 12px 1.25em;
padding:0; list-style-type:none;}

div#listcontainer li {border-bottom:2px solid #069; margin:0;
padding:.3em 0; text-indent:.5em}

div#listcontainer li:first-child {border-top:2px solid #069;}
```

```
div#listcontainer a {text-decoration:none; color:#069;}
div#listcontainer a:hover {color: #00;}
* html div#listcontainer ul {border-top:2px solid #069;}
```

只针对 IE 6 的星号 hack——IE 7 能够理解：first-child 伪类并忽略这个星号 hack

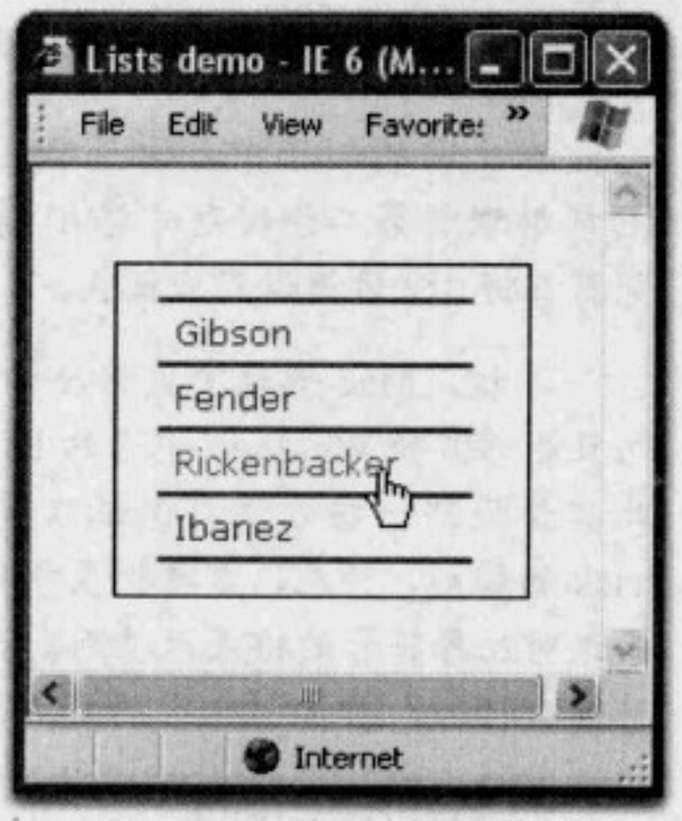


图 6-35 在 IE 6 中显示的当鼠标悬停在完成后的菜单链接上的效果（另见彩插）

星号hack和反斜杠hack

所谓 hack，是指那些没有按照既定的用途使用 CSS 的方式。通过 hack 可以使 CSS 规则针对特定的浏览器起作用，或者在特定的浏览器中隐藏自己。有关 hack 的一个非常常见的用法，就是为 IE 6 提供替代的 CSS 样式规则。

星号（*）hack

我们知道，所有祖先元素的祖先是 html 元素——所有元素都是它的后代。然而，IE 6 是唯一将一个未命名的元素作为 html 父元素的浏览器，因此，通过在选择符中使用这个元素，可以创建只有 IE 6（及更早版本的 IE）能够识别的样式规则。由于这个“超级祖先”元素没有名字，所以要使用通用 CSS 选择符 *（即星号，这个选择符的含义是“任何元素”）来引用它。比如，下面就是使用星号选择符的例子：

```
div * ul {……这里是 CSS 声明……}
```

对于这条规则而言，如果 ul 是 div 的子元素，那么该选择符就不会选择它，但是，如果它是 div 的孙子元素，则会选择它。这里的 * 选择符，实际上是表示任何位于中间的子元素。

使用同一个选择符创建只有 IE 6 能够理解的规则，需要这样来写：

*html……更多具体的选择符…… {……这里是 CSS 声明……}

例如：

```
div#box {border:1px solid blue;}
* html div#box {border:1px solid red;}
```

在上面的例子中，除了 Windows 平台的 IE 6 和 Mac 平台的 IE 能够理解第二条规则并将 div 的边框显示为红色之外，其他浏览器都将 div 边框设置为蓝色。

不过，Mac 平台中更符合标准的 IE 不仅能够识别 * 选择符，而且能够解释 Windows 平台的 IE 6 不能解释的某些 CSS 规则。因此需要找到一些方法以保证只有 Windows 平台的 IE 会应用星号 hack 的规则。为此，需要把星号 hack 的规则放到一对注释符号内，而这对注释符号的格式很特别，因为它利用了 Mac 平台中 IE 的一种独有行为。

反斜杠注释 hack

如果我们在注释中把一个反斜杠放到结束的 * 之前，如下面代码中突出显示的反斜杠所示：

```
/* 这里是一条注释 */
```

那么 Mac 平台中的 IE 则不会认为注释已经结束，因此会忽略下面的所有 CSS 规则，直至遇到常规的注释结束符号为止（如果你喜欢在选择符后面或者同一行中添加注释，在使用这种 hack 时就不要那么做了，因为注释的位置在这里是 hack 起作用的关键所在）。

比如：

```
/* 只针对 Windows 中 IE 6 设置规则的 hack \*/
* html div#listcontainer ul {border-top:2px solid #069;}
/* hack 结束 */
```

在这种情况下，Mac 平台的 IE 会忽略 *html 选择符，虽然它完全能够理解该选择符，但由于它认为第一条注释并没有结束，因此它会忽略第一条注释开始符号到第二条注释结束符号之间的内容——也包括 *html 选择符。由此可见，通过组合使用星号和反斜杠 hack，我们能够定义只针对 Windows 平台中 IE 6 的 CSS 规则。不过，鉴于 Mac 平台中的 IE 现在已经基本没有人使用了，我一般只使用星号 hack，不使用反斜杠 hack。具体用法由你决定。

在从本书网站 www.stylinwithcss.com 下载到的本章示例文件中，包含了几种不同的菜单样式。

下面，我们将列表的应用再提高一个层次，一起来看看如何创建多级菜单。

6.3.2 创建基于 CSS 的菜单

下拉菜单可以在占用非常少的屏幕资源的情况下，为用户提供大量的导航选项，同时也能反映出网站的整体结构。下面展示的菜单（可以在 Stylib 库中找到相应的文件），能够提供指向许多页面的链接，但在用户鼠标悬停在菜单上面之前则只会显示顶级选项。

下面就是一个多级菜单的例子（图 6-36）。

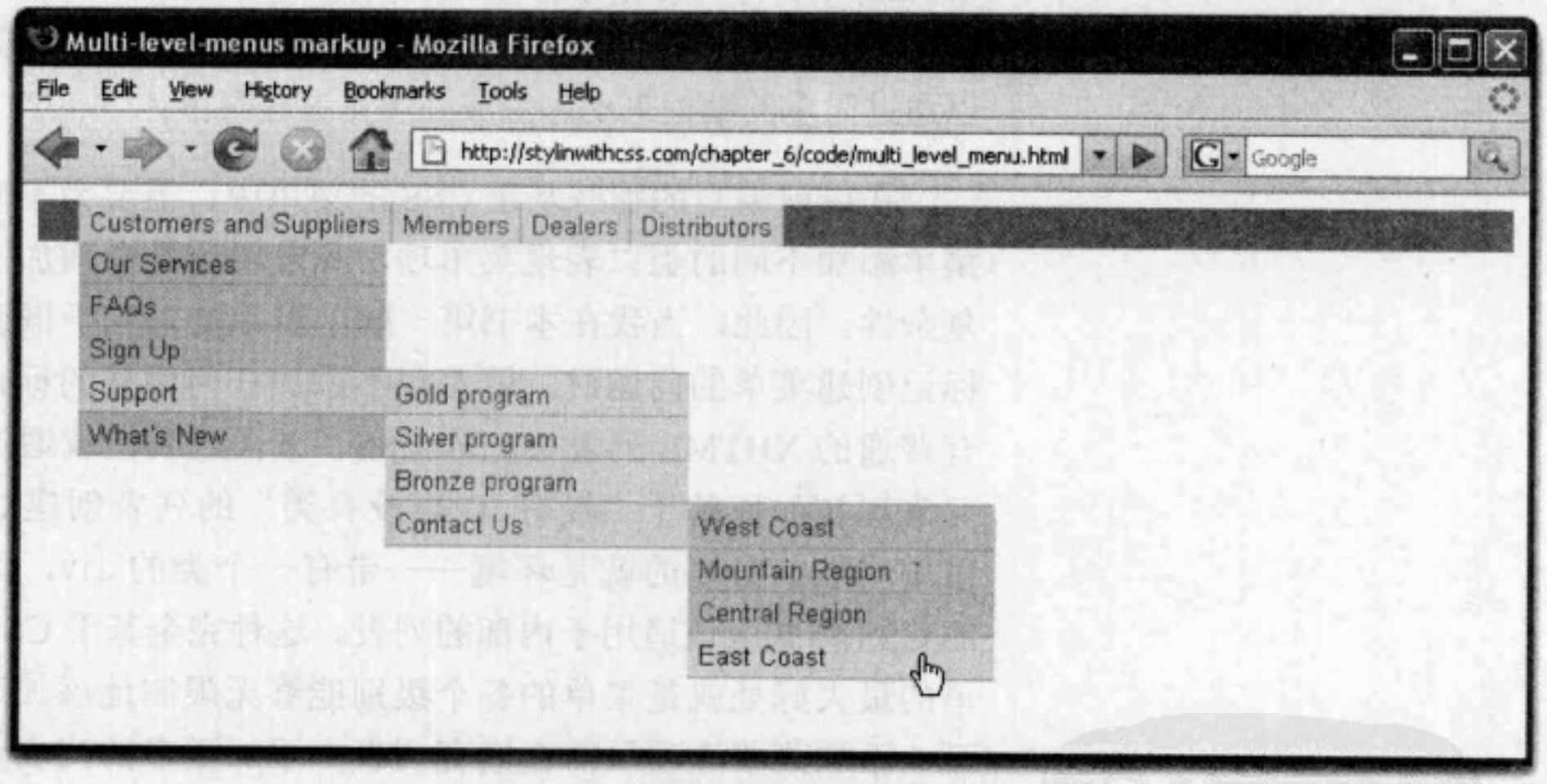


图 6-36 通过 CSS 编写的多级菜单

看过本书第一版的读者，可能觉得图 6-36 中的菜单很熟悉。但是，为了使相应的 CSS 更简洁（代码更少），同时也加强对 2004 年（第一版的写作时间）之后出现的新浏览器的适应能力，我已经完全重写了原先的代码。并且，将其中用户可以修改的 CSS，例如颜色、行高及字体大小从构建菜单的更复杂的代码中分离了出来。这样，你就可以在不影响底层“机制”的前提下，安心地修改菜单的外观。此外，由于已经解决了妨碍菜单项目响应单击的 bug，这些菜单在 IE 6 中也能够正常使用。

如图 6-36 所示，Stylib 库中的这个多级菜单能够支持到 4 个菜单级别。尽管再添加更多的级别也不难，但超过 4 级的菜单会导致用户操作不便。

因此，虽然这些菜单看起来相似（用于示例的 XHTML 标记仍然与本书第一版中相同），但 CSS 已经有了极大的改进。

如果查询我的电子邮件，就会发现这些下拉菜单的 CSS 代码是最受读者欢迎的。事实上，我也是在 2003 年之后厌烦了通过 JavaScript 实现这些菜单的复杂性，才开始编写第一版中的 CSS 菜单的。通过图 6-36 可以看出，这个菜单的深度可以达到 4 级，而且它也很容易在网站中实现。为此，你要做的就是编写一组简单的无序列表（同我们前面看到的导航组件一样），然后，将它们相互嵌套起来构成菜单的不同级别——具体嵌套方法稍后介绍。接着，把嵌套好的列表包装在一个 div 中，再为该 div 添加一个类，以便将 XHTML 标记与 CSS（即 Stylib 库中的 multi_level_menus_class.css）关联起来。仅此而已。如果你想通过网站的数据库来生成菜单选项，那么团队中的编程人员也一定会喜欢 XHTML 标记的简易性（只有嵌套的无序列表），因为他们可以通过简单的编程来动态生成并填充这些菜单。

在以前编写的那些基于 CSS 的菜单中，需要为不同级别的菜单添加不同的类以表现菜单项，我发现这样会增加不必要的复杂性。因此，当我在本书第一版中着手处理基于嵌套的列表标记创建菜单的问题时，就希望不添加任何额外的标记——只有普通的 XHTML 列表标记和 CSS。下面，我们就通过一个练习来理解如何基于“没有 ID 也没有类”的列表创建 CSS 下拉菜单。唯一必要的就是环境——带有一个类的 div，因而恰当的 CSS 样式应该适用于内部的列表。这种完全基于 CSS 创建菜单的最大好处就是菜单的各个级别能够无限制地添加选项。并且，添加菜单选项只需在嵌套列表中添加更多的列表项，不用对 CSS 进行任何改动。

1. 下拉菜单制作教程

与其现在就罗列出为多级菜单编写的复杂而冗长的 CSS，不如先来看一个更简单的例子，通过这个例子我们就能够理解 CSS 菜单背后的原理。对你而言，这是一次从零开始编写 CSS 的有趣的练习。这个练习虽然不太复杂，但也综合了到目前为止我们学过的所有 CSS 技术。因此，如果你能够顺利地编写完这个菜单中的 CSS，那么将会对解决各种 CSS 开发问题更有信心。

在这个练习中，我们创建一个两级菜单：一行水平的菜单选项和这些选项处于悬停状态时的弹出菜单。下面，我们从顶级菜单的标记开始。

```
<div id="multi_drop_menus">
  <ul>
    <li><a href="#">Item 1</a></li>
    <li><a href="#">Item 2</a></li>
  </ul>
</div>
```

我们注意到，这些标记与前面导航组件的标记结构相同，即一个 div 包含着带有两个列表项的无序列表，每个列表项中包含一个链接。要恰当地应用菜单样式，关键是要让这些元素像俄罗斯的小洋娃娃[①]一样在内部彼此正确地对齐。下面，我们尝试为标记中的 4 种元素类型（div、ul、li 和 a）应用不同的颜色或者背景，这样我们就能够在设计过程中看清它们的相互关系。

所有元素

```
#multi_drop_menus * {
```

移除菜单元素中默认的外边距和内边距

```
  margin:0; padding:0;
  }
```

包含 div

```
#multi_drop_menus {
  border:3px solid green;
  }
#multi_drop_menus ul  {
```

显示容器

```
  border:2px solid red;
  }
#multi_drop_menus li  {
```

显示容器

```
  border:2px solid blue;
```

移除每个列表项前的项目符号

```
  list-style-type:none;
  }
a {
  background-color:#DDD;
  }
```

① 即一种俄罗斯套装洋娃娃玩具，一般是 5 个并排的依次变小的洋娃娃。——译者注

这里，我们首先使用通用选择符 * 移除了所有元素默认的外边距和内边距。然后，为 div、ul 和 li 分别添加了绿色、红色和蓝色的边框。最后，为最里面的元素——链接（a），添加了浅灰色的背景，这也是目前菜单选项的背景颜色。

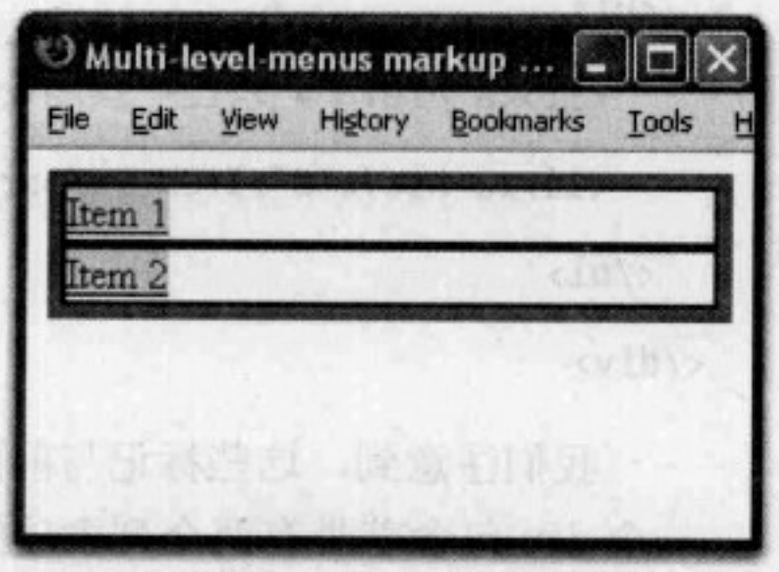


图 6-37　构成菜单的 4 种元素显示在了默认位置上（另见彩插）

如图 6-37 所示（通过我们添加的边框和背景），块级元素填满了浏览器窗口的宽度，而链接作为行内元素，其内容是紧缩的。列表项（li）相互堆叠在一起也很正常，因为它们是块级元素。当然，为了创建水平菜单，需要它们并肩排列，因此需要浮动它们，下面就来设置相应的样式（图 6-38）。我们将针对 li 元素的规则修改如下：

```
#multi_drop_menus li  {
border:2px solid blue;
  list-style-type:none;
  float:left;
  }
```

让列表项并肩排列

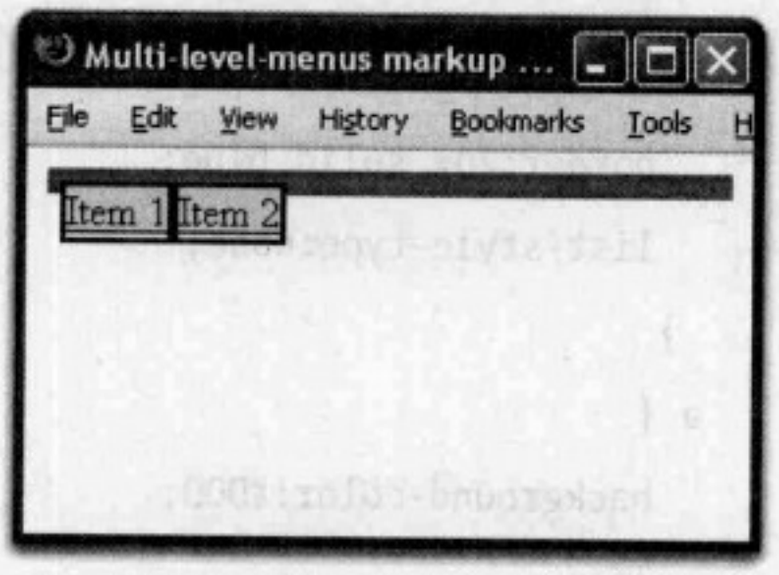


图 6-38　在浮动了列表项后，div 和 ul 紧缩起来（另见彩插）

哎，这些盒子怎么了？是这样，当我们浮动列表项时，ul（连同它的父元素 div）中没有了未浮动的内容，因此它们会紧缩起来，从而导致列表项悬垂到下方。实际上，我们希望这两个元素能够包裹住列表项，那么应该怎么做呢？如果你说“也浮动它们！”，那么，你显然是认真理解了第 4 章中浮动和清除的内容。因此，解决这个问题的简单方法如下所示（图 6-39）。

```
#multi_drop_menus {
  border:3px solid green
  float:left;
  }
#multi_drop_menus ul  {
  border:2px solid red;
  float:left;
  }
```

让 div 包裹住 ul

让 ul 包裹住 li

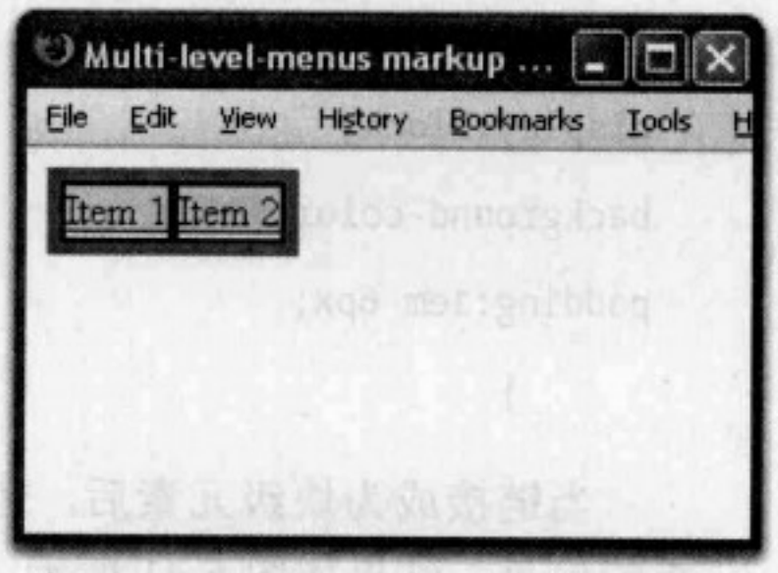


图 6-39 同时浮动 div 和 ul，会再次包裹住浮动的列表项（另见彩插）

这样，我们就取得了一些进展。浮动的元素紧紧地包裹住了 li 元素，而水平菜单也开始成形。接下来，我们通过在链接文本周围添加一些内边距来在文本周围创建一些空间（它们实在太挤了，而且已经接触到了包含元素的边界）。这里，为了示范正确创建菜单的关键所在，我们暂时为垂直的内边距设置一个超过正常需要的值（1 em）。

