畅销书作家、设计师、网页标准教父Jeffrey Zeldman再次更新了他经典的、颠覆行业的指南书。

网站重构

——应用Web标准进行设计（第3版）

Designi dards

[美] Jeffrey Zeldman Ethan Marcotte 著
傅捷 祝军 李宏 译
蒋芳（Windy） 审校

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

网站重构

——应用Web标准进行设计(第3版)

本书第3版是前二个版本的作者Jeffrey Zeldman与Ethan Marcotte合著的，内容涵盖了随着环境和技术的变化，Web标准所面临的挑战以及进行的改进。

第3版保持了本书一贯诙谐、有吸引力的写作风格，使复杂的技术和思想变得易于理解和消化。本书依然是Web标准应用的最佳指南，帮助你轻松地创建一个访问快速、低成本维护和开发、获得更多用户的网站。

其具体特点是：

◎大量的内容修订——用最新的思路和技术；

◎新的技术，包括HTML5、CSS3以及网页字体，对你的项目所产生的影响；

◎采取新的策略来推广Web标准；

◎改变“对IE6支持”的理解。

上架建议：计算机>网站设计

New Riders

ISBN 978-7-121-12775-5

9 787121 127755 >

定价：59.00元

策划编辑：孙学瑛
责任编辑：许 艳
封面设计：李 玲

网站重构

——应用Web标准进行设计(第3版)

Designing with Web Standards
(Third Edition)

[美] Jeffrey Zeldman　Ethan Marcotte　著
傅捷　祝军　李宏　译
将芳（Windy）　审校

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

畅销书作家、设计师、网页标准教父 Jeffrey Zeldman 再次更新了他经典的、颠覆行业的指南书。这已经是本书的第 3 版了，此次更新基本涵盖了随着环境和技术的变化，Web 标准所面临的挑战以及因此而发生的改善。第 3 版让基于标准的设计思想更加清晰，更加易于理解，帮助你在这个领域中保持聪明和领先。

一如既往，本书提供了更多明确的见解和新的例子，来阐述基于标准的设计的核心思想，为你的网站最终确定一个合理的设计与开发方法。

与前两版不同的是，在第 3 版，除了有很多的“Why”来提出问题，也提供了大量的“How”来解决问题。另外，在写作方法上作者力图用诙谐的、有吸引力的写作风格，使复杂的技术更易于理解消化，帮助你轻松创建一个访问快速、低成本维护和开发、更多用户的网站，帮助你用新技术使 CSS 布局适应多个浏览器，使网站内容更容易被搜索和访问到。

版权贸易合同登记号 图字：01-2009-7357

图书在版编目（CIP）数据

网站重构：应用 Web 标准进行设计 /（美）泽尔特曼（Zeldman,J.），（美）马克蒂（Marcotte,E.）著；傅捷，祝军，李宏译. —3 版. —北京：电子工业出版社，2011.3
书名原文: Designing with Web Standards, 3/e
ISBN 978-7-121-12775-5

Ⅰ. ①网… Ⅱ. ①泽… ②马… ③傅… ④祝… ⑤李… Ⅲ. ①网站－设计 Ⅳ. ①TP393.092

中国版本图书馆 CIP 数据核字（2010）第 008792 号

策划编辑：孙学瑛
责任编辑：许 艳
特约编辑：顾慧芳
印 刷：
装 订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×980 1/16 印张：23.75 字数：405 千字
印 次：2011 年 3 月第 1 次印刷
定 价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

依然感谢我的女儿 Ava。

——Jeffrey Zeldman

致 Elizabeth，我无与伦比的妻子，我们在自己的世界中共同舞动。

——Ethan Marcotte

第 3 版前言

当我和同事们在 1998 年创建 The Web Standards Project（网页标准计划小组，WaSP）的时候，我们并不知道有效的（valid）、语义化的标记（semantic markup）会让你的网站内容更容易被 Google 发现。是的，它确实有这个效果，但在 Google 出现之前，那并不是我们关注 Web 标准的原因。20 世纪 90 年代，一个网页设计高手的定义是：能够用 5 套代码方案为他的客户设计网站，让它在以下浏览器中都能正确显示：Netscape 3、Netscape 4、IE3、IE4 以及其他非主流浏览器。

如果还希望你的网站能在 Netscape 和微软的 IE3、IE4 浏览器里实现一些其他功能，那么你将不得不在同一个页面里使用两种不兼容的脚本语言，每种语言又有两个版本，总计四种不兼容的脚本语言。这些代码又混杂在使用复杂的表格布局（table layouts）的 HTML 代码里，结果就是：页面上至少多了 60%不必要的代码。当网站改版，重新设计，或者需要调整你的网站内容时，你会发现除了更高的成本之外，修改也非常困难，因为内容和布局完全混杂在一起。

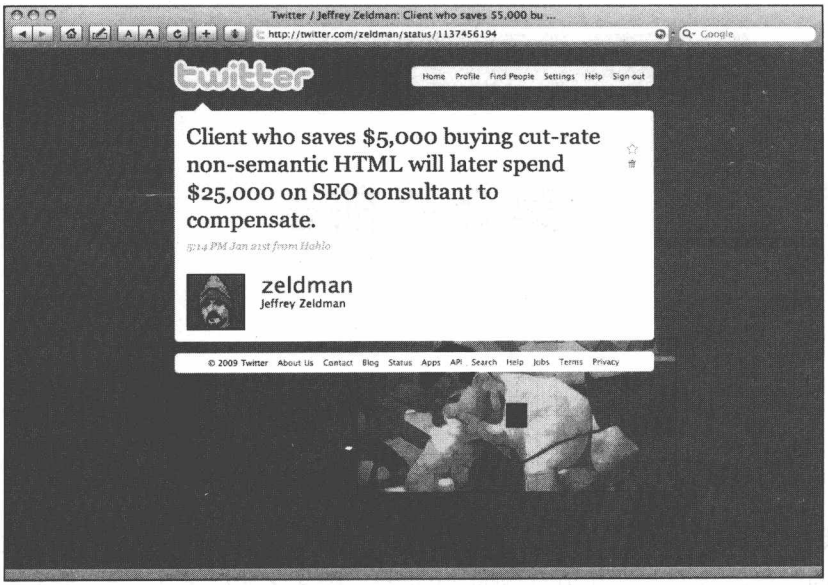


图 1　对于站点拥有者来说，遵循 Web 标准的一个巨大好处是：他的站点内容更容易被用户和搜索引擎找到。（www. twitter.com/zeldman/status/1137456194）

对于我们这些 WaSP 发起人来说，所有浏览器的私有代码都会严重威胁到 Web 发展。它导致每个网站至少浪费 25%的开发时间，从而成本提高至少 25%。而且，如果在 1998 年设计一个简单网站就需要写 5 套代码，那么未来的网页设计会变成什么样子？如果浏览器厂商为了竞争，故意推出不兼容的私有代码，那么我们这场自从印刷术发明以来最激动人心的出版和通信技术进步会不会倒退？10 年以后，我们是不是制作一个简单网站，就需要花费 12 倍的成本，写 12 套代码才能实现？

Web 标准——语义（X）HTML 标记、CSS 布局、基于 JavaScript 的不冲突脚本编程和 DOM，是以上所有问题的解决方案。而且，一直以来，WaSP 和其他组织对标准的倡导，最终说服了那些浏览器厂商支持标准，以及更多的专业人士来使用标准。

使用 Web 标准进行设计的好处（如图 1 所示），将在本书第 1 部分详细解释，它们包括如下内容。

- 恰当的语义化标记将使你的网站内容更容易被用户和搜索引擎找到。只要将代码从无语义的表格布局转换成结构良好的语义标记，网站就可以提高在搜

索引擎结果列表中的位置，提升它们的 Alexa 排名，并提高其他可发现性（Findability）的指标。（可发现性，是标准化的重要指标，以衡量搜索引擎优化（SEO）带来的好处。）

- 将网站的表现从结构和行为中剥离出来，使你的开发和测试更加容易，也更加便利，可以降低你的总预算，或者让你在网站可用性和内容开发方面的现金投入更宽裕。

- 将网站的表现从结构和行为中剥离出来，使你的网站更轻量级，从而提高性能。

- 将网站的表现从结构和行为中剥离出来，使用适当的有语义的标记，使你的网站更容易被各种浏览器和不同设备访问，包括移动设备和给残疾人士使用的浏览工具，例如屏幕阅读器和其他输入替代设备。使用 Web 标准设计网站同样有利于提供手机版本，如果你希望推出网站的手机版本，采用标准会使它变得更容易。而且在某些情况下可能根本就不需要推出手机版本，当然这也取决于你的网站内容。

- 使用 Web 标准而不是浏览器私有的代码设计网站，更加适应未来的发展。如果你的网站正确采用 HTML 4.01 或者 XHTML 1.0，并用 CSS2 布局，将得到各种浏览器的永久支持，即使 HTML5、CSS3 或者其他新的规范已经确定下来，甚至它们得到将来新的浏览器的出色支持，你采用现在的 Web 标准制作的网站仍将完美工作。而非标准的网站则没这么幸运，在好一点的情况下，它们会出现混乱，不好的话，网站将停止工作。

本书前两个版本已经翻译成 15 种语言，从保加利亚到韩国，基于标准的设计概念和方法已经被广泛地介绍给世界上成千上万的设计师和开发者，以及他们的客户。自从本书 2003 年开始销售以来，几乎所有承包商或自行开发的公司都已经有了至少一位 Web 标准的拥护者。这个行业在不断发展，不再只是那些领先的设计师们拥抱 Web 标准了，也不再只有那些前卫的承包商宣称自己在基于标准的设计方面很专业了。

而当本书的第 3 版面世的时候，Web 标准已经成为了设计主流，设计师和开发者开始争论 CSS3，甚至不少人已经开始尝试微格式（microformats）和 HTML5；符合 Web 标准现已成为包括苹果、Google、Opera、Adobe、微软等公司以及开源社区的软件必备项目。

这个版本经过了大幅度修订和重写，并重新组成了以下两个部分。

- 书的第 1 部分解释了传统的设计方法带来的问题以及 Web 标准如何解决这些问题。同时也提供了充足的论据给那些需要“推销”基于标准设计方案给那些仍然对标准持怀疑态度的客户、同事和老板的人。我们继续探查不断扩展的 Web 标准体系，展示新老标准是如何将整个网络变成一个动态的、有着众多的应用和高可访问性、可发现性（以及漂亮的外观）的内容平台。在第一部分的结尾我们展望一下 Web 的未来。

- 书的第 2 部分为读者介绍了 XHTML、HTML5 和 CSS，以及（更重要的）结构化，语义化的标记原则；简洁的、健壮的、优化的 CSS 布局，以及不冲突的脚本编程（unobtrusive scripting）。之后是侧边栏在布局和易用性上的重要性。在第 2 部分结尾，我们将深入一些网站设计项目去揭示它们关于标准的秘密。

我很感激我的新合著作者 Ethan Marcotte、我们的编辑 Erin Kissane，以及技术编辑 Aaron Gustafson，没有比这更好的团队了。如果这是一本好书，那么就感谢这些伙计吧。

第一次接触 Web 标准的新朋友，欢迎你们！还有那些早已经和我们共渡旅程的老朋友，欢迎你们回来！

译 者 序

一本好书，不仅仅带来知识，更重要的是传达一种理念、一种思想，引起我们更多的思考。《网站重构》无疑就是这样一本好书。它的出版犹如一声春雷打破了网页设计行业的沉寂，传统技术方式受到挑战并逐步被淘汰，Web 标准技术成为行业新的标准和开发指南。

如果说六年前，本书第 1 版出版时，我们还需要大声呼喊，我们要重视和推广 Web 标准理念。那么六年后，当本书即将出版第 3 版的时候，Web 标准已经成为网页设计行业的共识，成为 Web 开发通用的技术标准，成为网页设计师入门的基础知识。

关于作者

作者 Jeffrey Zeldman 是老牌的网页设计师，也是 Web 标准技术的捍卫者和“传教士”。他在 1998 年创立的 Web 标准组织（www.webstandards.org）曾经帮助 Netscape 公司和微软公司结束了浏览器私有技术之战；他的工作小组长期帮助和推进浏览器（例如 IE5、IE7 等版本）对 Web 标准的支持。本书自 2004 年第 1 版出版以来已经被翻译为 15 种语言，成为成千上万网页设计师学习网页技术的重要指南书籍。由于 Jeffrey Zeldman 对 Web 标准技术的长期推广和坚持，被《商业周刊》誉为“Web 标准国王”。

这是一本什么样的书

这是一本介绍 Web 标准的书籍。它是具有“里程碑”意义的书，已经在 Web 发展过程中开创一个崭新的时代！

它的“里程碑”意义对于中国的网站项目开发者和网页设计师来说尤为深刻。在本书出版之前，中国几乎所有的网页开发都采用表格布局法，《网站重构》出版后，以“CSS 布局”为典型代表的 Web 标准技术逐步为大家接受，直到今天被广泛采用。Web 标准技术不仅仅是一种技术的进步和更替，更重要的是它的这种思想和理念，深深影响了这一代网页设计师，让我们开始学会思考，在“漂亮”的页面背后，我

们长期忽略并应该关注的东西，包括：网站的成本、易用性、可维护性、SEO 以及可访问性。

本书的第 1 部分介绍了 Web 标准思想的产生、发展和推广，让我们了解采用 Web 标准的好处；书的第 2 部分结合实例，从技术细节上讲解了如何采用 Web 标准技术进行网页开发。通过阅读本书，你可以了解：

- 什么是 Web 标准？
- Web 标准有什么好处？
- 如何采用 Web 标准技术进行网页设计和开发？

适合什么样的人看

这本书适合的读者是：所有的网站设计师、开发者和网站管理者。

不论你是这一行业的新手还是资深人士，阅读本书都将让你获益匪浅。管理者可以了解如何降低网站成本，如何获得更多用户；设计师可以领悟到“表现和结构相分离”的真谛，用 CSS 自由地控制布局；开发者可以掌握如何降低维护成本，如何重用内容，如何延长网站的寿命。

对于已经了解和熟悉 Web 标准的开发者来说，本书第 3 版增加了更多实用的内容：

- Web 标准技术的最新发展动态和方向；
- 最新技术的介绍（包括 HTML5、CSS3 和网页字体），以及这些技术将对网站产生怎样的影响；
- 更多的实例应用和技术解决方案。

译者感言

从本书第 1 版出版到现在已经六年了，从 Web 标准“零”概念到目前行业内的普遍运用，不得不说是一个令人惊喜的变化。中国的网页设计师、开发者、管理者跟上了这一轮的技术变革。Web 标准在快速发展，虽然 Web 标准的思想核心依然是网站可用性，但随着更新的设备出现和技术的发展，许多老的 Web 标准开始淘汰，许多新的规范开始酝酿和发展，例如 HTML5、CSS3 等。学无止境，我们的设计师和开发者依然须要迎头赶上。

另一方面，随着中国互联网应用的普及和发展，中国网络开发者的地位和声音也逐步得到重视，万维网联盟W3C于2006年成立了中国办事处（www.chinaw3c.org），以便促进中国企业和开发者参与标准技术的研究和制定。但这方面我们还有比较大的差距，不论是标准制定的参与还是技术研究都还处于初级阶段。国际大型互联网公司，包括微软、苹果、谷歌、Mozilla 等都极力参与到标准的制定和推广中，值得国内的互联网公司重视和借鉴。

感谢

非常感谢电子工业出版社给我们翻译本书的机会，让我们得以见证和参与 Web 标准设计在中国的普及和推广过程。这一过程使我们自己获益匪浅。感谢博文视点公司总经理郭立、策划编辑孙学瑛的信任和鼓励，使得我们信心百倍地来翻译这本书，也感谢编辑许艳、顾慧芳，她们细心的工作帮我们弥补了很多细节上的漏洞。感谢许许多多给本书提出建议和意见的设计师朋友和网友。

译者

2010 年 6 月于北京

本书快览

第 1 部分

第 2 部分

目　录

第 1 部分

第 2 部分

第 1 部分

第 0 章

在开始阅读之前

本书是为那些希望自己的网站成本变得更低，运行得更好，访问者更多的网页设计师、开发者、网站所有者及管理者写的。如果你希望自己的网站不仅仅能适应目前的浏览器、屏幕阅读器和无线设备，也能适应明天的、明年的甚至更长远的新设备，那么本书正是你所需要的。

由于技术的迅速发展，我们中的大部分人已经不可避免地经历了几轮网络技术升级。每当浏览器升级或者新的网络设备出现时，它们好像都会破坏我们刚建成（或者刚支付了建设费用）的网站。

我们建立网站的目的似乎就是为了再次升级。很多情况下，升级改版并没有增加访问者需要的功能或者提高网站的可用性，而仅仅是为了跟上新的浏览器和设备，并且预算一定会超出我们的计划和开发周期。即使偶尔有一个新浏览器或设备的出现没有影响我们网站的情况，但我们采用的“向前兼容（backward-compatible）”技术（让站点在所有的浏览器版本中的外观和行为均一样）也会迫使我们花费大量人力和财力。“这些是在 Web 上做生意必须付出的代价”——我们已经习惯这样，甚至认为这样很正常。但是我们大多数人已经不能再承受这样的成本了。

0.1 终止不断“改版-淘汰”的怪圈

W3C 和其他标准化组织已经制定了相应的 Web 标准，这些标准也得到了大多数浏览器和设备支持，因此不论标准和浏览器如何发展变化，让网站的长久持续工作已经成为了可能。

什么是 W3C

W3C（World Wide Web Consortium，http//www.w3.org/ ）创建于 1994 年，该组织制定的 Web 规范和指导方针致力于推动 Web 发展，保证各种 Web 技术能很好地协同工作。大约有 500 个会员组织加入这个团体，它的创办人 Tim Berners-Lee（http//www.w3.org/People/Berners-Lee/）在 1989 年发明了 Web。W3C 推行的主要规范包括 HTML4、CSS2、CSS3、XML、XHTML1.0、XHTML1.1、HTML5（与浏览器厂商合作，由 Web 超文本技术工作组（the Web Hypertext Application Technology Working Group，简称 WHATWG，创建）以及标准的 DOM（ Document Object Model）等。

其他的标准组织还有欧洲计算机制造者协会（European Computer Manufacturers Association，ECMA），它标准化了 ECMAScript，也就是大家熟悉的“标准 JavaScript”。

本书将教你如何摆脱网站的“改版、淘汰、再改版”的怪圈，不再因为那些由于目光短浅而使用私有代码，导致种下恶果的解决方案而浪费时间和金钱，拒绝潜在的用户。本书将告诉你如何应对 IE 和其他一些浏览器偶尔不遵循 Web 标准的情况。同时，也将为你提供一些策略来应对一些顽固用户还在使用的老版本浏览器，可能你座位后面的同事用的就是。

有了这本书，设计师和开发者将能很快调整他们正在建设的网站，使它能够适应多种浏览器和设备，避免网站因为使用私有标记和代码技术而在未来被淘汰。读了这本书的网站拥有者将会立刻停止在怪圈上浪费金钱，他们将学会写出合适的需求文档来构建适应未来的网站。

0.2 不要死读书

这不是一本教条式的书。设计网站没有“最好”的方法，也没有一个绝对正确的方法能将 Web 标准融入到你的工作流程中。我不提倡死板地遵循标准，一些特别的网站和项目可以采用灵活的、变通的办法。这也不是一本纯理论的书，它是为那些需要认真完成工作的人们准备的。

我无意冒犯那些纯理论家，他们用自己的热情推动了 Web 标准的创建。我很佩服这样的人，并且幸运的是，我有一批这样的朋友，也从他们那里学到了许多。只是本书是给正在工作的设计师、开发人员，以及他们的客户和老板看的，这里探讨的 Web 标准技术将贯穿到具体的设计、内容和市场解决方案中。对于 W3C 规范中那些模糊的并且实现方法存在争议的地方，我将分享那些已经在标准社区达成一致的共识，来帮助你做出自己的判断和决定。

如果说这本书有一个教条的，或者有一个固执的、呆板的观点，那么就是：传统的设计方法成本太高了。

正在读这本书的人，没有一个能承担得起这样的开发成本，即：用传统的设计方法来建设现在的网站。每个站点都要用 6 套代码来写，在那个互联网泡沫过度膨胀的年代似乎是一个合理的做法，但现在，时代已经不同了。现在有了 HTML、XHTML、XML、CSS、JavaScript 以及 DOM，而且不光是它们自己，它们还成为了组件，可以构成合理的解决方案来解决那些自从有了 Blink 标签以来就困扰着网站所有者和开发者的问题。

渐进的标准，而不是一套不变的硬性规定

就像本书将要强调的，Web 标准是连续渐进的，不是一套不变的硬性规定。在转向 Web 标准的过程中，你的第一个甚至到第五个网站，表现和结构相分离的工作可能还都不完美；同样，你最开始在可访问性上的努力，也许只能达到 WCAG（“Web Content Accessibility Guidelines”网站内容可访问性指南的缩写）1.0 的一级，甚至还没有完全达到要求。（别害怕，正是因为很难判断是不是正确地遵循了 WCAG1.0，W3C 在 2008 年 12 月发布了完全不同于 WCAG1.0 的 WCAG2.0 规范）。

关键是你要“开始”。假如因为害羞不敢去健身房，那么你永远不能减去多余的脂肪。同样，如果我们不开始，我们的网站将永远无法实现向后兼容。删除“font”标签可能就是一个简单的开始，你可以用 H1 和 P 标签来替代无语义的它，这就是一个很好的起点。这本书将花费大量篇幅来介绍现代标记语言。

0.3 一些重要的定义

在本书中很多地方会提到“向后兼容”（forward compatibility），我的意思是：采用正确的方法设计和开发，任何发布的网页都能在多种浏览器、操作平台和网络设备中继续工作，就算将来发明了新的浏览器和设备也一样。Web 标准使这成为可能。

那么书中的“Web 标准”（Web standards）是什么意思呢？它的意思是，结构语言（例如 HTML、XHTML、XML）、表现语言（例如 CSS）、对象模型（例如 W3C 的 DOM）、以及脚本语言（例如 JavaScript），所有这些内容都会在本书进行解释。如果你是网站管理人员或网站拥有人，也不用担心那些技术章节，只要你的员工或者网站开发商能够理解就可以了。

专家指出，这些技术将为广大的网络用户带来巨大好处。总地来说，这些 Web 标准为合理的、先进的、高效的、低成本的 Web 开发方法提供了路标。另外，使用 Web 标准进行设计带来额外的好处是：你的网站将获得更好的可访问性。你将获得更多的用户、成本更低、改善公共关系，降低因为“无障碍访问”问题带来诉讼的可能性。（译者注：部分国家有法律法规，要求网站必须满足残障人士的访问要求。）

什么是 Web 标准组织

多年以来，W3C 一直称自己制定的规范是“建议”，这种称呼可能在无意中鼓励一些成员公司不严格执行 W3C 规范，例如 Netscape 和微软公司。Web 标准组织（The Web Standards Project，简称 WaSP）是 Web 设计师和开发者自发形成的一个群众团体。在 1998 年它成立的时候，就把 W3C 规范重命名为“Web 标准”，这是一个非传统的市场策略，有助于规范的重新定位，并使它们得到所有浏览器和网络设备的完全支持。

Web 标准组织（www.webstandards.org）提倡用标准来降低网站成本和复杂性，

使更多用户能够简单方便地访问。感谢 Web 标准组织，通过他们的努力，今天所有的浏览器都已经理所当然地支持 Web 标准了。Web 标准组织同时也为开发工具软件的开发商（例如 Dreamweaver）、网站所有者和设计师提供帮助。现在他们还提供 Web 标准的教育和培训，宣传效果已经扩大到了非英语国家。

书中说的“符合标准的浏览器”是指哪些浏览器？它包括：苹果公司基于 WebKit 的浏览器 Safari 3 和 4（当然包括 iPhone 手机上的浏览器）；开源的基于 Mozilla 的 Firefox3 及更高版本；Opera 软件公司的 Opera 9 以上浏览器及手机版本 Opera Mini；基于 WebKit 的 Google 浏览器 Chrome；以及微软的 IE 8（IE7 对 Web 标准的支持稍差些，IE6 则更差些）。以上这些产品有一个共同的特点，就是它们在不同程度上能正确解析和支持 Web 标准技术，包括 XHTML1.0、CSS2、JavaScript 和 DOM。（除了 IE，其他浏览器还支持 CSS3 和 HTMLL5 中的一些高级功能。）

这些浏览器都能完美的支持 Web 标准吗？当然不是，没有任何一款产品能够完全没有 bug。并且，标准本身也太复杂，标准之间的相互结合也很复杂。

那么是不是有些浏览器在标准兼容方面做得更好？我想每个有经验的开发者都能回答这个问题，我们大部分人都同意：即使是 IE 的最新版本，也不如 Safari、Firefox 和 Opera 在标准支持方面做得好。当然，很多用户还在使用对标准支持更差的低版本 IE 浏览器。

但即使你的用户自 IE6 流行以来就没有升级过他们的电脑，跨浏览器支持技术也已经非常可靠了，我们可以抛弃过时的方法、更聪明地工作，用 Web 标准设计来满足更多用户。而且由于标准天然的包容性，我们甚至也可以采用向前兼容的方法接受那些使用旧浏览器和设备的用户。

0.4 一种方法不能解决所有问题

本书虽然内容涵盖面广，并且已经出了三个版本，但对于 Web 标准的内容来说仅仅只是蜻蜓点水。还有很多关于 CSS、关于可用性、结构标记和 DOM、JavaScript 的内容不能被本书或其他任何参考资料所涵盖。正如我已经提到的，除了我们的观点之外，还有更多看待这些问题的方式。

自从本书第 1 版向全世界读者介绍 Web 标准以来，每星期都有关于“Web 标准”的新书问世。它们中的大部分只是在重复那些你已经知道的事情，也有一些书和本书的观点相矛盾，不同的技术有着不同的流派。

如果有两个设计师设计一个网页，那么可能会产生三种方案。每个设计师都有自己对布局样式、标记、导航条或颜色方面的看法，对于 Web 标准也是一样。不是每位读者都会立刻使用本书介绍的方法，但是，任何有远见、有思考的设计师、开发者或者站长都会赞同本书的先进概念。因为这样做能节省我们的时间、金钱，降低开销，延长网站寿命，让我们的内容更好地被访问。

如果这本书能够帮助你理解为什么 Web 标准技术能建立一个向后兼容的网站；如果这本书能够给你一些使用 Web 标准建站方面的技巧和帮助，那么就已经达到了它的目的。你可能不同意我的一些说法，但重点是不要因为一两个小分歧而停滞不前或者全盘否定；重点是开始改变，开始让你的网站获得更多的用户，存在更长时间，并且花费最少的成本。

0.5 欢迎优胜团队

W3C 的标准似乎已经多年没有什么进展了，但最新的浏览器技术规范（还没有全部完成）令人兴奋，例如 HTML5 和 CSS3。一些像 Google 这样的大公司开始在 Web 应用上使用这些新技术（radar.oreilly.com/2009/05/google-bets-big-on-html-5.html）。通过 CSS3 的新技术，为媒体带来了全新的设计思路。真正的排版技术已经触手可及（参见 www.zeldman.com/x/16），一些组织，例如苹果、MSN、维基百科和 WordPress 已经在它们的骨子里接受了 Web 标准，即使他们并不能完美地达到 100%纯语义标记验证（如图 01～图 04 所示）。

Web 标准是一系列工具，我们所有人都能用它们来设计和建立复杂而美观的网站，而这样建造的网站即使在未来，也能和现在一样正常显示。在这本书里，我将解释每个标准的作用，以及如何用它们建立一个向后兼容的网站。其余的就看你的了。