```
#multi_drop_menus a {
  background-color:#DDD;
  padding:1em 6px;
  }
```

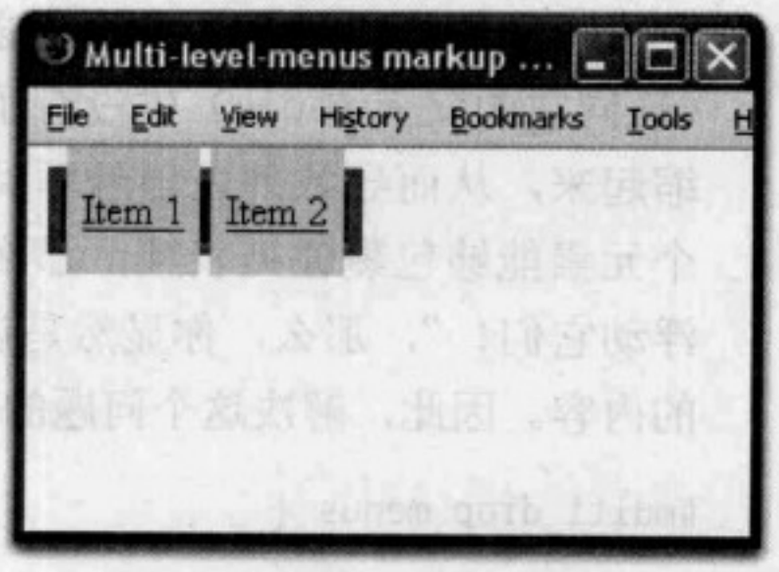


图 6-40 链接的灰色背景明显增大了，而且也在垂直方向上伸出并覆盖了包含元素的边框（另见彩插）

我们看到图 6-40 中似乎存在什么问题。链接的背景虽然变大了，但背景却垂直扩展并覆盖了包含元素，并没有向外推开并改变包含元素的大小。实际上，在为行内元素应用内边距和边框时，它们不会像我们期望那样影响包含元素。行内元素的垂直内边距不会像块级元素那样，沿着文档层次向上传递到包含元素。因而包含元素也不会调整自身大小来容纳被链接占用的更大区域。简单的补救方法就是把链接也转换为块级元素。

使链接的包含元素正确地调整自身大小

```
#multi_drop_menus a {
    display:block;
    background-color:#DDD;
    padding:1em 6px;
}
```

当链接成为块级元素后，包含元素就接收到了调整自身大小的信息，结果如图 6-41 所示。下一步是使链接在悬停时产生某种视觉上的反应（同时，由于我们已经展示了创建菜单的重点，也将把链接的垂直内边距设置为更合理的大小）。

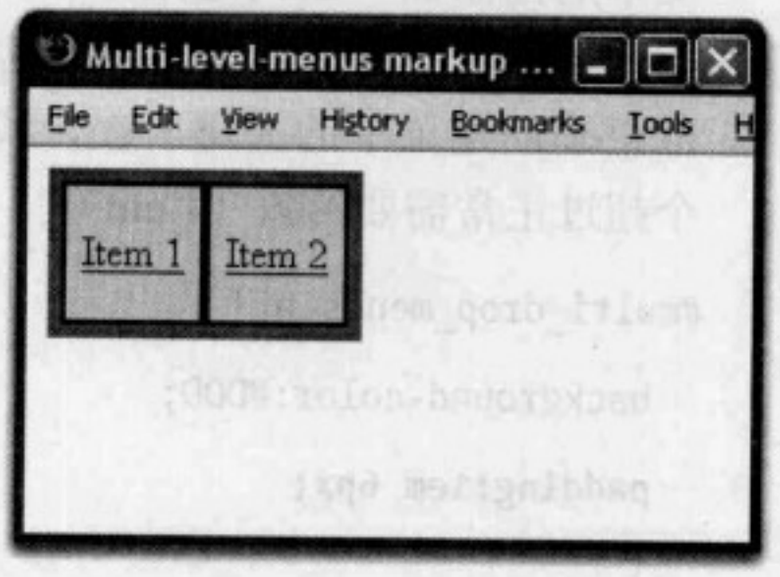


图 6-41 随着把链接转换为块级元素，包含元素调整了自身大小以容纳这些链接（另见彩插）

```
#multi_drop_menus a {
  display:block;
  background-color:#DDD;
  padding:.3em 6px;
        }
#multi_drop_menus a:hover {
  color:#CCC;
  background-color:#666;
}
```

为链接应用真实的垂直内边距

悬停状态下的文本颜色

悬停状态下的背景颜色

在悬停行为发挥作用后（如图 6-42 所示），为链接再应用一些边框和颜色样式会使水平菜单更美观。但是，为了在下一步（添加下拉菜单时）看得更清楚，我们在这里先不修改边框。在下一步之后，我们再通过美化菜单使其更具吸引力。

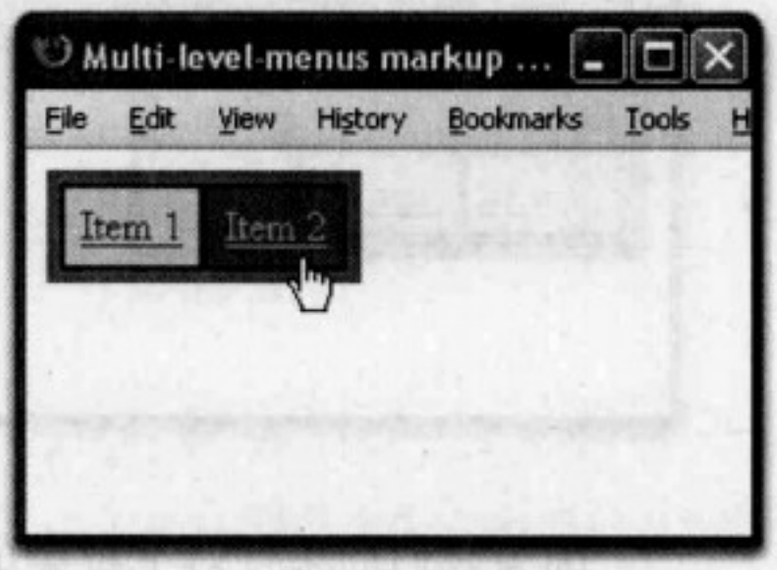


图 6-42 当链接处于悬停状态时，链接的背景和字体颜色都会发生改变（另见彩插）

2. 在菜单中创建下拉选项

菜单的各个级别是通过在列表内部嵌套列表创建的。下面，我们仔细地观察一下嵌套的标记：

```
<div id="multi_drop_menus">
  <ul>
    <li><a href="#">Item 1</a>
      <ul>
        <li><a href="#">Item 1a</a></li>
```

```
        <li><a href="#">Item 1b</a></li>
      </ul>
    </li>
    <li><a href="#">Item 2</a></li>
    </ul>
  </div>
```

在这个列表项（突出显示）中，我们添加了一个嵌套的无序列表，也就是下拉菜单。注意这个嵌套列表的位置——它恰好位于父元素（列表项）结束标签的之前。请记住，顶级列表项的结束标记要跟在全部二级列表的后面。如果没有正确地组织这种嵌套关系，那么将导致不会出现下拉菜单。下面，我们看一看这种嵌套布局的外观。

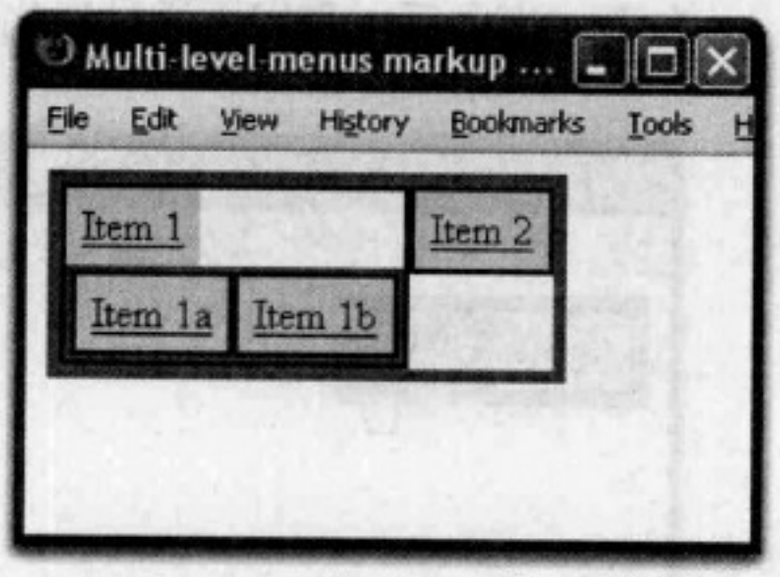


图 6-43 此时，下拉菜单样式全部继承自水平的顶级菜单，因此作为下拉菜单的列表项也会水平排列（另见彩插）

因为我们尚未针对下拉菜单编写样式，所以它们只是从顶级菜单继承了样式并水平排列，结果如图 6-43 所示。当然，为了让它们成为下拉菜单，必须将它们垂直堆叠。而实现这一点的方式，就是在我们已经多次看到的两级菜单之间设置绝对 / 相对的定位关系。

```
#multi_drop_menus li  {
    border:2px solid blue;
    list-style-type:none;

    float:left;
    position:relative;
    }
```

显示容器

移除每个列表项前的项目符号

让列表项并肩排列

为嵌套的 ul（下拉菜单选项）进行定位

```
#multi_drop_menus li ul {
  position:absolute;
  width:7em;
  }
#multi_drop_menus li ul li{
  width:100%;
  }
```

如果你试验一下通过取消 li 元素的浮动来达到堆叠它们的目的，那么将发现在 IE 6 和 IE 7 中菜单项之间会显示空隙。这也是我们让 li 元素填充 ul 元素的原因，而且事实上这也是一种更可靠的解决方案。采取这种方案，至少可以节省 3 小时的时间。

在将下拉菜单的定位环境设置为外部的列表之后，如图 6-44 所示，下拉菜单会令人满意地将自己定位在适当的位置上——恰好位于顶级菜单项下方我们想要它出现的位置。这里，我们将下拉菜单的 ul 元素的宽度设置为 7 em（你可以根据自己的需要设置这个宽度值），并为它的子元素 li 设置了 width:100%，以便强迫每个 li 元素填满 ul 的宽度。通过只允许在 ul 的宽度上放置一个 li 元素，最终使 li 元素相互堆叠起来，这也是我们想要的结果。为此，甚至不需要取消对它们的浮动。

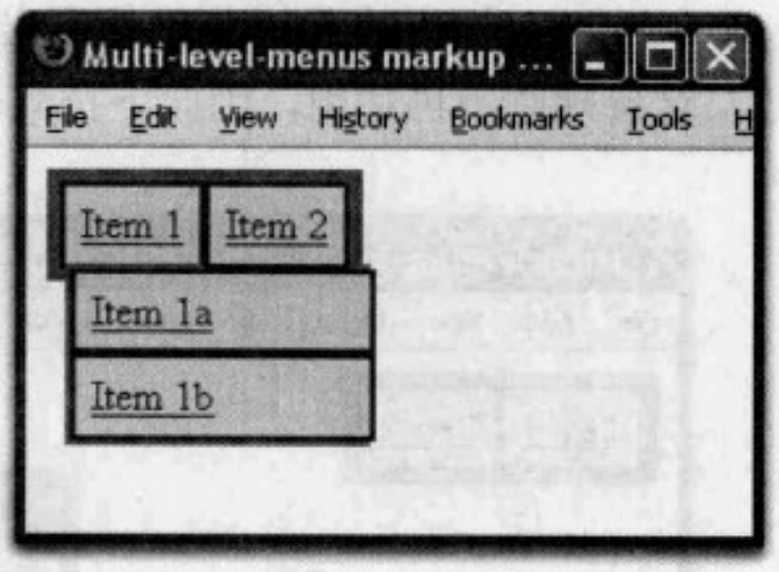


图 6-44 在为列表和子列表之间创建了定位关系之后，我们又强迫 li 元素填满 ul 元素，使子列表项堆叠起来从而创建了下拉菜单（另见彩插）

接下来，我们就要接触到有意思的地方了——让菜单活动起来。换句话说，当鼠标悬停在相关的顶级菜单项上时，再显示下拉菜单。

```
#multi_drop_menus {
behavior:url(../../lib/js_tools/csshover.htc);
font-family:lucida, arial, sans-serif;
border:1px solid #686;
float:left;
}
```

链接到这个文件可以实现在 IE 6 中显示下拉菜单

```
#multi_drop_menus li ul {
  position:absolute;
  width:7em;
  display:none;
  }
#multi_drop_menus li:hover ul {
  display:block;
  }
```

隐藏下拉菜单（当悬停时再显示——见下面）

当父元素（li）处于悬停状态时，显示下拉菜单

第一步是为构成下拉菜单的 ul 元素设置 display:none，从而隐藏下拉菜单。之后添加的规则中包含下面这个有意思的小选择符：

```
#multi_drop_menus li:hover ul {display:block;}
```

突出显示的部分意思是当父元素 li 处于悬停状态时为 ul 元素应用这一规则。看！只要菜单的顶级选项处于悬停状态（图 6-45B），相应的下拉菜单就会出现。当菜单未处于悬停状态时，这条规则不再适用，因此下拉菜单还会隐藏起来（图 6-45A）。

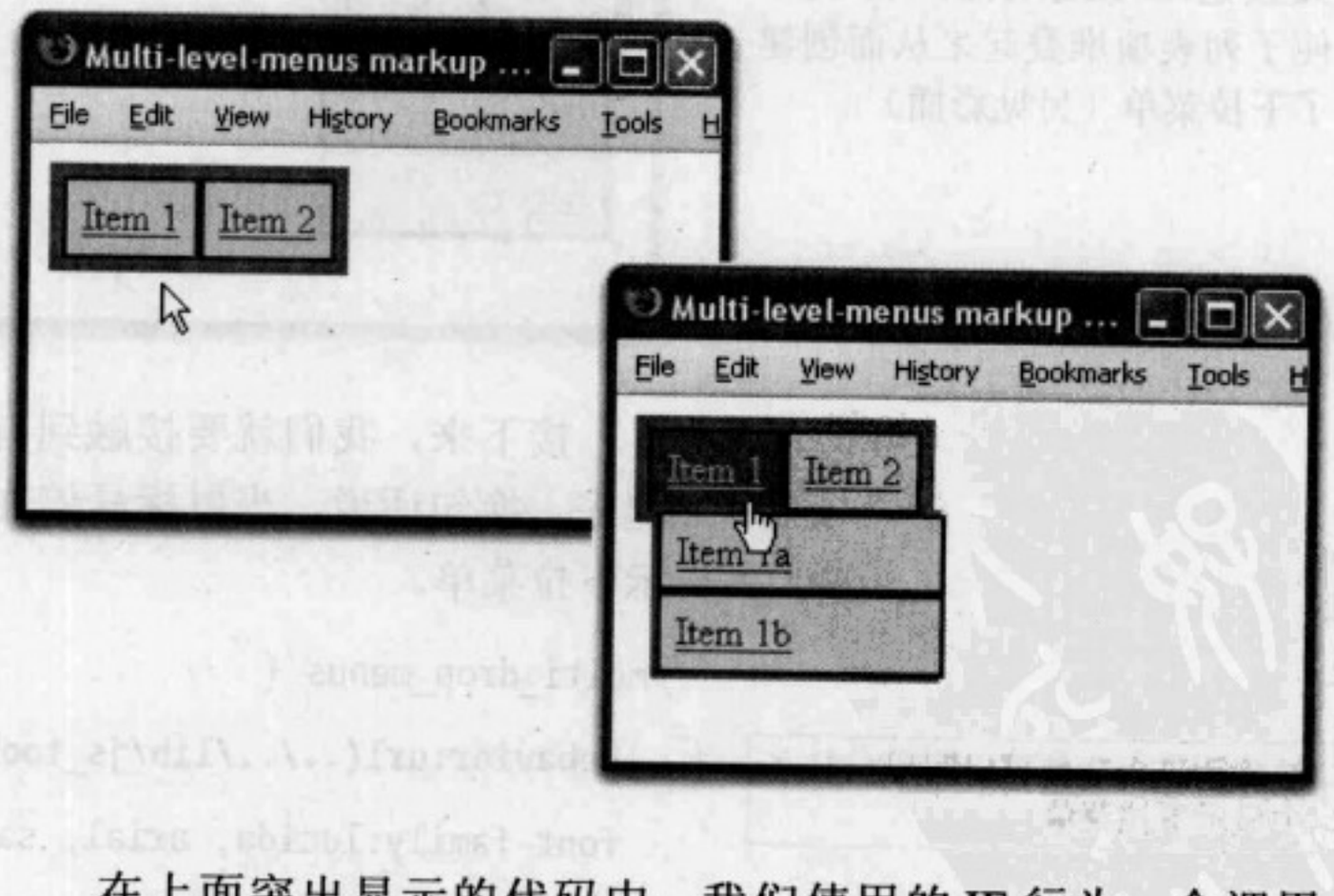


图 6-45A 和图 6-45B　通过两条简单的规则隐藏并在相关的菜单元素悬停时显示下拉菜单，添加的 csshover.htc 文件可以使 IE 6 在 li 元素上面响应悬停事件（另见彩插）

在上面突出显示的代码中，我们使用的 IE 行为，会调用名为 csshover.htc 的 JavaScript 脚本，这个脚本会导致 IE 6 能够在包括链接元素在内的所有元素上响应悬停事件，而这正是 CSS 菜单运行的关键所在。别忘了这个文件的路径是相对于页

面而不是相对 CSS 文件的。如果 CSS 处于一个单独的文件中，并且该文件与 XHTML 页面不在同一位置，那么就需要格外注意这个问题。要了解更多的内容，请参见第 4 章的提示条“IE 6 中的悬停行为”。

现在，剩下的工作就是清理那些对本节的练习提供了很大帮助的彩色边框，然后代之以更符合页面风格也更具有吸引力的样式。另外，我们也修改了一下标记（没有截图，但结果包含在了下载的文件中），不仅添加了第 3 个顶级菜单选项，而且也为每个选项添加了相应的下拉菜单。以下就是完成后的菜单的 CSS 规则（其中突出显示了装饰性的样式），最终的结果如图 6-46 所示。

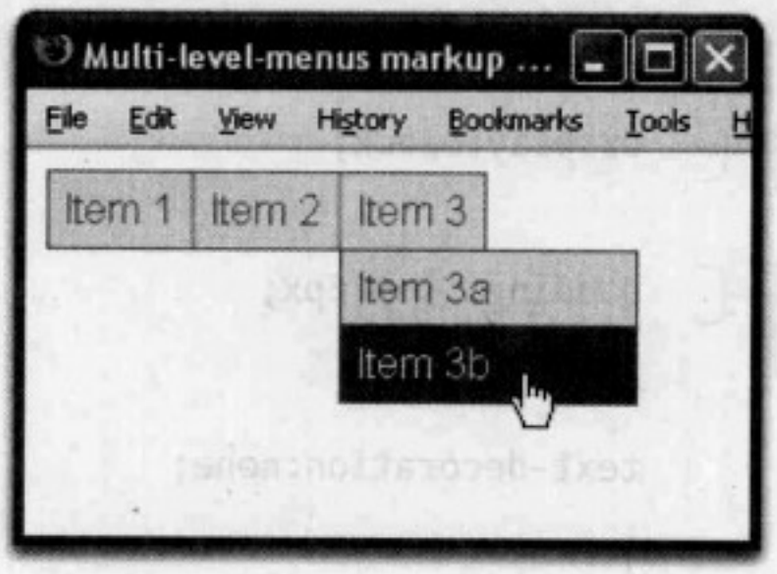


图 6-46 完成后的菜单，已经为你的 Web 设计生涯提升作好了准备（另见彩插）

```
#multi_drop_menus * {
  margin:0; padding:0;
  }
#multi_drop_menus {
  behavior:url(../../lib/js_tools/csshover.htc);
  font-family:lucida, arial, sans-serif;
  border:1px solid #686;
  float:left;
  }
#multi_drop_menus ul  {
  border:2px solid red;
  float:left;
  }
#multi_drop_menus li  {
  border:2px solid blue;
```

移除菜单元素中默认的外边距和内边距

让 div 包裹住 ul

移除参照用的边框

让 ul 包裹住 li

显示容器

```
列表项间的分隔符                    border-left:2px solid #ACA;
移除每个列表项前的项目符号          list-style-type:none;
让列表项并肩排列                    float:left;
为嵌套的 ul（下拉菜单选项）进行定位  position:relative;
                                    background-color:#DED;
                                    }
                                #multi_drop_menus li:first-child  {
第一个列表项左侧不需要分隔符        border-left:none;
                                    }
                                #multi_drop_menus a {
让链接正确地填充列表项              display:block;
将链接文本从列表项边界向内推开      padding:.3em 6px;
                                    color:#686;
                                    text-decoration:none;
                                    }
                                #multi_drop_menus a:hover {
悬停状态下的文本颜色                color:#DED;
悬停状态下的背景颜色                background-color:#464;
                                    }
                                #multi_drop_menus li ul {
相对于父元素（li）的定位环境定位下拉菜单  position:absolute;
隐藏下拉菜单（当悬停时再显示——见下面）    display:none;
设置下拉菜单的宽度                  width:7em;
精确定位下拉菜单                    left:-1px;
                                    }
                                #multi_drop_menus li:hover ul {
当悬停时显示菜单                    display:block;
                                    }
```