图 0.1　以产品精细设计著称的苹果，有一个秘密——它的网站采用 Web 标准建立（www.apple.com）

图 0.2　MSN，Alexa 排名 top100 第 7 名的网站，通过了 XHTML 严格校验（www.msn.com）

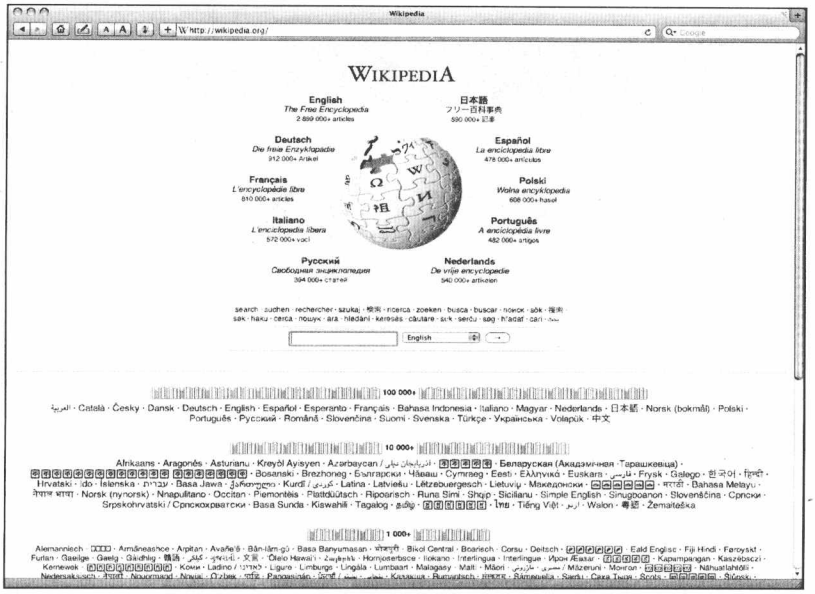


图 0.3　当你寻找答案时，你会选择维基百科。当维基百科寻求稳定性时，它选择了 XHTML 1.0 严格标记和 CSS 布局（www. wikipedia.org）

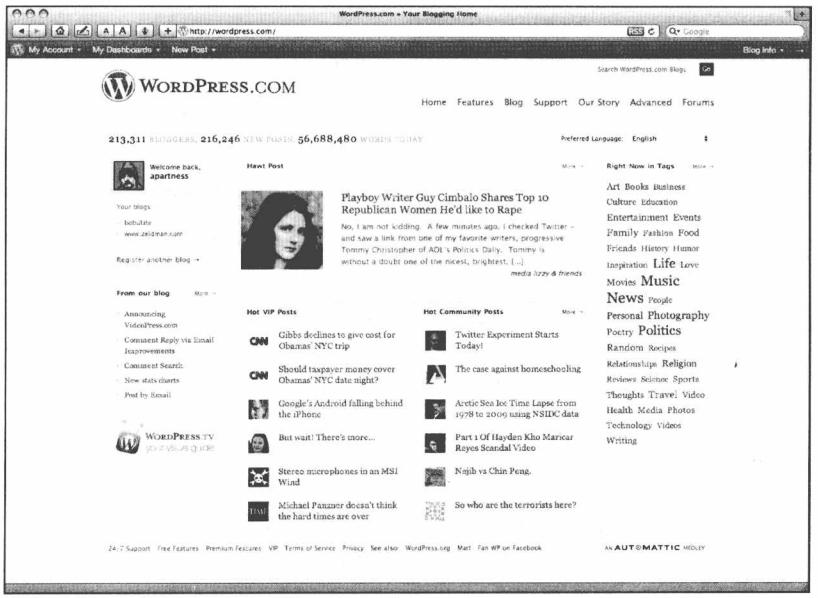


图 0.4　开源的博客平台 WordPress，Alexe 排名 top100 第 22 名的网站，通过了 XHTML1.0 过渡校验（www. wordpress.com）

第 1 章

99.9%的网站已经过时了

网络上有一种“病”困扰着几乎所有的网站，从简陋的个人主页到花费百万的商业网站都不能幸免。这种“病”隐藏得很深，以至大多数人并没意识到它的存在，因为它的病根深植在行业惯例之中。尽管网站所有者和管理者可能还不知道，但事实上 99.9%的网站都已经过时了。

那些站点也许在现在主流的桌面浏览器里看起来很好，但是一旦离开了容错的环境，弊病的症状和危害就开始显现。

在一些主流浏览器的最新版本中，包括 Microsoft 的 Internet Explorer、Opera 软件公司的 Opera 浏览器、苹果公司（Apple）的 Safari、Google 公司的 Chrome，以及 Mozilla（一种开放源码，基于 Gecko 的浏览器，Firefox 和 Camino 等浏览都是基于它的代码开发的），网站早先精心设计的布局已经开始部分变形，一些代价昂贵的 Web 交互程序也不能正常工作。在浏览器版本不断升级的同时，网站性能继续在恶化。

而在其他非主流浏览器中，在残障人士使用的屏幕阅读器中，以及在越来越多的上网手机中，大量的网站将无法工作，还有一些网站只剩下简单功能还算正常。根据需求和预算，网站所有者和开发者要么忽略了那些不知名的浏览器或设备，要么为了支持它们而专门定制标记和代码，就像以前对待“正规”的浏览器那样。

让我们一起来找出行业惯例中的有害部分，看看它是如何导致网站成本不断增加、如何导致网站开发越来越复杂而从未真正达到其建设初衷的。让我们来比较一下最新标准兼容浏览器和过去的浏览器之间有多少区别。

1.1　现代浏览器和 Web 标准

在这本书里，当我们提到“现代”或者“符合标准”的浏览器，就是指该浏览器能够理解和支持 HTML 和 XHTML、Cascading Style Sheets（CSS）、ECMAScript（通常也叫标准 Javascript）以及 W3C Document Object Model（DOM）标准。这些标准加在一起，可以帮助我们超越表现层的标记、不兼容的脚本语言以及它们所造成的网站不断升级、不断过期的怪圈。

符合标准的浏览器包括大奖获得者、开源浏览器 Firefox 3.5+；微软 Windows 系统下的 Internet Explorer 7/8 及其更高版本、苹果 Macintosh OS X 操作系统下的 Safari 3.0/4.0+；Google 公司的 Chrome 以及 Opera 软件公司的 Opera 9/10+（还有很多支持标准的浏览器，我不一一列举了），虽然我们称这些浏览器“符合标准”，但是请记住：还没有一个浏览器能够完美地支持标准。

然而缺少浏览器的完美支持不是逃避 Web 标准的理由。目前有上百万的人还在使用 Windows 系统下的 Internet Explorer 6。从标准支持的角度来说，这个浏览器是较差的（相对于 IE7+/Windows、Firefox、Opera 和 Safari 来说）。如果你的网站用户使用的是 IE6，是否就可以不考虑 Web 标准？你是否会要求他们升级 IE6 或者干脆拒绝他们？答案当然是否定的。符合标准的设计开发并不需要也不是“只为最新的浏览器版本设计”的。

同样，使用 XHTML 和 CSS 也不需要使用老浏览器的用户额外做什么，但是，使用 Web 标准设计的网站，并不会在 IE6 和其他符合标准的浏览器里看起来每个像素都一样。通过一些设计技巧，它们看起来可以完全不同，这样做没什么问题。事实上，一些设计师甚至提倡专门为 IE6 用户设计关键样式来提高可读性，而不去刻意模仿网站在其他（更符合标准的）浏览器里的外观和感觉。你可以阅读 Andy Clarke 的“IE6 通用 CSS”（www.forabeautifulweb.com/blog/about/universal_internet_explorer_6_css）了解更多这方面信息。

1.2 新版本新代码

现代浏览器不仅仅只是老浏览器的一个简单升级，它们的基础也发生了变化。很多浏览器的新版本完全是重新设计的，Mozilla Firefox、Camino 和其他相关的基于 Gecko 的浏览器已经不是从前 Netscape 4 的新版本。Opera 10 不是基于 Opera 浏览器老版本的代码开发的。这些浏览器已经为新版本写了新代码，也就是说，它们能够尽可能更贴切地符合本书所说的 Web 标准。

相比之下，20 世纪 90 年代的浏览器厂商专注于开发自己的私有技术（只有 Netscape 支持的或者只有 Microsoft IE 支持的技术）而基本上没有理会这些标准。

总体上，老式浏览器忽略了一些标准，但矛盾的是这种忽略并不是大问题。例如，如果浏览器根本不支持 Portable Network Graphic（PNG）标准，那么开发者就不会使用 PNG 图片。这样做没问题。可问题在于：那些浏览器实际上也支持了部分标准，甚至还以错误的方式支持了一些标准。正是这种漫不经心而且不一致的支持，造成了混乱的 Web 发布环境。

如果病人的阑尾发炎，合格的外科医生会做完整的阑尾切除手术。但如果相反，一个醉酒的实习医生来处理，他不但没有切除阑尾，还不小心损伤了其他器官，最后又了忘记缝合伤口，结果会怎样？老式浏览器对 Web 标准的支持就是这样，不完整、不合格、有害于网络的健康。

如果 Netscape 4 忽略作用在<body>标签上的 CSS 规则，并给页面的每个结构元素随意增加一些空白；如果 IE4 虽然能正确解析<body>标签，但是留白（padding）很糟糕，那么哪一种 CSS 写法是稳妥的呢？一些开发者因此选择不使用 CSS，另外一些开发者则针对 IE4 的缺陷写一个样式表，针对 Netscape 4 再写一个样式表。为弥补跨平台的字体和 UI 的细节差别问题，开发者或许还要为不同操作系统定义不同的样式。

CSS 不是唯一的问题，这些浏览器在 HTML 解析、表格表现、或者脚本语言处理等方面也无法达成一致。它们没有什么正确的方法来完全支持页面内容结构化、

页面的设计以及在页面上添加复杂的交互行为。

那个时候，如果要让网站能够在多个浏览器或操作系统中正常工作，设计师和开发者们不得不艰辛地应付不断增多的兼容问题，为每一个不同的浏览器定制设计一套非标准的标记和代码。现在已经不用这么做了，因为现在的浏览器都已经支持同样的开放标准。然而事实上这样的情况仍然在继续，依然有很多设计师和开发者在不必要地浪费资源，切分 Web，并导致网站的可访问性和可用性更差。

1.3　多版本问题

用非标准的标记和代码创建多个版本是导致网站过期的根本原因，但是，尽管这种做法成本很高，没有什么用，也不可持续，可还是有人这么做。

即使现在的浏览器支持 Web 标准，许多开发者也依然“无视”这一点。所以，尽管 IE6 已经能够处理标准的 JavaScript 和 DOM，他们仍旧继续写探测脚本来判断浏览器是否是 IE6，并编写针对 IE 的脚本来适应它，然后他们又不得不为基于 Mozilla 的现代浏览器写一些单独的探测脚本和代码，即使这些浏览器也能够处理标准的 JavaScript 和 DOM。

浏览器探测的危险

所有浏览器都有一个用户代理（UA）字符串，这是一个简短的文本字符串，包含了浏览器的名称、版本、平台信息，是浏览器的数字指纹。站长可以通过分析服务器日志中的 UA 字符串，来获取有价值的信息，以便更好地了解自己的用户。但是，更经常的是，站长使用 JavaScript 或者服务器端代码来判断用户的浏览器版本和平台，以提供针对性的网站版本。这种方法的问题是，字符串有可能无法获得，根据一些用户安全或者网络设置，浏览器不能把自己的 UA 字符串提供给服务器，这将导致浏览器探测脚本失败，最终用户无法看到他们想访问的站点。

此外，无论是通过安装扩展插件（例如 Firefox），还是作为默认功能的一部分（例如 Safari 和 Opera），所有的现代浏览器都允许用户修改和自定义它的 UA 字符串。事实上，在 8.02 之前的版本中，Opera 都默认定义自己是 IE，避免被许多只支持 IE 的网站屏蔽。

浏览器探测难免有错，最近新发布的 Oprera 10 测试版就是一个例子，这是第一款版本号达到两位数的浏览器。这原本是一个里程碑，值得庆贺。然而在它的 Beta 发布中，一些浏览器探测脚本无法正确判断 Opera 10 版本号的第二位，将版本 10 当成了版本 1。类似的问题非常普遍，最后 Opera 不得不改变了它的版本号，不再是 Opera10.00，而是 Opera9.80，最后一个单数字的版本（dev.opera.com/articles/view/ opera-uastring-changes）。

不幸的是，这种类似 2000 年千年版本的问题将等待所有尚未达到版本 10 的浏览器。除非我们不再采用浏览器探测，而采用更加标准的方法。

多版本带来了不断升级的成本和难题，当那些为早已消失的浏览器开发的私有脚本在现代浏览器和设备中无法工作时，网站拥有者是否应该投入更多金钱，开发第五、甚至第六个版本？如果没有预算制作这样的版本怎么办？大量的用户将被你的网站拒之门外。

即使当他们开始拥护 XHTML 和 CSS 等标准技术，那些刚从传统方法转型出来的设计师和开发者也经常抓不住要点。很多人不是使用标准来避免多个版本，而是为不同浏览器和平台建立不同的 CSS 版本（如图 1.1、图 1.2 所示）。

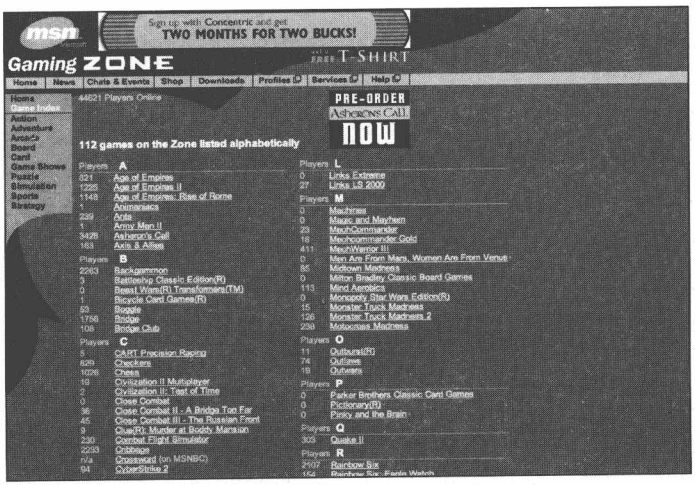


图 1.1　在 1999 年，MSN 游戏频道（zone.msn.com/blog.asp）使用了 3 个外联样式表文件和多个脚本（大多数是内联脚本，还包括大量的浏览器探测），但依然无法在最新的浏览器版本中正确显示。抛出多个版本的代码，往往解决不了问题

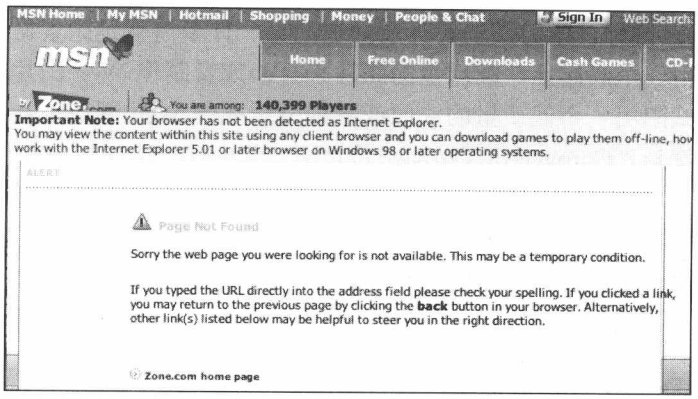


图 1.2　公平地说，上面这张截图是 4 年前的。到现在，尽管又写了更多代码，同样的页面还是更糟糕了。在微软推出第一款符合标准的浏览器 6 年之后，还是有部分微软网站依然不知道要使用 Web 标准设计

这样的做法既浪费时间，又浪费金钱。再多的版本都是不够用的。要制作那么多的版本，没有人能够承受，而这样做也还是无法解决问题，网站依然无法正常显示，用户依然可能无法访问。

1.4　垃圾代码来了

透过主要商业网站（例如 Amazon.com 和 eBay.com)的外表，检查他们内部混乱的非标准标记、错综复杂的脚本语言（通常含有失效的版本探测脚本），以及拙劣的 CSS 语法，你会感叹，这样的网站能在不同浏览器正常显示，真是一个奇迹。

通常，不符合标准的网站能在老式浏览器下正常显示是因为开发者购买了昂贵的网站发布软件，这些软件能自动生成多个版本，来适合不同浏览器和平台的差异。正如上文“版本问题”一节里提到的，Netscape 的 Navigator 和微软的 Internet Explore 最开始的四个版本不只是容忍非标准的标记和针对特定浏览器的私有代码，实际上，他们在鼓励随意发布和使用私有代码，想在一场构想拙劣的战争中获取更多的浏览器市场份额。

错误标记：最初，个人环境是无约束的

早期在学习计算机时，我们都学过一个短语“Garbage In, Garbage Out”（输入垃

圾，产出垃圾）。同样，学习图形设计所知道的第一件事情就是：素材质量决定最终产品的效果。用一个高起点、高质量的照片来打印或者制作 Web 图片，结果看起来都会很好。而试图用一个低质量的快照或者低分辨率的 Web 图像设计，最终结果往往很糟糕。这就是“Garbage In, Garbage Out”。

但老式浏览器却不遵守这样的原则。它们对输入宽松到荒谬的地步：它们接受所有残缺的标记、坏链接和 JavaScript 源文件，也不报错。并且在多数情况下，把它们当作正确的内容直接输出到页面显示。这导致前端设计师和开发者们养成了大量他们未察觉到的坏习惯，并影响到中间件和服务器端开发者，认为像 XHTML、CSS 和 JavaScript 这些技术是粗糙低级的。

不尊重工具的人就不大可能正确使用它。参考下面的 HTML 代码片段，这是从一个正在艰难的市场中竞争的公司电子商务站点上截取的：

```
<td width="100%"><ont face="verdana,helvetica,arial" size="+1"
color="#CCCC66"><span class="header"><b>Join now!</b></span>
</ont></td>
```

无意义的<ont>标签是为了抗议<font>标签而故意设置的——这个<font>标签贯穿整个网站重复达上千次，真是要感谢“高效”的发布工具！除了这个错误之外，这样的标记代码，看起来是不是很熟悉？类似的标记甚至也同样出现在你的站点里。要实现上面的代码效果，其实只需要下面一句代码：

```
<h3>Join now!</h3>
```

加上样式表里一个适当的规则，可以用上述更简单、更结构化的标记替代臃肿的、非标准的、不完整的标记。同时还能节约服务器和用户带宽，并在将来更容易转换到更灵活的使用语义标记的（可能包括能让机器读取的代码，例如后面将要讲到的微格式，microformat）站点上。上例中的电子商务站点还包含下列无效的 JavaScript 链接：

```
<script language=JavaScript1.1src=
"http://foo.com/Params.richmedia=yes&etc"></script>
```

这段代码的主要问题是，语言属性“language”和 source 标签连在了一起，没有空格分开。也就是说，浏览器被告知要去使用一个不存在的脚本语言

“JavaScript1.1src”。

在任何合理的检查方式下，这个站点应该报错。提示开发者所犯的错误，督促他们尽快修改。然而，直到最近，这个站点的 JavaScript 仍旧在主流的浏览器中“正常”工作，继续持续着不良发布的循环。我们已经不奇怪那些高级程序员为什么总是看不起前台的网页开发者了。

随着新的浏览器开始符合 Web 标准，它们开始越来越不能容忍不完整的代码和标记了。“输入垃圾，产出垃圾”的原则开始体现在浏览器世界，那些设计和开发网站的人们必须开始了解和掌握 Web 标准知识了。

1.5　代码分支可能影响你网站的长期健康

我知道许多公司都花费超过百万美元购买复杂而难用的内容管理系统（CMS）。这些 CMS 这么昂贵的原因之一就是能自动生成各种非标准的代码版本。而事实上，这些代码除了花费掉成堆的现金外，对网站并没有任何好处，真正的信息内容被埋没在大量非语义的标记中，使得可发现性极差（如图 1.3 所示）。那些拨号用户（以及智能手机用户）也不得不浪费大量的带宽在代码分支、层层嵌套的表格、空白像素等图片兼容技巧（hack）和过时无效的标记和属性上。

什么是代码分支（Code Forking）

代码是程序员写的用于创建软件产品、操作系统或者处理其他大量事务应用的语句。当一个项目有多个项目小组，代码就可能被“分支”到多个互不兼容的版本中，尤其是当每个开发组为了解决不同的问题，或者满足不同的进度要求的时候。总地来说，这种不一致和缺乏集中管理的情况很糟糕。

本书提及的“代码分支”指的是针对不同的非标准浏览器创建多个不兼容的版本，以应付不支持标准 JavaScript 和 DOM 的浏览器的实践（请参见上文“版本”问题）。

但是同时，这些多版本代码浪费了网站的带宽，甚至计算专家都难以估算。越大的站点浪费就越严重，大量的金钱被浪费在服务器调用、冗余、图片处理和不必要的复杂的代码和标记上。

图 1.3　快！在用错综复杂、非结构化上 HTML 标记创建的 Volkswagen.com 找到“Information on Volkswagen models and services”。提示：如果你觉得找起来很困难，那么对于读者和搜索引擎来说也一样（www.volkswagen.com）

精确的数字很难估算，但一般来说，如果一个站点精简 35%的代码，它也同样可以减少相同百分比的带宽成本。如果一个网站一年花费 2,500 美元的话，就可以节省 875 美元；如果花费 160 000 美元的话，就可以节约 56,000 美元。

来看看前面提到过的 Volkswagen.com（如图 1.4 所示）。尽管表面看上去是很简洁、典雅的设计，但页面的标记代码暴露了它落后，臃肿的表格布局、效率低下的脚本和大量内联的 CSS。注意，这不是一个业余网站：作为世界知名品牌的国际门户网站之一，每天被访问数百万次。如果把过时的 HTML 设计、hack 浪费的字节乘以这惊人的浏览量，结果大概是千兆带宽，更不用说服务器成本和巨大的管理费用。如果 Volkswagen 用轻便的 CSS 和语义 HTML 来替代原来臃肿的标记（如图 1.3、图 1.5 所示），那么将大大降低每个页面的带宽成本，反过来，也提高了公司的利润。

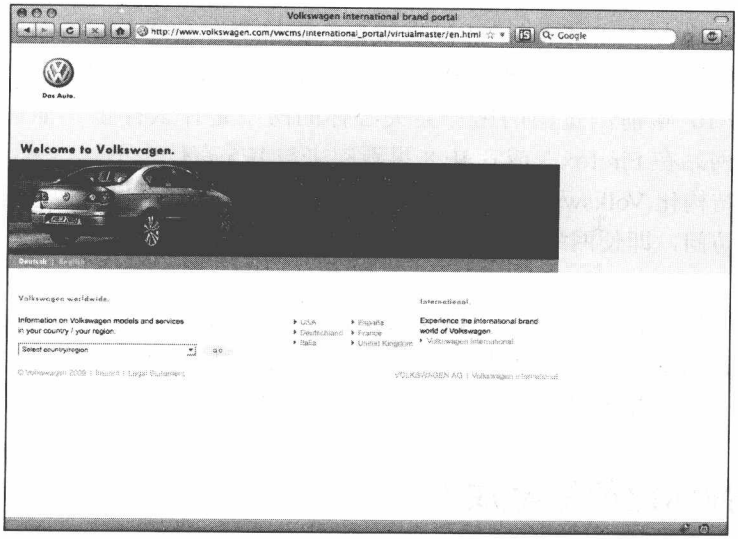


图 1.4　Volkswagen 网站看起来多么专业、漂亮、可以信赖

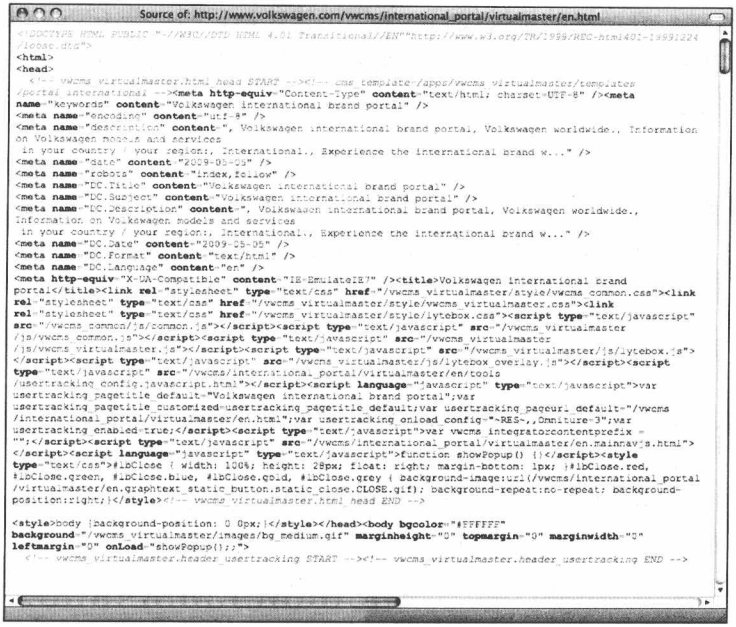


图 1.5　是很漂亮，但是当你翻开面纱，查看页面代码（在页面上点右键选择“查看源代码”）的时候，会发现用来创建这么简单的页面的标记和代码却错综复杂，难以理解

那么，为什么 Volkswagen 还没有改变？据推测，公司希望网站在那些不支持 CSS、已经灭绝的老浏览器里也能和现代浏览器里看起来一样。具有讽刺意味的是，没有一个用 10 年前浏览器的用户会关心你的网站是什么样子。他们也不会比较 Volkswagen 网站在 Firefox 3 或者 IE 8 里看起来与 IE5 有什么不同。采用本书介绍的技术来重新结构化 Volkswagen 网站，Volkswagen 可以确保他们的内容能够被所有设备和浏览器访问，即使网站细节不够到位。

作为一个如此聪明的公司，Volkswagen 也错失了这样的机会，你就可以知道，品牌经理们认为对待网络就像平面广告，网站设计团队认为需要保持向后兼容性，这样的陈旧心态有多顽固，甚至超过了理智、可用性或者他们自己的公司利润。

1.6 臃肿标记的隐藏成本

假设一个用传统方法制作的网页代码有 60KB，那么，替换掉过时的<font>标签，清除其他专有的、表现层的垃圾代码，采用结构化标记和一些 CSS 规则，同样的页面可以只有 30KB（在我们设计公司的实践中，常常可以用 22KB 甚至更少来替换 60KB，但是保守一点，至少可以节约 50%），看下面两个典型的情景。

T1 终结者

情景：一个提供小型业务或公共服务的网站，有自己的独立主机，经常有几百人在线，在清理掉表现层标记，用 XHTML 重新结构化页面后，代码缩减了一半，每月节约了 1000 美元的成本。

原理：在代码精简前，为这些用户提供服务需要租用 4 条 T1 线，每条线月租是 500 美元（一条 1.544MB/s 的 T1 线的正常成本），当裁剪文件尺寸 50%后，该网站发现只要 2 条 T1 线就可以有效提供服务，因此从这一处理上节约了 1000 美元/月。带宽节约后，同样将节约一些硬件费用，包括降低了大型数据中心的存储和电力成本，还有节约了机器以及机房的冷气，太好了，Web 标准还可以拯救地球！

更简单的标记，更快的页面显示，更小的服务器压力，购买、维护、升级更少的服务器。这种精简代码的做法对于现在多数动态的、基于数据库的内容网站（包括 blog）有立杆见影的效果。

带宽陷阱

情景：当商业主机托管变得流行起来以后，网站拥有者发现他们每月都在支付一种额外的费用：超额流量罚款。从几百甚至到几千美元。在采用 Web 标准、裁剪文件一半尺寸后，每月的账单恢复了正常，费用趋于合理。

原理：许多商业主机服务提供商每月给他们的用户分配一定流量的免费带宽。比如不超过 3GB。低于标准的时候，你每月只要付一般费用；超过了，则必须付更多。有时候会多很多。