```
#multi_drop_menus li ul li {
  width:100%;
  border-right:1px solid #686;
  border-bottom:1px solid #686;
  border-left:1px solid #686;
  }
#multi_drop_menus li ul li:first-child {
  border-left:1px solid #686;
  border-top:1px solid #686;

  }
* html #multi_drop_menus li ul {
  border-top:1px solid #686;
  }
```

让每个 li 元素都与 ul 同宽，从而实现堆叠

在下拉菜单四周添加边框

覆盖移除的顶级菜单左边框

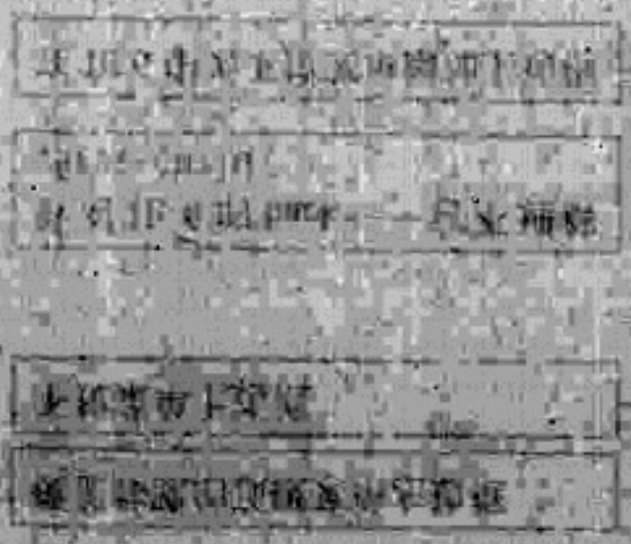
下拉菜单上边框

针对 IE 6 的 hack——它不理解 :first-child

在 IE 6 中为下拉菜单添加上边框

在本书测试的所有浏览器中（包括 IE 6），这个示例菜单都能够正常运行。不过，我仍然推荐使用 Stylib 中的文件 multi_level_menu_class.css，该文件的代码在真实的网站中有更广泛的应用，不仅支持多级菜单，而且，通过为包含元素添加第二个类，还能够使顶级菜单选项堆叠起来，从而方便地将它用作侧边栏中的导航。尽管实现这些菜单的代码比较复杂，但当编写完成后（对你来说是这样），则能够相当容易地在动态网站中实现下拉菜单功能。此时，后台系统只需输出 XHTML 列表的结构，并且同 JavaScript 的典型应用一样，不必再为菜单的每个元素自定义代码。

至此，本书中最长的一章就要结束了。但愿本章中展示的例子能够对你有所启发，以便你在此基础上创建出自己的界面组件。或者，至少能够在本章介绍的组件基础上进行修改，使它们为你所用。在接下来的最后一章中，我们将综合运用迄今为止学习的布局和组件技术，创建完整的网页。



第 7 章

构建网页

在本书最后一章中，我们要把前几章所学的知识综合起来，探索一下如何将页面组织成完整的网站。本章将介绍如何设置文件夹结构以组织 XHTML、CSS 及 JavaScript 文件，并示范几种将 CSS 文件链接到网页中的方式。其中还会涉及如何向网页中添加不同类型的界面组件，以及组织相关的 CSS。此外，还将讨论优化必要代码的技术，以便简化为标记中每个元素应用 CSS 样式的代码编写、查找和维护工作。

为了讨论上述内容，我们要为本书创建一个网站。这个网站的主页中将包含本书的综述及对网站内容的简介，网站的其他页面中会包含本书的目录、相关的 CSS 链接以及一个注册表单。

本章的焦点是网站的主页。我们要通过 Stylib 库中的一个页面模板来构建主页，然后再方便地添加其他 Stylib 库中的界面组件，例如我们在前几章看到的表单布局样式、下拉菜单以及文本和颜色样式等，最终以最少的自定义代码来完成相关的网页。除此之外，我们还会介绍一些在页面之间共享样式的方法以及辅助从视觉上分隔页面布局的背景图像技术。

要创建本示例中的页面或者其他网页不一定要使用 Stylib 库。既可以复制本书中的样式代码，也可以自己动手编写。但是，使用 Stylib 库中的代码要好于复制本书的代码，因为书中代码存在的问题在库中已经得到了修复。

7.1 本书网站简介

下面是完成后的主页在最大和最小宽度情况下的屏幕截图（图 7-1 和图 7-2）。

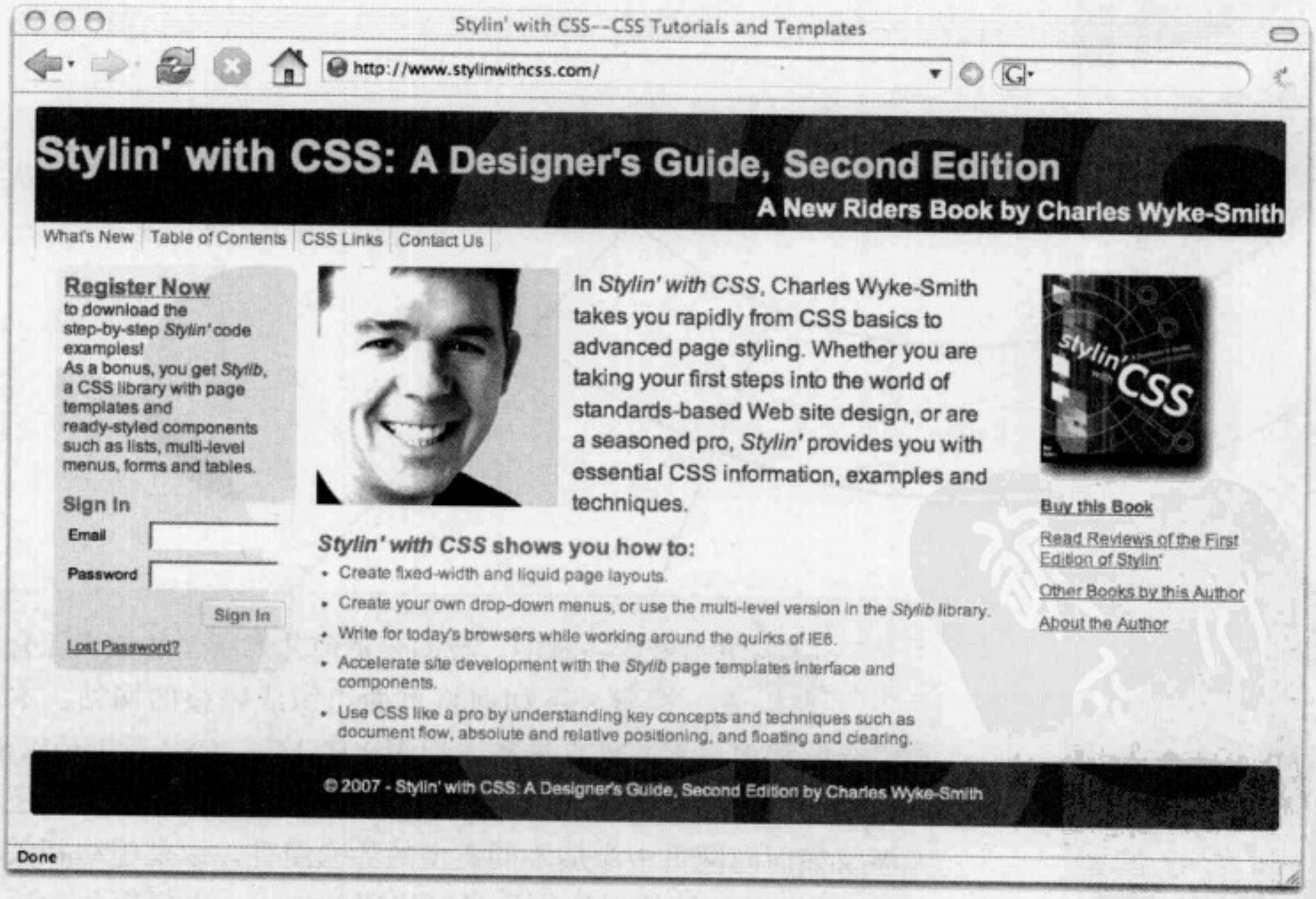


图 7-1　本书主页在最大宽度时的效果（另见彩插）

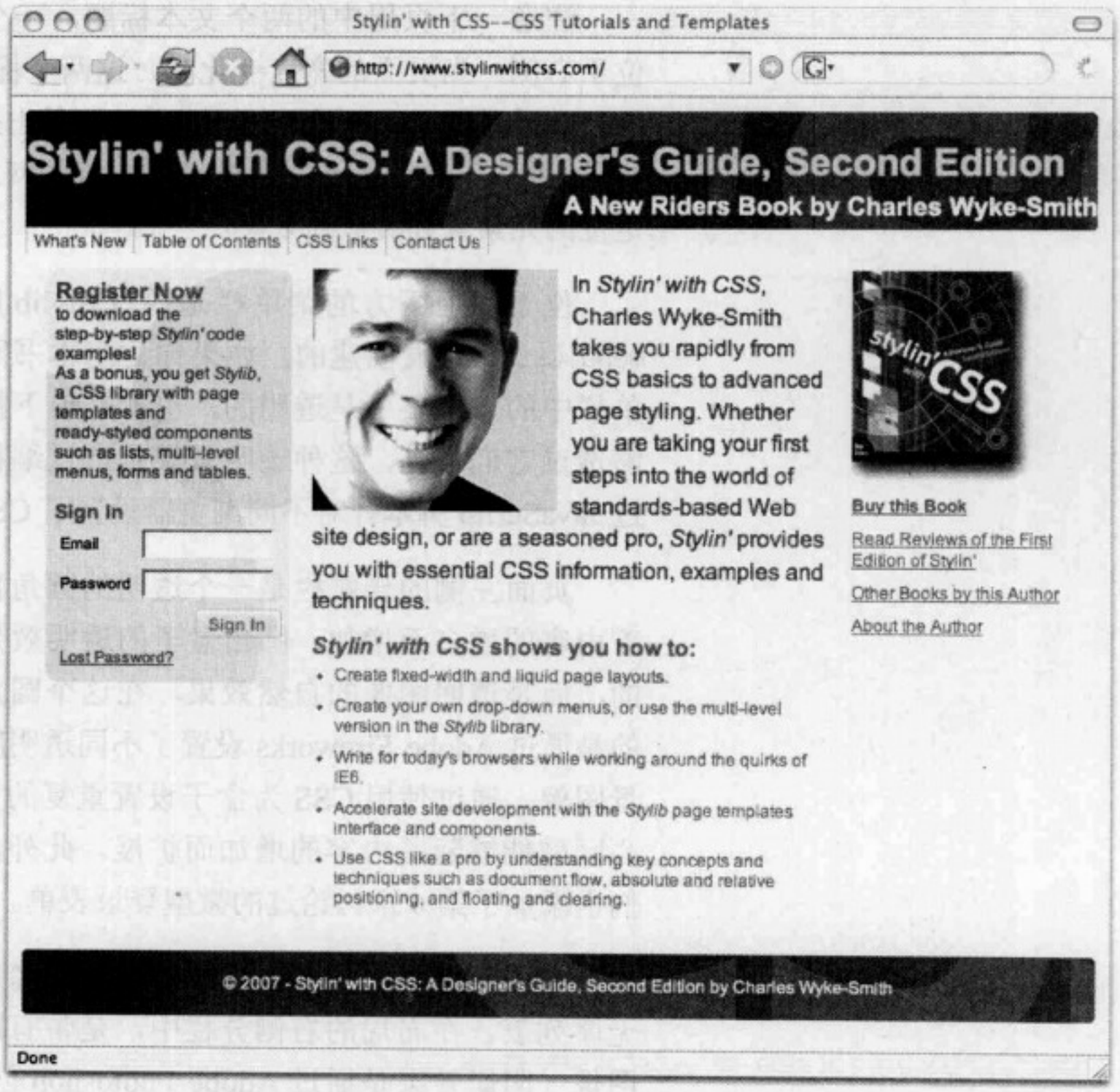


图 7-2 本书主页在最小宽度时的效果（另见彩插）

这个页面是在第 5 章介绍的三栏流动布局基础上构建的。在讨论如何构建这些分栏内容之前，我们先来看一看该页面中包含的关键功能。首先，我们注意到该页面中包含 4 幅背景图像。其次，页眉和页脚都应用了带大字母的暗色调背景图像。最后，通过使用 NiftyCorners 代码（参见第 5 章）为页眉和页脚创建了圆角。

此外，页面左侧的圆形图像位于 id 为 two_colum 的 div 中，该 div 包含了左侧和中间的分栏，因此圆形图像能够横跨两个分栏。而基于本书封面制作的大幅图像则位于 id 为 main_wrapper 的 div 中，该 div 包含了整个布局。这里，我们已经把大幅封面背景图像移动到页面右侧，远离了导航分栏，使其出现在中间和右侧分栏的后面。上述位于不同 div 中横跨分栏的两幅图像，从视觉上将侧边栏和中间分栏的内容联结为一个整体。

注意一下页眉中的两个文本标题，一个位于左侧，另一个位于右侧。当页面的宽度变化时，这两个标题的位置也会改变，这一点通过图 7-1 和图 7-2 能够明显地看出来。在第 5 章介绍的 Amazon 和 Jing 这两个采用流动式布局的网站中，也能够通过右定位的元素看到类似的效果。

位于页眉下方的菜单栏是通过 Stylib 库中的 multi_level_menu.css 样式表创建的。如果你访问本书网站，会发现这个菜单栏中的下拉菜单是透明的，也就是说下拉菜单下方的页面能够透过它们可见。这种透明效果是通过编程方式实现的，即通过 JavaScritp 脚本针对不同浏览器设置了 CSS 中的不透明度。

页面左侧的导航栏是一个透明的圆角盒子，它的高度会随着内容的增多而增加。圆角盒子的透明效果不是通过代码实现的，而是透明图像的自然效果。在这个圆角盒子中，我们使用的是通过 Adobe Fireworks 设置了不同透明度级别的 3 幅 .png 背景图像。通过使用 CSS 为盒子设置重复的背景图像，盒子的中心区域能够随着内容的增加而扩展。此外，在这个区域中，我们也添加了第 6 章讨论过的微型登录表单。

在内容分栏中，文本绕排在浮动图像周围，它们的下方是无序列表。在布局的右侧分栏中，是带有阴影效果的本书封面图像（阴影效果是通过 Adobe Photoshop 创建的），封面图像下方则是一个简单的无序链接列表。

7.2 设置文件夹结构

在开始构建网站时，第一步就是在计算机中设置文件夹——通常被称为本地文件夹。这个文件夹基本上包含完成后的需要上传到 Web 服务器根目录中的网站副本（参见提示条“根目录”）。当准备向 Web 服务器上传网站时，需要使用 FTP 客户端将本地文件夹的内容而不是文件夹本身，转移到 Web 服务器的根目录中。

如果你使用 Adobe Dreamweaver，那么这个文件夹就是你在设置网站的 FTP 信息时，选择作为“本地根文件夹”的文件夹。

根目录

根目录就是网站的根 URL 所指向的目录。URL 是因特网中每个文档的唯一地址。

每个域名，例如 stylinwithcss.com，都会通过 DNS（Domain Name System，域名系统）与一个 IP 地址（Internet Protocol address，因特网通信协议地址，即因特网中每台服务器的唯一数字名称）关联。当我们在浏览器中输入 URL 时，DNS 会查寻域名并找到关联服务器的 IP 地址，然后将请求转发到该服务器。

当服务器接收到对页面的请求后，会根据与请求关联的域名将请求发送到服务器上与该域相关的一个特定文件夹——根文件夹。如果 URL 中包含直接指向一个页面的路径，服务器就会将该页面提供给请求的 Web 浏览器。如果请求的 URL 只是一个简单的 www.stylinwithcss.com，没有指定具体的文件名，服务器会搜索根文件夹，如果其中包含名为 default.html（或 .htm）、home.html 或 index.html，则会自动提供该页面。

总而言之，根文件夹就是网站层次中的顶级文件夹，取决于 ISP（Internet Service Provider，因特网服务提供商）或网络管理员的设置，这个文件夹会与网站的网址关联。要了解有关 DNS 系统的更多信息，请访问 InterNIC 网站（www.internic.net/faqs/authoritative-dns.html）。

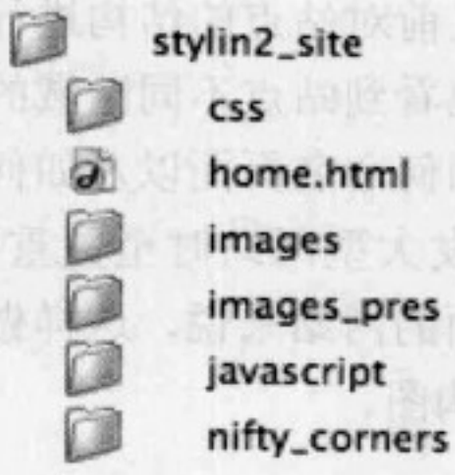


图 7-3 在 Adobe Dreamweaver 中为本书网站设置的本地文件夹

如果你没有使用 Dreamweaver 或其他编辑器内置的 FTP（File Transfer Protocol，文件传输协议）客户端，就需要一个独立的 FTP 客户端。无论使用何种 FTP 客户端，为了成功登录，都必须输入主机名、用户名和密码。然后，才能把计算机本地文件夹中的文件和文件夹上传到网站服务器的根文件夹。可以从 ISP（或者，如果你是在公司里搭建网站，应该是你们的网络管理员）那里获得 FTP 登录信息。

在根文件夹中（我们例子中的根文件夹是图 7-3 中的 stylin2_site），唯一绝对需要的页面是主页。在我们的例子中，主页是将 3_col_liquid_faux.html 文件重命名为 home.html 后得到的，这样它就会被 Web 浏览器自动加载。关于自动加载主页的详细信息，请参考提示条“根目录”中的内容。虽然把所有 XHTML 文件都放在网站的顶级文件夹中也没有问题，但把它们组织到子文件夹中则是一个好习惯。

关于为公司构建网站的说明

大型网站的开发一般都是由商业性的需求文档来驱动，该文档描述了用户对站点的需求以及为满足这些需要必须提供的功能。为实现满足需求的网站，下一步就是创建一份内容清单，用于描述网站中应该包含的内容，以及对网站功能的定义（包括需要编程实现的特性及工作原理）。然后，就是站点结构设计，也就是真正的设计工作的起点。

在创建文件夹结构时，我一般会在根文件夹中首先创建4个文件夹，分别命名为：css、javascript、images和images_pres。其中，前两个文件夹分别用于保存CSS和JavaScript文件。images文件夹保存所有与内容相关的图像——对于这个页面而言，就是本书的照片和图像。images_pres文件夹保存所有表现性的图像（例如背景），这些图像应该归类于视觉设计范畴而不是内容范畴。

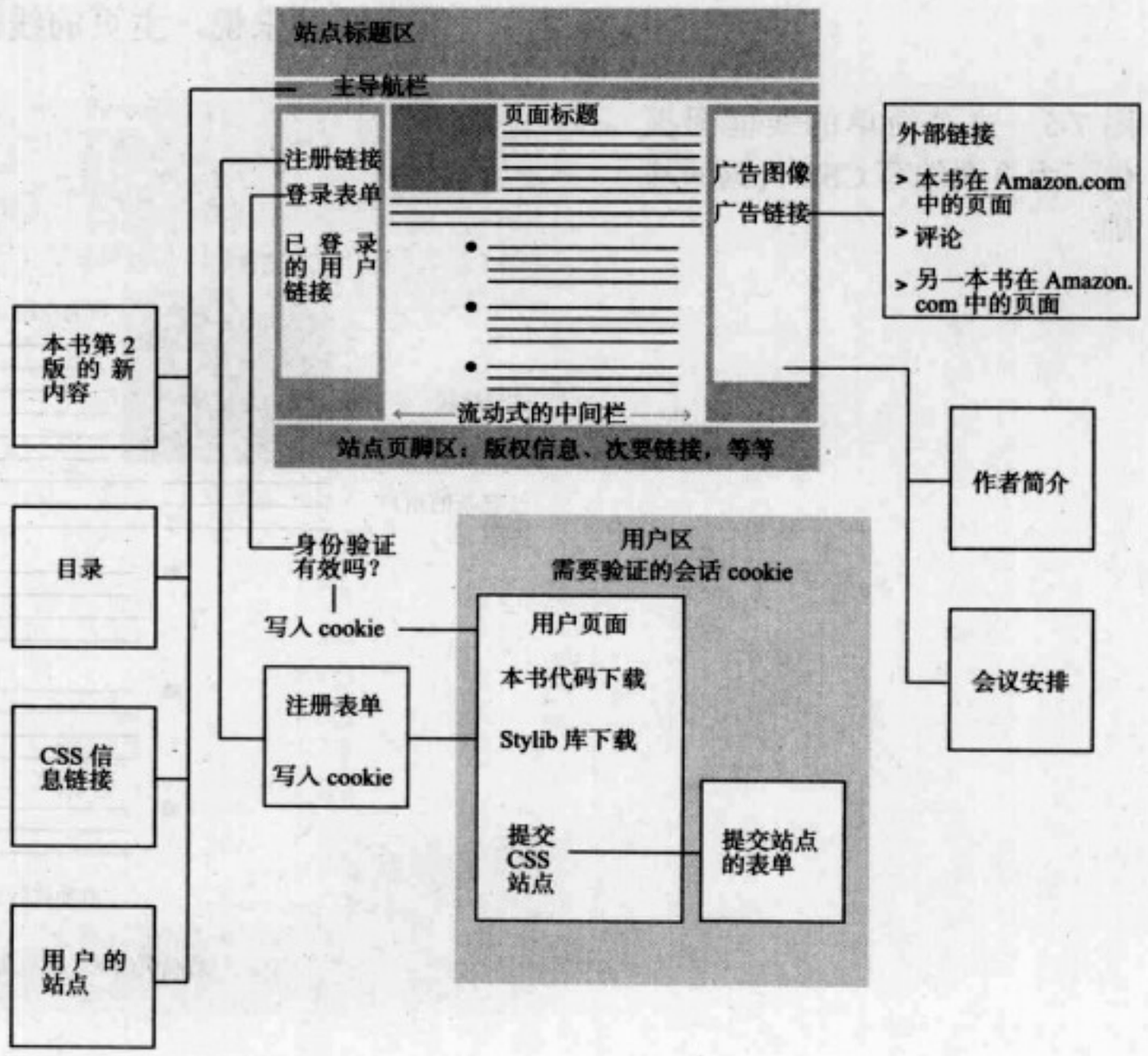
在两个不同的文件夹中保存图像，可以使更新网站设计更方便。因为只需更新images_pres文件夹中的图像，而无需改动内容相关的images文件夹中的图像。另外，当需要聚合网站的内容时，你也只需提供XHTML标记和images文件夹中的图像——相信我，当别人在自己的站点中显示你的内容时，不会希望显示你的背景图像。根据经验，如果一幅图像属于页面结构的组成部分，因而需要在XHTML中引用，那么该图像就应该放在images文件夹中。如果一幅图像是通过CSS引用，因而属于页面表现的组成部分，那么该图像就应该放在images_pres文件夹中。

7.3 创建站点结构

即使是对本例中这个简单的网站，也有必要绘制一幅结构图，以便在开始创建图像和编写代码之前对站点的结构进行预先规划。通过结构图我们可以更清楚地看到站点不同区域的内容之间的平衡关系，并帮助自己考虑如何命名页面以及如何对这些页面进行逻辑分组。这一点在开发大型网站时至关重要，而且，即使对本例这个只包含少量页面的网站来说，这样做也是必要的。图7-4展示了本书网站的结构图。

尽管图7-4的主要目的是表现站点结构，但其中也隐含地表现出了站点的外观。图7-4的上部就是一个页面布局的线框图（见图7-5），该图开始通过手工绘制，然后又在计算机中修饰而成。所谓线框图，是指以简单的线框、白底黑字的文本及灰色矩形代表的图像来表示的页面布局。当前，我们还不需要关心网站的视觉表现。通过图7-4，我们可以把问题集中在组织、层

图7-4 本书网站的简单结构图



次以及布局的整体平衡上面，不必因考虑阴影效果及如何精确地裁剪图像而分心。设计网站的视觉表现需要进行专门的讨论，但现在还不到时候。

Web设计不仅仅是视觉设计

在2007年10月于旧金山召开的Voices that Matter大会上，我听到一些设计者质疑结构驱动的方法对他们是否适用，他们说自己从来都是从页面的视觉设计着手，并在此基础上开始构建站点。首先，我认为设计的结构和表现方面可以同时着手，但在开始编码后，应该以结构为主导，因为结构是表现得以附着的基础。为此，在开始编写代码之前如果视觉设计能够就绪当然最好，而且有时也是必要的。通常，我都会把页面设计图打印出来，然后绘制表现div及其他元素的线框，并对相关的内容区域进行分组整合。对绘制出的线框，我都会为它们起个名字，这个名字将来就会成为div的ID值。如果你也是这样做的，那么可以将这些线框看成是线框图的基础，然后以结构为中心并在编写网页标记期间再完善视觉设计。

对这个网站来说，主页的线框图类似于图 7-5。

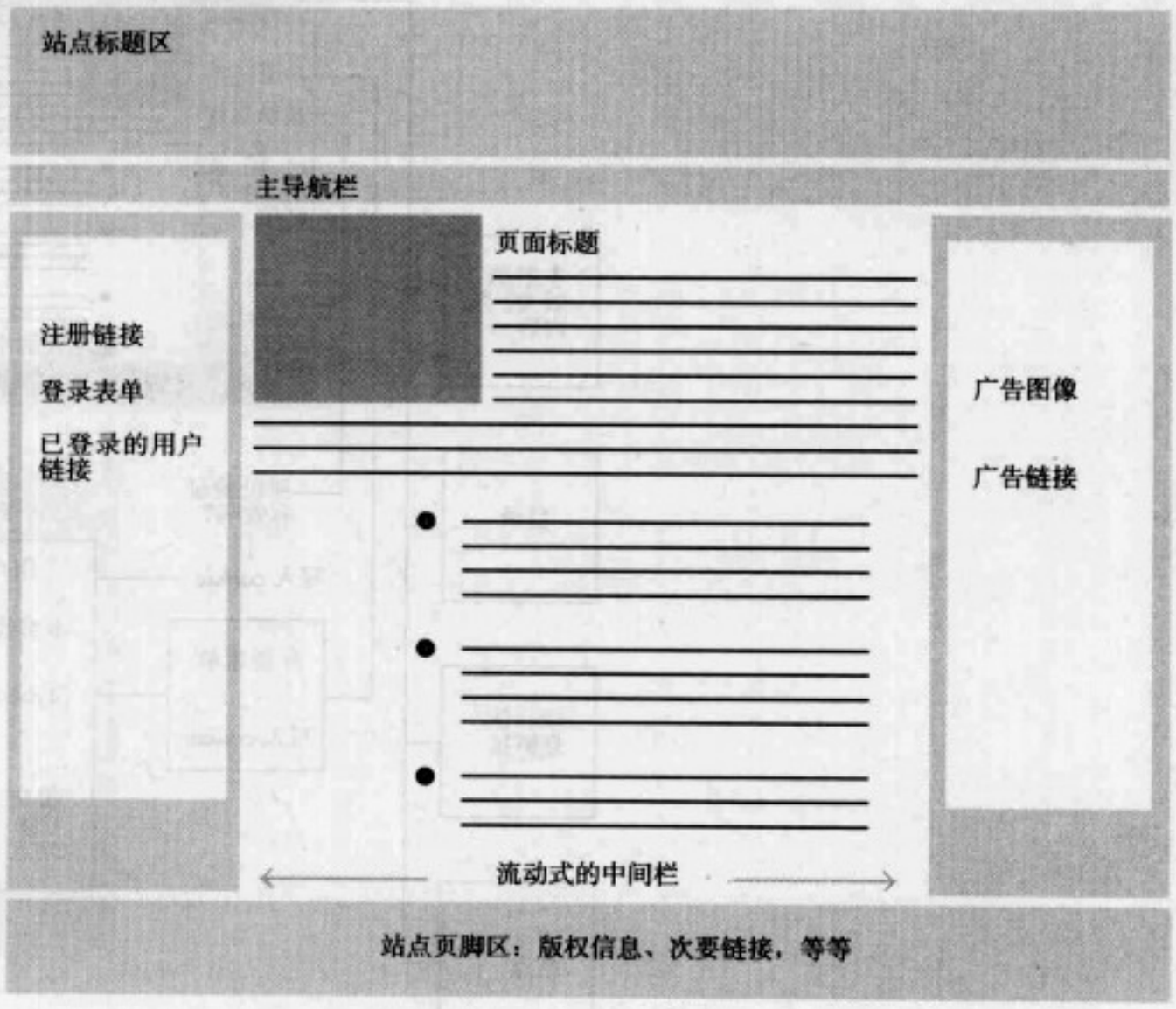


图 7-5　这个简单的线框图提供了为页面编写 CSS 代码的基础

通过 CSS 开发本身就意味着可以在视觉设计完成之前，把线框图转换为实际的原型网页。以第 4 章提到的 icyou.com 为例，我首先通过 CSS 编写了关键页面中基本布局的原型。在这一阶段，我只关注内容的组织和用户交互，不会考虑像颜色、字体和添加背景图像等之类的视觉问题。当然，在通过 CSS 构建具体的视觉设计效果时，还需要大量细小的修改和补充。我想表达的意思是，在摆脱过去“切完图后再放到表格中”的开发方法，并迁移到结构驱动的 CSS 方法之后，编程与视觉设计可以以协作和迭代的方式进行。从而，改变以前两者只有一方驱动开发过程，并且在一方未完成之前不能开始另一方的做法。

因此，我们在第一轮首先关注页面内容的布局，不过多考虑颜色和图像。具体来说，就是要关注如何通过内容布局显示其层次和关系。

为了创建本例中的三栏流动式页面，还需要用到第 5 章为这种布局形式开发的标记和 CSS。由于有了 Stylib 库，我们不用再为此重新编写代码——只需从库中获取即可。标记代码保

存在 3_col_liquid_faux.html 中。我把这个文件从库的布局文件夹复制到了根文件夹中，并将其重命名为 home.html。这样，当用户输入 www.stylinwithcss.com 时可以自动加载该页面。不过，在这里我们不需要复制第 5 章中用到的人造分栏图像，因为需要为本书的网站制作新图像。

7.3.1 从库中复制必要的 CSS 文件

下面，我们要把相关的 CSS 文件 3_col_liquid_faux.css 复制到 CSS 文件夹中——不用重命名。而且，还需要复制另外一些有用的样式表 text_n_colors.css、multi_level_menus.css 以及两个用于表单的样式表 form_layout_2_col.css 和 sign_in_form.css。

另外，由于 3_col_liquid_faux.html 模板使用了 NiftyCorners 代码来实现页眉和页脚的圆角效果，而且我们也希望保留这一效果。所以，需要把 NiftyCorners 文件夹也复制过来，并放到与 CSS 和 JavaScript 文件夹并列的顶级目录中。

除此之外，还需要一个 JavaScript 文件 minmax.js 来控制 IE 6 中的布局。因此，需要从 Stylib 库的 JavaScript 文件夹中将它复制到站点的 JavaScript 文件夹中。

下一步，我们会采取一种从未介绍过的方式将这些 CSS 文件与 XHTML 文件关联起来。这种关联方式能够在站点的页面迅速增多时提供更大的便利。

7.3.2 @import 规则

以前，我们介绍过使用 link 元素关联外部 CSS 文件和 XHTML 页面的方法。不过，通过 @import 规则也可以实现样式表与网页的关联。@import 规则是一种 CSS 规则，不是像 link 那样的 XHTML 元素。为了在 XHTML 页面中使用它，需要把它包含在一个 style 元素中——这样它就会如同第 2 章我们看到的嵌入式样式一样了。为此，需要在文档头部添加如下代码：

```
<style type="text/css">
  @import url(css/mystylesheet.css);
</style>
```

以上代码等价于：

```
<link href="css/mystylesheet.css" rel="stylesheet" />
```

也就是说，这两种方法都可以实现样式表与网页的关联。不过，@import 很有用是因为它作为一种 CSS 规则可以在样式表中使用。换句话说，我们可以在页面中链接一个样式表，而在该样式表中使用一系列 @import 规则，加载一组样式表。

这样一来，就可以避免把特定网站所需的数百行样式代码全都保存到一个样式表中；相反，我们可以把这些 CSS 规则分别保存到几个独立的样式表中，然后通过在 XHTML 页面中使用单个 link 标签加载它们。通常，这些样式表可以分别针对布局、文本或者其他较大的界面组件，比如表单和菜单等，这些组件一般只存在于某些页面中。显然，这样可以使我们根据实际的需要来加载样式表。

为了方便起见，我把一组样式表添加到一个“导入”（我的术语）样式表中，然后再在所有需要这些样式表的页面中导入这一个样式表。这个导入样式同其他样式一样，都位于 CSS 文件夹中，其作用就像是这个样式表集合的指针。通过将不同的站点样式表组合到多个导入样式表中，就能在每个页面中通过一个简单的 style 标签来关联该页面所需的全部样式表。

例如，在本书网站的主页中，可以看到以下代码：

```
<link href="import_3liq_txt_signin_menu.css" media="all"
rel="stylesheet" />
```

当然，这行代码关联了样式表 import_3liq_txt_signin_menu.css。

而在这个样式表中，（如它的名称所暗示的）导入了如下样式表：

```
@import url(3_col_liquid_faux.css);
@import url(text_n_colors.css);
@import url(sign_in_form.css);
@import url(multi_drop_menus_class.css);
```

这样，主页所需的全部样式表都关联完毕。而且，即使再添加或删除样式表也不会影响到主页。

虽然这一技术很有用，但必须注意的是，在一个页面中关联大量样式表是影响站点性能的重要因素——参见提示条“站点性能”。

小心FOUC

尽管使用@import规则同样可以在XHTML中导入样式表，但这种技术也存在一个问题。当页面中只有@import指令而没有link或script标签时，IE 6有时会在页面加载时瞬间显示没有应用CSS样式的页面内容——通常不是一个令人满意的效果。这个问题被称为FOUC（即Flash Of Un-styled Content，闪现未应用样式的内容）。不过，在XHTML中使用link标签来代替@import引入样式表可以避免这个问题。而且，只要XHTML中包含一个link标签，就不会出现FOUC问题。因此，如果想杜绝这种问题，那么最好是使用常规的link标签向页面中导入样式表（本章例子中正是这样做的）。也就是说，如果页面头部存在一个link标签或者一个script标签（用于向页面中关联JavaScript文件），那么页面就不会触发FOUC问题。要了解与FOCU有关的更多内容，请访问http://bluerobot.com/web/css/fouc.asp/。

对于注册页面，由于它的内容区域完全是一个表单，因此在侧边栏中也不需要出现小型登录表单，所以应该创建另外一个导入样式表并命名为import_3liq_txt_form_menu.css，该样式表中导入了以下样式表：

```
@import url(3_col_liquid_faux.css);
@import url(text_n_colors.css);
@import url(form_layout_2_col.css);
@import url(multi_drop_menus_class.css);
```

尽管@import规则可以在任何样式表中使用，但必须把它们放在样式表的最顶部，位于其他CSS规则前面——否则，它们会被忽略。

然后，我们使用下面这行代码将它与注册页面关联起来：

```
<link href="import_3liq_txt_form_menu.css" media="all"
rel="stylesheet" />
```

请注意，导入样式表名以import_开头，可以方便地识别它们。

读者也可以根据自己的站点分区来重新命名这些导入样式表，例如：import_product_listings.css。

站点性能

将多个样式表分门别类地组织起来，是一种管理大量样式表的有效方法。但是，千万不要把所有页面都会用到的CSS分割到过多的样式表中。假设每个页面都要加载5个以上的样式表，那么即使这些样式表很小，也会导致站点性能的明显下降。通常，浏览器请求每个文件并与服务器建立连接所需的时间，要远远超过下载文件所需的时间。如果你不能确定是否存在这个问题，那么可以使用Firebug（Firefox浏览器的一种调试用的附加组件——可以在www.getfirefox.com → Add-ons中找到）来检测使用多个样式表的速度。然后，把所有样式都复制到一个样式表中，并将这个样式表链接到页面，然后再检测一下。如果你发现了显著的差别，就说明需要把样式合并到更少的样式表中了。

不过，影响页面呈现时间（即，从按下回车键或单击一个链接到页面在浏览器中显示完成所需时间）的因素也是多方面的。这些因素包括网络延迟、服务器速度、必要的请求数量（每一幅图像、每一个外部样式表和JavaScript文件都需要一次单独的请求）以及将内容呈现到屏幕上的CSS和JavaScript的复杂程度等。其中一些因素是我们能够控制的，另外一些则不是。Yahoo!的Yslow是一个扩展Firebug能力的Firefox附加组件，通过该组件可以帮助设计者进一步搞清楚影响页面性能的原因。如果你在运行一个大型网站，我建议你通过Firebug来运行页面并在Yslow选项卡中查看网站的性能。有关站点性能的讨论和这样那样工具的链接，可以参考http://davidherron.com/book-page/190-measuring-website-speed。

这样，我们就有了一个名为home.html的三栏布局模板，而且通过导入样式表为该页面关联了4个样式表。这个页面的外观如图7-6所示。

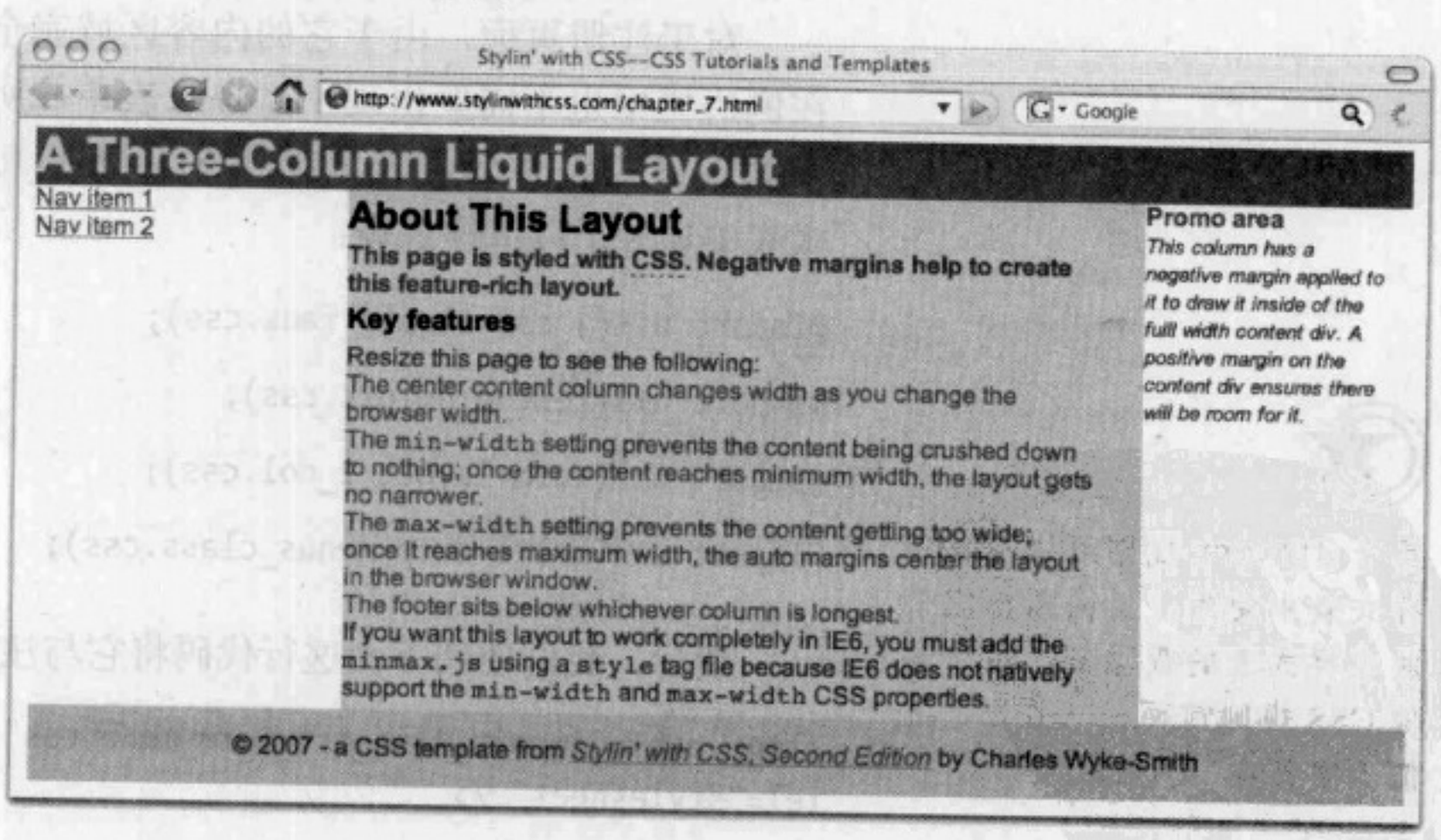


图7-6 在为3_col_liquid_faux模板链接了相关了CSS文件及text_n_colors CSS文件后，网站就具备了雏形

此时，只有3_col_liquid_faux.css和text_n_colors.css会实际影响这个页面。因为，即使sign_in_form.css和multi_drop_menus_class.css也链接到了页面中，但页面中还没有添加与之对应的标记。稍后我们会介绍表单和菜单的标记。

7.3.3 与文本和颜色有关的样式表

text_n_colors.css 样式表是我不久前为一个项目编写的。从那之后，我发现将其作为开发网站的起点非常方便。这个样式表的作用是为最常用的 XHTML 元素指定某些总体上的样式，以便将浏览器默认样式修改为更符合我的偏好。这种创建反映个人设计偏好的样式表，并将其作为每个网站设计起点的思想是一个有意思的概念，而 text_n_colors.css 样式表就是这样一个例子。为了使其中的“起点样式”尽可能发挥最大效用，我已经着手为页面布局中的主 div 采取了标准化的命名方式（例如：header、navigation、content、promo 和 footer 等），通过学习本书你应该会注意到这一点。假如这些名字作为某个特殊项目中 div 的 ID 不合适，那么我会选择相关性更强的名字，然后对 ID 和样式表中的类名进行简单地搜索替换，以便样式表与标记中“改进的”名字协调一致。

在 text_n_colors.css 样式表中，我针对颜色和文本大小创建了几组不同的规则，并通过一个描述性类名（例如，lime、olive）来进行组织。这样，每组中的每个样式规则都会使用一个上下文选择符，其中包含描述性的类名和一个 div 的 ID（例如：.lime #nav p {color:#444;}）。然后，通过为页面的 body 标签添加一个类名（在本例中是 lime），就能够确定每个主 div 的文本样式。（如果你没有理解上面的这些介绍，请参考下文中的样式表部分）。最后，我基本上都会针对具体的项目调整这些样式，不过，这个样式表确实为我提供了一个不错的起点。在这个样式表的基础上，经过修改会得到令我满意并整体协调的文本和元素背景，同时，也会为我节省几个小时的工作量。

通常，我会把 text_n_colors.css 样式表的一个副本粘贴到项目的 CSS 文件夹中，然后就可以在不直接修改其中样式的前提下，在样式表的末尾添加针对该站点的样式——这样就会覆盖前面需要改变的样式。在本例中，我们也会采取这种方式。

当前，我们从 Stylib 库复制到站点 CSS 文件夹中的这个 text_n_colors.css 样式表，包含着数百行代码，这是因为我在其中已经定义了几种不同的配色方案。当设计阶段结束时，我们会再回到这个样式表中，将不需要的样式全部删除——假如我基于类名为 lime 的颜色样式进行设计，那么最终就会删除与其他颜色方案相关的规则。为节省版面，我们这里不会展示该样式表中的所有规则，而只列出在完成

站点设计的过程中，不同于原始 text_n_colors.css 文件中的样式。下面，我们分块展示相关的样式，同时每块之间都视情况添加了注释。