在一个影响很差的案例中，因为超出限定流量，一家主机提供商要求独立设计师 AI Sacui 为他的非营利性网站 Nosepilot.com 支付 16 000 美元额外费用。这是一个极端的例子，虽然 Sacui 最后没有支付这笔费用，但经历了漫长的司法诉讼，因为那家主机提供商改变了服务条款而没有通知他。谁愿意冒高额账单风险或者去跟一个反复无常的主机服务提供商打官司呢？

当然，不是每一个主机提供商对待流量超额都采取高额收费的措施。不论你的站点大还是小，是百万人访问还是只有少数团体会员访问，你的文件越小，占用带宽就越小。顺便说一句，如果你拥有一个热门站点，最好选择那些流量不限制的主机提供商。

简洁代码对比压缩代码

我在一次进行 Web 标准方面的演讲时，一位听众提出不同意见，他说干净的、结构良好的代码标记所节约的带宽对于采用压缩 HTML 技术的公司来说是微不足道的。

的确，除了精简代码这种方法，你还可以通过一些服务器端技术来压缩代码标记。例如：Apache Web 服务器加载的 mod_zip 模块可以在服务器端压缩你的 HTML，在用户浏览器端重新把它解开。

那位开发者举了个例子：如果 Amazon.com 在垃圾代码上浪费了 40KB，但是用 mod_zip 压缩掉 20KB，Amazon 为垃圾代码付出的费用会比演讲中提到的要少。

事实上，Amazon 没有使用 mod_zip，很少有商业网站使用这个工具，可能是因为在发送压缩页面之前需要额外加载的缘故。但是，从另一个角度说，越小的文件压缩以后也越小。如果你从每页 80KB 压缩到 40KB 节约了费用，那么你为什么不先从 80KB 精简代码到 40KB，然后再压缩到 20KB 以节约更多费用？每一页节约的数量看起来很小，但这个值是累加的，长时间下来，可以大大减少运营费用。

干净的、结构良好的标记的好处不仅是节约带宽，而且还会得到会计和客户的赞扬。这对于那些采用压缩 HTML 技术的网站也同样具有现实意义。

1.7 向前兼容（backward-compatibility）是一个谎言

对开发者来说，什么是“向前兼容”？如果你问他们，他们会说就是“支持我们所有的用户”。谁会对这样的观点有异议？

事实上，“向前兼容”就意味着使用非标准的、私有的标记和代码来保证每一位访问者有相同的体验，不管他们喜欢 IE2 还是 Opera 10。“向前兼容”的理论听起来很伟大，但实际上这样做成本太高，实践中也完全是一个谎言。事实的真相是，没有什么真正的向前兼容，总是有一个切断点。例如，不论 Mosaic（第一个可视化浏览器）还是 Netscape1.0，都不支持基于表格设计的 HTML。需要再解释清楚一点吗？好，那些用“古老”浏览器看网站的人们不可能和用稍微新一点的浏览器浏览的用户获得同样的视觉体验，比如 Netscape1.1 或者 MSIE2.0。

那些坚持“向前兼容”观点的开发者和客户不可避免地要设定一个“基线浏览器”。例如 IE6 以上版本。网站将不支持更老版本的浏览器（IE 5.5 的用户就遭殃了）。开发者为了履行他们支持基线浏览器的承诺，针对不同版本的浏览器细节，用非标准的处理方式建立多个版本，写多个脚本来适应不同的浏览器，通过 UA 字符串检测用户的浏览器并运行不同的代码以使页面看起来最好。正是因为这样做，便使得页面尺寸不断加大，服务器压力不断增加，从而保证“改版-过期”的怪圈继续持续下去。

屏蔽用户不利于业务发展

一些公司愿意多花点钱在向前兼容上，以保证在老浏览器上能和新浏览器一样正确看到他们的网站；另外一些公司则决定只支持一种浏览器。在减少成本的目标压力下，越来越多的网站只针对 IE 浏览器设计，甚至在某些情况下只针对 Windows 平台，因此他们失去了 15%～25%潜在的访问者和客户（如图 1.6～图 1.10 所示）。

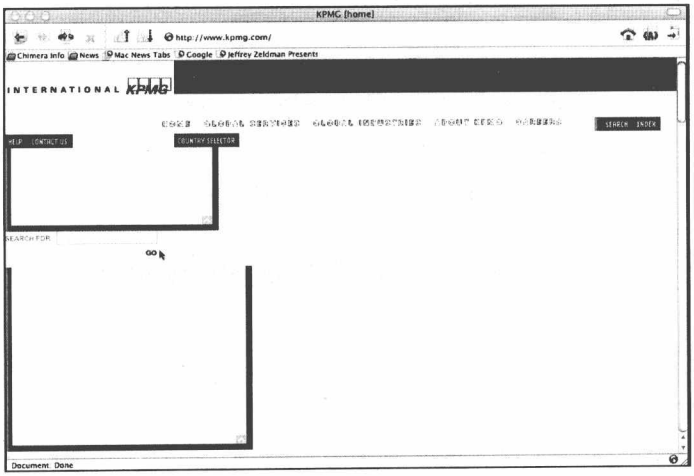


图 1.6　KPMG 的首页在 Netscape 浏览器中的截图，大约在 2003 年。能找到它的导航吗？这要感谢只支持 IE 的代码

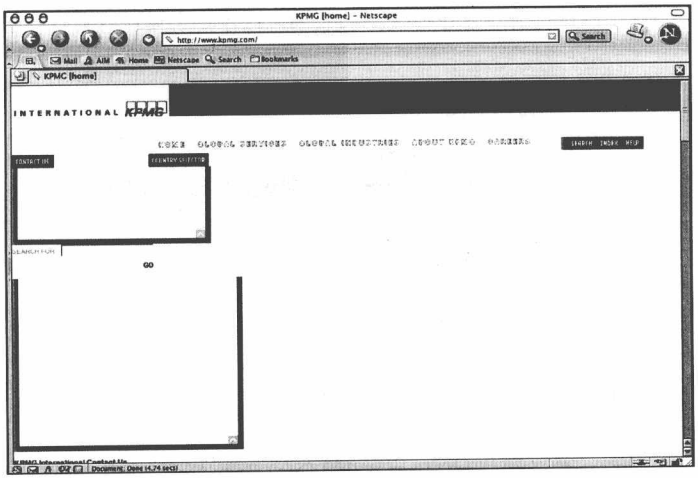


图 1.7　网站同样不能在 Netscape 7 中正确显示

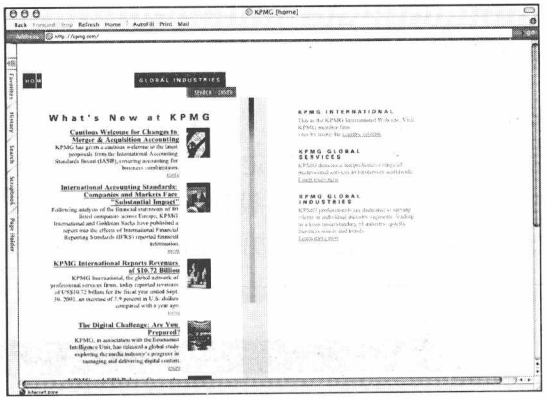


图 1.8　好吧，如果网站只支持 IE，那么在 IE5/MAC 下能看吗？显然也不行

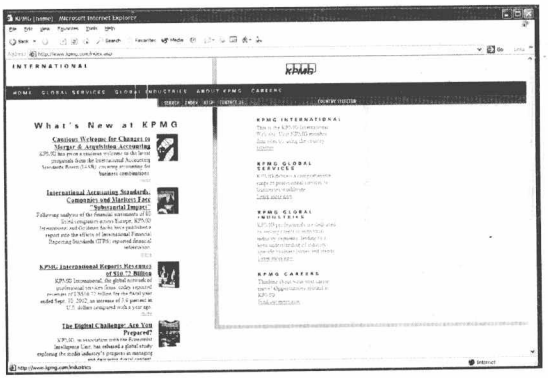


图 1.9　网站在 IE6/Windows 下的效果，这才是它设计的真正样子

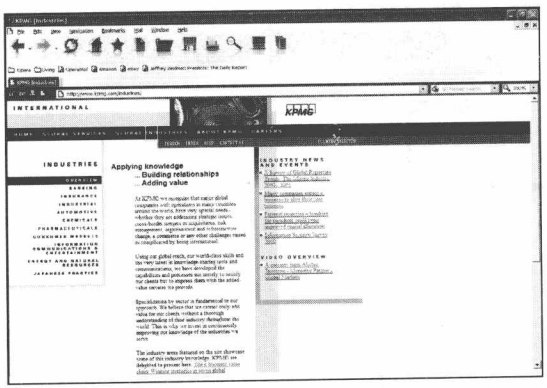


图 1.10　还好，网站能在 Opera 7/Windows 下显示（但必须在 Opera 定义自己为 IE 的情况下，如果定义自己为 Opera，则同样会显示不正确）

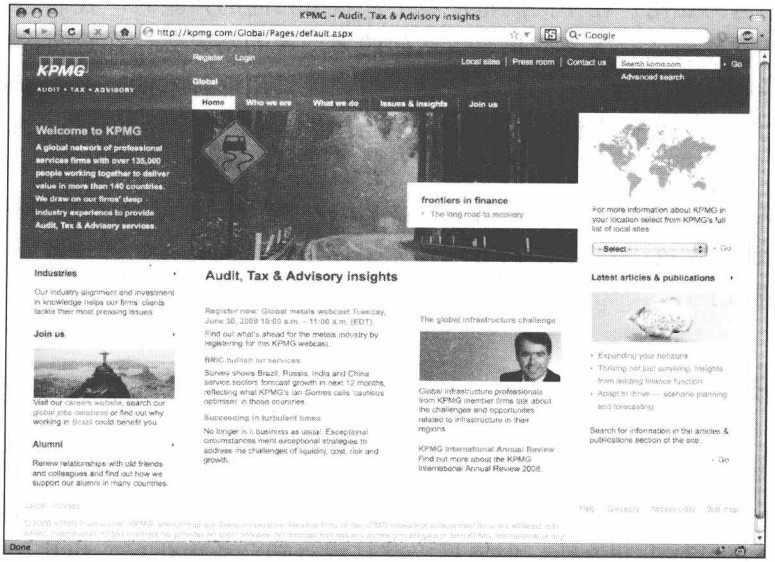


图 1.11　在对网站进行重构之后，今天的 KPMG 网站已经能在多个浏览器和操作系统中显示正确。虽然还有很多可以改善之处，但是使用非 IE-only 的标记就已经达到了现在的效果

我不能理解为什么有些公司宁愿放弃 1/4 的潜在客户，所有理性的投资者和必须为公众服务的非营利组织都会惊诧于这种目光短浅的做法。根据最近的调查统计报告（www.internetworldstats.com），截止到 2002 年 5 月有超过 16 亿的因特网用户，你可以自己计算损失。

即使你真的不介意你的站点损失 25%的访问用户，继续坚持“IE-only”方法也是不明智的，因为你不能保证 IE（或者桌面浏览器）在 Web 领域能持续保持主导优势。举个例子：在写这本书的时候，Firefox 正在不断蚕食 IE 的市场份额；另一方面，越来越多的用户通过手机浏览器来访问网络(例如基于 Webkit 的 Opera mini 浏览器)。当这种情况变得普遍时，设计观点就必须跟着改变。仅仅为 20 世纪个人桌面浏览器所进行的设计会变得越来越落后，越来越不明智。

此外，正如本书所展示的一样，使用 Web 标准就可能设计出适用于所有浏览器和设备的站点，简单而快捷（如图 1.11 所示）。

当我们努力让用户在不同浏览环境下都获得相同的体验时，我们忘记了网站真正的本意：网站对所有人都应该是可访问的多层次的富媒体。在短暂的网络繁荣期间，当设计师和程序员们为了赶上产品需求，而采用非标准的、多浏览器版本方法建立网站的时候，我们忘记了网站的本意，这也是导致众多网站过期的原因。

但是就在你读到这些文字的时候，落后的网站开发方式已经快要到尽头了，伴随的是无数网站跟着它一起走向过期。如果你是网站拥有者、管理者或者创建者，警钟已经敲响了！

1.8 治疗

“一次创建，随处发布”，这是 Web 标准的许诺，也是一个非常令人向往的概念。今天它已经实现，我们将在这本书里探讨这种实现方法。尽管今天主流浏览器终于都开始支持这些标准和方法，但信息还没有传达到许多正在设计和开发的设计师和开发者那里，大量的新站点依旧建立在非标准的标记和代码之上。希望本书能改变这种状况。

经过 W3C 会员和其他标准组织的制订，像 CSS，XHTML，标准 JavaScript 和 W3C DOM 等技术已经被 2000 年之后的所有浏览器支持，并可以设计和做到如下的内容：

- 在图形桌面浏览器中实现更精确的布局、定位和排版，并允许用户按他们的需求修改表现方式；
- 可以开发工作在多浏览器和平台上的复杂交互行为；
- 遵守可用性法律和指导规范，却无须牺牲美观、表现和精确；
- 以前重新设计网站需要几天或者几星期，现在只需要几小时，从而减少成本和避免无聊的工作；
- 支持多种浏览器，却不需要考虑多版本的开发和成本，只需要很少或根本就不需要代码分支；

- 支持非传统的设备和新设备，从无线设备、可以上网的智能手机到盲人阅读器、屏幕阅读器等残疾人士使用的设备，不需要再开发特殊版本，更不需要付出费用；

- 为任何网页提交适合打印的版本，不需要建立通常的“专门打印页”或者依赖昂贵的专业发布系统来建立类似的版本；

- 从过去混乱的标记转变为现在或者未来真正的语义 Web；

- 确保这样设计出来的网站不但能正确工作在现代标准兼容的浏览器或平台上，也可以正确显示在老浏览器中，即使它们的表现并不完全相同；

- 保证这样设计的站点能继续工作在将来的浏览器和设备上，包括那些还没有发明和仍在想像中的设备，这是向后兼容的许诺；

- 以及更多，就像本书将展示的……

在开始学习如何使用标准来达到这些目标前，我们必须先回头检验一下老的方法，找出老技术是如何造成网站不断过期的原因，在第 2 章“根据标准设计和制作”将展示所有这些内容。

不想重温历史？那就直接跳到第 3 章去感受一下扑面而来的新鲜空气吧。

第 2 章

根据标准设计和建造

在创建标准和浏览器支持标准以前，设计师和开发者是如何建造网站的呢？用他们能用的任何办法。看看下面的 Suck.com（一个最早独立发行的、幽默的 Web 期刊，如图 2.1 所示）在每天的头版，读者最关注的位置，用一种犀利的、疯狂的写作风格，巧妙而公正地展现自己的观点。现在，这种做法随处可见，每个人都在自己的博客上更新头条故事。但在 20 世纪 90 年代中期 Suck 刚成立的时候，大多数网站都把自己的内容深藏在漂亮的欢迎页、导航条，以及文章列表页之后。

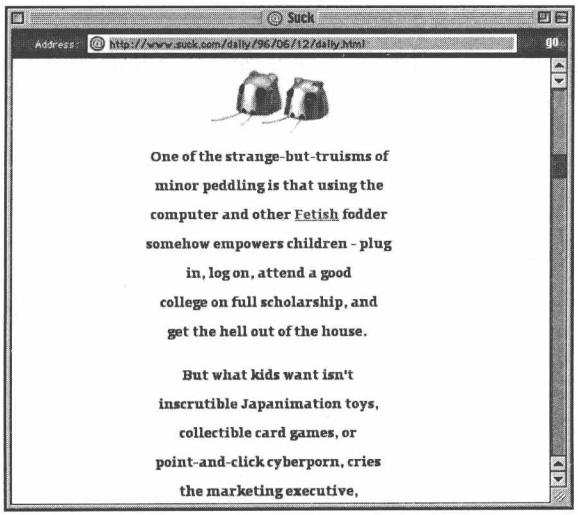


图 2.1　Suck，一个简单明了的商业网站先驱者（www.suck.com）

在大多数商业网站文字含混隐喻的时代，Suck 以勇往直前的文字、不夸张的比喻让人耳目一新。同样，在大多数商业网站过度设计的时代，Suck 极简的网站外观显得尤为突出。

那个时候，自学成才的 HTML 设计师和系统管理员练习着使用金属边框和高技巧的发光效果，把它们和毫无设计的混乱信息堆砌在一起，许多网站的建设者都利用 Netscape 1.1 提供的各种原始特性进行布局，从背景图片重复到用专有的 标签居中等等，还有一些站点到现在还在展示这些技巧（如图 2.2 所示）。因此，在当时网站热衷于做加法的普通情形下，Suck 大胆的减法显得格外突出。

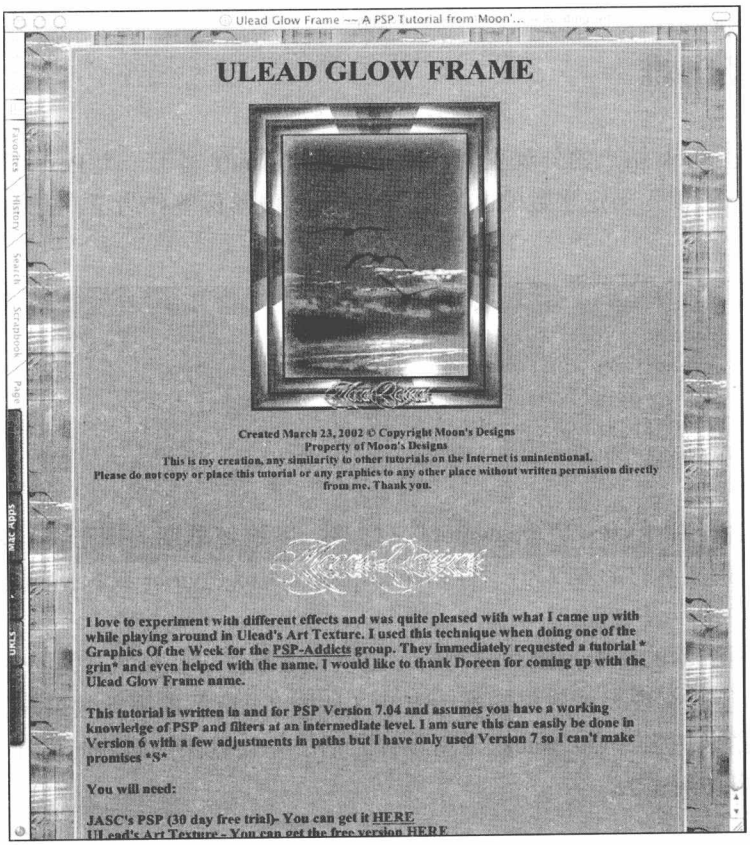


图 2.2 “Moon’s Design”网站的 Ulead 发光 Frame 的教程（moonsdesigns.Com/tutorisls/frames/glows.html）是 20 世纪 90 年代中期的典型 Web 设计。内容居中放在中间的 HTML 表格中，表格使用了一个重复平铺的背景，包含这个表格的页面还用了另一个平铺的图片作为背景

为了达到与众不同的以内容为主、外观简洁的目标，Suck 网站创建者 Carl Steadman 和 Joey Anuff 不得不放弃传统方法。那时，HTML 缺乏设计编辑工具，主要原因是：当发明 Web 的物理学家 Tim Berners-Lee 构思时，HTML 是一个结构化的标记语言（http://www.w3.org/MarkUp/html-spec），源自 SGML，而不是一个类似 Adobe 的 PostScript 或者标准样式表的那样的设计语言（那时，CSS 尚未被 W3C 批准为标准；一旦批准，将至少需要 4 年时间，来使浏览器支持标准及完善对标准的支持）。

那么，Steadman 和 Anuff 是如何控制站点的表现呢？他们利用创造力、发明和很多卷数字管道胶带创造性地实现了目标。

2.1 历经磨难

为了创建 Suck 站点的外观表现，Steadman 和 Anuff 写了一个 Perl 脚本来计算文本中的字符，每读入一定字数的文本插入一个<p>段落标签作为下一行的开始：

```
<p>One of the strange-but-truisms of
<p>minor peddling is that using the
<p>computer and other Fetish fodder
<p>somehow empowers children - plug
<p>in, log on, attend a good
<p>college on full scholarship, and
<p>get the hell out of the house.
```

整个内容外加上<tt>（“typewriter”）标签，以迫使早期的图形浏览器（主要是 Netscape1.1）把文本的字体改为一种叫 monospace（类似 Courier 或者 Monaco）的字体。

这样运用基本的排版控制和强制的标题模拟实现了以上结果，这种 HTML 处理方法是 1995 年能实现设计效果的唯一方法（如图 2.1 所示的视觉效果来自 1996 年，在设计技术稍微加强一点，Suck 重新设计之后——依然保持简洁的设计。原始的设计已经不能再访问了）。

网页设计师们广泛采用着类似的迫使 HTML 符合排版设计效果的创造性方法，并在早期的像 Lynda Weinman 和 David Siegel 这样的作者写的 Web 设计经典教程中被广为传授。HTML 的创造者面对这样的 HTML 变形只能瞠目结舌，但是当客户强

烈要求好看的 Web 界面时，设计师别无选择。

在今天，许多设计师依旧在工作中使用这些方法，许多设计类的书籍依旧教这些过时——而且在当前的 Web 上效率低下的技术。一本在其他方面很优秀的网页设计书籍甚至直接建议它的读者使用 Font 标签和“HTML 脚本”来控制字体样式。事实上，Font 标签早就被反对使用了（W3C 说：请不要使用这种过时垃圾），HTML 也不是一种脚本语言。但是像这样错误的、毫无意义的建议广泛地存在于网页设计书籍中，因此，愚昧和无知一直在 Web 开发领域中延续。

2.2　无标准时代的设计成本

正因为对 HTML 创造性的控制，Suck 达到了与众不同的外观，但是花费了两倍的成本：网站拒绝了一些访问者，并且对它的创造者来说，很难升级。

如果使用早期的 Mom-and-Pop 屏幕阅读机（一种为视障人士设计的有声阅读浏览器）大声朗读 Suck 的文本，由于有不断的段落标签，会在几个单词后就停顿一会儿，例如使下面精彩的社论不停地中断：

```
One of the strange-but-truisms of ... [讨厌的暂停]]
minor peddling is that using the ... [讨厌的暂停]]
computer and other Fetish fodder ... [讨厌的暂停]]
somehow empowers children—plug ... [讨厌的暂停]]
in, log on, attend a good ... [讨厌的暂停]]
college on full scholarship, and ... [讨厌的暂停]]
get the hell out of the house.
```

在理想条件下分解这些话都比较困难，更何况 Suck 复杂的句子结构被没有语义的段落标记分割后变得更加难以理解。这些声音间断给屏幕阅读机用户带来无法克服的理解问题，对他们来说这个站点是不可用的。

为了实现 Suck 在图形浏览器中的布局所使用的 HTML 技巧使得部分用户无法访问，同样它们也给 Suck 的创建者在每次升级的时候制造了障碍。

因为他们的设计依赖于 Perl 和 HTML 处理，所以不可能采用模板。每天 Suck

必须花费几小时的工作来更新网站。当 Suck 网站迅速流行，最后成为了一个社团的门户站点后，网站创建者被迫要雇用一个创作团队，而不仅仅是几个人。Suck 烦琐的手工更新根本满足不了它每天都要发布的需求。

这些困难只会发生在它们那个时代，它们只是早期商业站点的一个轶事。在欣赏并思考先驱者的灵活设计时，我们不禁莞尔：Web 发展究竟为什么如此曲折？但是，目前大多数的商业站点仍旧不理会 Web 标准的出现，奇怪地依赖于劳动密集型工作方式，并继续为处理那些方法造成的问题而痛苦。这些旧的实践应用得如此广泛，以至于许多设计师和开发者从未停下来想一想。

2.3 时髦的站点，古老的方法

从 1995 年跳到 2003 年，我们先来看一个电影记录片，名字叫“Hooked: The Legend of Demetrius“Hook” Mitchell（吸毒成瘾：德米特里厄斯-米切尔传奇）”。也许这部记录片不是你喜欢的十大影片之一，但它因为其独立的特色参加过 20 个以上的电影节，并获得了众多奖项。作为影片的营销手段，发行商建立了一个专门网站来帮助他们更好地推广这部电影（www.hookmitchell.com）（如图 2.3 所示）。这是一个精心设计的网站，但却采用了传统的表格布局技术，这就是典型的现代网站使用过时技术的案例。

记录片讲述了一个外号“Hook”的业余篮球运动员的传奇故事。他出生贫寒，却成为街头篮球中的天才，最后因毒品陨落。虽然该网站设计效果有效地表现了影片的特色，但在漂亮的外表下，堆满了“污垢”，大量表格的标记代码和 Gif 间隔控制图片导致了以下缺点：

- 如果制片人需要对网站进行任何更改，甚至小到增加一个链接，则整个网页完全报废，需要重写。改变任何图片的尺寸也将导致 HTML 表格布局变形（如图 2.4 所示）。

- 因此，即使最小的改变，网站也将显著提高成本。图片必须重新设计、切割和优化，表格标记必须重新写。因为用 JavaScript 控制链接翻转效果，如果像增加一个新的链接这样的简单任务都需要几个小时的修改，显然这意味着问题。

图 2.3　记录片网站（www.hookmitchell.com）创建于 2003 年，却采用了 1998 年的标记技术，外表夺目，但代码陈旧

- 网站拒绝了许多潜在的访问者。因为这样的实现方式，网站无法支持那些使用屏幕阅读器的用户、文本浏览器的用户，以及手机用户。如果使用非图形浏览器（如图 2.5 所示），页面看起来就像这样：

```
spacer.gif
spacer.gif spacer.gif
spacer.gif
spacer.gif spacer.gif
[And so forth...]
spacer.gif spacer.gif
bottom.jpg
```

在这个网站上没有一点为非图形环境准备的内容，甚至站点的搜索结果也没有一点对记录片推广有用的信息（如图 2.6 所示）。“spacer.gif”是制片人希望传达给用户的信息？我们对此表示怀疑。

我不是说图形不好，或者复杂的设计不好，恰恰相反，图形是非常重要的、美观是需要的。Hooked 网站的外观设计确实是一个良好的美观体验，但是这种体验不应该导致部分用户无法访问。

图 2.4　还是这个网站，我们将表格的边框显示出来，揭示其排版方法和其潜在的维护难题

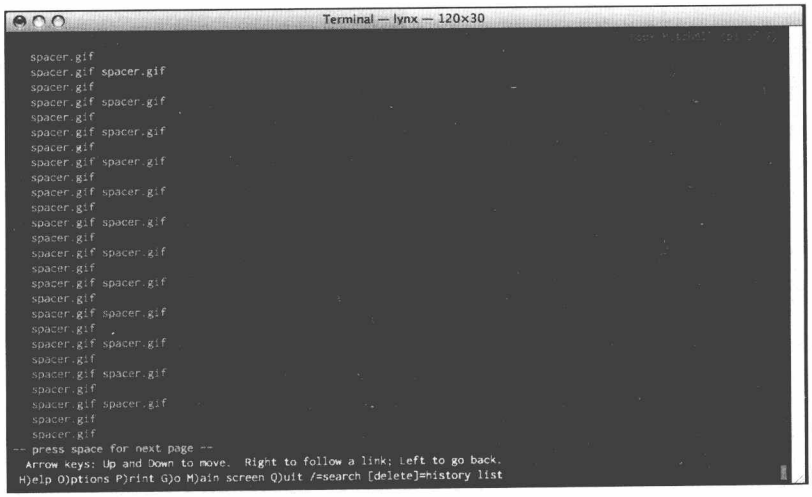


图 2.5　在文本浏览器中，例如 Lynx（http: //lynx. isc. Org），记录片网站没有提供任何信息。图片没有提供 alt 属性，所以将直接显示文件名而不是可读的标题给用户。要看看你的网站在非图形环境下如何表现，可以用 Lynx 或者 JAWS 那样的屏幕阅读器（或者 OS X 的 VoiceOver）测试一下。这里有一个免费的在线 Lynx 模拟器（www. delorie.com/web/lynxview.html），不过只能给测试网站的站长使用