```
* {
  margin:0;
  padding:0;
  }
body {
    font: 1em Lucida, Arial, sans-serif;
}
h1, h2, h3, h4, h5, h6, ul, ol, dl {
    font-family: 'Trebuchet MS', Arial, serif;
    }
```

首先，我们将所有元素的内、外边距设置为 0，再为所有元素设置一种字体系列。然后，为标题和列表设置特殊的字体系列。接下来，需要设置标题的文本大小：

```
h1 {font-size:2em;
    }
h2 {font-size:1.5em;
    line-height:1.25;
    padding: 0 0 0 0;
    }
h3 {font-size:1.25em;
    line-height:1.5;
    }
h4 {font-size:1.125em;
    }
h5 {font-size:1em;
    }
h6 {font-size:.875em;
    }
```

这样就重新设置了标题的相对大小。浏览器在默认情况下为大标题设置了过大的尺寸，因此最大和最小的标题文本之间的差距也很悬殊。通过设置以上样式，较小的标题会变得更大一些，从而缩小了标题字体大小的变化范围——不过，即使是最小的标题也比段落文本要大。如果标题文本还不如常规的文本大，那标题就没有什么意义了。下面，我们来为最常用的XHTML元素设置一些基本的样式：

```
p {font-size:1em;}

code {font-size:1.25em;}
* html code {font-size:1.1em;}

cite {
  font-size:.85em;
  font-style:italic;
  }
blockquote {
  width:30%;
  font-size:.7em;
  margin:0 0 1em 1em;
  padding:.3em .4em;
  border-top:2px solid;
  border-bottom:2px solid;
  }
blockquote cite {
  display:block;
  font-size:.85em;
  }
abbr, acronym {
  border-bottom:1px dashed #000;
  cursor:default;
```

针对IE 6的hack，因为IE 6中的默认字体更大一些

```
}
address {
  margin:0 1em .75em 1em;
  }
img {
  border:0;
  }
a:hover {
  text-decoration:none;
  }
```

以上是针对一些最常用的 XHTML 元素设置的基本样式。请注意，这些规则的选择符是纯粹的元素选择符，不包含上下文。因此，它们会影响页面中出现的那种字体中的所有元素。

接下来，我们再关注一下主 div 的背景颜色和最常用的 XHTML 元素的字体颜色。text_n_colors.css 样式表中包含了几种不同的颜色方案，但这里只展示本例中用到的 lime 类的方案。

```
.lime #main_wrapper {background-color:#FFF;}
.lime #header {background-color:#507EA1;}
.lime #nav {background-color:transparent;}
.lime #content {background-color:#CFE673;}
.lime #promo {background-color:transparent;}
.lime #footer {background-color:#BFCCD6;}

.lime h1 {
  color:#D6E2EC;
  }
.lime h2 {
  color:#000;
  }
```

```
.lime h3, .lime h5 {
  color:#000;
  }
.lime h4 {
  color:#507EA1;
  }
.lime h6 {
  color:#507EA1;
  }
.lime p {
  color:#555;
  }
.lime ul, .lime ol, .lime dl, .lime cite {
  color:#507EA1;
  }
.lime blockquote {
  color:#738040;
  }
.lime cite {
  color:#555;
  }
.lime table, .lime form {
  color: #507EA1;
  }
.lime a {
  color:#507EA1;
  }
.lime a:hover {
  color:#738040;
  }
```

```
.lime #nav a:hover, .lime #promo a:hover {
  color:#507EA1;
  }
.lime #nav a:hover, .lime #promo a:hover {
  color:#666;
  }
```

这样，通过为 body 标签添加 lime 类，以上样式就会应用到 ID 为指定值的 div 包含的相应的元素上。下面，我们再看一看包含新内容的标记在与这些样式配合的情况下是什么样的。

7.3.4　页面中的标记

以下就是网站主页中的 XHTML 标记：

激活 text_n_colors.css 中的 .lime 类

```
<body class="lime">
<div id="main_wrapper">
  <div id="header">
    <div id="header_inner">
      <h1>Stylin' with CSS: <span>A Designer's Guide, Second
      Edition</span></h1>
      <h3>A New Riders Book by Charles Wyke-Smith</h3>
    </div>
</div>
<div id="threecolwrap">
  <div id="twocolwrap">
    <div id="nav">
      <div id="nav_inner">
        <h4><a href="sign_up.html">Register Now</a></
        h4><p>to download the step-by-step <em>Stylin'
        </em> code examples!</p>
        <p>As a bonus, you get <em>Stylib</em>, a CSS
        library with page templates...</p>
      </div>
    </div>
```

header 结束

nav_inner 结束

nav 结束

中间的分栏（content）

```
<div id="content">
  <div id="content_inner">
    <img src="images/cws_cropped_bw_170px.jpg" alt=
    "Charles Wyke-Smith - author of Styin' with CSS" />
    <p>In <em>Stylin' with CSS</em>, Charles Wyke-
    Smith takes you.....examples and techniques.</p>
    <h4><strong><em>Stylin' with CSS</em> shows you
    how to:</strong></h4>
    <ul>
      <li>Create fixed-width....</li>
      <li>Create your own drop-down menus...</li>
      <li>Write for today's browsers...</li>
      <li>Accelerate site development...</li>
      <li>Use CSS like a pro...</li>
    </ul>
  </div>
</div>
```

content_inner 结束

content 结束

twocolwrap 结束

```
</div>
<div id="promo">
  <div id="promo_inner"> <img src="images/book_cover.
  jpg" alt="Stylin' with CSS - second edition by Charles
  Wyke-Smith" />
    <ul>
      <li><a href="#">Buy
          this Book</a></li>
      <li><a href="#">Read
          Reviews of this Book</a></li>
      <li><a href="#">Other
          Books by this Author</a></li>
      <li><a href="#">About the Author</a></li>
      <li><a href="#">The Author's Conference Schedule
      </a></li>
```

```
            <li><a href="#">Other Books We Recommend</a></li>
          </ul>
        </div>
      </div>
    </div>
    <div id="footer">
      <div id="footer_inner">
        <p>&copy; 2007 - Stylin' with CSS: A Designer's Guide,
        Second Edition by Charles Wyke-Smith</p>
      </div>
    </div>
  </div>
</body>
```

promo_inner 结束

promo 结束

threecolwrap 结束

footer_inner 结束

footer 结束

main_wrapper 结束

这样，我们就得到了如图 7-7 所示的结果。

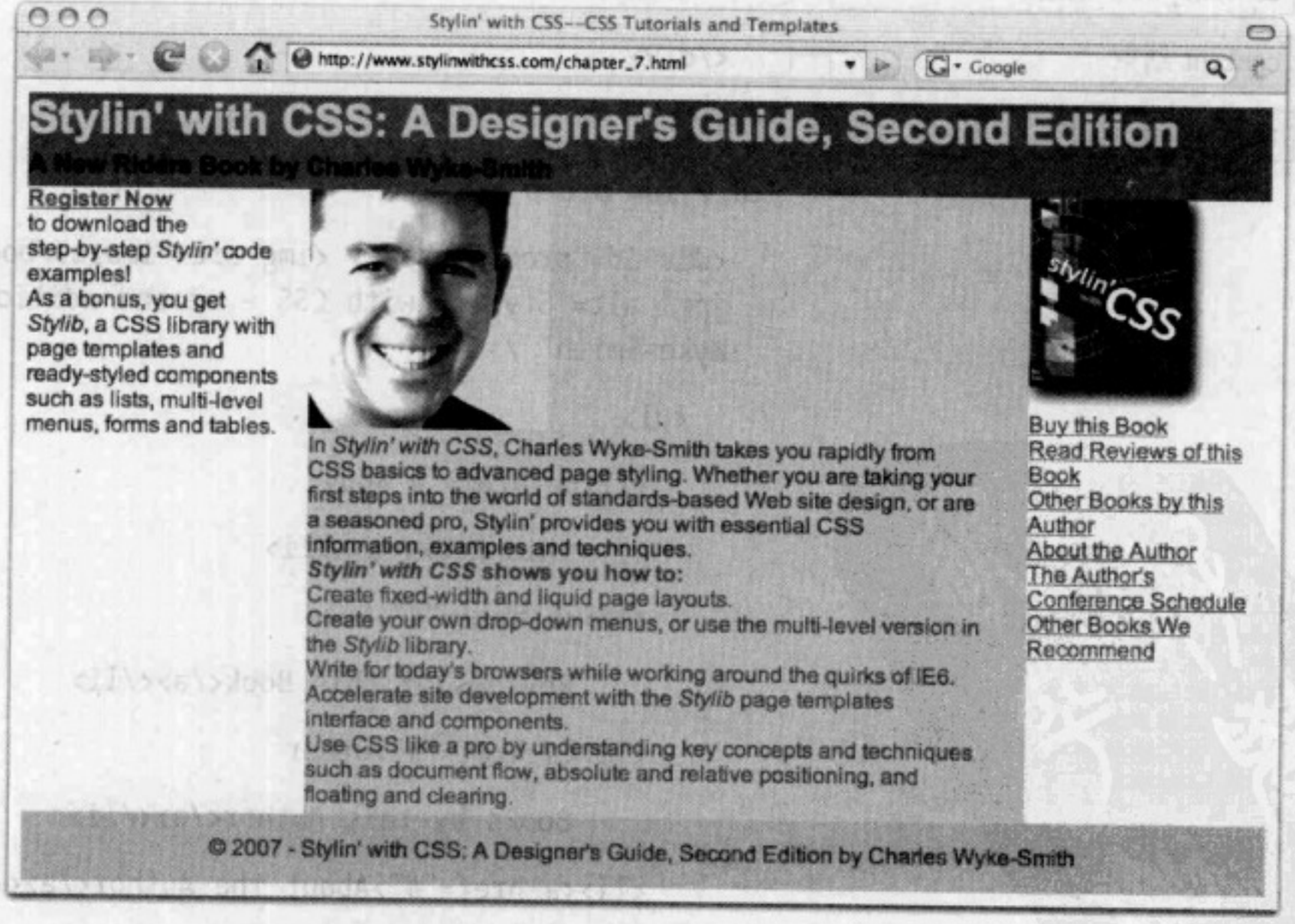


图 7-7 模板中作为占位符的内容已经被本站点的实际内容所替换

此时此刻，我们已经将标记好的内容添加到了预先存在的XHTML页面中。而且，又使用了预先存在的3_col_liquid_faux.css和text_n_colors.css样式表为这个页面应用了样式。虽然这样使我们的进度提前了很多（图7-7），但接下来仍然需要添加另外一些样式，以便页面拥有自己独特的外观。

7.3.5 背景图像

首先来添加背景图像。总共有4幅背景图像，其中一幅用于页眉、一幅用于页脚，剩下的两幅用于页面的主区域。我们先来看一看页眉和页脚的图像（图7-8A和图7-8B）。这两幅图像是在Adobe Photoshop中制作的，而且只是将较大的文本叠放在了暗灰色的背景上，然后输出为.gif格式。这两幅图像的宽度为880像素，这是我为本例的流动布局规划的最大宽度。

图7-8A和图7-8B　页眉和页脚的背景图像全都是880像素宽的.gif图

当把这些图像放到images文件夹后，要让它们显示到页眉和页脚的背景中就很简单了。打开3_col_liquid.css，然后在其中添加如下规则：

```
div#header {
  background:url(../images/gray_header.gif) repeat-y #383838;
  }
#footer {
  clear:both;
  background:url(../images/gray_footer.gif) repeat-y #383838;
  }
```

注意，对这两幅背景图像我们都使用了repeat-y（垂直重复）——尽管在正常情况下，哪一幅图像也不会重复显示。这样做的目的是，如果页眉或页脚变得比背景图像还高（这是不应该发生的情况，不过当有人无意间添加了过多的文本，或者用户通过“查看”菜单选择了太大的字体时就有可能发生），那么相应区域背景图像的下方不会出现空白间隙——此时，应该

页眉和页脚的背景颜色最初来自于text_n_colors.css样式表，因此我们需要从该样式表中移除相应声明。

会看到垂直重复的背景图像。同样，为防止图像加载失败，我们也添加了灰色的背景。这样，背景图像始终会显示在灰色的背景颜色上，正常情况下，如果图像填充了背景，那么背景颜色是不可见的，正如本例所示。在添加背景图像的同时声明一种背景颜色是好习惯，而且在我们这里，这种做法尤为重要。这是因为，我们在暗色调背景图像上使用了浅颜色的文本，这些文本在背景图像由于某种原因不能加载时，几乎不可能在默认的白色页面背景中显示出来。

除了页眉和页脚的背景图像之外，还要将一幅圆形重叠的图像添加到包含左侧和中间分栏的twocolwrap div中，将另一幅基于本书封面制作的图像应用到main_wrapper div。

```
div#main_wrapper {
```

省略了其他样式

```
  background:url(../images/cover_circles.jpg) no-repeat
  300px 0;
  }
#twocolwrap {
```

省略了其他样式

```
background:url(../images/full_arc.gif) no-repeat;
 }
```

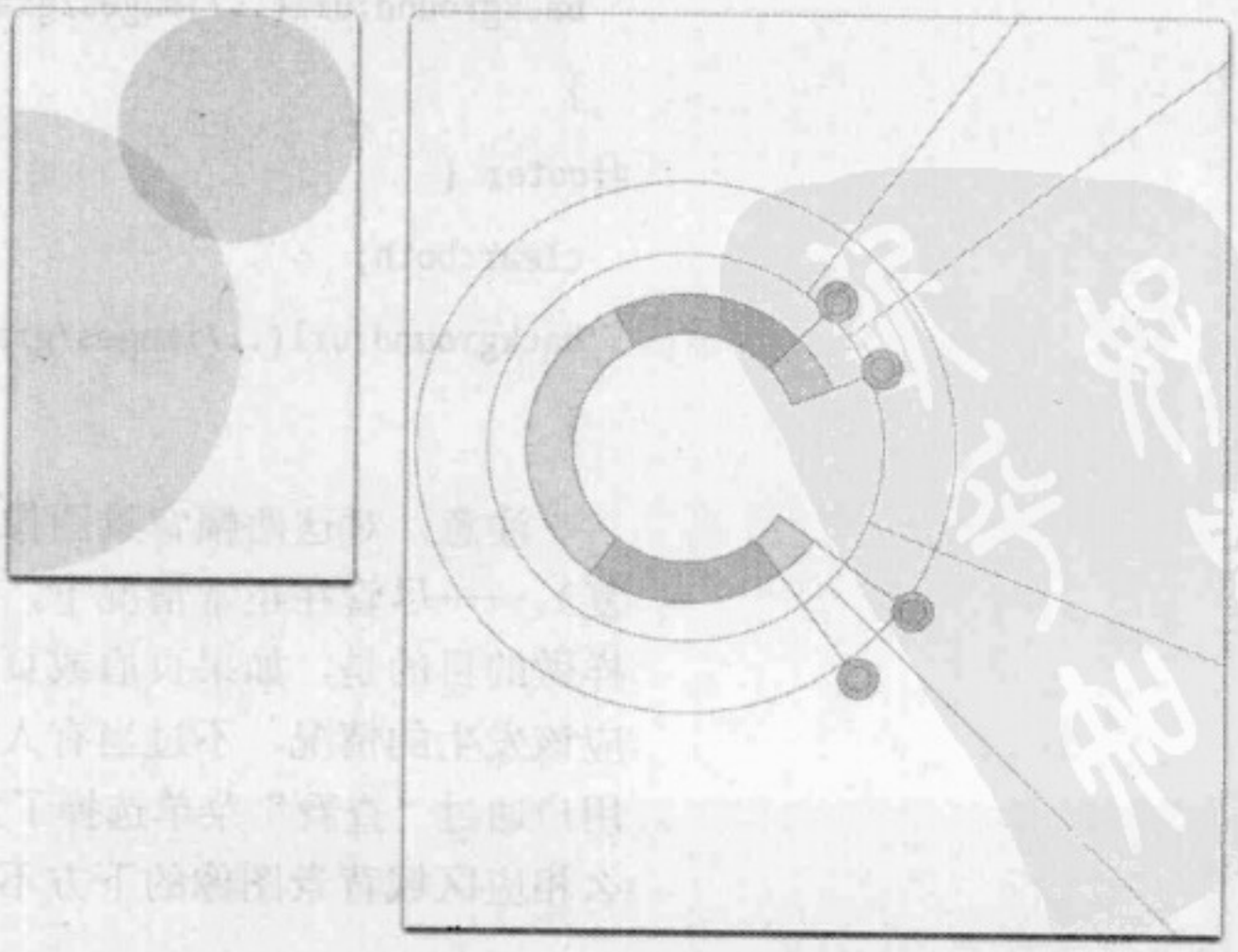


图7-9A和图7-9B　这里的两幅图像full_arc.gif和cover_circles.jpg会分别出现在左侧和中间分栏以及中间和右侧分栏的后面。它们的边框在原图中实际上是不存在的，这里只是为了清晰才在插图中添加的（另见彩插）

注意一下位置单位的用法（突出显示的代码）。main_wrapper div 包含了整个布局，因此为了让 cover_circles.jpg（图 7-9B）恰到好处地定位在内容区域后面，必须将它向右移动 300 像素。而 full_arc.gif（图 7-9A）的默认位置正是我们期望的，所以就不必额外地声明位置了。

此时，还需要在 text_n_colors.css 文件中找到为内容区添加背景颜色的样式，并修改这些样式，以便背景颜色由浅绿色变成透明。否则，main_wrapper div 的背景图像不会透过上面的 content div 显示出来。如果两个元素重叠（就像这里的 main_wrapper 和 content 一样），那么子元素会出现在父元素的上方。