图 2.6　因为页面上除了没有意义的文件名之外没有任何内容，所以搜索引擎的结果是一些不知所以的文件名，它的排名无疑也会受到影响。如果你想搜索记录片 hook mitchell，看到 spacer.gif、bottom.jpg 你会去访问吗？我不这么认为

虽然视觉效果很有吸引力，Hooked 网站还是典型的传统方法布局：代码暴露了它的不易访问性和对低效昂贵的发布技术的依赖。幸运的是，大部分网站已经放弃了传统方法而采用更标准的技术。但是尽管我们已经取得了很大进展，依然有一些网站还在采用传统技术建站（参见下面的“酒店没有房间了”小节），当客户需要对那些网站做一些改变的时候，它们精心设计的布局就乱了。要么花费客户的钱，要么增加建站者自己的成本，又或者实现的是一个视觉混乱的解决方案，损害着网站的可信性和可用性。（例如：我们可以在 Hooked 网站的顶部增加三个并排的链接以提供新的快速访问渠道，但是这个增加将破坏网站严格控制的外观，并使用户产生困扰）。

酒店没有房间了

可悲的是，非标准的建站技术，例如 Hook Mitchell 网站所用的那些，至今仍在使用。

来看另外一个案例，我们来到当今希尔顿（Hilton）酒店网站的搜索页面（www.hilton.com/en/hi/hotels/search/index.jhtml）。想订个房间？自求好运吧。在 CSS

技术发布超过 10 年和浏览器都符合标准的年代，希尔顿酒店的搜索页面仍然用无语义的表格布局，依然使用过时的<font>标签和 1995 年的“GIF 透明图片控制间隔”办法，以及其他陈旧方法。即使所有非结构标记没有浪费大量的带宽，庞大的嵌入在每个页面里（而不是外联）的 JavaScript 脚本也使页面迅速膨胀，减缓了页面显示速度。最糟糕的是，找到内容成为了一个可访问性的噩梦，图片 alt 属性的缺失使声音阅读器胡言乱语。如果 JavaScript 被禁用，页面上很多区域将停止工作。因此，如果你的浏览器不能符合希尔顿酒店网站的严格要求，就得换个酒店订房了。

尽管可能是最差实践的样例，希尔顿酒店的搜索引擎并不孤独，还有更多又大又使用过时技术的网站漫游在寒冷的网络空间（如图 2.7、图 2.8 所示）。我们有理由倡导聪明的、基于 Web 标准的技术，这样，可以把那些错误留在过去。

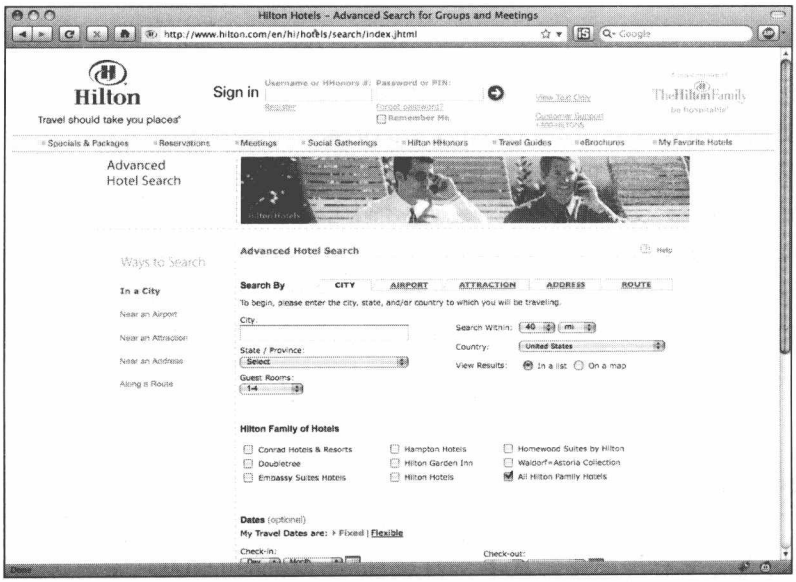


图 2.7　希尔顿酒店搜索页面（www.hilton.com/en/hi/hotels/search/index.jhtml），可以从 Hiltom.com 直接访问到。这原本是一个简单的表单，却因为过度的表现标记而“超重”：透明 GIF 图片、嵌套的表格和成堆的过时标记。快去预订吧，“1999 年”会为你亮起一盏灯

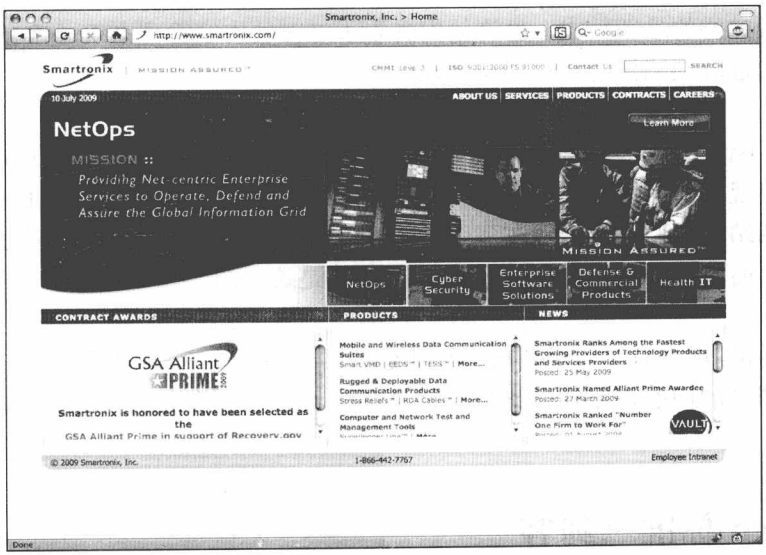


图 2.8　在 2009 年，Smartronix 公司（www.smartronix.com）获得 950 万美元（www.bizjournals.com/washington/stories/2009/07/06/daily78.html）来管理 recovery.gov 网站（一个以跟踪政府预算和预防浪费为目的的网站）的重新设计。从 Smartronix 的无效标记来判断，大部分的预算都浪费在带宽上了

反击！

如果你厌倦了那些过时的、非基于标准的改版，因为这会使网站用户更加难以访问，也会阻碍 Web 标准为客户所接受的进程，那么可访问性专家和标准专家 Joe Clark（这个名字会多次出现在本书中），已经建立起一个名为“失败的改版运动”的网页（参见 http://blog.fawny.org/category/web-standards/failed-redesigns）来传播标准，同时也让那些还在使用 1999 年建站技术建站或改版的网站创建者感到羞愧。

对上面提到的网站来说，它们的创建者现在可能已经知道，以桌面为中心的布局已经跟不上发展，网站可能在一般的浏览器下看起来正常，但一旦访问环境改变超出设计边界，这些网站将无法访问，至少，会把潜在的客户屏蔽在外。

那些设计师还在使用过时的技术，把网站的内容淹没在大堆标记中，把用户和一些浏览器关在门外。他们为什么会这么做？可能是所受教育的问题：也许他们用

的是他们最熟悉的最新技术，也许他们受限于过时的内容管理系统（CMS），或者一个针对老的过时浏览器的客户端编辑软件。

但是，不论出于何种原因，这些问题不仅仅出现在希尔顿酒店或者 Hook 网站上。它们也会出现在那些针对一些特定浏览器做特定代码，而不是采用 Web 标准提供普适访问的网站；它们也会出现在那些页面标记非结构化的网站，他们强迫 HTML 来实现布局，而不管 HTML 的设计初衷本来就不是这样的。

幸运的是，现在有了一个新的设计和建造网站的方法——一种解决所有用传统方法创建的网站的问题，不需要破坏界面美观，不会损害商业利益的方法，那就是：Web 标准。

2.4 Web 标准三剑客

图 2.9 显示了 Web 标准是如何来解决我们前面讨论过的问题的，它把网页拆分成三个独立组件：结构（Structure）、表现（Presentation）和行为（Behavior）。

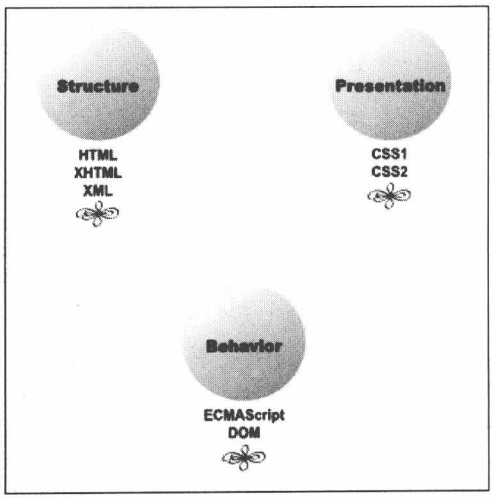


图 2.9　结构、表现和行为：Web 标准将任何网页都分成这三个组件

2.4.1　结构

一种标记语言（HTML 4.01: www.w3.org/TR/html401; XHTML 1.0: www.w3.org/TR/xhtml1），包含文本数据，并按它的结构（语义）含义进行分类，例如：标题、二级标题、段落、编号列表、定义列表，等等。

在页面上，下面的文本是定义列表<dl>中的一部分。“Structure”被<dt>标记定义为小标题，内容片段将被<dd>定义为内容数据：

```
<dl>
<dt>Structure</dt>
<dd>A <em>markup language</em> (<a href="http:// www.w3.org/
TR/html401">HTML 4.01</a>; <a href="http://www.w3.org/TR/
xhtml1">XHTML 1.0</a>) contains text data formatted according
to its structural (semantic) meaning: headline, secondary
headline, paragraph, numbered list, definition list, and so
on.</dd>
<dd>On the web, this text would likely be part of a definition
list &lt;dl&gt;. The subhead, &#8220;Structure,&#8221; would be
marked up as a definition title. The paragraphs you&#8217;re
now reading would be wrapped in definition data tags.
</dd>
</dl>
```

或者，为了更好地表示语义，我们可以用两个段落<p>和一个小标题<h3>元素来实现：

```
<h3>Structure</h3>
<p>A <em>markup language</em> (<a href="http:// www.w3.org/
TR/html401">HTML 4.01</a>; <a href="http://www.w3.org/TR/
xhtml1">XHTML 1.0</a>) contains text data formatted according
to its structural meaning: headline, secondary headline,
paragraph, numbered list, definition list, and so on.</p>
<p>On the web, this text would likely be part of a definition
list &lt;dl&gt;. The subhead, &#8220;Structure,&#8221; would
be marked up as a definition title. The paragraphs you&#8217;re
now reading would be wrapped in definition data tags.</p>
```

’的含义

如果你是那 1/5 确实花时间读了这个 XHTML 例子的读者之一，你可能想知道“&#8217”是什么意思。非常简单，这是标准 Unicode 转义字符串的撇号“’”。而“&#8220”，是 Unicode 中双引号的开始，“&#8821”则是双引号的关闭。

XML（http://www.w3.org/TR/REC-Xml），可扩展标记语言，能够比 XHTML 提供更多的选择。但现在我们暂时只使用 XHTML 1.0，XHTML 1.0 是一种过渡性标记语言，也是自从 2000 年以来 W3C 一项稳定的标准，就像 HTML 一样可以在当前几乎所有浏览器和网络设备上工作。正确书写（指没有错误，没有非法标记和属性），符合标准的 XHTML 或者 HTML 标记很容易移植，它可以“工作”在浏览器、屏幕阅读器、文本浏览器及未来的无线设备中。

这种标记也可以包含设计师认为必要的其他结构，例如，内容和导航可能被这样标记：

```
<div id="content">[Your content here.]</div>
<ul id="navigation">[Your navigational menu here.]</ul>
```

这种标记语言也能包含嵌入对象，包括图片、Flash 或者视频片段，还有相应的标签和属性，这些属性可以为那些不能在浏览器环境里看到这些对象的访问者提供等价的文本信息。

校验？语义？是什么意思？

- 页面校验通过，意思是它包含的代码没有错误（例如：忘记关闭标签）；没有不合法的标签和属性（例如：表格中的高度属性“height”，在 XHTML 中是不合法的）。校验可以通过在线的免费软件进行（http://validator.w3.org）。

- 标记是“语义化”的，意思是标记和它们所要表达的含义是相近或者一致的。例如，定义一个标题用 h1，因为它是页面上最重要的标题，而不是因为需要这些字看起来大一些。在本书中，我有时候说“结构标记”，意思就是“有语义的标记”（结构标记的说法源自每个 Web 文档都有一个类似大纲的结构）。

一个网页通过校验，并不一定符合语义。例如，一个用表格布局 HTML 页面，只要它的代码没有错误、没有不合法的标记和属性，这个网页就能够通过校验，但网页也可能是符合语义但没通过校验的。通常，我们说符合标准的设计是指既通过校验、又符合语义的网页。

2.4.2　表现

表现语言（CSS 2.1: www.w3.org/TR/CSS21; CSS 3: www.w3.org/Style/CSS/current-work）用来为网页提供格式，控制字体、布局、颜色，等等。CSS 能够代替旧式的 HTML 表格布局。在任何情况下，CSS 都可以替换非标准的 Font 标签和那些浪费带宽的、过时的垃圾代码，就像这样：

```
<td bgcolor="#FFCC00" align="left" valign="top"><br><br><br> </td>
```

这样的垃圾代码，我们只要为元素的边框属性加一个样式规则就能实现。

因为表现与结构相分离，所以可以实现改变一部分而不影响另一部分。例如：你可以在多个页面使用同样的布局，或者改变文本和链接而不破坏布局。你或你的客户可以在任何时候自由地改变 XHTML，不需要担心布局被破坏，因为文本就是文本，它不像设计语言那样要负担双重责任。

同样地，你可以改变布局而不改变标记。读者抱怨你的站点字号太小吗？改变全局样式表中的一个规则，整个网站将立刻改变。需要一个打印页版本吗？写一个打印样式表，不论它们在屏幕上看起来是什么样，你的页面都可以打印得非常漂亮。

2.4.3　行为

一个标准的对象模型（W3C DOM 参见 http://www.w3.org/DOM）与 CSS、XHTML 和 ECMAScript 262（http://www.ecma.-international.org/publications/standards/Ecma-262.htm，JavaScript 的标准版本）共同工作，使你创建出可以运行在多平台和浏览器上的交互行为和效果，不再需要为特定浏览器编写脚本。

2.5 实际应用

如果 Suck 网站现在才创建，那么像（X）HTML 和 CSS 之类的 Web 标准将可以使他们专心投入到写作中去。用一个基本的（X）HTML 模板来完成文档结构，用 CSS 将控制外观表现，不再需要额外的设计工作。段落标签将明确表示出段落的起点和结束，不需要在每一行之间强行进行间断（CSS 完全可以做好这些）。

在任何一个类似 IE、Firefox、Opera 和 Safari 的图形浏览器中，样式表将保证 Suck 站点看起来与设计师的意图一致。语义化的标记将保障 Suck 的内容不仅可以显示在浏览器中，还可以显示在移动设备、屏幕阅读器和文本浏览器中，不再需要设置假的段落间隔或那些类似的当成设计工具来用的标记。

作为一个以内容为核心的网站，Suck.com 是用 XHTML/CSS 技术实现其内容和风格的最佳网站候选者：用 CSS 控制风格布局，用 XHTML 控制内容结构。

同样，Hook 网站也可以用 CSS 来实现它的布局，既节约了带宽，又可以避免修改部分内容就要重新设计整个页面的烦恼。网站设计师可以使用 CSS 的背景属性来定位它的主要图片（比如一张 JPEG 或者 PNG 图片），从而替代一大堆的切片图片；也可以用任何一种经过时间考验的 CSS 定位方法，来轻松实现一个或多个菜单的图形变换效果。

除了将表现逻辑从标记中剥离出来并用 CSS 实现之外，设计师还可以使用 XHTML 或者 HTML 或者说明属性（例如 alt 和 title 属性）来确保网站内容能够面向所有用户，而不是让它对一些用户来说毫无含义（如图 2.5 所示）。一个样式表就可以控制无数的网页风格、降低维护成本以及未来设计修改的成本。

幸运的是，自从本书第 1 版出版以来，无数的网站已经接受这了些技术。看看马里兰州艺术学院网站（MICA，http://www.mica.edu/），最近刚由 Happy Cog 公司改版，在引人注目的设计下，你会发现代码同样有吸引力。由于采用了 Web 标准技术，MICA 的网站内容能够在任何地方展现（如图 2.10、图 2.11 所示）。

图 2.10　马里兰州艺术学院 MICA 的主页（www.mica.edu），把学生们引人注目的作品放在中间的重点位置，是 XHTML 和 CSS 等 Web 标准技术帮助实现了这一切

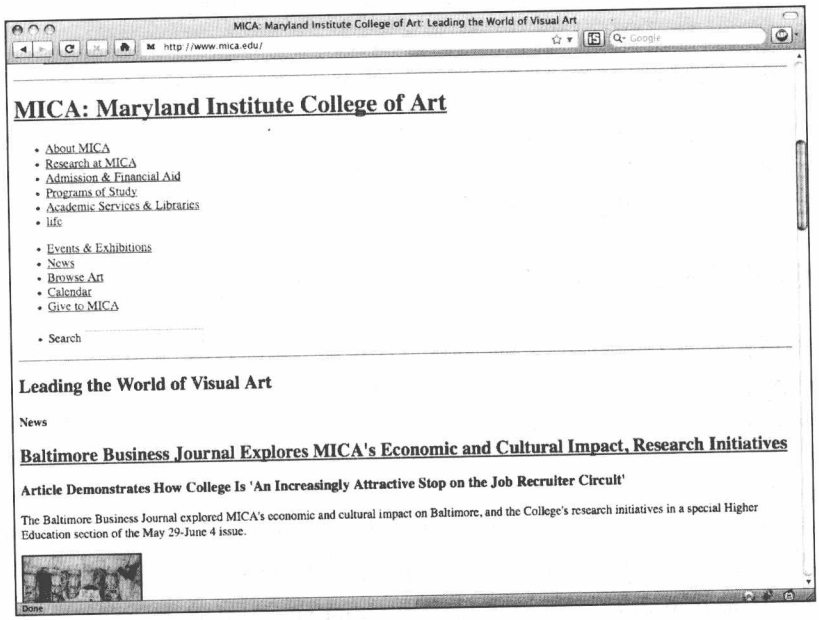


图 2.11　当样式表被禁用的时候，网站的内容仍然可以访问，语义丰富的标记很容易阅读，在那些只能显示文本的浏览器中的阅读也差不多同样友好

2.6 Web 标准组织：可移植性

Web 标准组织（WaSP）（如图 2.11 所示）成立于 1998 年，起初的目的是说服 Netscape、Microsoft 和其他浏览器开发商彻底支持本书介绍的标准。经过长期坚持不懈的努力以及有效的策略（又称“大喊，抱怨和恳求”的方法），最终使得浏览器开发商接受了 WaSP 的观点，即：Web 要向前发展，就必须采用共同的标准协同工作。

口是心非

在努力推进浏览器符合标准的过程中，颇有讽刺意味的是：微软，本身作为 W3C 的会员，而且已经为 Web 标准做出重大贡献，却不得不“被迫”完全支持自己参与制定的标准。想想看吧！

浏览器终于开始在真正意义上支持标准之后，Web 标准组织在 2002 年重新启动，目的是鼓励设计师和开发者学习和使用这些强大的、来之不易的标准技术。为了表明 Web 标准组织新的使命，WaSP 的网站从交流论坛到教程都进行了改版和设计。

就像预期的那样，站点在符合标准的浏览器中看上去很好（如图 2.12 所示），在老的、支持标准不完善的浏览器中看起来也可以接受（如图 2.13 所示）。而且站点还超越了 PC 浏览器的范畴，不需要额外附加的标记、代码或者设备检测，就能工作在其他的设备上（看吧，没有多版本代码）。

一个文档服务所有需求

Web 标准组织站点严格采用 XHTML 1.0 建造，采用 CSS 布局排版。它没有 Palm 版本或 WAP 版本（当时很流行），也不再需要多版本，当你使用 Web 标准去创建的时候，一个文档可以服务于所有的需求。

图 2.14 展示了 webstandards.org 在 palm 中的显示；图 2.15 展示了网站在微软的 PocketPC 中的显示。最令人惊讶的，图 2.16 显示了网站在苹果公司早已停产的手持设备 Newton 中能正常显示。Newton 截图者 Grant Hutchinson 告诉我们：在当时，没有一个现代网站能在这么古老的系统上正常显示。

图 2.12　2002 年建立、采用 CSS 技术的 Web 标准组织的主页，在 MAC OS X 操作系统中的一种早期浏览器 Camino（基于 Mozilla）中的显示效果（www.webstandards.org）

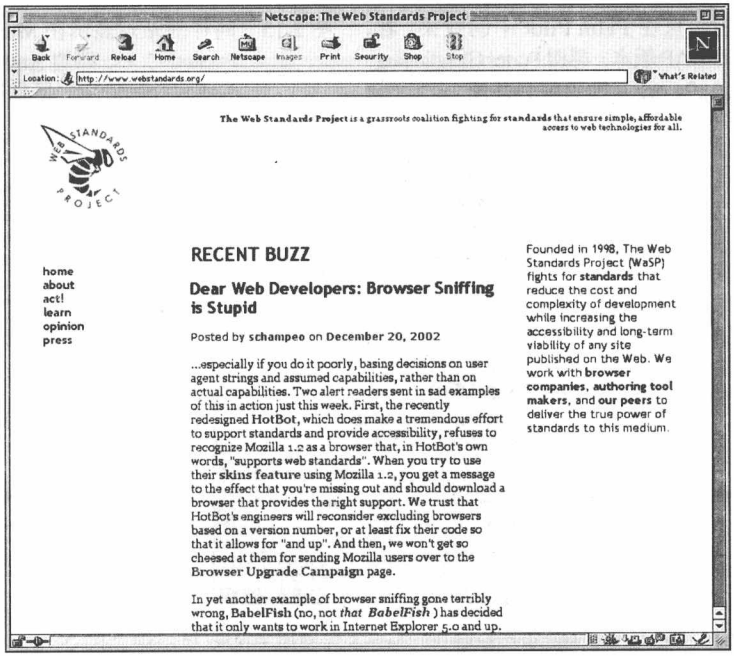


图 2.13　同样的站点在已经淘汰的、典型非标准支持浏览器 Netscape 4 中看起来还不错。也就是说，不需要一个特别针对 Netscape 4 的版本

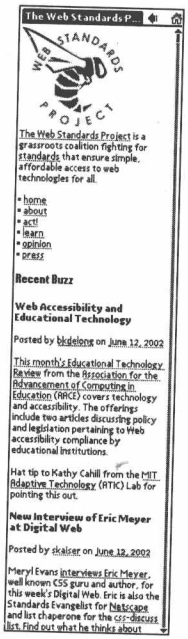


图 2.14　同样的站点在 Palm Pilot 中的效果。看，不需要 WAP 版本。感谢 Porter Glendinning（www.g9g.org）提供屏幕截图

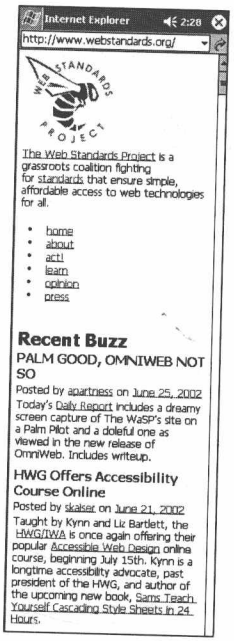


图 2.15　同样的站点显示在 Microsoft 的 PocketPC 上。感谢 Anil Dash（www.dashes.com/anil/）提供屏幕截图

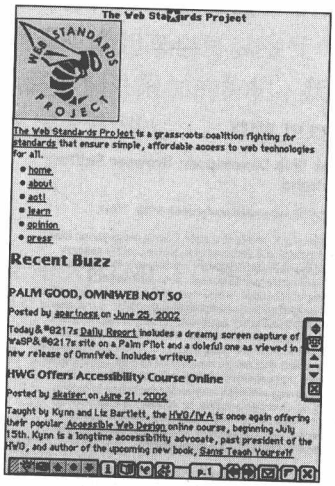


图 2.16　还是 Webstandards.org 网站，这次使用 Apple 早已停产的 Newton 手持设备。感谢 Grant Hutchinson（www.splorp.com）提供屏幕截图

对于那些希望用最小的代价获得最多的客户的设计师和站长们来说，这无异于

天赐佳音。严格使用 XHTML 和灵活使用 CSS 将使那些忙于创建多版本的设计师和开发者获得自由。

当你阅读到本书的时候，WaSP 网站也许又改版了，有些屏幕截图可能会不相同，但这些观点依然是正确的。

2.7 “A List Apart”：一个页面，多种显示方式

“A List Apart”（www.alistapart.com）是我们的设计公司 Happy Cog 于 1998 年创办的在线杂志，目的是向网页设计师长期教授和提升基于标准的设计技术。在 2001 年 2 月，当支持标准的浏览器终于面世的时候，我和同事一起将它的网站改版为只用 CSS 布局的网站，并鼓励其他设计师在他们的站点上也做同样的改变，大约数十万的设计师这样做了。

A List Apart's 的形式和结构证明了语义标记和 CSS 布局的优势。这种方法使我们的站点能够支持老的浏览器和那些不支持 CSS 的设备，而不用创建新的版本。支持 CSS 的浏览器可以显示正常布局（如图 2.17 所示）；不支持 CSS 的浏览器则可以按同样的文档结构显示内容（如图 2.18 所示）。

图 2.17　A List Apart （www.alistapart.com），“为网站建造者服务”的网站。这个漂亮的布局是 Jason Santa Maria 设计的，这是网站在符合标准的浏览器中看到的效果

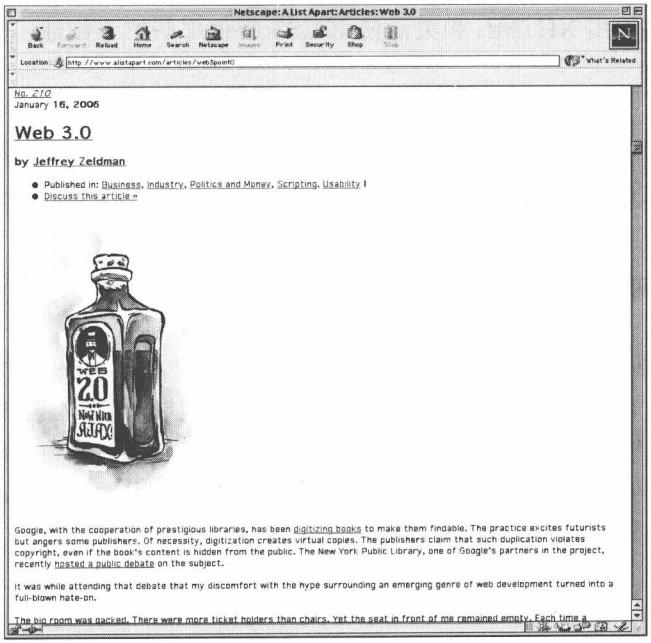


图 2.18　同样的站点，在 4.0 浏览器中的效果。不支持 CSS 也没问题，网站依旧能阅读和理解。标记会按浏览器默认样式展现基本布局

事实上，一些访问者很喜欢这个方法：在 A List Apart 网站用 CSS 重新设计后的一段时间内，Netscape 4 的用户访问增加了。因为他们可以看到一个简单清晰的页面，不再是先前惨不忍睹的页面。那些持有“标准会伤害老浏览器用户”观点的人们，从这个例子可以得到相反的结论：只要合理使用，标准可以帮助每一个人。

进一步说，在大多数情况下，使用这种语义标记加 CSS 布局的方法，可以帮助你建立一个“良好打印”的版本，而不需要制作另外的页面（如图 2.19 所示）。

2.7.1　屏幕以外的设计

图 2.19 展示的是 A List Apart 一篇文章打印出来的效果。如你所见，侧边栏已经隐藏，读者打印的时候并不需要这些导航内容。站点发行号前的大圆点也变成了简单的链接，字体和颜色已经针对打印而加以了优化。不管这些 URL 是否出现在屏幕上，每个链接 URL 都会有效地显示出来。实现这种神奇效果只是增加了一个专门的打印样式表，这个样式表是 Eric Meyer（“Eric Meyer on CSS”一书的作者）为网站写的。在网站的一篇文章中（www.alistapart.com/articles/goingtoprint），Eric 详细解释

了建立打印样式表的原理和技巧。

A List Apart: Articles: Web 3.0　　02/09/2006 03:34 PM

JANUARY 16, 2006　　No. 210

Web 3.0

by JEFFREY ZELDMAN

Published in: Business, Industry, Politics and Money, Scripting, Usability

Google, with the cooperation of prestigious libraries, has been digitizing books (http://books.google.com/) to make them findable. The practice excites futurists but angers some publishers. Of necessity, digitization creates virtual copies. The publishers claim that such duplication violates copyright, even if the book's content is hidden from the public. The New York Public Library, one of Google's partners in the project, recently hosted a public debate (http://www.nypl.org/research/chss/grd/grdreg.cfm?id=1661) on the subject.

It was while attending that debate that my discomfort with the hype surrounding an emerging genre of web development turned into a full-blown hate-on.

The big room was packed. There were more ticket holders than chairs. Yet the seat in front of me remained empty. Each time a hopeful standee approached the empty chair—and this happened every few nanoseconds—the poor schmoe seated next to it had to apologetically explain, "Sorry, the seat is occupied."

It soon became clear that the kindly schmoe was reserving the seat, not for a friend or colleague, but for a stranger who had imposed that duty on him. While the kindly fellow defended the other man's throne against a steady stream of resentful ticket holders, the stranger was off somewhere knocking back the library's free champagne. I wondered what kind of jackass would ask someone he didn't know to save his seat for thirty minutes at an oversold event. When he finally arrived, I found out.

A taste of ass

"Were you at the Web 2.0 conference?" the arriving man asked, by way of thanking the other for saving his place. The kindly schmoe signified in the negative. This was all the encouragement our man needed to launch into an adjective-rich and fact-poor monologue that was loud enough for half the room to hear.

It soon appeared that "Web 2.0" was not only bigger than the Apocalypse but also more profitable. Profitable, that is, for investors like the speaker. Yet the new gold rush must not be confused with the dot-com bubble of the 1990s:

"Web 1.0 was *not disruptive*. You understand? Web 2.0 is *totally disruptive*. You know what XML is? You've heard about well-formedness? Okay. So anyway—"

http://www.alistapart.com/articles/web3point0　　Page 1 of 5

图 2.19　从网页到打印：ALA 的文章可以实时地，以合适的排版打印出来，这要感谢打印样式表

现在需要领会的重要概念是：用一个单一的轻量级文本，一个打印样式表，A List Apart 网站不再需要独立制作一个“打印友好”的版本，同样你的站点也不需要。除了有一些例外：那些使用分页显示格式的站点，像 www.nytimes.com 或 O’Reilly Network（www.oreilly.com）的一些站点仍需要设计一个打印友好页面，以便将整个文章连接成一个完整的文档。但是，他们也可以通过使用打印样式表获得便利，显然，输出全文的那个页面可以使用打印样式表。

让我们来总结一下上述两个站点所获得的好处。

2.7.2 节约时间和成本，增加扩展性

如果采用标准进行设计和构建，意味着你不再需要为每个站点创建多版本，很容易看出这将节约巨大的时间和成本：

- 不再需要针对特殊浏览器的版本；
- 不需要为老浏览器准备基本版本；
- 在大多数情况下，不需要开发手机专用版本；
- 在大多数情况下，不需要建立额外打印友好版本；
- 不需要检测浏览器和平台，也不需要为了适应不同的浏览器和优化设备而改变服务器端组件。

许多人想给他们的用户提供手机服务，但可能负担不起开发成本。现在他们不用再担心成本问题了，因为 XHTML 和 CSS 标准的出台，不费吹灰之力，他们就可以获得大批的新读者和用户了。

严格符合标准也提供了一个可访问性问题的解决方案。如果你的站点能够在一个 Palm Pilot 上工作，它也就能像在 JAWS 的屏幕阅读器上工作一样。当然，你需要进行测试来确认一下，可能需要一些额外的工作来保证它们确实可用。我们将在第 14 章和第 2 部分的一些章节来深入讨论可访问性问题。

2.8 我们去向何方

我们不能继续过去的开发设计方法来制作未来的网站，我们的出路只有标准化，尽可能的采用 Web 标准。这意味着什么？让我们来看一看。

向后兼容的要求

- 完全分离结构、表现和行为；

- 使用校验正确的 CSS 布局，表格仅仅用在其真正本意上：表格数据的展示，例如：电子表格、地址簿、股票报价、事件列表等等；
- 使用校验正确的、严格或者过渡的 XHTML 1.0（或者 HTML4.01/5）标记；
- 注重结构。没有表现标记用于结构（严格的），或者只有少数表现标记还存在（过渡的）；
- 抽象出结构化标记来描述设计元素，例如“Menu（菜单）”要好于“Green Box（绿箱）”
- 用基于 DOM 的脚本语言控制行为：如果你需要对代码进行分支，那么探测对象支持，而不要探测浏览器版本；
- 可访问性属性和测试；

为什么你应该关心

今天，有效的 CSS 布局和语义标记使网站具有严格的向后兼容性，已经被广为推荐。加上广泛的兼容标准的浏览器支持，通过上述技术，不符合标准的浏览器也能访问你的内容，虽然也许不是你网站的每个元素都能正常访问。由坎贝尔（Campbell-Ewald）设计的美国海军网站（如图 2.20 所示）采用了纯 CSS 布局和严格 XHTML 1.0 标记，还有一些 Flash 内容使页面显得生动有趣（如图 2.21 所示）。

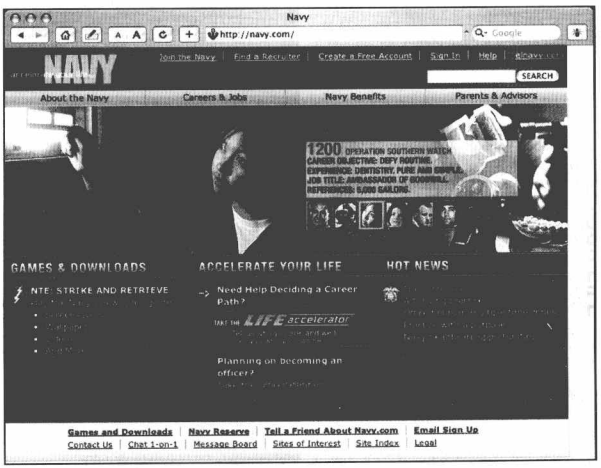


图 2.20　美国海军网站（www.navy.com）采用严格的 XHTML1.0 标记和纯 CSS 布局（混合了一点 Flash），由坎贝尔（Campbell-Ewald）设计

益处

- 向后兼容性：可以在现有和未来的浏览器或设备（包含手机）协同工作；
- 用更少的工作获得更多的用户；
- 不需要多个版本；

图 2.21　2009 年，美国政府白宫网站（www.whitehouse.gov），采用 XHTML 1.0 过渡标记对内容进行结构化、用 CSS 驱动表现，以及用基于 DOM 的 JavaScript 增加行为

- 更少的可访问性问题。这样设计的网站能对所有人开放；
- 还原标记的雅致、简洁和逻辑性；
- 还原文档本身的结构；
- 更快、更容易、更便宜的发布和维护，降低了成本；更多的预算（如果有的话）可以投入到写作、设计、程序、艺术、图形、编辑和可用性测试中；
- 更容易合并到提供动态发布和模板驱动的内容管理系统中
- CSS 布局使一些用 HTML 表格做不到的设计成为可能；
- 站点可以在将来的浏览器和设备中继续工作。

须要考虑的事项

- 站点极有可能在老浏览器中看上去相当简单朴素。
- 即使在 2009 年，浏览器对 CSS 的支持还是不完善，可能需要一些 CSS 技巧（“CSS hacks”）主要用于解决 IE 老版本中的问题。
- 基于 DOM 的交互行为将不能在 4.0 版和更老版本的主流浏览器、屏幕阅读器、文本浏览器和多数的无线设备中工作。但是如果你采用适当的方法，不支持 JavaScript 的用户依然可以访问到你的内容（详见下文）。

本书的第 2 部分解释了标准是如何工作的（单独工作以及协同工作），提供了解决关于各种各样 Web 设计和商业问题的技巧和策略。但在我们开始深入学习之前，先要讨论一些可能已经长期困扰你的问题。

如果标准能够促进协作能力，增强可访问性、工作流程和维护，减少带宽浪费，降低成本，那为什么不是所有的设计师和开发者都一直使用标准来创建网站呢？

为什么不是所有的客户都大声叫嚷要求使用符合标准的方法呢？为什么甚至需要我们写一本像这样的书，乞求你的客户、同事、老板和供应商来阅读呢？为什么六年以后，还必须再写本书的第 3 版？为什么还有新的不符合 Web 标准的网站建立？为什么 Web 标准还没有被更广泛地理解和使用呢？

一个原因是缺乏知识。这个问题简单，只要学习，你很容易就成为一个 Web 标准网站设计大师。不断实践建设简单精练、可访问性好、打开快速的网站（内容也很容易被搜索引擎找到）。这样边做边展示，用事实和结果来赢得队友、经理和客户的尊重。越是努力为更多的用户建立符合标准的网站，你就能越轻松完成业务目标，也能更简单和更便宜地维护、更新，这不仅有利你推广 Web 标准，也可以引导和发展新用户。

另一个许多网站看起来还没有接受和理解标准的原因是，一些站长、管理人员和技术主管反对使用 Web 标准，通常他们的理由都是不成立的，没有基于调查和分析。如果你的老板或者客户，用一个又一个可笑的理由告诉你不能使用 Web 标准，说服他们是你的工作。第 3 章节将告诉你怎么去做。

第 3 章

温和劝导

这可能是你读过的书中最短的一章，也是我写过的书中最短的章节。但是，尽管很简短，这一章有它的价值体现。你将在本章学到当你的老板、同事或者客户反对标准时应该如何应对。我不能保证你的老板和客户像冰淇淋遇到烧烤架那样立刻融化，马上赞同基于 Web 标准的设计，但至少他们会解冻，其他就靠你自己了。

拒绝的理由：我们在 IE6 上标准化

你该怎么说：“非常棒！那样我们就可以在网站上使用标准的 HTML、CSS 和 JavaScript。”

你该怎么做：按照本书的建议，用 Web 标准设计网站，在支持标准的浏览器上开发，例如 Firefox3+、IE8+、Opera10+或者 Safari 4+，然后对设计进行必要的微调（用本书第 2 部分介绍的方法），使之能在老版本的 IE 上正常工作。这里说的“微调”，意思是确保外观在 IE6 中看起来差不多，不用担心圆角、段落缩进和首字母放大等的细节差异。或者，设计一个简单的页面，使它能在所有浏览器（包括 IE6）中看起来都一样，然后增加额外的层来逐步实现类似圆角、段落缩进和首字母放大效果，这种设计方法又称为渐进增强法。第三种方案，在有些案例中，你可能希望网站在 IE6 下比在其他浏览器中有一个更好的视觉体验。关于此方案的详细信息，请阅读第 1 章的“现代浏览器和 Web 标准”小节里的“IE6 通用 CSS”内容。

拒绝的理由：“我们在 IE7 上标准化。”

你该怎么说：“非常棒！那样我们就可以在网站上使用标准的 HTML、CSS 和 JavaScript。”

你该怎么做：与前面的解决方案一样，比 IE6 遇到的问题更少一些。

拒绝的理由：“我们需要在 SEO（搜索引擎优化）上节约一点。”

你该怎么说：“非常棒！那么最好的办法就是提供精简的、适合用户的语义标记。”

你该怎么做：向他们解释结构化的、易于访问的标记为什么能提高搜索结果。（如果你无法解释，那就复印本书的几页，只是不要告诉我的出版商。）

拒绝的理由：“难道 Flash（或者其他技术）不能更好地得到多平台支持？”

你该怎么说：“这是一个常见的误解。Flash 在视频播放系统或者作为一个体验增强组件时有它的长处，但是在可维护性和可扩展性上不如（X）HTML、CSS 和 JavaScript。虽然现在的浏览器在设计和脚本方面对 Flash 的跨平台支持与对 Web 标准的支持一样好，但是无法替代 Web 标准在 SEO 和可访问性上的优势。”

你该怎么做：找一个复杂而精美的网站在多种浏览器（例如 IE、Firefox、Safari 和 Opera）以及多种操作系统（例如 Linux、Mac OS、Windows）中显示，这将彻底打击只有 Flash 可以跨平台的信念。选择一个有复杂交互的网站（当然脚本是符合 Web 标准的），在不支持 JavaScript 的平台上展示，网站能够正常执行。而 Flash 网站则无法在不支持 Flash 的平台上显示。再来看 SEO（可发现性）和可访问性，让 Google 搜索引擎的结果来告诉我们这个故事，虽然 Flash 网站也有可能有较好的可发现性和可访问性，但是很少有 Flash 开发者注意这样做。相反，即使 HTML 新手，只要使用语义标记和好的标题，就能提高搜索引擎的评分。眼见为实，谁胜谁负，一目了然。

拒绝的理由：“我们的网站已经在所有浏览器里看起来都一样了。”

你该怎么说：“我们来研究一下，这是基于什么基础的，它能维持多久？”

你该怎么做：用变通但坚定的方法说服营销人员、品牌总监和 CEO 们。你可以说许多理由，只要你的基调定位在商业理由上，而不是传播 Web 标准，通常都能完成任务。

首先，当广告总监说网站在所有浏览器都必须显示一样的时候，小心地试探他真正的意思。在少数情况下，他的意思仅仅是标志（Logo）必须正确显示。大多数情况中，品牌人员真正的要求是网站在 IE6 和 IE8 里的外观和工作完全一样。（为什么是 IE？因为通常提出这一要求的人士不大可能讨论 Firefox、Opera 或者 Safari。）

如果他们只讨论 IE6，没有说 Netscape 4 或者 IE5，那么用如上所述的标准技术设计，采用渐进的方法在更强大的浏览器里增加新的细节，并获得你的客户或者老板的许可，忽略 IE6 用户的一些细节体验。如果站点在 IE6 中看上去很好，使用正常，那么你诚实说明哪些小细节被省略了（可以解释成为了维护方便和节省带宽），也会获得执行经理的批准。

当然，这位执行经理也可能需要你的一点点帮助，才能从“不行”转变为“可以”。例如，你可能需要用两种代码方式来制作同一个网页。第一种使用精简的、语义化的标记；第二种使用那些成堆的、无含义的标记（例如为了在 IE6 中实现圆角效果写的标记），然后计算出第一种方法节省的带宽，再乘以网站的访问量，最后用美元和美分得出一年节省的费用。这个数字不能保证执行经理了解你的方法，但至少是一个开始，开始理解你说的说法。

创建一个电子表格，测试不同用户在网页加载时间上的差异，特别是网速较慢的拨号用户和 DSL 用户。结论是页面加载时间越慢，用户没等阅读到公司信息就离开的可能就越大（Jakob Nielsen 称加载缓慢是设计第一大错误）。这时候，在可能失去用户的前提下，再坚持“在不同浏览器里看上去每一个像素都要求一样”就显得毫无意义。

分析网站的日志，看看还有多少人使用 IE6，如果低于 20%，你可以提出这一论点：为了 20%用户的小细节完美，而增加 80%用户的加载时间是愚蠢的，何况那 20%用户可能根本不关心这些细节。（不要花太多时间去证明用户是否关心网站细节，毕竟，在设计的过程中，你不可避免会遇到很多细节问题。）

如果你的执行经理依然犹豫不决，那么问她，她认为有多少 IE6 用户会同时打开 Firefo3.5 和 IE 6，并排窗口来比较网站的视觉细节差别。除了网站开发人员进行测试，没有人会这样做。显然，那些坚持在不同浏览器里每一个像素都相同的设计可能只满足不到 1%的用户需求。浪费时间、金钱和带宽，为这极少数用户设计，不符合商业理念。

也可以提醒她，当越来越多的用户通过手机访问网站，他们开始习惯看见同一个网站的不同版本，例如 mobile.nytimes.com 与 nytimes.com, m.flickr.com 与 flickr.com 等等（如图 3.1、图 3.2 所示）。从这个角度来看，一个过时的浏览器只是变成了一大堆网站查看工具的其中之一。

图 3.1　读者接受纽约时报手机版本的外观和交互与...

如果用户能接受网站在旧手机、智能手机和现代浏览器中有不一样的版本，那么他们也可以接受（甚至不用通知）旧浏览器和新浏览器中略有不同的体验。

对于有经验的执行经理，你甚至可以提出更巧妙的观点：网站的每个用户应该根据自己的兴趣和选择获得不同的体验。差不多就是说，存在几种可能的视觉体验。假设你从一开始就（或为最近一次重新设计）采用了标准信息结构或用户体验过程为主导的方法进行网站开发，当执行经理接受不同人物角色导致不同使用路线的时候，她也就认同了不同用户将有不同需要的看法。同样，大学网站为“学生、老手、校友和捐助者”这些不同的用户角色提供不同的体验服务，那么使用旧浏览器和新浏览器的用户获得一些不同的视觉体验也合乎逻辑。大多数拥有和管理网站的人因为实践理解并接受用户角色这个概念，虽然这来源于心理学知识，但已经和市场营

销、用户统计联系在了一起。“不同的用户体验是好的”到“不同的视觉体验也是好的”仅仅一词之差。一个聪明而备受尊敬的商人，可能可以理解这一点。

图 3.2 ...桌面浏览器版本不相同。移动设备的广泛使用，使用户开始接受相同网站的不同版本。这对那些被要求“网站在每个浏览器里看起来都一样”的开发者来说无疑是个好消息

如果这些都不能说服你的老板或者客户坚持传统观念，还有两个选择：

- 考虑一个简单的设计，一个在多个平台都能工作的页面，不要有无意义的标记。如果对不同平台之间的像素精度是基本要求，那么用尽可能少的垃圾代码实现目标外观效果。（这样的设计要求往往可以用“drop-shadow”阴影和“gradient-fest”渐变效果就可以实现。）

- 如果一个简单的设计也不行，你可以使用更多的有语义的调用（hooks）以及尽可能少的技巧（hacks），把每一个垃圾的 div 标签建档保存，这样来年或者两年以后，当使用老浏览器用户百分比下降时，你可以方便地将它们替换为精练、干净的代码。

如果有人坚持要求网站在所有比 IE6 还老的浏览器看起来效果一样你怎么办？比如，有客户告诉你，他们要在 Netscape 4 或者 IE 5 上“标准化”，那么很明显，你只能另找客户。在你的辞职信上，一定要包含一个链接地址：dowebsitesneedtolookexactlythesameineverybrowser.com（如图 3.3 所示）。

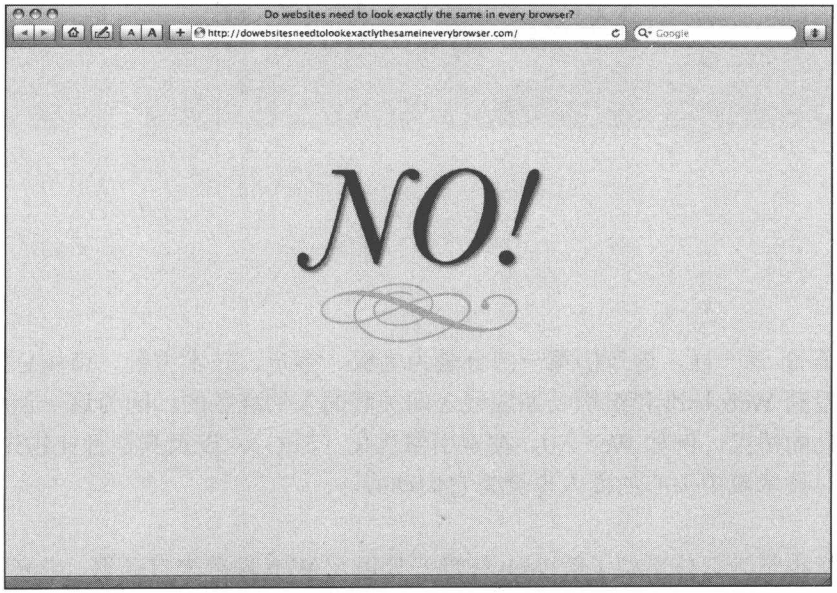


图 3.3　这个可爱的站点回答了这个问题：“网站需要在所有浏览器里都看起来一模一样吗？”

第 4 章

Web 标准的未来

和第 3 章一样，这章的第一部分是为老板、客户、技术主管、市场总监以及那些还没看到 Web 标准和组织长期发展之间关联的人们准备的。因为这一点就是我们可以借助的桥梁，例如 Web 2.0、搜索引擎优化（SEO），以及其他行业领先的创意，即使是对技术漠不关心的商人也会为它们动心。

本章也是为那些希望了解 Web 标准好处以及 Web 标准来自哪里，未来将如何发展的网页设计师和开发者准备的。如果你对 HTML5 充满期待（或者担心），如果你对 XHTML2 项目的停止觉得放心（或者失望），或者你是努力工作的专业人士，但对我所说的不甚了解，那么这章也是为你准备的。

让我们从你老板或者客户需要知道的基本内容开始。

4.1 可发现性、聚合、博客、播客、长尾、Ajax（以及其他使标准成功的理由）

企业老板、客户、信息和市场人员，你们在听吗？这是你们必须知道的：尽管对标准的误解妨碍了标准的采用，但标准已经在许多技术前沿获得成功，并且正在迅速地改变着我们的科技、商业、网络及非网络出版行业。事实上，在过去的五年里，Web 标准已经成为转变市场、获得赢利的数字创新的决定性因素。

看看播客或者博客，即使是市场经理的助理也听说过它们。是什么技术支持它们运转？是 RSS，一个 XML 应用程序。这个 XML 应用程序还做些什么？它帮助传统的报纸杂志把内容迁移到一个有更多用户阅读的地方——这就是网络。

也许市场主管已经读过关于“长尾”经济理论，积少成多，小买卖累积起来也能成就大生意。在 2004 年 10 月，《连线》杂志的主编及 TED 大会的组织者 Chris Anderson 发现：亚玛逊（Amazon）书店超过一半的收入来自销售榜排名 130 000 以后的图书，也就是说那些非畅销图书市场可能比常见图书的市场还要大（www.wired.com/wired/archive/12.10/tail.html）。

网络提供给网民个性化、定制化的产品和服务，这是传统商店难以提供的。什么能最好地抓住长尾？是那些最容易被找到的内容。Peter Morville，现代信息构建理论创造者，把这个有助于成功的品质称为“可发现性（findability）”

为了使他们的产品在网络上更容易搜索到，许多公司花费了上百万美元的费用在 SEO（搜索引擎优化）上。然而也有一些公司负担不起 SEO 费用，但依然能很好地被搜索到，很好地占领长尾市场，他们的秘密是什么？他们写干净的、关键词丰富的、易懂的、用户关心的内容，用有语义的标记将他们的文本推到成堆的数字信息顶部。加上合理的编辑，CSS 布局和结构化的 XHTML，这些成为打开“可发现性”大门的金钥匙。掌握这种方法的公司兴旺发达，不了解这种方法的公司日渐衰落。（对于那些崇拜 SEO 但忽略语义标记的人们，建议看一下本书前言中 Twitter 屏幕截图，把它们打印下来并随意贴在 T 恤、杯子和保险杠上。）

如果说 2000 年的网络看起来了无生气，那么现在它再次进入花期，尽管其他行业不景气，美丽的互联网“赚钱”之花正在发芽，这一切都要感谢基于 Web 标准的新思想和新技术。

4.1.1　通用语言 XML

XML（可扩展标记语言，Extensible Markup Language 的缩写）标准（www.w3.org/TR/REC-xml），于 1998 年 2 月提出，并在软件行业引起了风暴。这是结构化文档和数据第一次有了一个通用的、普遍适应的格式，不仅可以应用在 Web 上，也可以应

用在任何地方。全世界很快就喜欢上了它，就像小男孩天性喜欢玩泥巴一样。虽然预言家说的“XML Web”时代不会很快到来（XML 取代 HTML 感觉就像飞行汽车和时间机器那么遥远），但特定的 XML 应用已经使媒体获得新生，XML 已经让消费者和商业软件应接不暇。

XML 之父有三位

Tim Bray、Jean Paoli 和 C.M.Sperberg-McQueen 为世界带来了 XML。Bray 有一个注释版本（www.xml.com/axml/testaxml.htm）提供了 XML 和 SGML 的详细说明，还有一些小幽默呢。

XML 和 HTML 的比较

XML 虽然和 HTML 来源于同样的技术（类似 HTML，它也使用标记、属性和值来格式化文档结构），但 XML 和 HTML 完全不同。

HTML 是构架网页的一种基本语言。它由一些固定数量的标记语言和一些不严谨的规则组成。在 HTML 中，你可以随心所欲地关闭或者不关闭某些标记。这种宽松的规则使人们创建 Web 页面变得很容易，甚至不需要知道他们在做什么都行——当然，当时的意图就是这样。

在早期，宽松规则的 HTML 是一个好方法，那时候网页只有基本的内容而没有更多其他信息，HTML 的低门槛推动了网页的普及。但在今天，更大、更复杂的网站，经常需要频繁通过发布工具重新组装页面，经常需要将内容在数据库和 Web 页面或者无线设备之间来回传送。但 HTML 缺乏这种统一的规则，从而阻碍了数据的转化。我们很容易把文本转化成 HTML，但是很难把标记在 HTML 文件中的数据转换成我们需要的其他格式。

同样，HTML 仅仅是一种格式化语言，而不是一个能确切描述自己的语言。它没有任何信息来描述它所格式化的内容，因此限制了我们用其他格式重用这些内容的能力。（微格式，后面我们将要讨论到，在尝试丰富语义 HTML 方面获得了是一次相当成功的尝试。）当然，HTML 是专门用于 Web 的。

相对而言，基于 XML 的标记，则由一些统一的规则组成并且具备超越 Web 领

域的能力。当你用 XML 创建一个文档的时候，你不仅仅是准备把它们在 Web 页面中显示出来，还是在对标记进行编码，从而使得任何支持 XML 的环境都能理解你的 XML 文档。

同一父母，多个孩子

很明显，XML 是一种能创造其他语言的语言。只要遵循 XML 本身的规则，图书馆管理员们可以通过适合图书目录的定制标记自由地创建 XML 标记。音乐公司也可以创造出他们自己的 XML 文档结构，包括艺术家信息、唱片的信息、作曲家信息、制片人信息、著作权信息、版权信息等标签。作曲家可以通过一个叫 MusicML 的 XML 定制架构来进行谱曲。

这些自定义的 XML 语言我们称为应用，并且它们也是 XML，因此它们都能互相兼容。也就是说，一个 XML 解析器能够理解所有的这些应用，每个应用都能很容易地和其他应用进行数据交换。因此，不需要额外的工作，就可以将一个唱片公司的 XML 数据库中得到的数据很容易地转换到图书馆目录册中，并且不会产生错误，也不会陷入到软件不兼容的困境之中。

专业软件的基本要素

XML 的格式化、可理解和交换数据的强大能力使得它就像可口可乐一样无处不在。XML 不仅可以保存网上和企业数据库的内容，而且也可以成为面向数据库处理软件（例如 FileMaker Pro）和更多非面向数据库软件的通用语言。从高端的应用程序到商业产品（比如微软的 Office 和 OpenOffice），都能看到基于 XML 的文件格式。