```
.lime #content {background-color:#FFF #CFE673;}
```

这样，我们就得到了图 7-10 所示页面外观。

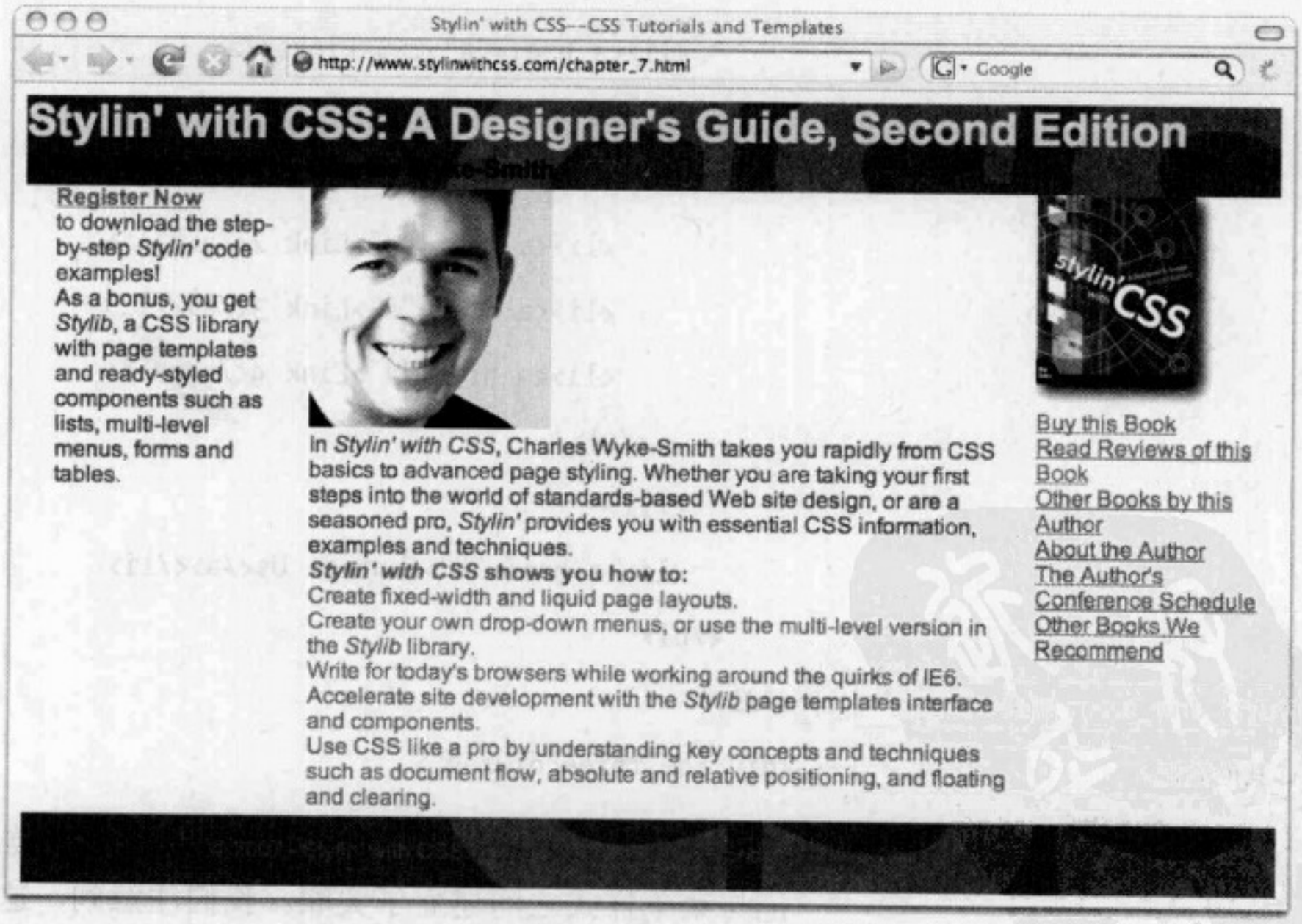


图 7-10 为布局添加了背景图像之后的页面外观

现在，页眉和页脚中的文本样式还需要进行一些调整。不过，我们会在处理其他文本时一并考虑。在此期间，我们先来添加下拉菜单。

7.3.6 下拉菜单

创建下拉菜单是相当简单的，因为构成菜单的标记无非就是一个嵌套的列表结构。在编写完列表标记之后，我们所要做的只是将其与已经编写好的 CSS 建立关联，然后再根据自己的偏好修改菜单的颜色。前面我们已经把菜单的样式表与页面建立了关联，因此只需添加一些标记，菜单就可以创建出来。

```
header 结束                    </div>
                    <div class="multi_drop_menus">
                     <ul>
                        <li><a href="#">What's New</a></li>
                        <li><a href="#">Table of Contents</a></li>
                        <li><a href="#">CSS Links</a>
                          <ul>
                            <li><a href="#">Link 1</a></li>
                            <li><a href="#">Link 2</a></li>
                            <li><a href="#">Link 3</a></li>
                            <li><a href="#">Link 4</a></li>
                          </ul>
                        </li>
                        <li><a href="#">Contact Us</a></li>
                      </ul>
multi_drop_menus 结束          </div>
（其他标记）                   <div id="threecolwrap">
```

这样，通过为 div 添加 multi_drop_menus 类，就在这些标记与菜单样式之间建立了关联。我们注意到，这个菜单 div 在页面标记中位于页眉和包含分栏的 threecolwrap 之间。如图 7-11 所示，菜单标记的位置决定了它恰好出现在页眉的下方。

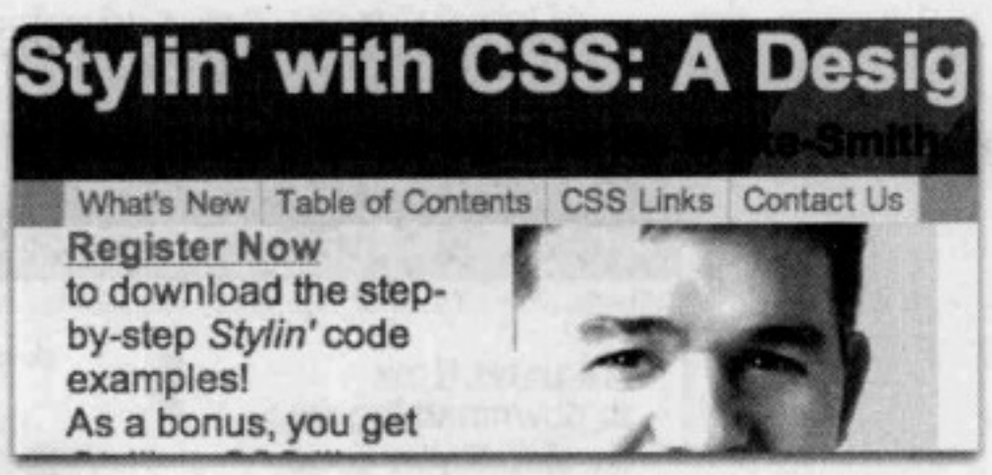


图 7-11 由于页面中已经关联了 multi_level_menu.css，菜单标记立即就应用了样式，尽管还需要在页面中对它进行调整（另见彩插）

显然，使用预先编写的库组件，可以使我们迅速得到一个相当令人满意的结果。此时，我们要做的就是调整菜单的颜色，并在菜单下方创建一些空间。通过菜单样式表进行一些局部的修改，就可以做到这一点。在菜单样式表中，所有用户可以修改的样式都集中放在了顶部，并与标记的顺序一致。而且，每个样式都有说明其用途的注释，因此查找和修改需要调整的样式也很容易。

```
div.multi_drop_menus {
```

针对 IE 6 及更早版本

```
  behavior:url(../../lib/js_tools/csshover.htc);
```

菜单的字体系列

```
  font-family: arial, sans-serif;
```

菜单文本相对于父元素的大小

```
  font-size:.8em;
  background-color:transparent; background-color:#AA9;
```

在菜单与内容之间创建空白

```
  margin:0px 0 10px 0px;
  }
```

菜单左侧的竖线

```
div.multi_drop_menus ul {
```

删除此规则可以使菜单完全居左

```
  border-left:1px solid #CCB;
  margin-left:20px;
  }
div.multi_drop_menus li {
```

一级菜单项

```
  background-color: #FFF; background-color:#EEB
```

创建分隔线

```
  border-right:1px solid #CCB;
  }
div.multi_drop_menus li:hover {
```

悬停时出现的菜单项

```
  background-color:#F0F7D9; background-color:#EE8;
  }
```

经过以上修改，将会得到图 7-12 所示的效果。

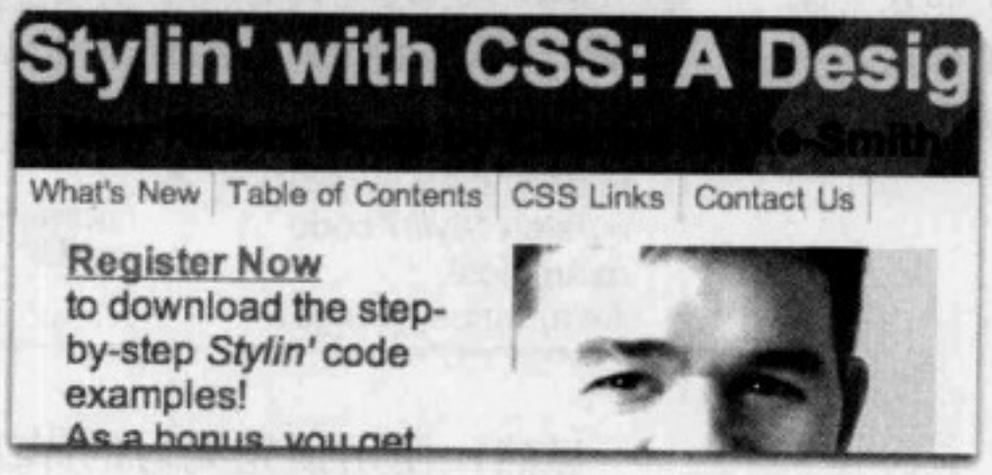


图 7-12　改进之后的菜单样式（另见彩插）

之前横跨页面的菜单 div 仍然横跨页面，只不过现在该 div 变成了透明的。通过删除 ul 的左外边距，使整个菜单向左移动到了布局的最左端。最后，通过添加很小的下外边距，在菜单和内容之间创造了必要的空间。

下拉菜单项的透明效果

multi_level_menu.css 文件也具备为菜单的下拉选项添加透明效果的功能。由于样式表的编写已经完成，所以现在要做的只是为菜单的包含 div 再添加一个 transparent 类，如下所示：

```
<div class="multi_drop_menus transparent">
```

然后，菜单会立即呈现出透明效果（图 7-13）。

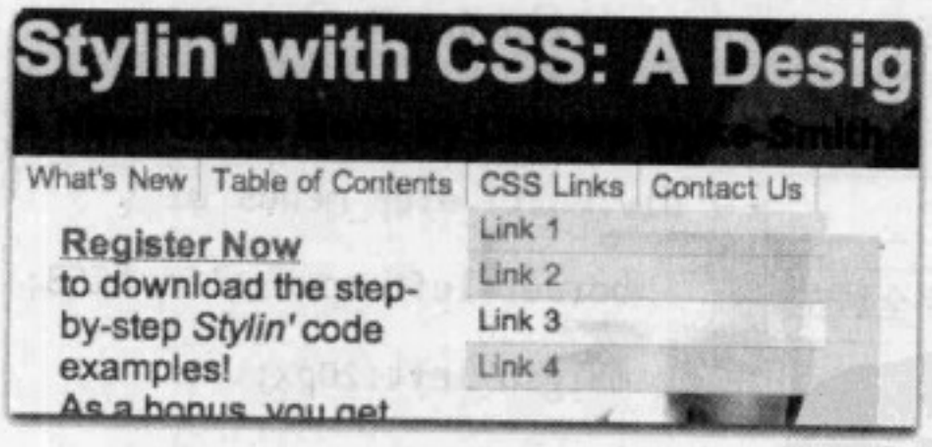


图 7-13　只要为菜单的包含 div 添加 transparent 类，可以使所有的下拉菜单项呈现出美观的透视效果（另见彩插）

实际上，CSS 规则中使用的术语是 opacity（不透明度）——当然，它是 transparency（透明度）的反义词。因此，元素透明度越高，opacity 值就应该越小。

然而，令人摸不着头脑的是，在不同的浏览器中存在 3 种创建不透明度的方法：

取值范围 0～100 —— (1) IE 方法：filter:alpha(opacity=90);

取值范围 0～1 —— (2) Mozilla（Firefox）方法：-moz-opacity:0.9;

取值范围 0～1 —— (3) CSS3 标准方法（Safari 及其他 SBC）：opacity:0.9;

因此，菜单样式表中的透明度规则如下所示：

```
div.multi_drop_menus.transparent ul ul li {
    opacity:0.9;
    -moz-opacity:0.9;
    filter:alpha(opacity=90);
    }
```

CSS3——取值范围 0～1

Firefox——取值范围 0～1

IE——取值范围 0～100

当你在自己的项目中使用这个菜单时，可以根据需要调整样式表中的透明度级别。不过，千万不要忘记同时包含这 3 种方法。

7.3.7 透明的侧边栏面板

如果你现在看一看完成的页面（参见图 7-1），会发现带有灰色背景和圆角效果的左侧边栏也是透明的。不过，此处的透明度不是通过编程实现的，而是以图像的方式创建的。

圆角盒子是 Web 2.0 设计时代的标志，而且创建这一效果应该很容易——毕竟，为 div 创建圆角的功能是 CSS3 的特性之一。不过，除了基于 Mozilla 的浏览器——如使用专有方法支持这一特性的 Firefox 之外，当前的其他浏览器都不支持这一特性。除了第 5 章介绍的通过 JavaScript 来模仿这种效果的 NiftyCorners 脚本，另一种创建圆角盒子的方法就是使用图像。

通过JavaScript创建圆角盒子

圆角盒子正日趋流行，下面是一些以编程方式实现圆角盒子的链接，供读者参考：

Editsite.net 的 Rounded Corners

http://www.editsite.net/blog/rounded_corners.html

Spiffy Corners.com 的 Spiffy Corners

http://www.spiffycorners.com/

Alessandro Fulciniti 的 Nifty Corners

http://www.html.it/articoli/niftycube/index.html

VertexWorks 的 Thrash Box

http://www.vertexwerks.com/tests/sidebox/

除了使用 JavaScript 为 div 创建圆角效果之外（参见提示条“通过 JavaScript 创建圆角盒子”），我们还要介绍一种通过图像创建圆角的方法。在元素盒子的大小固定的情况下，创建圆角只需要生成图像并将其添加为相应元素的背景即可。但是，关键的技巧在于怎样使盒子随着其中内容的增多而扩展。如果想让以固定宽度图像为背景的盒子垂直扩展（也就是本例中的目标），那么最简单的方法就是创建 3 幅图像，分别作为 3 个包含元素的背景。

单纯解释不如实际示范，因此我们先来看一看这 3 幅图像：上方的图像构成盒子的顶部圆角，中间的图像可以通过重复自身填充盒子的中部，下方的图像构成盒子的底部圆角（图 7-14）。

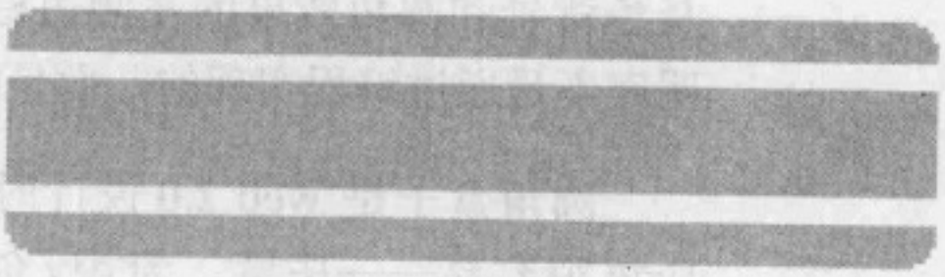


图 7-14 这是 3 幅构成盒子的灰色图像。它们的宽度都是 170 像素，而且透明度也都是 37%

这 3 幅图像是在 Adobe Fireworks 中创建的，最初它们是一个高 60 像素的灰盒子（十六进制颜色值为：#CCCCCC;）。在“属性”面板中，我把它们的圆角平滑度设置为 16，把透明度设置为 37%。在“修改”菜单中，我将“画布颜色”设置为透明，以确保为生成圆角而去掉的部分能够显示出盒子下方的背景。然后，我通过“修改”菜单中“画布大小”窗口把画布裁剪为原来的一小部分（上、中或下），并分别输出为 .png 图像。接着，再使用“撤销”命令恢复原来的画布，并再次裁剪得到并输出另一小部分。创建这 3 幅小图像总共花了大约 7 分钟。

1. 调整左侧分栏的宽度

在将这些小图像添加到左侧分栏之前，我们想让左侧分栏更宽一些，以便在背景盒子的两侧创建一些空间。当前，该分栏的宽度为 170 像素，我们需要把它加宽为 200 像素——这样就会在 170 像素宽的盒子两侧各留出 15 像素的外边距。在这个流动的布局中调整分栏宽度也很简单，只要记住每个分栏的宽度设置值都会对应相邻分栏的外边距即可。

```
#nav {
  float:left;
  width:150px;
  width:200px;
  }
#content {
  width:auto;
  margin-left:150px;
  margin-left:200px;
  margin-right:170px;
  padding:0;
  }
```

因为这里的布局是流动的，所以无需调整包含布局的外部div。而且，对于固定宽度的模板来说，也是如此。

注意，这里我们要把一幅背景图像放到另一幅背景图像上方。前面，我们将 full_arc.gif 添加到了 twocolwrap[①] div，该图像从视觉上也会出现在 nav div 的背景中。现在，我们需要把前面用于创建透明盒子的 3 幅小图添加到 nav div 中的 div。由于这些 div 是 twocolwrap 的后代元素，因此它们的背景图像会出现在 twocolwrap 背景图像的上方。换句话说，twocolwrap 中“最远的”背景图像（两个重叠的圆形）将会透过 nav 分栏的透明盒子显示出来——这是很令人惬意的一种视觉效果。

2. 为左侧分栏添加透明的背景图像

因此，接下来的问题就是：把透明的背景图像添加给哪些元素呢？方案如下：将中间的背景图像添加到 nav_inner div。然后，再在标记中增加两个 div 元素：一个位于 nav_inner 之前——用于显示上方的圆角图像；另一个位于 nav_inner 之后——用于显示下方的圆角图像。这样，上方和下方的圆角图像就会与中间的背景图像融为一体。

① 原文 two_col_wrap 有误。本段下面两处情况也相同。——译者注

下面，我们向标记中添加新 div。

左侧的分栏

```
<div id="nav">
  <div id="top_of_box">
    <!--holds top of rounded box as background image-->
  </div>
  <div id="nav_inner">
    <!--holds center of rounded box as background image-->
    <!--all the content for the left column goes in here-->
  </div>
  <div id="bottom_of_box">
    <!--holds bottom of rounded box as background image-->
  </div>
</div>
```

top_of_box 结束

nav_inner 结束

bottom_of_box 结束

nav 结束

以下是相关的 CSS

```
#nav #top_of_box {
  height:8px;
  background:url(../images/gray_rnded_box_top_trans.png)
  no-repeat;
  margin:0 0 0 15px;
  }
#nav_inner {
  background:url(../images/gray_rnded_box_trans.png)
  repeat-y;
  }
#nav #bottom_of_box {
  height:8px;
  background: url(../images/gray_rnded_box_btm.png)
  no-repeat;
  margin:0 0 0 15px;
  }
```

即使针对 nav_inner div 的 CSS 位于 3_col_liquid.ccs 文件中，我们也有必要把这些特殊的 nav_inner 样式放到 text_n_colors.css 样式表里（其中包含我为这个页面专门编写的作为库中 CSS 补充的规则）。因为把这些样式同盒子其他部分的 div 样式放在一起是很直观的。不过，除此之外，我们还需要对 3_col_liquid.css 样式表中的 nav_inner 样式作一点小小的调整。以下是原始的样式规则：

两个 div 都具有相同的样式

```
#nav_inner, #promo_inner {
  padding:.5em .5em 1em 1em;
  }
```

我们要将其调整为如下所示：

```
#nav_inner {
  margin:0 15px;
  padding:0 6px;
  }
```

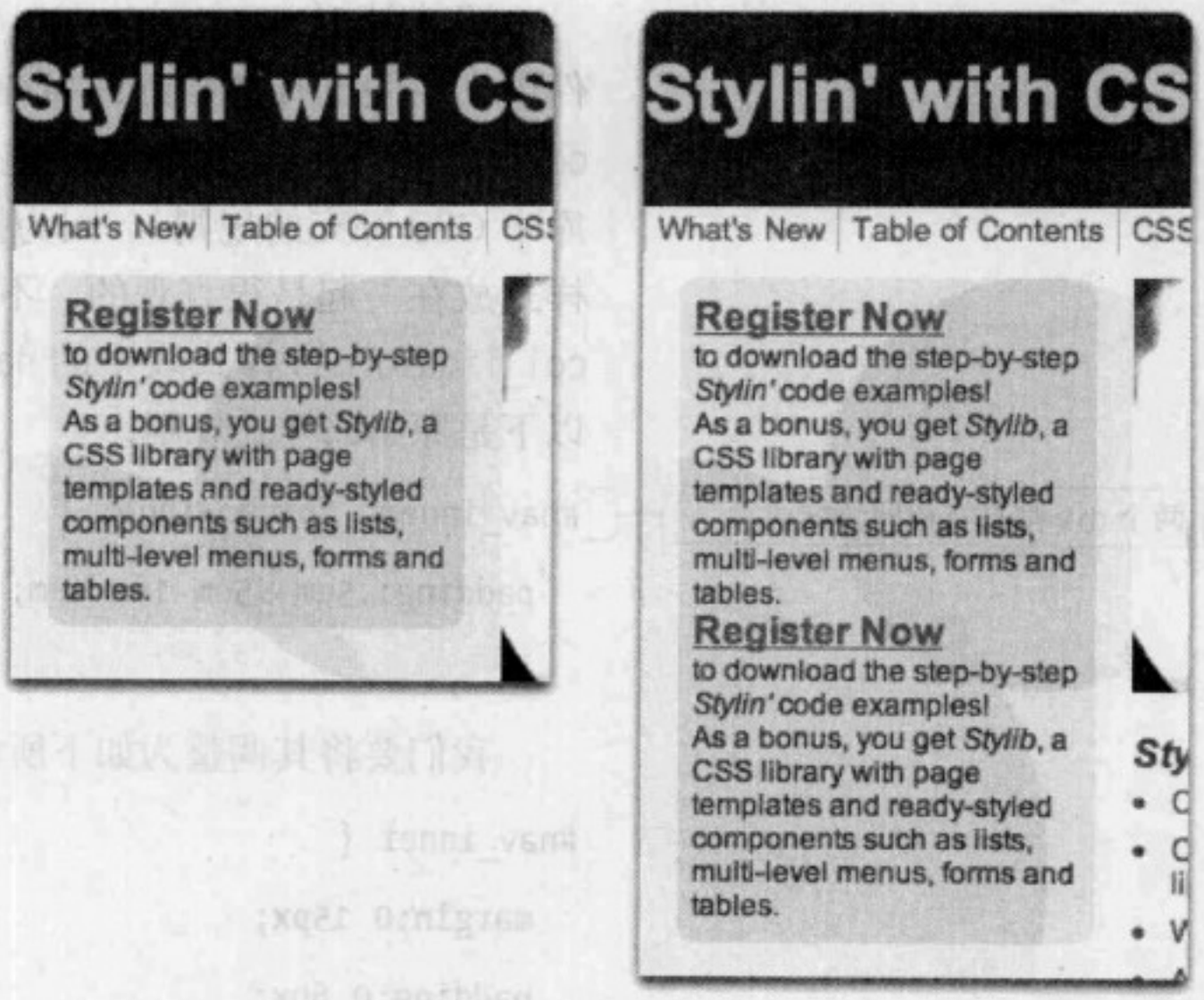
保持同以前一样的样式