打印设计软件 Quark XPress 及 Adobe InDesign 可以导入导出 XML，并且支持创建基于 XML 的模板。可视化的 Web 编辑工具同样具备“XML 智能”（例如 Dreamweaver），从而使得在打印页面、Web 布局以及支持你的在线商店和全球目录运行的数据库之间的数据转化变得更容易（至少也会变得有可能）。

不只是可以解析 XML，有一些软件竟然就是由 XML 组成的。Dreamweaver 就是由一些最终用户也能看得见的 XML 文件组成的，通过自己动手编辑这些文件就可以修改应用程序了。早在 2002 年，在 A List Apart 上 Carrie Bickner 的一篇热门文章

（www.alistapart.com/articles/dreamweaver）解释了如何编辑 XML 文件使 Dreamweaver 4（对，就是 Dreamweaver 4）能生成有效的 XHTML。用这种方式定制 Dreamweaver 和出售这些定制化版本甚至成为了一个行业，我知道的一个家伙靠此赚来的钱已经买了一栋房子。

消费者软件也喜欢采用 XML 技术。在你的 PC 机、Mac 计算机或者 PDA 上的个人信息管理软件都可以读写 XML 信息。当一个数码相机在一张照片上记录拍摄时间、照片大小、文件大小和其他类似信息的时候，很有可能就是采取 XML 来记录这些数据。当你的父亲通过电子邮件给你发送了一个 7MB 的度假照片包，日落时刻的美丽照片很可能采用的就是 XML 格式。嘿，你父亲也在用网络标准哦。

甚至一些惹人喜爱的图像处理软件也能处理 XML 格式（比如 Apple iPhoto）。当你打印一张家庭照片时，打印能够正确输出，这些都要归功于 Macintosh OS X 操作系统中保存在 XML 数据中的打印设置。（事实上，整个基于 UNIX 的 OS X 操作系统都采用 XML 方式储存。）苹果 iTunes 的 Windows 版或者 Mac 版，也都是用 XML 导出播放列表的。

比 White Rapper 更流行

为什么 XML 能发挥这么多不同的计算机厂商的想像力，并且把它运用到他们的产品中呢？因为 XML 由各种标准化的特性组成：可扩展性（个性化定制能力）、可移植性（把数据从一种格式转化到另外一种格式的能力），以及在 XML 应用程序或者基于 XML 的软件产品和其他同类程序之间进行相对无缝的数据交换能力。

作为一种不受专利和版权限制的开放标准，XML 扫除了过时的、私有的、不被广泛接受并且成本高昂的格式。如果你在软件中集成了 XML 或者创造自己的基于 XML 的个性化语言，W3C 并不会让你支付任何费用。不仅如此，接受 XML 就像病毒传播一样，越多的厂商发现 XML 的好处，就越能加速 XML 向其他厂商的传播，就越容易能在一个厂商和另外一个厂商的产品之间进行数据传输。

而且，XML 确实有用。过去如果你能从一个程序中导出用 tab 字符分开的文本，并且导入到另外一个程序中（一般都会丢失一些信息和进行手工重新排版），办公室的同事一定会以你为偶像。现在，XML 能帮助厂商创造出让用户更好协同工作的软

件产品，并且更加灵巧而不会让用户感到困难。用户是会很乐意掏出钱包来购买的！

不是万能药，但能在电视中使用

我并不是说 XML 是解决任何软件问题的万能药。比如在图片压缩比例方面，采用 JPEG 二进制格式的数据要比文本格式更好。我也不要求所有软件都必须采用 XML，虽然大多数专业软件和大众软件都采用了 XML，并且数量还不断增长。我没有说市场上声称支持 XML 的软件都是完美地支持了 XML（即使是在网络行业，如本书所言，首席执行官反对 XHTML 就是因为 IE 依然把 XHTML 当 HTML 处理）。但尽管执行上不够完美，XML 作为 Web 标准还是改变了软件产业和我们在家庭和工作场所使用的硬件。

即使是那些现在不支持 XML 的产品，也相信他们最终将会支持 XML。在 2002 年 4 月，销售惨淡的中间件市场令人悲哀，在 iTV 标准协会（iTV Production Standards Initiative，www.itvstandards.org）的规范下，一个由交互电视和技术提供商组成的组织成立了。这个组织的任务是：推出和支持一个基于 XML 的标准，从而“允许制片人可以一次编写交互式内容，并且能发送到所有的机顶盒和所有的 PC 平台上去”。听起来很熟悉？这正是 Web 标准组织在 20 世纪 90 年代中后期浏览器大战中倡导的 W3C 标准。

建立稳健数据的五种方法

在 Web 上，工作在大型的协作项目或者特定的系统上的 IT 专业人员、开发者和内容专家，越来越多地选择 XML 作为数据格式。因为有五个因素而选择了 XML，其中许多在前面的讨论中都已经提到了。

- 就像 ASCII 文本文件一样，XML 是一个单一的、通用的文件格式，用来和其他系统进行交互。

- 不同于 ASCII 文本文件（或者 HTML），XML 用一种智能的、自识别的格式。XML 不仅保存数据，它还有保存数据的数据（元数据），协助检索和其他功能。

- XML 是一种扩展性语言，容易定制以适应任何商业及学术领域，或用来创建其他基于 XML 的语言，来处理特定任务，比如数据同步或者 Web 服务的传递。

- XML 基于如下规则：可以保证数据传输到其他数据库，转化成其他格式或者被其他 XML 应用程序处理中的一致性。

- 通过附加的 XML 协议和基于 XML 的帮助语言，从 Web 页面到打印目录到年度报告，XML 数据可以自动地转化成各种各样的数据格式。在 XML 出现之前，这样强大的功能对于开发者来说都只是一个梦。没有人会怀疑 XML 可以轻松地提高效率。

4.1.2 发明的源泉

虽然对 XML 的完整讨论超出了本书的范围，但是接下来的例子，会让你加深对 XML 的了解和接受程度，它也说明了在 Web 领域之外，XML 的派生语言和协议是如何解决那些最聪明的开发人员也一度难以解决的问题的。

RDF（资源描述框架）（www.w3.org/RDF/）

这种基于 XML 的语言为那些在网络上交换元数据的应用程序提供了一个连贯的结构。具体来说，RDF 可以集成图书馆的分类和目录；收集和整合新闻、软件，以及各种内容；促进不同的集合之间的沟通和交流（例如：个人收藏的照片和音乐）。RDF 的能力同样在软件中体现，如果你正好使用基于 Mozilla 的浏览器，那么打开它的目录并查看，你会发现其中包含了 RDF（以及 CSS）文件。具体来说，打开每个配置文件夹，你都可以找它对应的一套基于 XML 的文件。

RDF 看上去是一个困难的、令人糊涂的语言，但是如果能够正确运用，它将产生出色的创意。Jo Walsh（frot.org）用 RDF 完成了对地理空间关系的标注（space.frot.org）。在一篇采用基于 RDF 分类法的文章中，作者 Paul Ford 为成千上万的设计师描述了一个真实语义网的概念，而他们曾认为那样的语义网只是空中楼阁（www.ftrain.com/arbs_and_all.html）。还有更多关于 RDF 的好玩案例，可以在 XML.com 看看 Tim Bray（XML 之父之一）的“什么是 RDF？”（www.xml.com/pub/a/2001/01/ 24/rdf.html）。

RDFa (www.w3.org/TR/xhtml-rdfa-primer)

W3C 创建 RDFa 的目的是用它在“人类和电脑（数据网络）”之间建立一座沟通的桥梁。就像微格式（虽然一些微格式社区的成员不赞同这样的说法），RDFa 为现有的（X）HTML 元素（例如 a 和 rel 元素）增加了语义。想进一步了解，你可以花半小时时间阅读 Mark Birbeck 的“RDFa 简介”（www.alistapart.com/articles/ introduction-to-rdfa）以及“RDFa 简介 II”（www.alistapart.com/articles/introduction-to-rdfa-ii）

XSLT（Extensible Stylesheet Language Transformations，可扩展样式表转换语言，www.w3.org/TR/xslt）

这种基于 XML 的标记语言，可以对 XML 数据进行提取和排序，并将它格式化为 HTML 或者 XHTML，以方便在线即时浏览。如果你喜欢，XSLT 可以将你的数据转换为 PDF 或者纯文本；或者用它来驱动不断更新的表格以及可以缩放的矢量 SVG 图形。XSLT 甚至可以在同时做这些事情。这里有一个实际应用教程，是 J.David Eisenberg 的“使用 XML”（www.alistapart.com/articles/usingxml）

RSS 2.0（Rich Site Summary，站点内容聚合，blogs.law.harvard.edu/tech/rss）

营销人员应该关注这一技术。RSS 是一个轻量级的描述网站的 XML 词汇，可以用它来告诉用户站点什么时候更新，更重要的是（对营销人员更有吸引力的是）也可以用它来把更新的内容发送给用户。还记得你经常听到，如何使你的网站更有“黏性”？提供 RSS 是一个非常好的方法，不是你希望的那样把用户黏在你的网站，用了 RSS，可以让你的内容去黏你的读者。

RSS 最早是由 Dan Libby 开发的，用来组装 AOL/Netscape 的“我的 Netscape”门户程序的。当在 2001 年 4 月份 AOL 对它失去兴趣后，Dave Winer 的 UserLand 软件公司推动了这个规范继续前进。后来 Winer 离开了 Userland，去了学术机构。现在，RSS 规范在创作共用协议（Creative Commons license）下由哈佛大学贝克曼研究中心（Berkman Center）发展。（cyber.law.harvard. edu）。

现在，RSS 2.0 已经被上百万的个人、公司、博客和社区网站所使用，使它成为在 Web 上最为广泛接受的 XML 格式（如图 4.1、图 4.2、图 4.3 所示）。它的简单却强大的聚合能力，很容易将博客和播客聚合起来（参见注释："你能在我的网络讲座里引用你的播客！"）。所有的博客系统软件都支持 RSS 2.0，也基本支持名为 Atom 的竞争规范。对于那些靠 RSS 集成的网站和产品来说，它们是"饲料"，是内容来源。它们是一种服务，当你的站点更新时，会及时通知搜索引擎（www.pingomatic.com）。

你能在我的网络讲座里引用你的播客！

播客是一种语音文件，可以被下载到你的电脑、Apple iPod、iphone 或者其他 MP3 播放器里，然后在体育馆、汽车、飞机上听。如果你在线，RSS 2.0 会提示你，有订阅的内容更新了，通常开始下载新的内容并同时删除老的内容。就像博客，任何人都可以建立播客，一些最流行的播客都是由业余人士建立的，就像那些热门的博客作者一样。

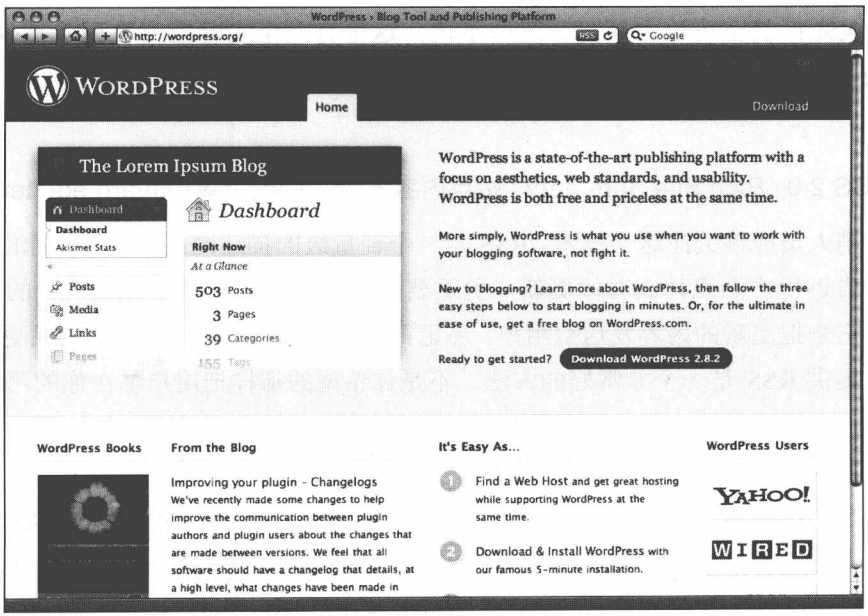


图 4.1　符合标准的博客信息发布系统 WordPress（www.wordpress.org），支持 RSS 订阅

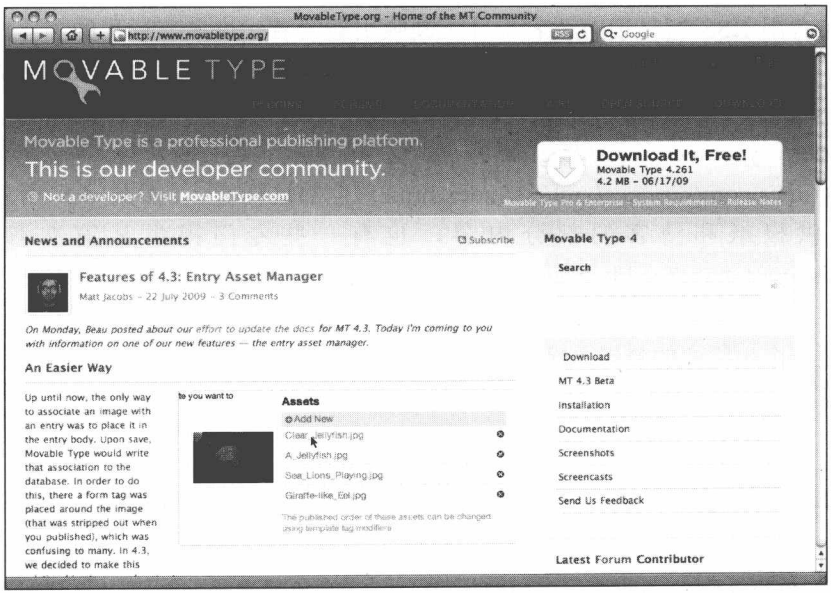


图 4.2　Movable Type（www.movabletype.org），也同样支持 RSS

图 4.3　社会化媒体网站，从 Twitter 到 Flickr 都允许你的好友订阅你的 RSS（www.flickr.com）。每当我发布一个新照片，订阅我 RSS 的好友就能从 RSS 阅读器中知道我发了新照片。好吧，我只是想上传一张我孩子的照片，你能怪我吗？

发布者用 RSS 留住现有读者，并不断吸引新读者。许多有远见的独立出版商都这样做。今日美国（USAToday）网站发布 RSS（content.usatoday.com/marketing/rss/index.aspx），BBC（英国广播公司）也这样做（如图 4.4 所示），Amazon 以及 Yahoo（developer. yahoo.com/rss/#biglist）也这样做。连线新闻和纽约时报（www.nytimes.com/services/xml/rss/index.html）也都这样做。（如图 4.5 所示）一些 RSS 提供报纸和博客的部分内容，也有一些专门提供某篇文章的讨论（www.alistapart .com/feed/hattrick/rss.xml）。

这是一个出版商的梦想、营销人员的兴奋、销售人员的收入来源。（令人讨厌的是，因为电视和报纸的表现下滑，越来越多的销售和市场人员在 RSS feeds 中加入广告。）

XMLHttpRequest—不仅仅是为了 Ajax(en.wikipedia.org/wiki/XMLHttpRequest)

作为 IE 浏览器 ActiveX 的一部分，由 Microsoft 创建，现在同样被 Apple 的 Safari、Mozilla 和 Opera 浏览器支持，XMLHttpRequest 对象通过 JavaScript 将从服务器获取的 XML 数据展现出来，却不需要刷新页面。

图 4.4　就像所有的时髦新闻站点一样，BBC（英国广播公司）网站也向读者提供头条新闻、世界科技（视频）和最新出版故事的 RSS 源

图 4.5　你希望订阅 Jason Santa Maria 网站的哪一部分？（www.jasonsantamaria.com）

这种技术可以通过不间断的互相协作使访问者获得丰富的用户体验。在一篇被广泛阅读的论坛文章中，顾问 Jesse James Garrett 把这种应用开发方法命名为 Ajax（www.adaptivepath.com/publications /essays /archives/000385.php），这个缩略词促使这种技术更容易为市场所接受。

当你听市场人员、投资者和开发者谈论“Web 2.0”应用的时候，他们多数的意思是指那个产品用 XMLHttpRequest、XML 和 JavaScript 构建，并且用 CSS 和 XHTML 制作页面。在本书的第 3 版准备印刷的时候，狂热的“Ajax”依然在不断抢夺富媒体应用的市场份额，它也是 Facebook、Flickr 等社交网站背后的力量。

XMLHttpRequest 不仅仅是为了 Ajax，HTML、JSON、文本以及其他更多都可以异步发送，为电脑前的用户提供长时间的乐趣（www.hedgerwow.com/360/ajax/rss-json/demo.php）。事实上，虽然 XMLHttpRequest 一直能够实现这些功能，但直到 Ajax 使得网页应用像桌面程序一样生动我们才开始关注它。注意，Ajax 已经帮助 JSON 成为了 XML 的一个替代品。JSON 已经内含在大多数的 Web 后台开发语言里，包括 PHP 和 Ruby on Rails。

XML-RPC (www.xmlrpc.com)

另外一个来自 UserLand 软件公司的创新就是 XML-RPC，它是一个规范，可以使运行在完全不同的操作系统或者不同的环境中运行的进程能够通过因特网进行调用。除此之外，XML-RPC 还可以用来在 Web 发布工具中进行站点的自动化管理。

Web 发布工具

一个简短的调查显示，本章前面讨论的一些基于 XML 的软件产品，对于聪明的开发者来说是免费的。反过来，开发者们经常会为设计师、开发者和作者们提供符合他们需求的新产品。

个人信息发布工具，比如 WordPress（如图 4.1 所示）和 Movable Type（www.movabletype. org）（如图 4.2 所示）采用 XML-RPC 来进行方便的站点管理，用 XML RSS 来自动进行同步及分发内容到其他具备 XML 能力的站点。WordPress 和 Movable Type 给它们的用户提供了极其强大的内容发布功能，而正是 XML 赋予了它们生命力。

因此，随着个人信息发布工具（包括博客）的传播，XML 不仅仅被那些资深的开发人员使用，而且也被那些从来没听说过 XML 标准或者不会书写 XML 文档（有时候甚至连 HTML 也不会）的人应用来完成他们的工作。

你的服务

XML 的逻辑也促进了整个 Web 服务市场。基于 XML 的简单对象访问协议（SOAP）（www.w3.org/TR/soap），使得在分布式、平台独立的网络环境中交换数据、访问服务、对象和服务器、编码、解码，以及图片处理都变得非常便利。XML 的强大能力使得 SOAP 能够不用考虑多个平台和产品间的复杂性。

SOAP 只是在急速增长的 Web services（Web 服务，www.w3.org/2002/ws/）领域中的一个协议。David Rosam（dangerous-thinking.com）这样定义 Web Service：

> Web Service 是由基于 XML 和相关协议的可重用的软件组件构成，可以使商业体系以接近零的成本进行交互。它们可以用来在公司或组织内部

进行快速低成本的系统集成，或者放在因特网上让你的客户、供应商及合作伙伴访问。

这从商业的角度来看非常出色，但最神奇的是支持这些 Web 服务的内在的库，我们称之为 API（en.wikipedia.org/wiki/API）。API 让一个 Web 服务可以衍生出无数个派生服务。这些服务通常遵循 GNU（www.gnu.org/copyleft/gpl.html）协议或者创作共用协议（www.creativecommons.org）协议，以保证那些“子产品”不陷入法律纠纷中。

因为 Google Maps（maps.google.com）、Flickr（如图 4.3 所示）和 Amazon.com 支持 API，独立开发者可以使用中心数据来建立自己的分散服务。在 2005 年，芝加哥的记者和 Web 开发者 Adrian Holovaty，开源的 Django Web 框架作者之一，创建了第一个前 API 的 Google 地图聚合应用：创建了 chicagocrime.org （如图 4.6 所示）。这个网站在 Google 的开放地图 API 的过程中起到了一定的作用。Holovaty 后来在此基础上更进一步，建立了“一个微社区新闻实验”网站 EveryBlock （如图 4.7 所示）。

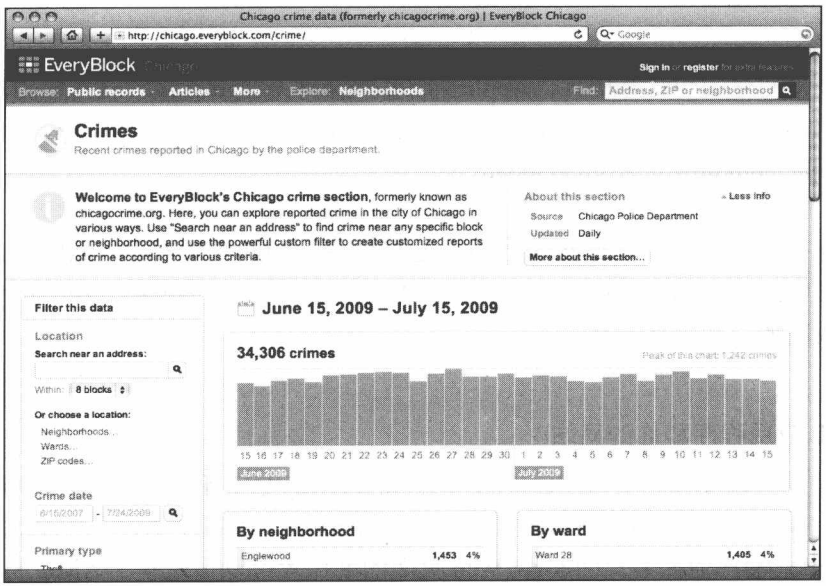


图 4.6　建在巨人们开放 API 肩膀之上，EveryBlock 网站的芝加哥犯罪部分（chicago.everyblock.com/crime，前身是 chicagocrime.org），通过从芝加哥警察部门获得的信息能够在 Google 地图上显示犯罪在哪里发生。这是 Adrian Holovaty 发明的，也是第一个前 API 的 Google 地图聚合应用，这鼓励了 Google 更加开放它的地图 API

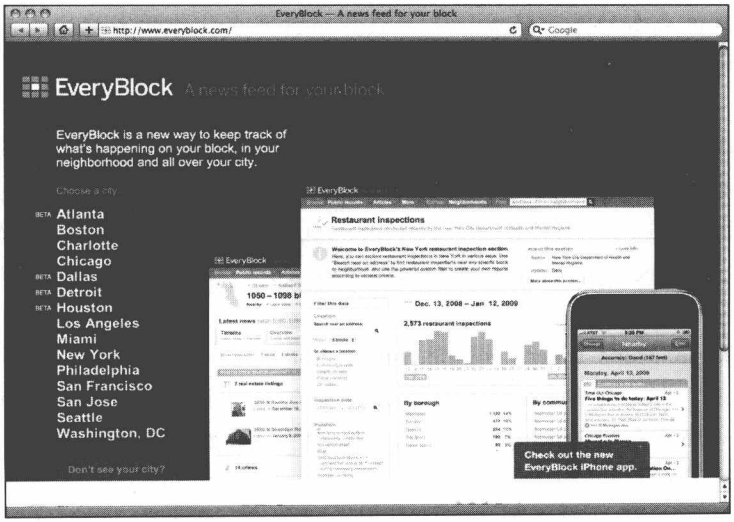


图 4.7　EveryBlock（www.everyblock.com）建立在 chicagocrime.org 基础上，采用 Web 标准技术将数据世界和新闻结合起来，这是否是报纸的未来呢？

在去中心化的想法基础上，Apple 的 Dashboard Widgets 技术就是让用户自己写应用程序（通过 XHTML、XML、CSS 和标准 JavaScript）把远程数据拉到自己桌面（www.apple.com/downloads/dadhboard）。

是什么让 Widgets 以及 EveryBlock 这样的网站如此激动人心？原因就是他们拥有的知识以及勇于创新的精神。他们就像 19 世纪后期的无声电影一样，虽然现在还只是自我陶醉，却预示着爆发。

XML 应用程序和你的网站

可缩放矢量图形格式 SVG（www.w3.org/TR/SVG/）和可扩展超文本标记语言 XHTML（www.w3.org/TR/2002/REC-xhtml1-20020801/）都基于 XML 语言。Illustrators 用 SVG 格式输出他们客户的 logo，Web 页面的创作者可以使用基于 XML 的 XHTML 来编辑他们的页面，不管他们是否明白 XML。

因为都基于 XML，拥有相同的规则，所以这些语言能够很容易在一起工作，也能和其他类型的 XML 一起协同工作，例如数据库里存储的 XML 格式的数据。一个

SVG 图形可以根据访问者的检索自动产生变化，或者根据从 XML 格式的新闻数据流获得的数据不断地更新。

例如，当地电视台新闻频道的网站可以使用这种功能在拥挤高峰时段提供实时地铁交通信息。当一个交通堵塞消除后，或者另外一个堵塞开始的时候，新闻流把这个信息发送到服务器，在那里这些信息被格式化为 XHTML 格式，变成用户可以阅读的文本内容，并且可以用 SVG 更新交通情况图。同时，这些数据还可以通过 RDF、RSS 和其他新闻组织共享，或者采用 SOAP 来帮助城市管理者查找问题并进行相应的处理。

虽然基于 XML，SVG 图像可以非常容易地在一些软件产品中生成，比如 Adobe 公司的 Illustrator 10 CS2（www.adobe.com/products/illustrator）。类似 Flash 矢量图像，只要用一点点带宽，SVG 图像就可以扩展充满整个显示器屏幕。和其他标准的 Web 页面组件一样，SVG 的图像可以通过标准 JavaScript 和 DOM 进行操作和处理。而且，SVG 文本内容在默认状态下可以用鼠标进行选择，不管它是伸展的还是变形的。Firefox 默认内置 SVG 支持插件。

与生俱来的兼容性

因为都是源自相同的技术并且遵守同样的规则，所有的 XML 应用程序都互相兼容，这样可以使得开发者们很容易根据需求通过其他程序处理这个 XML 数据，或者直接开发新的 XML 应用程序，整个过程都不必担心产生不兼容。

今天，XML 已普遍存在于专业和大众软件中，广泛应用在中间件和后台系统的开发中，是新兴的 Web 服务市场领域的本质。XML 的成功超越了所有人的想像力，它解决了“不兼容和技术死角”的噩梦。

软件制造商们，都不会冒着流失客户的风险来排斥 XML，他们认识到支持 XML 会使他们的软件产品能和其他的软件产品一起工作，并且能在持续变化的市场中生存下来。经营管理者和 IT 专家，也都不愿让专有的遗留系统继续控制他们企业最宝贵的数据资源，通过把数据转化成开放的 XML 格式可以解决这个问题。独立的小型开发团队通过 XML 的力量能与大公司进行竞争。在 XML 上，只尊重智力而不是投入的预算。

在当今数据驱动的时代，专有格式的数据将被淘汰，即便它们曾经非常流行。XML 降低了技术的门槛，任何人都可以参与。XML 是一个 Web 标准，而且它已经在工作着。

好的标准有一个特点：用这种标准能完成一项工作，并且能和其他标准配合得很好。这称为可互操作性（W3C 这样称呼），或者可以称之为组件协作。不论你怎么称呼它，XML 比过去任何专有的 Web 技术都有巨大的进步。在 Web 标准下，竞争者们也学会了协作。

4.1.3 标准的未来

感谢 Web 标准组织（WaSP），使得浏览器市场也学会了支持相同的标准。技术的合作带来了意想不到的结果，这些曾经苦苦竞争的公司也学会了在一起很好协作，而且经常是在一些令人吃惊的方面。

在 2002 年 7 月，Microsoft 向 W3C 的 HTML 工作组提交了“用于支持 W3C 的 HTML4.01 测试开发套件的一套测试及可测试性的声明”（http://lists.w3.org/Archives/Public/www-qa-wg/2002Jul/0103.html）。这是由 Microsoft、Openwave 系统公司、美国在线、Netscape 和 Mozilla 共同贡献的。Opera 软件公司（Opera 浏览器的开发者）和 Web 标准组织也一同参与了讨论。

校验程序及其规范

W3C 的校验程序可以让浏览器的开发者们检测他们的软件是否符合标准，或者是否还需要做更多的工作。现在还没有针对 HTML 4.01（也是 XHTML 1.0 基于的标记语言）的校验程序。当缺少校验程序的时候，那些浏览器开发商只能通过他们自己的方法来遵从标准。

此外，当缺少校验程序的时候，标准的制定者发现，他们自己处于一个待验证的状态。当你缺少实际验证环境的时候，怎么能确认你发明的技术可以充分解决那些问题？这就像在纸面上设计一个汽车，而如果没有汽车工厂是不可能建造出你想要的汽车的。

为了标准制定者本身的利益，也为了浏览器开发商的利益，校验程序姗姗来迟。

校验程序是怎么工作的

当 Microsoft 着手开始着手解决由于缺乏校验程序而引起的问题时，他们并没有排斥竞争对手，而是邀请他们及外部组织（WaSP）来参与这个标准兼容的工作。就像大家所期待的，那些竞争对手和外部组织十分乐意地接受了这个邀请。虽然这些工作的专利技术或者版权都是免费的，而且工作的结果或者其他衍生的技术都归 W3C 所拥有，而不论 Microsoft 还是他的竞争对手们都没有意见。

通常情况下，Microsoft 不会关心怎么做对 Netscape 是最好的，不论 Netscape 是否对它的帮助感兴趣；更不会浪费脑细胞考虑对 Opera 有益的事情。其他公司也不会把金钱浪费在无私的冒险上。但是，为了 Web 的健康发展，他们都聚到一起来了，相互合作，不是为了争夺一个时髦的专利技术，而是为了“小小”的 HTML 4。

忽略那些商业新闻的炒作，这个事件本身意味着巨大的改变。Microsoft 大方地给 W3C 贡献出了“HTML 校验程序”，是给商业对手们一个信号，Web 还将按照这种方式继续前进。不再因为私有“创新”技术而忽视标准，Web 标准使浏览器制造商结合在了一起。

这是符合自然规律和商业逻辑的，下一步浏览器厂商将采取“联合创新”的方式建立新的 Web 标准，而不是被动地等待 W3C。由于 XHTML 2.0 方向的不确定性、W3C 流程的繁复以及 Web 2.0 驱动的应用重于文档倾向，接下来发生的事情是不可避免的。

4.2　HTML5 的诞生

2005 年，在 Ian Hickson 的领导下，来自 Mozilla 基金会和 Opera 软件公司的工程师们建立了 WHAT（the Web Hypertext Application Technology 的缩写，意为 Web 超文本应用技术）工作小组（www.whatwg.org）。“这是一个针对浏览器开发商和对此感兴趣的合作伙伴的、松散的、开放的合作组织”，它的目标是“通过制定将在主流浏览器中实现的技术规范，为统一的 Web 应用开发环境定义需求。”

虽然 Mozilla 基金会和 Opera 软件公司也是 W3C 会员之一，但 WHAT 的工程师有时候也对 W3C 的标准开发进度缓慢而抱怨。WHAT 重点关注的是实用的浏览器相

关问题，它对 HTML 的偏爱超过了 XML，并保持与 W3C 相对独立。当然 WHAT 选择与 W3C 合作并不是对着干，故 WHAT 很快提交了 HTML5 第一个草案给 W3C 等待批准。

通过在不同公司产品线的工作和对重点领域的关注，例如，指定所有浏览器应该怎么处理 RDF 控件、菜单和工具条，WHAT 小组希望快速跟进 Web 标准，理顺浏览器开发，从而让所有浏览器都统一支持更新的标准。

我们将在第 7 章“HTML5：新的希望”详细探讨 HTML5。现在，我们先看看 HTML5 的一些目标以及它们将如何突破现有的标记。

HTML 5 带来新的语义

虽然 CSS 是一种布局语言，但它是没有语义的，它也不表明页面的结构。HTML 和 XHTML 虽然包含了文档的大纲结构，但也没有明确提示页面结构。HTML5（www.whatwg.org/html5）的提出改变了这种状况，并解决了“DIV 堆砌”的情况，采用类似“header、nav、footer、section 以及 aside”等布局元素来说明页面布局。Lachlan Hunt 的文章“HTML5 预览（www.alistapart.com/articles/previewofhtml5）”解释了使用简单、美观清晰元素背后的意图。在同一篇文章中，他也解释了 HTML5 如何增强窗体（form）、API 和多媒体控制，这样将“给作者更大的灵活性和更强的互操作性。”

Lachlan Hunt 是 HTML5 的支持者，John Allsopp 则是反对者，在文章“HTML5 的语义”（www.alistapart.com/articles/semanticsinhtml5）中，他解释道：

> 我们需要为 HTML 建立一种机制，可以让开发人员明确无误地为标记添加丰富的、更容易理解的语义。这也许是 HTML 项目最紧迫的唯一目标。
>
> 但建立这个机制不是那么简单：任何解决方案都有许多限制，最重要的限制也许就是向前兼容。解决方案不能破坏已有的成千上万的网页，它们将继续被使用许多年。任何无法向后兼容的解决方案都不会被广泛的使用，开发者会很快畏缩。

> 解决方案也必须向后兼容，并不是说它必须工作在未来的浏览器上，这是浏览器厂商的责任，而是说它必须可扩展。我们不能指望一个单一的解决方案就能满足我们未来对语义的所有需求。我们需要开发一个可以扩展的解决方案，以满足未来的需要。
>
> 这两个制约因素是目前面临的巨大挑战，但是作为一个存在十年以上的语言环境，而且它在全球沟通平台中的地位不可取代，这个挑战必须被解决。

对 HTML5 的其他问题担心包括：它对不良 HTML 的宽容阻碍了干净的、结构化的、语义标记的发展运动；将网络的未来交给一个有着相当（或没有）固定想法的小团体的手里是否公平稳妥，以及制定这个规范过程的疑问。（目前，有两个团体在同时研究 HTML5：一个是 WHATWG，由 Hickson 领导；另一个是 W3C 工作组，也是由 Hickson 领导。如果 Hickson 在 WHATWG 小组中增加一个元素到 HTML5，而同样由 Hickson 领导的 W3C 工作组却不接受这个元素，由 Abbott 和 Costello 表演的经典的小品“谁是第一个？”相当有趣，可是如果未来的标记语言需要这样不停地平衡，就会有麻烦。）

在 2009 年 7 月 2 日，W3C 宣布它停止了所有关于 XHTML2 的工作（www.w3.org/News/2009#item119），人道地解决了 XHTML 2 的痛苦。但这也使那些对 HTML5 有疑问的人们更加不安，让那些“标准人”怀疑 XHTML 已经死亡（www.zeldman.com/2009/07/07/in-defense-ofweb-developers）。（既然 XHTML2 没有继续，那么 XHTML1.0 将继续工作直到你我退休，而 XHTML2.0 中最好的思想正在被转移到 HTML5 中，HTML5 将支持 HTML 和 XHTML 语法，就算这样也还是困扰着一些人。）

虽然本书还是愉快地推荐 XHTML1.0 过渡方法或者严格方法（正如在第 1 版和第 2 版中所提倡的那样），但是每个设计师都应当学习 HTML5，并在工作中使用现代浏览器已经支持的那部分。无论你是在个人项目中尝试 HTML 5，还是在一个真正的网站中使用它（例如我用在 aneventapart.com），这取决于你自己、你的客户、你的浏览器版本和不同的网站。那些主要设计和开发 Web 应用的人们最有可能体验 HTML 5 的力量。

4.2.1 IE 浏览器和 Web 标准

随着公司的发展，只有苹果公司比微软更神秘。2005 年后期，微软开始与 Web 标准组织 WaSP 合作，而不是秘密开发新的浏览器版本。微软希望确保 IE7 能够比以前的版本都更准确的支持 Web 标准。

当微软准备发布支持标准的杰作 IE8 的时候，再次与 Web 标准组织进行合作。颇具喜剧效果的是，在默认使用标准模式（可能导致那些专门为 IE 优化的网站出现脚本和 CSS 错误），还是需要使用一个 meta 声明（这样将需要开发人员专门将 IE 当作一个特别的浏览器来对待，否则不支持标准）来切换标准模式和非标准模式上，微软一直左右摇摆。两种方法都有合理的解释，但是当这成为微软和 Web 标准之间的问题，没有人觉得这个争论是有理的，即使微软让步也仅仅是减少一时抗议，标准之争就像拔河一样会继续你来我往。

有关详细信息，请看 Aaron Gustafsin 的“超越 DOCTYPE：Web 标准、向后兼容以及 IE8”（www.alistapart.com/articles/beyonddoctype）。文章宣布 IE8 将提供先进的、基于选择性加入的标准支持。如果你想了解更多，阅读 Eric Meyer 的“从切换到针对：一个‘标准人’的旅程”（www.alistapart.com/articles/fromswitchestotargets）、Jeremy Keith 的“他们对浏览器射击，不是吗？”（www.alistapart.com/articles/theyshootbrowsers），以及 Jeffrey Zeldman（作者本人）的“版本定位：威胁还是恐吓？”（www.alistapart.com/articles/minorthreat）。在积极的方面，现在所有的主流浏览器都能很好地支持 HTML4.01、XHTML 1.0、CSS1、CSS2.1、标准 JavaScript 以及 DOM。这些曾在 1998 年 Web 标准组织刚成立时努力追求但没期望能全部实现的目标。

只要涉及 IE，设计师和开发者就需要决定是“支持”IE6 用户，还是支持另一方面：CSS3、Firefox、Safari 和 Oprea（甚至是 IE8）。不同的决定将进入不同的旅程，我们将在第 2 部分讨论这些。

4.2.2 创作和出版工具

在浏览器大战期间开发的，市场领先的、专业的可视化编辑工具，例如 Adobe 的 Dreamweaver，通过生成标记初步解决了浏览器兼容问题，自动生成的代码针对 3.0 和 4.0 浏览器优化。在浏览器可以运行非标准、无效的 HTML 标记的大环境下，

Dreamweaver 也产生非标准的标记；当浏览器开始支持标准，像 Dreamweaver 这样的编辑工具也需要改变。2001 年，在 Web 标准组织的帮助下，Dreamweaver 开始支持标准。

Web 标准组织中的“Dreamweaver 特别小组”由 Drew McLellan 和 Rachel Andrew 领导，创建于 2001 年，致力于帮助 Dreamweaver 的工程师提高用领先的专业化 Web 编辑工具 Dreamweaver，以生成能与标准兼容的，具有高可访问性的站点。这个特别小组的历史可以通过 www.webstandards.org/act/campaign/dwtf/获取。这个小组的目标主要如下。

- Dreamweaver 应该生成有效的标记（有效的标记只使用标准的标签和属性，并且不包含任何错误）。
- Dreamweaver 应该允许在 XHTML 和 HTML 版本中进行选择，而且为其中任何一个选择匹配一个有效的 DTD。（DTD（Document Type Definition）称为文档类型定义，它会告诉浏览器在这个 Web 页面中采用的是何种标记语言，参见第 5 章）。
- Dreamweaver 应该遵照文档的 DTD，并且产生和它相对应的标记和代码。
- Dreamweaver 能够让用户轻松地创建所有人都能够访问的文档。
- Dreamweaver 能够对 CSS 2 进行更精确的处理，由此通过 CSS 格式化的页面能够在 Dreamweaver 可视化环境中正常显示。
- 在没有得到用户的同意下，Dreamweaver 不应该通过插入内联样式来破坏合法的 CSS 布局。
- Dreamweaver 的用户应该充满信心，因为他们用 Dreamweaver 创建的页面是有效的，而且具备很好的可访问性。

在 2002 年 5 月发布的 Dreamweaver MX 中已经实现上述目标，产品对标准的支持一直不断改善至今。

在 2006 年，Web 标准组织的 Molly Holzschlag 与微软合作确保它的可视化 Web 编辑器 Expression Web Designer(www.microsoft.com/products/expression)能够完全支持标准。该产品能很好地支持 XHTML，而且也像 Dreamweaver 一样支持 CSS。

没有哪款可视化的编辑器能为手工编码匹配 CSS 和语义标记，但是喜欢可视化编辑器的专业人士将有两个遵从标准的选择。（WaSP 也曾经和微软一起协作来改善 VB studio 的兼容工作。）

校验是通往快乐之路的准备

如今，越来越多的设计师开始使用 Web 标准来创建美丽的、可用的和可访问的网站了。越来越多的开发者使用 Web 标准开发出了新的产品、为数字市场提供新的思路。Ajax 是一匹新的黑马，可发现性（findability）成为王牌，现在的合同都是用（X）HTML、CSS 和 JavaScript 创建的。然而，即使最初的模版校验已经通过，客户和设计师都充分答应支持 W3C 规范，（X）HTML/CSS 校验在大型商业网站还是应用得比较少的。

过时的内容管理系统（CMS）和数据库还是会连累网站出现许多校验错误，其他的包括第三方广告系统，也会使用无效的方法和不正确的 URL 处理。但是，理解了为什么基于标准的网站却不能达到完美的校验的原因，并不等于可以纵容 CMS 厂商或者广告服务商忽视 Web 标准。

为了网络的进步并发挥更大潜力，出版工具必须符合标准。网站所有者和管理者应该告诉 CMS 厂商以及广告服务商遵循标准很重要，正如设计师和开发者告诉浏览器厂商那样。当足够多的客户都这样做时，销售商将改善他们的产品，并且 99.9% 的网站将开始远离过时和淘汰。

第 2 部分

第 5 章

现代标记语言

本书的第 1 部分描述了使用过时的 Web 设计方法所产生的创作和商业上的问题，简要地说明了使用标准来设计和创建站点所能得到的好处，并且描绘了使用这些方法将获得的美好前景。本书下面的部分将从概述转向具体的技术，我想最好从再看一眼 Web 标记语言的基础知识开始，包括我们应该使用哪种语言，以及我们应该如何标记这些熟悉的元素，例如标题、段落、列表（提示：在语义上）。

在重新考虑标记的时候，很多设计师和开发者都有思想上的畏惧。尽管那些能够花数个星期的时间来设计专业站点的人们对旧有的 HTML 语法已经了然于胸了，可是难道就不能花一点时间学习更新的、功能更强的语言吗？例如，学习一些如 PHP、Ruby on Rails、ASP 或者 ColdFusion 等服务器端的技术，不是比以前那样浪费时间重新考虑 HTML 表格和段落标记等入门技术重要吗？

这两类技术其实都很重要。服务器端技术对于创建响应用户请求的动态站点是非常重要的。在数据库中保存内容，并通过 PHP 或者类似的技术在必要时取存内容，即便是传统的信息站点也能通过这种方式获得好处。几乎所有的现代网站，包括最简陋的个人博客，都有这样的功能。就像本书讨论的 Web 标准一样，服务器端的脚本语言和基于 Web 的框架（如：Ruby on Rails、CakePHP、Django 和 Symfony）也从界面中提取内容。就像 CSS 把设计师从无意义的影响带宽的表格元素内容片断中解脱出来一样，用 PHP 那样的脚本语言以及 MySQL 那样的关系型数据库管理系统

也把站点建造者从蠢笨、劳累的手工制作每个页面的工作中解放出来。

什么是 PHP?

PHP（www.php.net）是一个开放源码的通用脚本语言，主要适用于 Web 开发并且可以嵌入到 HTML 和 XHTML 中。它采用了 C、Java 和 Perl 的一些语法特性并且非常易于学习。PHP（有些拗口的官方解释，PHP（的意思是超文本预处理器 Hypertext Preprocessor）有很多功能，其中一项功能已经变得非常流行：当和 MySQL（www.mysql.com）结合起来使用的时候，PHP 能够让设计师和开发者轻松地开发动态站点和 Web 应用程序。

PHP 是免费的，这是 PHP 得以流行的原因之一（另外一个原因是：它有很多配置和调试工具）。开放源码开发者和独立的 Web 设计师一直沉醉于这个语言，用它开发网页，驱动产品，直到它年轻的表亲 Ruby 的出现.（如图 5.1、图 5.2 所示）。由于 PHP 优良的可扩展性（更别说它还是免费的），一些基本上不拥抱开放源码运动的大公司，也喜欢上了 PHP。比如，从 2002 年开始，PHP 语言就开始为 Yahoo.com 服务了（最近则是通过 Symfony[www.symfony-project.org]开源的 PHP 框架），而 IBM 采用的是 Php+Zend+CakePHP 框架（www.internetnews.com/ent-news/article.php/ 3485806）。

PHP 可以运行在 Microsoft 的服务器端平台上，但它更经常和 Apache 服务器结合一起使用。Apache 可以在 Windows 和大多数的 UNIX 平台上运行。就像大部分 Linux 软件分发版本一样，Apple 公司基于 UNIX 的 Mac OS X 操作系统也包含 PHP 和 Apache 服务器软件。

PHP 并不要求网站的前端采用 CSS 布局和有效的语义标记，但是通常你会发现，基于标准的网站，它的背后都采用了 PHP。

但是如果那些动态生成的页面的可访问性不高，并且不能兼容大部分的浏览器和设备，或者有很多混乱的垃圾代码，那它们就可能没什么多大用处。如果那些动态页面不能在浏览器及一些设备上显示，或者不能在 60 秒钟内完成在拨号网络下的加载，那么服务器端的技术就不可能发挥它们的全部功能。如果因为你的内容淹没在毫无意义的语义标记里面，导致你的用户或者搜索引擎找不到你的网站，那么你所有的优质内容，良好设计，智能后台处理都是没有意义的。

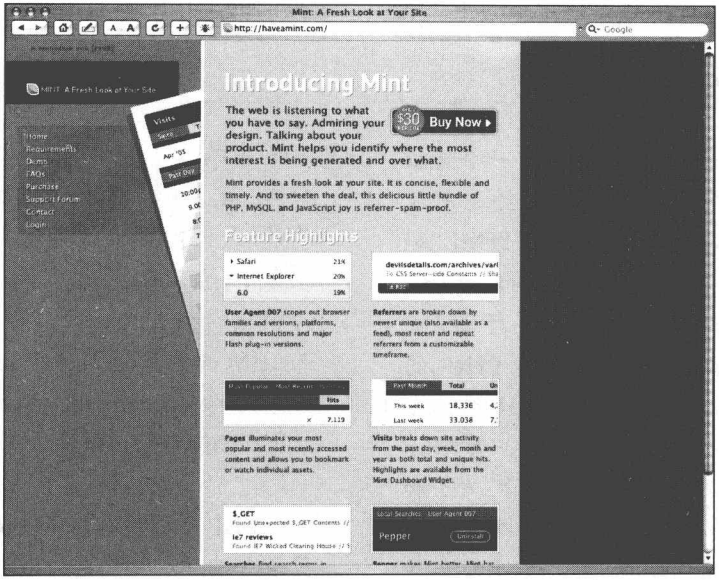


图 5.1　设计师兼代码开发者的 Shaun Inman 使用 PHP、MySQL、JavaScript 和 CSS 开发了 Mint（www.haveamint.com），一个可扩展的网站访问报告工具，它可以用来告诉你哪些人访问了你的网站，哪些人连接了你的网站，以及他们使用的浏览器还有其他更多的信息

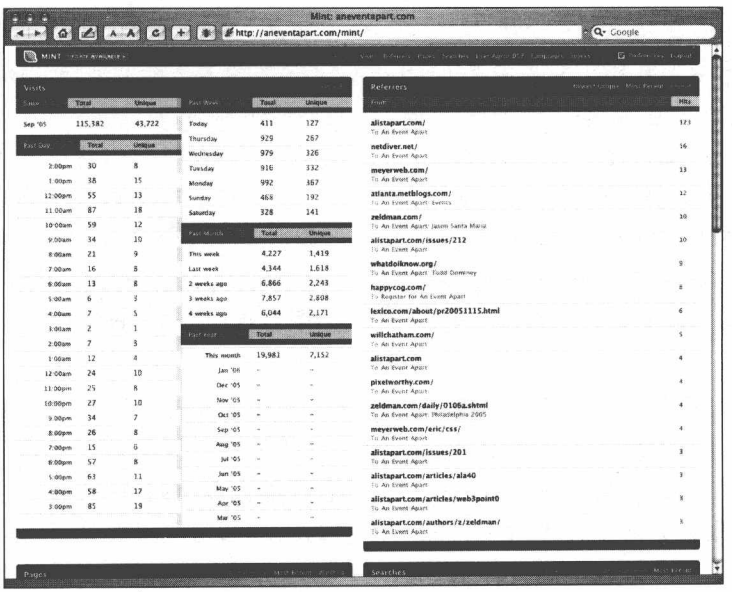


图 5.2　一个典型的 Mint 安装后的界面（该图来自 An Event Apart 会议网站）

什么是 Rails?

Ruby on Rails（www.rubyonrails.org）是一个专门为快速、产品化开发和持续编程进行了优化的开源 Web 框架（如图 5.3 所示）。可能你会问什么是 Ruby? Ruby 是一种由 Yukihiro Matsumoto 在 1995 年开发的面向对象编程语言，并且在开放源码许可下作为免费软件分发。它综合了 Perl 和 Ada 类似语法和面向对象的特征，并且还吸收了 Lisp 和 Python 其他语言的一些特性。

图 5.3　Ruby on Rails（www.rubyonrails.org），一个开源开发框架。它的指导原则是："不要重复"和"约定优于配置"

那什么是 Rails? Rails 是一个 MVC（模型-视图-控制器）框架。它是由 David Heinemeier Hansson 在 2004 年 7 月从 37signals 的 Basecamp 项目管理程序（如图 5.4 所示）中剥离出来的一个框架。（1.0 版本在 2005 年 12 月份发布并开放代码），由于相信它的指导原则能够让开发者们能够更好、更快地编程，许多开发者都投入其中。

在大多数的编程环境下，为了在 Web 屏幕上增加（或者减少）一个变量，你都必须书写冗长的代码。并且每次你坐下来创建新的应用程序时，任何一个小小的改动都必须要这么做。这就好比你每次需要书写商业信笺的时候都需要创建一个新的字处理程序一样。Ruby on Rails 使得我们可以抛弃这种模式。使用 Rails，开发者不再要为任何小小改动而编程，他们只需要更改一下配置。类似 PHP，Rails

也不必有一个标准兼容的 Web 前端展现，但是两者经常结合在一起。XHTML、CSS、JavaScript 和 Ruby on Rails 组合在一起就能搭建一个流行的网站应用，如 Twitter（如图 5.5 所示）。传统的网站应用如 yellowpages.com（如图 5.6 所示）也已经使用 Ruby on Rails（和 XHTML）了。其他语言，如 PHP 和 Python，也已经提供类似 Rails 的 MVC 框架了。

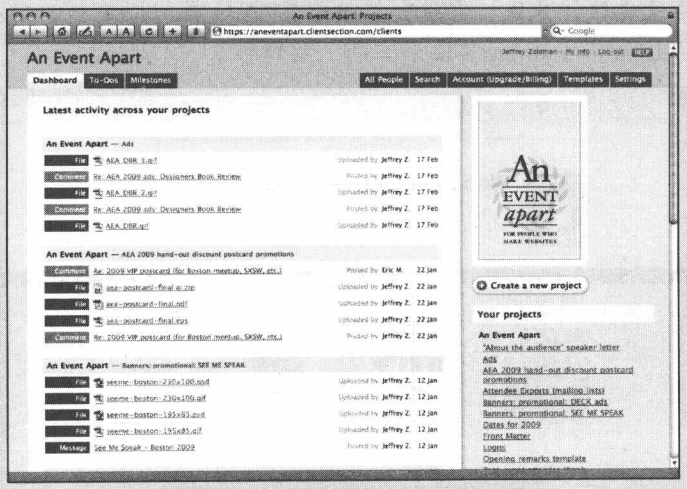


图 5.4　37signals 的 Basecamp（www.basecamphq.com）衍生出了很多创意和应用。它不仅是一个非常完美的项目管理应用，还是 Ruby on Rails 得以诞生的母体

图 5.5　（www.twitter.com），一个疯狂流行的消息服务，它由 CSS、XHTML1.0 Strict 和 Ruby on Rails 构建

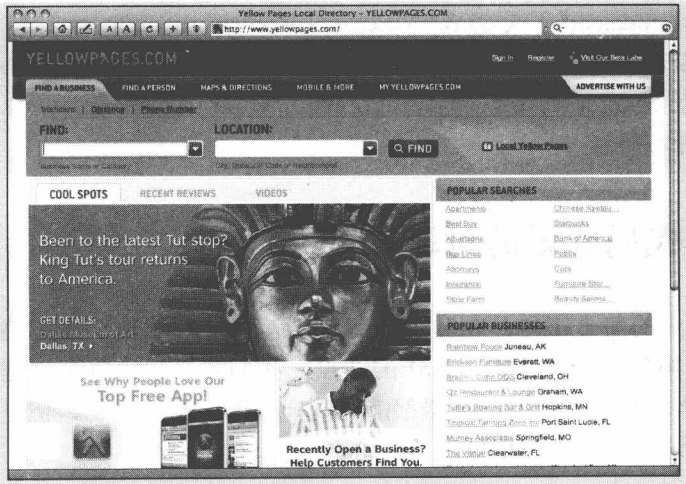


图 5.6　黄页网站（www.yellowpapes.com）由 Ruby on Rails and XHTML 1.0 Transitional 搭建

简而言之，服务器端技术和 Web 标准缺一不可，服务器端和数据库技术越发达，站点就越强大，但是只有当这些站点的内容在语义上更加结构化，它们才能驱动这些内容达到最好的境界。而这些恰恰是我们大多数人（也是大多数内容管理系统）忽视的地方。

其他平台

Microsoft 公司的 ASP.NET 2.0（www.asp.net），还有 Adobe 公司的 ColdFusion8（www.adobe.com/products/coldfusion/）是另外两个流行的可以用来创建动态站点内容的脚本语言。这两种语言都有自己的优点，而且他们都有大量狂热的爱好者和用户社区。Java 服务器端脚本（JSP）是另外一种动态技术，一般应用于大型企业系统，因为这些内容超出了本书的范畴，所以我们不再介绍。

注意，除非你特别小心，一些用以上语言开发的信息发布系统会破坏你符合标准的模版。在我的设计公司参与的一个使用 ASP（pre.NET 2.0）开发的应用中，所有的页面渲染都是非法的，直到我们引进了 HTML Tidy（tidy.sourceforge.net）来进行检查，情况才有所改善，这就像往一件干净的衣服上扔泥巴，然后再把它们冲洗下来一样，但是这种方式也让我们从一个纵容不良代码的系统中建造出了合法的网页。如果不用 Tidy，也可以使用别的一些工具，例如 TinyMCE

（tinymce.moxiecode.com）或者 WYMEditor（www.wymeditor.org）来帮助建立良好的代码。准标记语言，类似 Dean Allen 的 Textile（www.redcloth.org/hobix.com）和 John Gruber'的 Markdown（www.daringfireball.net/projects/markdown），也可以帮助站长避免非标准的标记。

5.1 垃圾代码标记的可耻秘密

在这个行业的前十年内，Web 设计就像给满满一屋子刚学会走路的挑食宝宝喂饭一样。为了构建一个能够完美工作的站点，我们就必须给每个浏览器不同口味的食物。2001 年以后，当今的浏览器都能吃同样营养口味的食物，但是大多数专业人员还没有掌握这些，还在使用原来的方法。

劣质的食物会使动脉硬化，牙齿腐蚀，还让人们缺乏活力。同样，垃圾代码标记会经常达不到用户的短期需求，并损害站点内容的长期健康。但是直到今天，事实上大多数浏览器还容忍着那些隐藏的垃圾代码，就像我们在第 1 章中讨论的一样。