```
#promo_inner {
  padding:.5em .5em 1em 1em;
}
```

以上调整是为了在包含圆角盒子的 3 个 div 两侧各添加 15 像素的外边距（突出显示的代码）。这样会把构成的盒子居中在 200 像素宽的分栏当中——盒子的宽度为 170 像素，两边各有 15 像素的外边距。因此，当前 nav_inner 的宽度实际上同它背景中圆角盒子的宽度相同。为此，我们可以为 nav_inner 两侧再各添加 6 像素的水平内边距，以便内容不会因扩展接触到背景盒子的边界。

现在，随着为 nav_inner 中添加内容，其背景会重复自己以适应增高的元素，与此同时，盒子底部的圆角图像也被推向下方并永远与整个盒子融合为一体。如第 2 幅图像所示，不管为 nav_inner 中添加多少内容，背景盒子都会随之扩展（图 7-15A 和图 7-15B）。这一效果的关键在于为 nav_inner 指定背景图像的 CSS 规则中使用了 repeat-y，这个值使得背景图像可以按照需要重复任意多次，而上方和下方的图像则始终与中间的图像保持紧密的契合，始终能构建出圆角盒子的效果。

图 7-15A 和图 7-15B 上方、中间和下方的图像相互连接构成了一幅完整的图像。在右侧的屏幕截图中，通过复制了相同的内容可以看到盒子随内容增加而扩展的效果（另见彩插）



我想说的是，以这种方式来创建透明的圆角盒子终归有点繁琐，我希望将来能够出现比使用 JavaScript 或圆角图像更简单的方法。同样，IDWIMIE——IE 6 无法显示重复的 .png 背景图像，因此，我们需要通过星号 html hack 在 IE 6 中简单地去掉背景，如下所示：

```
* html #nav #top_of_box, * html #nav_inner, * html #nav
#bottom_of_box {background:none;}
```

由于这个盒子主要是为了增强页面的可读性和视觉外观，因此看不到该背景图像也不是主要问题。

如果你确实希望在 IE 6 中使用 .png 图像的透明效果，可以在 http://www.twinhelix.com/css/iepngfix/ 中找到能够修复这一问题的 IE 行为文件。

7.3.8 添加注册表单

下面，我们在 nav_inner div 中现有文本的下方添加注册表单。因为第 6 章我们已经创建了登录表单的 XHTML 和 CSS（也正是这里所需要的），因此这一步会介绍得比较简单。我们这里使用了图 6-21 后面的代码，下面就不重复展示这些代码了（也可以在 Stylib 库中名为 sign_up.html 的文件中找到相同的标记）。这里代码需要放在 nav_inner[①] div 结束标签的前面。

① 原文 inner_nav 有误。——译者注

```
<div id="nav_inner">
  <div class="register">
    <form id="tiny_form_vert" action="#" method="post">

    </form>
  </div>
</div>
```

登录表单由此开始

表单元素的标记

register 结束

nav_inner 结束

在本章的开始，我们介绍了如何将针对这个表单的 CSS 文件与主页进行关联，因此下面就来看一看应用了样式之后的表单。注意相应的 CSS 在这里创建了一个不同的表单布局——标签与文本字段位于同一行中，而不是像第 6 章那样位于文本字段上方。这样一来，也就减少了表单在垂直方向上的高度。

这里，同第 6 章中标记的唯一一个不同之处，就是对作为提交按钮的 input 元素进行了修改。此处使用了一幅图像作为按钮，而不是通常的系统生成的按钮。为将图像添加到 input 标签，需要使用下面的代码：

```
<input type"image" src="images/form_button.gif" />
```

通过图 7-16 可以看到这个图像按钮的效果。

以下是针对这个表单的 CSS 代码，包含在 Stylib 库的 sign_up_form_2_col.css 文件中。

移除表单中所有元素的内、外边距

```
#tiny_form_vert * {
  margin:0;
  padding:0;
  }
form#tiny_form_vert {
  float:left;
  margin:.5em 0 0 0em;
  }
form#tiny_form_vert .formsection {
  width:150px;
  float:left;
  padding:.25em;
  }
```

为 IE 设置固定的宽度

使 formsection div 包含浮动的元素

```
#tiny_form_vert label {
  float:left;
  width:38%;
  margin-top:.35em;
  color:#000;
  font-size:.7em;
  }
#tiny_form_vert input[type="image"] {
  margin:0;
  display:block;
  }
#tiny_form_vert p {
  display:none;
  }
#tiny_form_vert p.error {
  display:block;
  color:red;
  font-size:.75em;
  }
#tiny_form_vert li {
  font-size:.7em;
  list-style-type:none;
  }
#tiny_form_vert a {
  color:#069;
  }
#tiny_form_vert a:hover {
  color:#336;
  text-decoration:none;
  }
```

为旁边的输入字段留出空间

设置标签的颜色

设置标签文本相对于父元素的大小

移除图像按钮上下方的空间

使按钮支持外边距

针对消息 / 错误列表的样式

在不存在 error 类的情况下隐藏段落——参见下面样式

当存在 error 类时显示段落

减少字体大小

移除列表的项目符号

未悬停时链接的颜色

悬停时链接的颜色

当处于滚动状态时移除链接的下划线

此时，可以将表单元素及其子元素都放入到包含元素中。而在页面关联了必要的CSS文件并且为表单div添加了正确的类之后，样式马上就可以起作用，结果如图7-16所示。

图7-16 位于文本下方应用了样式之后的登录表单。其中，透明的背景盒子为适应这个新增的表单也进行了垂直扩展（另见彩插）

7.3.9 文本样式

现在，剩下的工作就是为页眉、内容区、宣传区（右侧分栏）及页脚中的文本添加样式了。

1. 页眉

对于页眉中文本的效果，我希望标题完全居左对齐，而子标题则完全居右对齐。正常情况下，我们会在文本和包含容器的边界之间留出一定的空间（就像我们在左侧导航分栏中所做的那样），但是我希望在这里通过使用黑色和灰色为页眉创建一种生动的外观，同时，也通过文本紧贴布局边界创建出一种视觉上的张力。

为进一步强化这种效果，我们还将把两个标题向两个不同的方向移动。这样，当流动布局的大小发生变化时，两个标题的相对位置就会发生改变，从而增强了整体效果。同样，两个标题之间还要保持一定的距离，而通过在对应元素上使用外边距可以解决这个问题。

文本：Stylin' with CSS

```
#header h1 {
    margin:.5em 0 0 0;
}
```

文本：a designer's guide, second edition

```
#header h1 span {
    font-size:.85em;
}
```

子页面的页面标题（不在主页上）

```
#header h2 {
    margin:4em 0 0 6.25em;
    color:#999;
}
```

主页中子标题“A New Riders book...”

```
#header h3 {
    float:right;
    margin:0 0 0 0;
    color:#FFF;
}
```

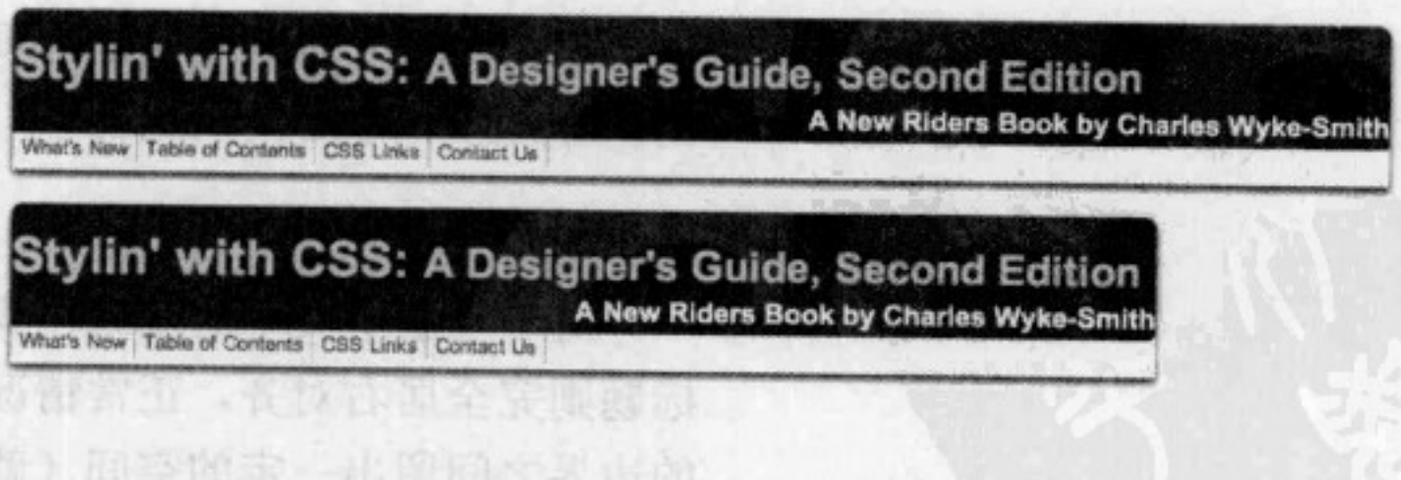


图 7-17A 和图 7-17B 随着页眉的变小，向右浮动的子标题移动到了左对齐的主标题下方（另见彩插）

以上简单的样式产生了如图 7-17A 和图 7-17B 所示的视觉效果。

在第4章中，我们强调过为浮动元素应用宽度的重要性。但对于浮动的图像来说，这是没有必要的，因为图像本身就具有固定的宽度，所以不需要为这里浮动的图像设置宽度。

2. 内容区

对中间分栏内容区的调整，主要是将文本向上提升到照片的旁边。这是通过向左浮动照片实现的——无论什么时候，我都会同时为照片的右侧和下方添加 0.5 em 左右的外边距，以确保图像不会与文本接触。

```
#content img {
  float:left;
  margin:0 10px 5px 0;
  }
```

然后，需要为图片旁边的段落添加一些样式。这里，我们不想放置过多的文本，而且，也希望列表从图片的下方开始。因此，这也是一次通过足够的行间距将文本区域变大，并使其突出显示的机会。

```
#content p {
  font-size:1em;
  line-height:140%;
  margin-bottom:.75em;
  }
```

最后，通过调整了一些列表的默认样式完成了页面中内容区的样式（图 7-18）。

```
#content ul li {
  margin: 0 0 0 16px;
  padding:.3em 0;
  font-size:.8em;
  }
```

确保项目符号不会伸出包含 ul 的左侧

在列表项之间创建垂直空间

为项目文本设置较小的字体大小

In *Stylin' with CSS*, Charles Wyke-Smith takes you rapidly from CSS basics to advanced page styling. Whether you are taking your first steps into the world of standards-based Web site design, or are a seasoned pro, *Stylin'* provides you with essential CSS information, examples and techniques.

***Stylin' with CSS* shows you how to:**

- Create fixed-width and liquid page layouts.
- Create your own drop-down menus, or use the multi-level version in the *Stylib* library.
- Write for today's browsers while working around the quirks of IE6.
- Accelerate site development with the *Stylib* page templates interface and components.
- Use CSS like a pro by understanding key concepts and techniques such as document flow, absolute and relative positioning, and floating and clearing.

图 7-18 应用了样式之后的内容区域。浮动的图像带有少量的右和下外边距，从而与文本之间保持了适当的距离

3. 右侧的宣传分栏

在这个分栏中，只有一幅本书的封面图像和一个链接列表。对于图像来说，由于 3_col_liquid_faux.css 样式表已经为其包含元素设置了内边距，因此不需要作任何调整。剩下的就是为列表设置一些样式，以便移除项目符号、减少字体大小、设置适当的文本间距。而且，我也为第一个列表项添加了一个类，以便可以额外加粗它的文本（尽管我更希望使用 :first-child 选择符以省去这个类，但你也知道——IDWIMIE6）。通过下面的 CSS，可以得到如图 7-19 所示的结果。

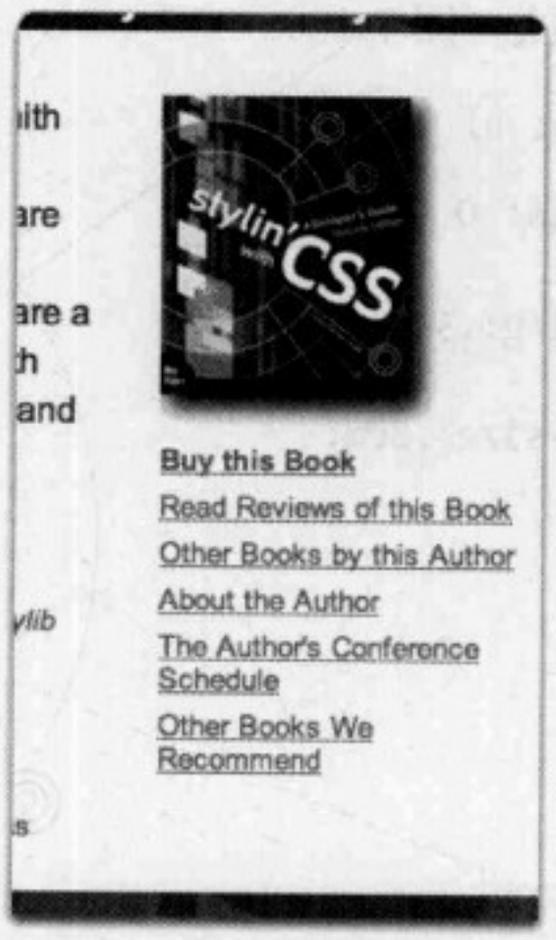


图 7-19 通过为列表设置一些简单的样式完成了对右侧分栏文本的样式化

```
#promo li {
  list-style-type:none;
  font-size:.8em;
  line-height:120%;
  padding:.3em 0;
  }
#promo li.big_link {
  font-weight:bold;
  }
```

整个布局中只剩下为页脚应用样式了。

4. 页脚

对于页脚，除了将原来的字体变小一些并为文本应用白色前景之外，也不需要什么过多的样式。

```
#footer p {
  font-size:.75em;
  color:#FFF;
  }
```

作为最后一步中的一个过程，我们需要修正指向 Nifty-Corners 文件的路径，以便页眉和页脚具有原始模板中的圆角。而且，在此也要利用 NiftyCorners 支持单独指定圆角的功能，不为页眉的左下角应用圆角，以便下方的菜单与它完美地契合为一个整体。此外，还要修正 minmax.js 文件的路径，以便 IE 6 能够支持在 CSS 中为布局指定最小和最大宽度。

此时，页面头部中的代码应该如下所示；必须确保代码中的路径名与实际要链接的文件吻合。这里的代码与本章开始展示的目录结构是一致的。

```
<script type="text/javascript" src="js_tools/minmax.js">
</script>
```

加载和使用 NiftyCorners

```
<script type="text/javascript" src="nifty_corners/javascript/
niftycube.js"></script>
<script type="text/javascript">
```

```
window.onload=function(){
Nifty('div#header','top br transparent small');
Nifty('div#footer','transparent small');
AddCss ("nifty_corners/css/niftyCorners.css");
}
```

为页眉的上边两角和右下角设置小型的透明圆角

为页脚设置透明的圆角

设置 Nifty 的 CSS 文件相对于页面的位置

注意，上面调用 NiftyCorners 中 JavaScript 函数的代码是如何指定圆角的（突出显示部分）——这里，我们指定了上边（top）和右下角（br），唯独没有指定左下角，因此左下角仍然是方角。通过设置透明度（transparent），也确保了背景图像的可见性。

就这些了——我们的页面到此结束（图 7-20）。

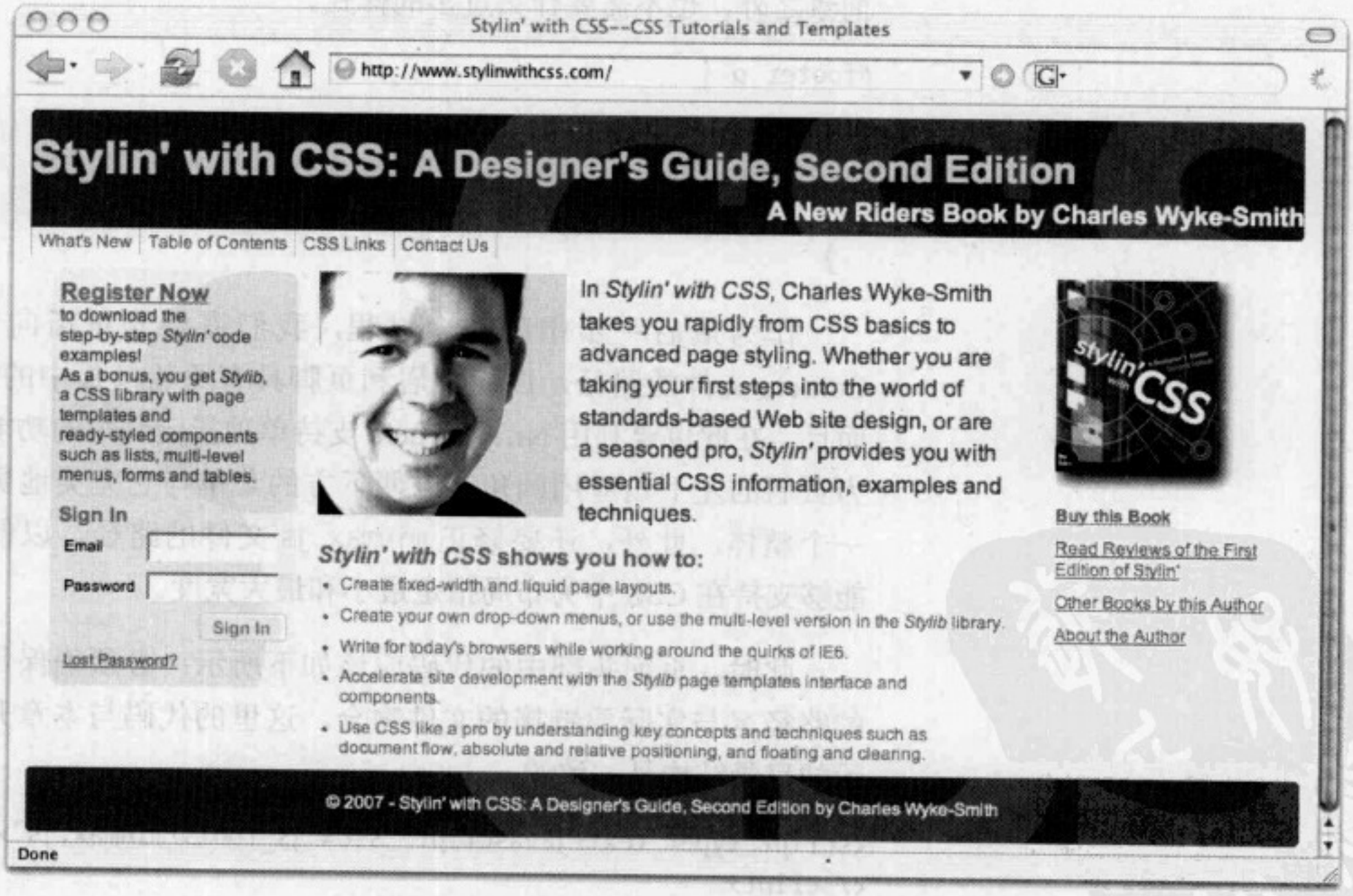


图 7-20 完成后的页面外观（另见彩插）

7.4 结束语

CSS 是为网页应用样式的一种强大机制。深入理解 CSS 工作原理，可以确保你（Web 设计者）的设计成果，能够最大限度地在各种浏览器中得到精确呈现。在本书中，我尽力把自己所知道的最有用的技术和见解和盘托出，并且，希望读者能够以 Stylib 库和书中的示例代码为基础，创建出更多的页面布局及界面组件。

诚如所知，我深深地喜爱 CSS。因此，我也希望本书展示的 CSS 令人称奇的能力会影响到你，并且也为你提供了必要的知识，使你能够以此为起点实现自己的创意梦想。现在，该轮到你挥洒自己的设计天才了。

附录 A

XHTML 标签参考

本表第 3 列中的 DTD（Document Type Definition，文档类型定义）表示相应标签的 DOCTYPE 中可以使用的 DTD：严格型（Strict）、过渡型（Transitional）和框架型（Frameset）。要了解与 DTD 有关的更多信息，请参考本书第 1 章。

不推荐（deprecated）意味着该标签可以使用，但正在逐步被淘汰，因而最好不要使用。通常，之所以不推荐使用某个标签，是因为存在对应的 CSS 替代属性。例如，用于居中文本的 标签，可以使用 CSS 属性 text-align:center; 来代替。注意，使用不推荐的标签不能通过严格型的 DOCTYPE 验证。

XHTML 标签

标　签	说　明	DTD
<!--...-->	定义注释	STF
<!DOCTYPE>	定义文档类型	STF
<a>	定义锚	STF
<abbr>	定义简写词	STF
<acronym>	定义首字母缩写词	STF
<address>	定义地址元素	STF
<applet>	（不推荐）定义 applet	TF
<area>	在图像地图中定义区域	STF
<b>	定义粗体文本	STF
<base>	为页面中的所有链接定义基准 URL	STF
<basefont>	（不推荐）定义基准字体	TF
<bdo>	定义文本显示的方向	STF
<big>	定义大号文本	STF
<blockquote>	定义较长的引用	STF
<body>	定义页面主体	STF
 	插入一个换行符	STF
<button>	定义按钮	STF
<caption>	定义表格标题	STF
	（不推荐）定义居中的文本	TF
<cite>	定义引用	STF
<code>	定义计算机代码文本	STF
<col>	定义表格列	STF
<colgroup>	定义表格列组	STF
<dd>	定义一个定义说明	STF
<del>	定义删除的文本	STF
<dir>	（不推荐）定义目录列表	TF
	定义文档中的一个部分	STF
<dfn>	定义一个定义术语	STF
<dl>	定义一个定义列表	STF
<dt>	定义一个定义术语	STF
	定义强调的文本	STF
<fieldset>	定义字段集	STF

（续）

标　签	说　明	DTD
<font>	（不推荐）定义文本的字体、大小和颜色	TF
<form>	定义表单	STF
<frame>	定义子窗口（框架）	F
<frameset>	定义框架集	F
<h1> 到 <h6>	定义标题 1 到标题 6	STF
<head>	定义关于文档的信息	STF
<hr>	定义水平线	STF
<html>	定义 html 文档	STF
<i>	定义斜体文本	STF
<iframe>	定义内联子窗口（框架）	TF
<img>	定义图像	STF
<input>	定义输入字段	STF
<ins>	定义插入的文本	STF
<isindex>	（不推荐）定义单行输入字段	TF
<kbd>	定义键盘文本	STF
<label>	定义表单控件的标签	STF
<legend>	定义字段集的标题	STF
	定义列表项	STF
<link>	定义资源引用	STF
<map>	定义图像地图	STF
<menu>	（不推荐）定义菜单列表	TF
<meta>	定义元（meta）信息	STF
<noframes>	定义无框架区域	TF
<noscript>	定义无脚本区域	STF
<object>	定义嵌入的对象	STF
	定义有序列表	STF
<optgroup>	定义选项组	STF
<option>	定义下拉列表中的选项	STF
<p>	定义段落	STF

（续）

标　签	说　明	DTD
<param>	定义对象的参数	STF
<pre>	定义预格式化的文本	STF
<q>	定义较短的引用	STF
<s>	（不推荐）定义加删除线的文本	TF
<samp>	定义示例计算机代码	STF
<script>	定义脚本	STF
<select>	定义可选择的列表	STF
<small>	定义小号文本	STF
	定义文档中的一个部分	STF
<strike>	（不推荐）定义加删除线的文本	TF
	定义加粗的文本	STF
<style>	定义样式	STF
<sub>	定义下标文本	STF
<sup>	定义上标文本	STF
<table>	定义表格	STF
<tbody>	定义表体	STF
<td>	定义表格单元格	STF
<textarea>	定义文本区	STF
<tfoot>	定义表注	STF
<th>	定义表头	STF
<thead>	定义表头	STF
<title>	定义文档的标题	STF
<tr>	定义表格行	STF
<tt>	定义打字机文本	STF
<u>	（不推荐）定义加下划线的文本	TF
	定义无序列表	STF
<var>	定义变量	STF
<xmp>	（不推荐）定义预格式化的文本	

附录 B

CSS 属性参考

背景

属性	说明	值	IE	F	N	W3C
background	用于在一条声明中设置所有背景属性的简写属性	*background-color* *background-image* *background-repeat* *background-attachment* *background-position*	4	1	6	1
background-attachment	设置一幅背景图像是固定还是随着页面其余部分滚动	scroll fixed	4	1	6	1
background-color	设置元素的背景颜色	*color-rgb* *color-hex* *color-name* transparent	4	1	4	1
background-image	设置元素的背景图像	url(*URL*) none	4	1	4	1
background-position	设置背景图像的开始位置	top left top center top right center left center center center right bottom left bottom center bottom right *x% y%* *xpos ypos*	4	1	6	1
background-repeat	设置背景图像是否/如何重复	repeat repeat-x repeat-y no-repeat	4	1	4	1

浏览器支持：IE：Internet Explorer；F：Firefox；N：Netscape。数字表示支持相应属性的最早版本。

W3C：W3C列中的数字表示定义相应属性的CSS推荐规范（CSS1或CSS2）

边框

属　性	说　明	值	IE	F	N	W3C
border	用于在一条声明中设置4个边框的简写属性	*border-width* *border-style* *border-color*	4	1	4	1
border-bottom	用于在一条声明中设置下方边框所有属性的简写属性	*border-bottom-width* *border-style* *border-color*	4	1	6	1
border-bottom-color	设置下方边框的颜色	*border-color*	4	1	6	2
border-bottom-style	设置下方边框的样式	*border-style*	4	1	6	2
border-bottom-width	设置下方边框的宽度	thin medium thick *length*	4	1	4	1
border-color	设置4个边框的颜色，可以带有1～4个颜色值	*color*	4	1	6	1
border-left	用于在一条声明中设置左侧边框所有属性的简写属性	*border-left-width* *border-style* *border-color*	4	1	6	1
border-left-color	设置左侧边框的颜色	*border-color*	4	1	6	2
border-left-style	设置左侧边框的样式	*border-style*	4	1	6	2
border-left-width	设置左侧边框的宽度	thin medium thick *length*	4	1	4	1
border-right	用于在一条声明中设置右侧边框所有属性的简写属性	*border-right-width* *border-style* *border-color*	4	1	6	1
border-right-color	设置右侧边框的颜色	*border-color*	4	1	6	2
border-right-style	设置右侧边框的样式	*border-style*	4	1	6	2
border-right-width	设置右侧边框的宽度	thin medium thick *length*	4	1	4	1
border-style	设置4个边框的样式，可以带有1到4个样式值	none hidden dotted dashed solid double groove ridge inset outset	4	1	6	1

（续）

属　性	说　明	值	IE	F	N	W3C
border-top	用于在一条声明中设置上方边框所有属性的简写属性	*border-top-width* *border-style* *border-color*	4	1	6	1
border-top-color	设置上方边框的颜色	*border-color*	4	1	6	2
border-top-style	设置上方边框的样式	*border-style*	4	1	6	2
border-top-width	设置上方边框的宽度	thin medium thick *length*	4	1	4	1
border-width	设置4个边框的宽度，可以带有1～4个宽度值	thin medium thick *length*	4	1	4	1

分类

属　性	说　明	值	IE	F	N	W3C
clear	设置元素的某一边不允许出现浮动元素	left right both none	4	1	4	1
cursor	指定光标显示的类型	*url* auto crosshair default pointer move e-resize ne-resize nw-resize n-resize se-resize sw-resize s-resize w-resize text wait help	4	1	6	2

（续）

属　性	说　明	值	IE	F	N	W3C
display	设置如何/是否显示元素	none inline block list-item run-in compact marker table inline-table table-row-group table-header-group table-footer-group table-row table-column-group table-column table-cell table-caption	4	1	4	1
float	设置一幅图像或一块文本出现在另一个元素的哪一侧	left right none	4	1	4	1
position	以静态、相对、绝对和固定的位置定位元素	static relative absolute fixed	4	1	4	2
visibility	设置一个元素是否可见	visible hidden collapse	4	1	6	2

尺寸

属　性	说　明	值	IE	F	N	W3C
height	设置元素的高度	auto *length* %	4	1	6	1
line-height	设置行间距	normal *number* *length* %	4	1	4	1
max-height	设置元素的最大高度	none *length* %	-	1	6	2

（续）

属　性	说　明	值	IE	F	N	W3C
max-width	设置元素的最大宽度	none *length* %	-	1	6	2
min-height	设置元素的最小高度	*length* %	-	1	6	2
min-width	设置元素的最小宽度	*length* %	-	1	6	2
width	设置元素的宽度	*auto* % *length*	4	1	4	1

字体

属　性	说　明	值	IE	F	N	W3C
font	用于在一条声明中设置所有字体属性的简写属性	*font-style* *font-variant* *font-weight* *font-size/line-height* *font-family* caption icon menu message-box small-caption status-bar	4	1	4	1
font-family	设置显示元素中文本的字体系列名称的优先级列表和（或）通用字体系列名称	*family-name* *generic-family*	3	1	4	1
font-size	设置字体大小	xx-small x-small small medium large x-large xx-large smaller larger *length* %	3	1	4	1
font-size-adjust	为元素指定一个特征值，用于保持首选字体的x-height（字母x的高度）	none *number*	-	-	-	2

（续）

属　性	说　明	值	IE	F	N	W3C
font-stretch	压缩或拉伸当前的字体系列	normal wider narrower ultra-condensed extra-condensed condensed semi-condensed semi-expanded expanded extra-expanded ultra-expanded	-	-	-	2
font-style	设置字体的样式	normal italic oblique	4	1	4	1
font-variant	设置文本是以小型大写字母还是以常规字体显示	normal small-caps	4	1	6	1
font-weight	设置字体的粗细	normal bold bolder lighter 100 200 300 400 500 600 700 800 900	4	1	4	1

生成的内容

属　性	说　明	值	IE	F	N	W3C
content	使用 :before 和 :after 伪元素在文档中生成内容	*string* *url* counter(name) counter(name, *list-style-type*) counters(*name*, *string*) counters(*name*, *string*, *list-style-type*) attr(X) open-quote close-quote no-open-quote no-close-quote		1	6	2

（续）

属　性	说　明	值	IE	F	N	W3C
counter-increment	设置每次出现相应选择符时计数器递增的数量	none *identifier number*				2
counter-reset	设置每次出现相应选择符时将计数器设置为什么值	none *identifier number*				2
quotes	设置引用标记的类型	none *string string*	-	1	6	2

列表和项目符号

属　性	说　明	值	IE	F	N	W3C
list-style	用于在一条声明中设置所有列表属性的简写属性	*list-style-type* *list-style-position* *list-style-image*	4	1	6	1
list-style-image	将一幅图像设置为列表项的项符号	none *url*	4	1	6	1
list-style-position	设置列表项的项目符号在列表中的位置	inside outside	4	1	6	1
list-style-type	设置项目符号的类型	none disc circle square decimal decimal-leading-zero lower-roman upper-roman lower-alpha upper-alpha lower-greek lower-latin upper-latin hebrew armenian georgian cjk-ideographic hiragana katakana hiragana-iroha katakana-iroha	4	1	4	1
marker-offset		auto *length*	1	7		2

外边距

属　性	说　明	值	IE	F	N	W3C
margin	在一条规则中设置所有外边距属性的简写属性	*margin-top* *margin-right* *margin-bottom* *margin-left*	4	1	4	1
margin-bottom	设置元素下方的外边距	auto *length* %	4	1	4	1
margin-left	设置元素左侧的外边距	auto *length* %	3	1	4	1
margin-right	设置元素右侧的外边距	auto *length* %	3	1	4	1
margin-top	设置元素上方的外边距	auto *length* %	3	1	4	1

轮廓

属　性	说　明	值	IE	F	N	W3C
outline	用于在一条声明中设置所有轮廓属性的简写属性	*outline-color* *outline-style* *outline-width*	-	1.5	-	2
outline-color	设置元素轮廓的颜色	*color* invert	-	1.5	-	2
outline-style	设置元素轮廓的样式	none dotted dashed solid double groove ridge inset outset	-	1.5	-	2
outline-width	设置元素轮廓的宽度	thin medium thick *length*	-	1.5	-	2

内边距

属　性	说　明	值	IE	F	N	W3C
padding	在一条规则中设置所有内边距属性的简写属性	*padding-top* *padding-right* *padding-bottom* *padding-left*	4	1	4	1
padding-bottom	设置元素下方的内边距	*length* %	4	1	4	1
padding-left	设置元素左侧的内边距	*length* %	4	1	4	1
padding-right	设置元素右侧的内边距	*length* %	4	1	4	1
padding-top	设置元素上方的内边距	*length* %	4	1	4	1

定位

属　性	说　明	值	IE	F	N	W3C
bottom	设置元素的下方边缘相对于父元素下方边缘向上/下偏移的距离	auto % *length*	5	1	6	2
clip	设置元素的形状。元素将被剪切为该形状并显示	*shape* auto	4	1	6	2
left	设置元素的左侧边缘相对于父元素左侧边缘向右/左偏移的距离	auto % *length*	4	1	4	2
overflow	设置元素内容溢出时的处理方式	visible hidden scroll auto	4	1	6	2
position	将元素定位在静态、相对、绝对或固定的位置	static relative absolute fixed	4	1	4	2
right	设置元素的右侧边缘相对于父元素右侧边缘向左/右偏移的距离	auto % *length*	5	1	6	2
top	设置元素的上方边缘相对于父元素上方边缘向下/上偏移的距离	auto % *length*	4	1	4	2

（续）

属　性	说　明	值	IE	F	N	W3C
vertical-align	设置元素中内容的垂直对齐方式	baseline sub super top text-top middle bottom text-bottom *length* %	4	1	4	1
z-index	设置一个元素的堆叠次序	auto *number*	4	1	6	2

表格

属　性	说　明	值	IE	F	N	W3C
border-collapse	设置表格的边框是折叠为单边框，还是像标准HTML中那样保持分离	collapse separate	5	1	7	2
border-spacing	设置单元格之间的距离（只适用于“分离”边框模型）	*length length*	5M	1	6	2
caption-side	设置表格标题的位置	top bottom left right	5M	1	6	2
empty-cells	设置是否显示表格中的空单元格（只适用于“分离”边框模型）	show hide	5M	1	6	2
table-layout	设置用于显示表格单元格、行和列的算法	auto fixed	5	1	6	2

文本

属　性	说　明	值	IE	F	N	W3C
color	设置文本的颜色	*color*	3	1	4	1
direction	设置文本的方向	ltr rtl	6	1	6	2
line-height	设置行间距	normal *number* *length* %	4	1	4	1
letter-spacing	增大或减小字符间距	normal *length*	4	1	6	1
text-align	元素中文本的对齐方式	left right center justify	4	1	4	1
text-decoration	为文本添加装饰	none underline overline line-through blink	4	1	4	1
text-indent	缩进元素中的首行文本	*length* %	4	1	4	1
text-shadow		none *color* *length*				
text-transform	控制元素文本中字母的大小写	none capitalize uppercase lowercase	4	1	4	1
unicode-bidi		normal embed bidi-override	5			2
white-space	设置对元素文本中空格的处理方式	normal pre nowrap	5	1	4	1
word-spacing	增大或减小单词间距	normal *length*	6	1	6	1

伪类

伪　类	用　途	IE	F	N	W3C
:active	为激活的元素添加特殊样式	4	1	8	1
:focus	为获得焦点的元素添加特殊样式	-	1.5	8	2
:hover	为处于鼠标悬停状态的元素添加特殊样式	4	1	7	1
:link	为未经访问的链接添加特殊样式	3	1	4	1
:visited	为访问过的链接添加特殊样式	3	1	4	1
:first-child	为作为其他元素第一个子元素的元素添加特殊样式	-	1	7	2
:lang	为特殊元素指定一种语言	-	1	8	2

伪元素

伪元素	用　途	IE	F	N	W3C
:first-letter	为文本中第一个字符添加特殊样式	5	1	8	1
:first-line	为文本中第一行添加特殊样式	5	1	8	1
:before	在元素前面插入内容		1.5	8	2
:after	在元素后面插入内容		1.5	8	2

媒体类型

媒体类型 *	说　明
all	用于所有类型的设备
aural	用于语音合成设备
braille	用于盲文触觉反馈设备
embossed	用于盲文打字机
handheld	用于小型或手持设备
print	用于打印机
projection	用于以投影方式展示项目方案，如幻灯片
screen	用于计算机屏幕显示
tty	用于使用固定行距字符网格的媒体，例如电报机或终端机
tv	用于电视机类型的设备

* 用作 link 标签中的 media 属性或者 @media 属性的值。关于 @media 的例子，参见：http://www.w3schools.com/css/css_mediatypes.asp

本附录中的表格经 W3 Schools（一家为 Web 开发人员提供免费在线学习资源的备受赞誉的网站，网址为 www.w3schools.com）授权使用。可以在 W3 Schools 网站上找到这些表格，其中每一部分都有相关的例子，并配有对每个标签和属性的详细解释。

“我读过不少同类图书，本书显然是设计师学习 CSS 的最佳起点。”

—— Harvey A. Ramer，资深Web设计师

“还在为学习 CSS 而抓耳挠腮？那还等什么呢，从这本书开始吧。我在读本书的时候，常常会恍然大悟地发出感慨：‘哦，原来别的书讲的是这个意思啊。’……这是你初学道路上的明灯。”

——Amazon.com 读者评论

写给大家看的CSS书（第2版）

Second Edition

Stylin’ with CSS: A Designer’s Guide

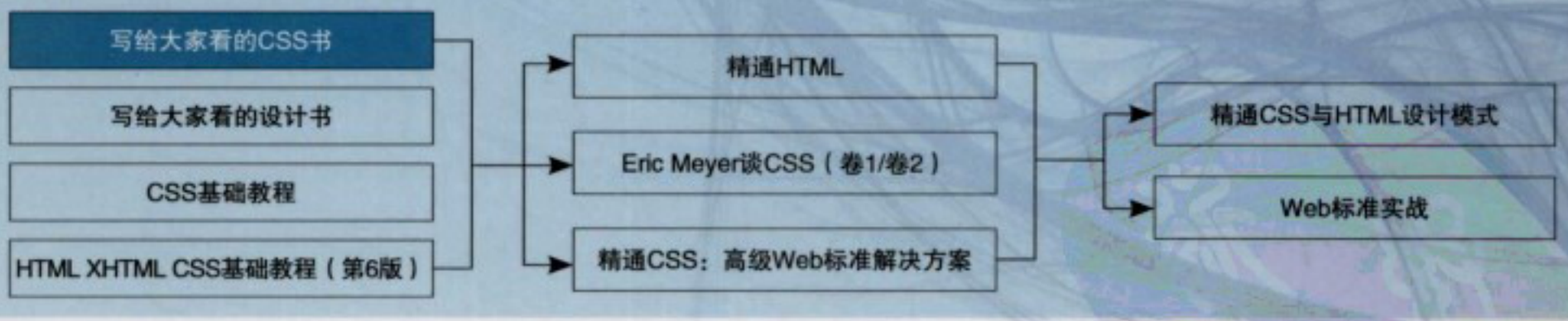
什么，还没有用过 CSS？那你也太落伍啦。今天，CSS 已经成为 Web 设计与开发人员不可缺少的核心工具。通过 CSS 集中管理网页样式，你可以轻松控制数百甚至上千个网页共享的设计方案，随心所欲地更换主题，甚至同时实现网页、平面等各种媒介的设计，从而显著缩短开发周期。更酷的是，CSS 还能够实现许多绚丽的、符合 Web 标准的动态效果。而且，CSS 也是迈向 Web 2.0 的重要技术基础之一。

但是，接触过 CSS 的人都知道，CSS 并不那么简单，你不仅要与以往只使用 HTML 设计网页形成的许多错误观念做斗争，而且 CSS 中许多概念理解起来也绝非易事。

没关系！在这部世界性的畅销书中，技术专家 Charles Wyke-Smith 用通俗易懂的语言为读者讲解了使用 CSS 所需要知道的各种知识、技巧和经验。全书的组织非常人性化，层层递进，读者即便没有任何编程背景，也会在不知不觉中到达前所未有的技术秘境，将自己的 Web 设计水平提高到一个新的层次。

Charles Wyke-Smith 世界知名的 Web 技术专家。目前是 Benefitfocus.com 公司用户体验总监。此前，他曾担任著名娱乐媒体 eStar.com 的 Web 开发副总裁，富国集团、ESPN 等大公司的 Web 设计顾问。

图灵Web设计阅读路线图



New Riders

本书相关信息请访问：图灵网站 http://www.turingbook.com

读者/作者热线：（010）88593802

反馈/投稿/推荐信箱：contact@turingbook.com

上架建议 计算机 / 网页制作

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-19364-3

ISBN 978-7-115-19364-3/TP

定价：49.00 元