在本章和后续章节中，将探讨我们忘记已久的清晰的、有语义的代码标记，并且学习怎么用结构化的思维方式代替那种认为 Web 标记只是二流设计工具的想法。同时，我们还要考察 XHTML 1.0，当前的 Web 页面标记标准语言，讨论它的目标和好处，并且探讨从 HTML 转化到 XHTML 的策略。我们也将开始考虑 HTML5（www.w3.org/TR/html5）这个新鲜事物。

不正确的 URL

脚本语言生成的页面经常带有很长的 URL 链接，包括在 HTML/XHTML 中禁止的符号“&”。在 HTML 和 XHTML 中，这个字符（&）用于表示一个实体，比如’，它表示印刷体的撇号。在 ColdFusion 中，这个问题可以通过使用一个名叫 URLEncodedFormat()的函数来解决。ASP 同样有个类似的函数，叫做 HTMLEncode。PHP 则是用 urlencode()（us3.php.net/manual/en/function.urlencode.php），rawurlencode()（us3.php.net/manual/en/function.rawurlencode.php），以及 htmlentities()（us3.php.net/manual/en/function.htmlentities.php）。通过以上所有方法，在输出之前通过这些函数传递 URL，开发者们可以（也应该）能够避免这个问题。

一个奇怪的巧合就是，正确的 XHTML 编码方法鼓励结构化的代码标记，不鼓励在它内部进行表现层的处理工作。而在“XHTML 1.0 Transitional（过渡版本）”中，这样的处理方法只是“不鼓励的”，这就是说，如果必须使用，那么你还是可以使用的。但是它更鼓励我们用别的方式来达到相同的设计效果（比如，使用 CSS）。在 XHTML 1.0 和 1.1 文件中，XHTML1.0 及 XHTML1.1 严格版本中，任何对表现层的处理都是禁止的：如果在你的页面中使用它们，你的 Web 页面将无法通过 W3C 的标记校验。W3C 这个标记校验服务是知名的“The Validator（校验服务）”（如图 5.7 所示）（如果你现在还不熟悉它，那么在学习基于标准进行设计和构建 Web 站点的过程中会了解的。见词条，“进行校验”）。

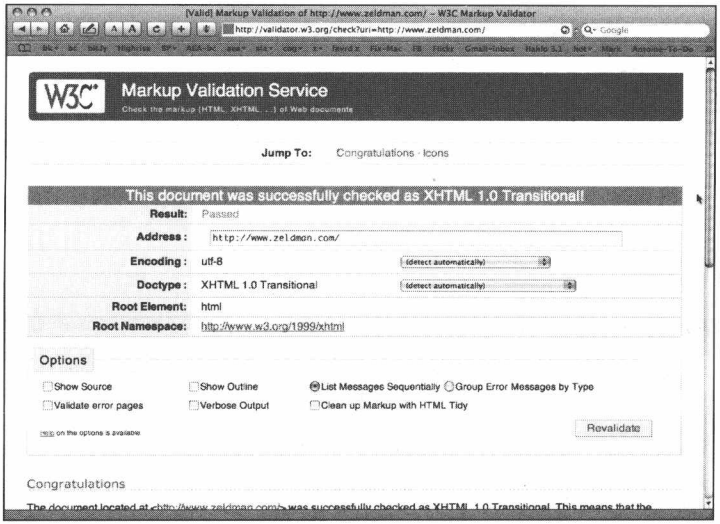


图 5.7　设计师和开发者们可以使用 W3C 免费的在线校验服务（validator.w3.org）来确认他们的页面符合标准

进行校验

W3C 的校验服务（http://validator.w3.org/）可以测试用 HTML 4.01、XHTML 1.0 和 XHTML 1.1 制作的网页是否符合标准规范。对于样式表，它也提供了相应的 CSS 校验服务（http://jigsaw.w3.org/css-validator/）。在 htmlhelp.com 的 Web 设计组织也维护着一个经常更新的可靠的标记语言校验服务（http://www.htmlhelp.com/tools/validator/）。这三种服务都是免费提供的。

提示：对于那些高级开发者们，如果你在用 HTML5，那么并不是所有校验服务都能校验你的工作。而 validator.nu （如图 5.8 所示）的服务能够校验 HTML5。W3C 的校验服务现在也能很好地校验 HTML5 了。快去试试吧。

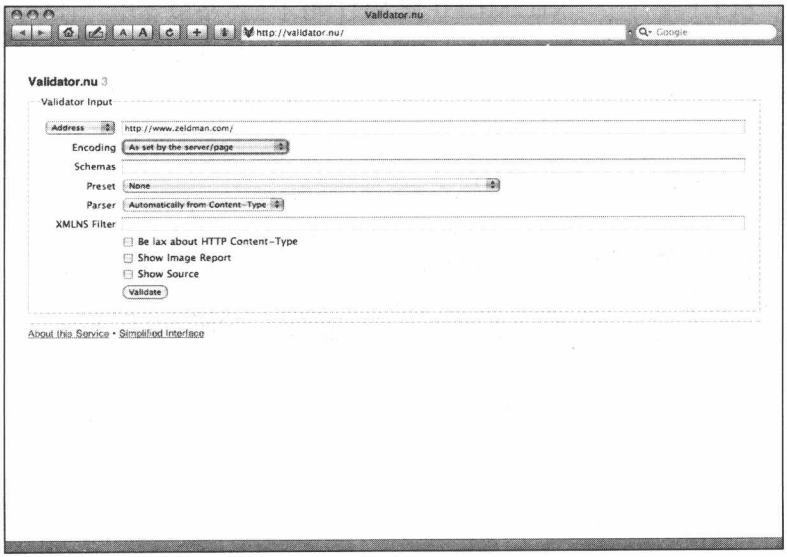


图 5.8　看上去不太起眼，但是 Validator.nu 是一个强大的校验工具，可以校验 XML、hHTML5 等。（www.validator.nu）

不论选择 XHTML 严格的方案还是过渡方案，你会再次发现一个令人羞愧的事实："您所有知道的东西都是错误的。"换行标记（
）曾被您雪花般地撒开来用来模拟一个列表；<h>不是用来作为标题等级划分而是用于显示需要；透明的像素 GIF 图形，则用来生成空白（whitespace）：之后你才会发现它们真正的本义。

你要开始学会结构化的思考，而非着重于表现层的处理工作。要让标记回归标记的原本意义。即使采用了一些表现层表格和其他不推荐使用元素的过渡布局，你还是可以学习用 CSS 做更多的工作，比如清除您的 XHTML 中复杂和冗余的表格单元颜色和对齐属性，用全局样式表中的一个或者两个规则来代替它们。当我们学习一个 Web 标记的新语言时，也可以同时忘掉我们多年养成的坏习惯。好了，让我们努力研究吧！

5.1.1　重新阐述了什么

根据 W3C 的官方解释，“XHTML（http://www.w3.org/TR/xhtml1/）是一个用 XML 语法对 HTML 重新阐述的语言”。如果用稍微准确一点的语言来说，XHTML 是一个基于 XML 的标记语言，并且看起来和 HTML 有些相像，只有一些小的但却很重要的区别。虽然有些主流的现代浏览器可能对它们处理起来有些不同，就像我们将在第 6 章“HTML：Web 重构”中讨论的，但对于 Web 浏览器和其他用户代理程序，XHTML 的工作方式和 HTML 完全相同。对于设计师和开发者，用 XHTML 1.0 设计 Web 和用 HTML 也基本一样。不过两者还是有些规则不同，并且 XHTML 还有一两个新元素，我们下面会接着讲述。

在前面，我们讨论了 XML，也叫可扩展标记语言，一个“超级”的、可以被程序员用来定制其他标记语言的语言。XHTML（可扩展超文本标记语言）就是这样一种用 XML 定制的语言。XHTML 1.0 是最早的，也是向前兼容程度最高的 XHTML 版本，从此对于浏览器和其他用户代理来说，有了容易学习的、最大兼容、最少麻烦的版本。

其他基于 XML 的应用程序和协议非常多，它们流行的部分原因是因为它们具有花更少的代价和产生更少的兼容问题而进行数据交换和转化的能力，这也是使用 XHTML 的优点。类似的协议包括富站点摘要 RSS（blogs.law.harvard.eud/tech/rss），可缩放矢量图形 SVG（http://www.w3.org/TR/SVG）、同步多媒体集成语言 SMIL（http://www.w3.org/TR/SMIL）、资源描述框架 RDF（http://www.w3.org/RDF）（在第 4 章有更多的介绍）

所有这些协议都在当今新兴的 Web 领域扮演着重要的角色，但是没有一个能像 XHTML 那样，对设计师和开发者有那么重要，也没有 XHTML 那么容易学习。

为什么要用 XML 来“重新阐述”HTML 呢？只有一个原因：XML 是一致的，而 HTML 则不具备这个特性。在 XML 中，如果你开始了一个标签，就必须在后面有它的结束标签。而在 HTML 中，有些标签从来没有结束标签，而有些必须要有，还有一些可以通过设计师的意愿来决定是否要有结束标签。这种不一致性会引发一些实际问题。比如，即使在 HTML 中是可以不闭合表格（<td>）标签，但部分浏览器会拒绝显示此类页面。XHTML 强迫我们必须要结束所有的元素，从而可以用来帮

助我们避免类似的浏览器问题，并且能够减少测试和调试的时间，同时不需要浪费脑细胞来记住哪些标签要结束而哪些不需要。

更重要的是，如果你用基于 XML 的语言开发页面，你的站点就会很好地和其他基于 XML 的语言、应用程序和协议进行交互。

如果 XML 那么重要，为什么还要创建一个工作方式类似于 HTML 的、基于 XML 的标记语言 XHTML 呢？这是因为虽然 XML 非常强大而且无处不在，但现在原始的 XML 数据还不能服务于大多数 Web 浏览器，还不能期待它们更智能地做一些事情，比如显示一个精细格式化的 Web 页面。旧的浏览器也不能识别原始的 XML，在本质上说，XHTML 是一个桥接（过渡）技术，结合了 XML（有几分）的强大功能及 HTML 的简单特性（在很大程度上）。

5.1.2 执行概要

大体上可以这样理解，XHTML 就是一个扮演着类似 HTML 的角色的 XML，它可以同时显示在新旧 Web 浏览器中，也同样可以工作在大多数的因特网设备中，比如从古老的（20 世纪 90 年代）Newton 到 Palm 和 IPhones，这使得它变得轻便、实用和有效率。

XHTML 就像 HTML 一样易于学习和使用，对于那些没有坏习惯需要忘记的初学者是很容易的，可能对于那些早在 20 世纪 90 代就开始坚持 Web 设计和开发的老手倒稍微有点困难。

XHTML 是当前标记语言的标准（用来替代 HTML 4），而且它用来为 Web 内容设计严格的、逻辑的文档结构，并且能够很好地和其他的 Web 标准（例如 CSS 和 DOM）进行协作，还能够很好地与其他现有的、未来的基于 XML 的语言、应用程序和协议进行交互。我们先看看下面这些重要的内容，然后再来列出 XHTML 的好处。

严格版还是过渡版

在本书的前两个版本中，我们推荐的是更加宽容的 XHTML1.0 过渡版本，这种过渡能力是 XHTML 最大的好处：与现有开发方式有最好的兼容性，并且非常易于学习和进行转化操作。如果你在忘记一些旧的习惯或更新一个原有站点的话，XHTML1.0 过渡版是一个好选择。

另一方面，现在许多标准的狂热追随者们更倾向于喜欢 XHTML1.1 更严格的标准，甚至 XHTML 1.1 严格版，如果他们能解决 IE 之外的浏览器 MIME 格式问题的话。默认选择严格版是一种流行趋势，它可以告诉世界（至少是查看源文件世界）你无条件支持标准，而不仅仅是兼容。

5.1.3　XHTML 2 为你我而生

在撰写本书第 1 版的时候，XHTML 2.0 的规范草案已经提交给开发社团进行讨论。六年后当我进行第 3 版的创作的时候，XHML2.0 还处于草案的阶段（www.w3c.org/TR/xhtml2），这 3 年，它并没有更新。这可能表明，业界并没有追捧 XHTML2，事实上，尽管它有很多有意思的想法，但在开发者社区并没有得到推广。2009 年 7 月 2 日，W3C 放弃了 XHTML2（www.w3.org/News/2009#item119），网页标准团体抓狂了（www.zeldman.com/2009/07/07/in-defense-of-web-developers）。

XHTML 2.0 原本准备向语义化靠得更近，甚至意味着要抛弃目前的开发方式。这个初始化草案纯粹是个理想化的版本。XHTML2.0 故意设计成不向前兼容于 HTML 和 XHTML 1.0。它放弃了一些我们熟悉的约定，包括 img 元素（而采用 object 来代替），
标记（用一个新的<line>标记，后来还变成了<l>），还有历史悠久的 anchor（锚链接），而用新的标记叫 hlink 来替代。

开发者最抱怨的是在最近的草案中，<a>（anchor）元素又回来了。但是在修订版本的 XHTML2.0 中，任何页面元素都可以拥有一个 href 属性，必须说明的是，许多开发者都喜欢让每个元素都拥有 Href 属性的这个想法，当然，HTML5 也有同样的思路。到目前为止，唯一的问题是，大多数浏览器还不支持这个。

至于 img 元素，它最终出现在 XHTML2 中（www.w3.org/TR/xhtml2/mod-image.html#sec_20.1），但不推荐使用。我们还是假设要用 object，尽管 Object 不能在 IE8 之前的任何 IE 浏览器工作（说好听一点，IE8 终于支持 Object 了，酷!）。

只有少数人对初始版本的 XHTML2.0 规范有些许欢迎，其他人都抱怨它太过理想，并且和实际 Web 建站之间存在沟渠（好吧，这么说的可能就是我）。大多数设计师认为它根本就不值得关注。六年过去了，除了一些技术前沿人士在关心之外，大多数人还是不理会它。相比需要支持 2 种“未来的标识”语言来说，我们宁可只专注在其中一个的开发上，W3C 明智地决定停止 XHTML2.0 的左右相关工作。

Xhtml 2.0 可能会死掉，但是没有浏览器会停止支持 XHTML1.0，因为没有浏览器打算停止支持 HTML4。符合 HTML 4.01 规范的网站仍然会在未来几年内维持正常工作状态。这同样也适用于完全用 XHTML 1 规范开发的网站。

另外，如果 HTML5 成为将来网站建设的语言，那为什么还用 XHTML?这是一个公平的问题，花费 10 年推广 XHTML 的我们会在夜深人静的时候问问自己.，如果最终版的 HTML5 即将来临，如果明年所有浏览器都支持 HTML5 的新元素（例如，任何元素都有 href 属性，又例如，页面结构元素 footer 等），增加网站 XML 应用的兼容性比费劲去学习 XHTML 更有意义，但是 HTML5 仍然没有完成，尤其是 IE 不支持该语言的大部分新增内容，因此，就目前来说，在 HTML 和 XHTML 之间进行选择有 10 个主要的理由，我把它们归结到下面的列表中。

5.2 5 个坚持用 HTML 的原因

1. HTML 可以工作在所有的浏览器上，所有浏览器（包括 IE）都能正确地解析 HTML MIME 格式。

2. 虽然 HTML5 不可能在几年内完成，但是所有现代浏览器都支持它的的部分功能，现在是学习强大的，新版的 HTML 的最佳时机。

3. HTML 的容错性强过 XHTML。

4. HTML 没有像 XHTM 一样要求每个元素都需要关闭，因此，占用带宽少。(HTML5 的 DOCTYPE 比任何其他 DOCTYPE 占用带宽都少。)

5. HTML5 是第一个为富网络应用设计的标识语言，这也就是为什么像 google 这样巨大的互联网公司押宝在这个上面的原因。如果你工作在网络应用中，如果你觉得可以接受 HTML5 的方向，那就潜心研究它吧。

另外，在当前任何浏览器中，一个存在的 HTML 文档声明不再自动触发 Quirks 模式，这一点相比 XHTML 而言不是什么优势，但是 HTML 的使用不再使开发者限于越来越多的不必要的 Quirks 模式风险中。在第 7 章，我们将讨论 HTML5，它和 XHTML 有何不同，以及它的元素、规则和语法。

5.3　5 个使用 XHTML 1 的原因

1. XHTML 是当前用来代替 HTML 4.0 标记语言的标准。

2. XHTML 就是为和其他基于 XML 的标记语言、应用程序以及协议进行良好的交互工作而设计的，这正是 HTML 缺少的优点。

3. XHTML 比 HTML 有更好的一致性，所以它很少会产生功能和显示的问题。

4. 用 XHTML 可以帮助你去除书写表现层标记的坏习惯，它也能帮助你避免可访问性问题和在不同厂商的浏览器上显示不一致的问题。（如果使用结构化的 XHTML 并且把它所有或者大部分的显示采用 CSS 控制，你将再也不用担心自己的页面在 Firefox 和 IE 上产生差异，比如说，设置了宽度的空表格单元。）

5. 正如练习打节拍器，能让给你成为一个更好的音乐家，XHTML 的重点是良好的格式和规则给常年书写无意义的“Tag soup（tag 堆砌）”HTML 的设计者和开发者提供了一个完善的平台。就算从现在起 2 到 3 年时间切换到 HTML5，你也会因为在严格的条件下学习了语义标记而写出干净，有效的标记。

5.4　不使用 XHTML 1 的原因

1. 你不知道 XHTML 的规则。

幸运的是，我们可以做些事情来了解 XHML 规则，见第 6 章

第 6 章

XHTML 和语义标记

我们本来想把这一章命名为“XHTML：规则简单，容易上手”。因为，第一，这章讨论的规则和教程是非常简单和容易的。第二，“简单”和“容易”对于 Web 设计书籍就像超级市场门口的“最新!”和“免费!”的标语，对吸引用户注意和鼓励他们进行尝试是惯用的有效手段。

我们当然希望这一章的内容能够引起人们兴趣和尝试。为什么呢？因为在掌握了这一章简单、容易的观点后，你将会重新思考创建网页的方法，并且开始改变。我的意思不是说你将用新的标签代替过去几年所用的标签，而是你将以和以往不同的方式进行思考和工作。而“简单规则，容易上手”的标题不能涵盖这些意思。

另一方面，“除了炫目的 Flash 外，唯一的真实有效方法”是我们放弃的另外一个标题，因为看起来太复杂了。而让 Web 内容拥有更丰富的语义，有更好的可发现性是 XHTML 的主要意图。所以，最后就用了“XHTML 和语义标记”这个标题。

在这一章里，我们将学习 XHTML 的入门知识，并探究语义标记和表现标记之间的机制和关联。如果已经在实际的设计开发过程中使用了一些 Web 标准，你就会很熟悉本章的内容，但是即使对一些老手来说，当他们打开这一章的“知识宝箱”时，也还是会非常惊奇的。

延伸阅读列表

- Ambient Findability: What We Find Changes Who We Become，作者 Peter Morville（O'Reilly 出版，2005），书中引入了 Findability（可发现性）的概念，并在信息架构和业务层面就信息和连通性之间的收敛进行了研究。（www.amazon.com/Ambient-Findability-What-Changes-Become/dp/0596007655）作者是现代用户体验设计之父，并与 Louis Rosenfeld 合著有热销书"Information Architecture for the World Wide Web"。

- Building Findable Websites: Web Standards, SEO, and Beyond 的作者由 Aarron Walter（New Riders 出版，2008），提供了实用的，基于 Web 标准的技术，以帮助更多人找到你的网站，帮助用户在你的网站发现好的内容，并鼓励用户回来访问（www.buildingfindablewebsites.com），这个网站是一个美观简洁的教学工具，包含实战经验、原创设计和代码实例。作者集网页设计者、开发者、大学讲师于一身，并且是 Web 标准项目的成员之一。

6.1　转换到 XHTML：规则简单，容易上手

只要你看到一些简单的规则和指南，就会发现：从传统的 HTML 转换到 XHTML1.0 是快速和平滑的（我真的不知道用什么短语来形容）。如果你会写 HTML，你就能写 XHTML。如果你从来没写过 HTML，你也可以直接写 XHTML。让我们从简单而容易的基础开始，好吗？下面是 XHTML 的规则。

6.1.1　用正确的文档类型（DOCTYPE）和命名空间（Namespace）

XHTML 文档刚开始的一些元素告诉浏览器如何解析它们，也告诉校验服务——如何采用统一规则校验它们。这些元素中的第一个就是 DOCTYPE（"document type"——文档类型的简写）声明。这个方便的小段代码用来说明你用的 XHTML 或者 HTML 是什么版本。DOCTYPE 总是写在所有标记的前面，其原因只有 W3C 委员会的成员才知道，DOCTYPE 也总是全部大写。

为什么需要一个 DOCTYPE

XHTML 允许设计师或开发者创造个性化的、不同类型的文档，每种文档被不同的规则约束。这些规则都是在一个叫文档类型定义（DTD）的 XHTML 规范的基础上定义的。DOCTYPE 声明告诉校验服务和现代浏览器根据你定义的 DTD 来描绘你的标记。相应地，这些信息告诉那些校验服务和浏览器如何处理你的页面。

DOCTYPE 声明是一个符合标准的网页的关键组成部分；除非你的 XHTML 确定了一个正确的 DOCTYPE，否则你的标记都不会生效。另外，你选择的 DOCTYPE 会影响大部分浏览器显示页面的方法。如果没有注意，结果可能会让你大吃一惊。

XHTML 1.0 提供了 DTD 的三种选择和三种可能的声明：

- 过渡型（Transitional）——最宽松的 DTD，它宣称的目标是“自己活也让别人活”
- 严格型（Strict）——“挥着鞭子的冷酷的”DTD，逼着你不能使用表现层的标记和属性。
- 框架型（Frameset）——20 世纪 90 年代最流行的布局方式，可以在你的设计中使用框架（frame）。

6.1.2 哪一种 DOCTYPE 适合你

6.1.1 节我们介绍了三种类型的 DTD，其中 XHTML1.0 过渡型 DTD 最接近于我们熟悉和喜爱的 HTML，也就是说，它是唯一一个还容忍表现层的标记结构、垃圾元素和属性的 DTD。

href 链接的 target 属性就是这样的，我们不赞成使用。如果你希望在一个新窗口打开一个链接页，或者你不想这么做但你的客户却坚持——过渡型 DTD 是唯一允许你使用 target 属性的 XHTML DTD：

```
<p>Visit <a href="http://www.whatever.org" target="_blank">whatever.org</a>
in a new window.</p>
```

```
    <p>Visit <a href="http://www.whatever.org/" target="bob">whatever.org</a> in
a named new window.</p>
```

在 XHTML 1.0 Strict 下，要在新窗口打开一个链接页，你需要写 JavaScript，也需要确认链接能否在不支持 JavaScript 环境下工作。不管你是否要在新窗口打开链接页，XHTML 过渡方法让你处理起来轻松。

XHTML 1.0 过渡型同时容忍在表格单元使用背景颜色，以及类似一些其他应该用 CSS 代替的标记。如果你的 DOCTYPE 声明已经规定了使用 XHTML 1.0 Strict，但页面中包含了这些不赞成使用的背景颜色属性，那么校验服务将把它们作为一个错误标记出来，一些符合标准的浏览器就会忽略它（也就是说，它们将不显示背景颜色）。相反，如果你把它声明为 XHTML 1.0 过渡型的 DTD，背景颜色（bgcolor）就不会被标记为错误，浏览器也将识别它而不会忽略它。（明白了？你已经开始知道了一个特定存在的 DOCTYPE 将如何影响内容的显示，以及在指定的浏览器里如何显示。）

6.1.3　严格与过渡：我们这个时代的大战役

现在，很少有这本书的读者，特别是这本第 3 版的读者，去花费力气使用 bgcolor 这个属性，（你只会在不得已的情况下使用，比如为了大的 Netscape Navigator 3 用户群体，不过也没有读这本书的人会不得不这样做）。那为什么不在每一个项目里使用 XHTML 1.0 Strict？答案是，你可以，如果你的项目自始至终都支持标准的话。

如果知道你在做什么，你能完全掌控网站， 只编写结构性的，语义标记代码——使用 CSS 控制样式，用不冲突的脚本控制行为，那么 XHTML 1.0 Strict 会是一个很好的选择。

但是当你的搭档有能力反复修改你美丽的基于标准的设计，那么过渡型还是好一点。例如：当你知道加入第三方不规范的代码，如广告代码，News Feed，将影响你的规范代码时，使用过渡型会好一点。在 2009 年，唉，这种情况仍然在发生：当一个不好的内容管理系统（或者一个好的内容管理系统集成了一个坏的模板）会让你的漂亮的设计变丑陋，HTML 代码失效时，过渡型会是一个安全的选择，当你网站编辑可能将一些不规范的 HTML 代码加入你美丽的模板时，过渡型可能是最好的选择。（根据不同的 CMS 和 CMS 的权限配置，可以有效地防止这些情况发生。）

概括地说，当系统和访问者可以超越你的控制破坏你的网站，或者对那些向现代 Web 标准过渡的设计师来说，XHTML 1.0 过渡型是一个完美的 DTD。而严格型是为这些人准备的…

- 了解他们自己在做什么；
- 确保代码标记只有结构和语义元素，没有表现（没有 font 标签、table 布局，以及其他过时的表现元素和属性）或行为（例如：没有内嵌 JavaScript，没有 JavaScript 链接），等等；
- 不会让编辑、客户、市场人员、非标准 CMS 系统以及未来可能不称职的开发者篡改你的工作。

嗨！Frameset 怎么样？

Frameset DOCTYPE 是用在那些使用<frameset>元素的文档上的。实际上，必须在你的<frameset>文档里使用这样的 DOCTYPE。开发人员是不是应该使用框架，是我们这个时代的另一大争论，答案是不。

嗨！DOCTYPE 放在哪里？

DOCTYPE 声明必须放在每一个 XHTML 文档最顶部，在所有代码或标记之上。放在<head>元素、<title>元素和 meta 元素，以及样式表和 JavaScript 文档之前。当然，也在你的内容之前。简言之，DOCTYPE 声明在所有这些东西之前。

（熟悉标准的读者可能会疑惑为什么我没有提到在 DOCTYPE 声明前面可以放置一个元素，也就是说，可选的 XML 声明。我会在下面的段落介绍）。

XHTML 1.0 DOCTYPE 快速指南

这儿有三个 XHTML 1.0 DOCTYPE，以及它们的典型用例：

XHTML 1.0 Transitional

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

何时使用：当使用过渡型标准时；当你需要使用一些纯表现性的 HTML 元素或者内嵌脚本时；或者当系统或者访问者可以超越你的权限破坏你美丽的标记时。

XHTML 1.0 Strict

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

何时使用：搭建一个你完全能控制的一个纯语义标记的网站时。

XHTML 1.0 Frameset

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

何时使用：当有人拿枪指着你的头，强迫你使用框架的时候。

6.1.4　紧随 DOCTYPE 之后的是 Namespace

紧跟在 DOCTYPE 声明之后是一个 XHTML 命名空间（namespace）声明，放在增强的<html>元素中，类似这样：

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

XML 的命名空间是收集元素类型和属性名字的一个特定的 DTD，命名空间声明允许你通过一个在线地址指向来识别你的命名空间，就像上面例子中的 www.w3.org/1999/xhtml。后面两个附加的属性，指定你的文档用英文，以及你使用的 XML 版本也是英文。附加的属性是任选的，没有它们你的文档也是有效的。

使用 DOCTYPE 和命名空间声明后，你的 XHTML Transitional 1.0 页面的开头看起来就像这样：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

剪切，粘贴，继续！

如果你不喜欢按照书上的例子输入标记和代码，可以直接到 zeldman.com，

alistapart.com 查看源代码，复制和粘贴到你的文档中央。注意：我的意思不是说可以任意复制和粘贴那些站点的内容和设计（那些都是有版权的），我只是说可以自由地从那些页面中获取一些代码，比如 DOCTYPE 和 namespace 声明。

6.1.5 声明你的字符集

为了被浏览器正确解释和通过标记校验，所有的 XHTML 文档都必须声明它们所使用的字符编码的类型，可能是 Unicode、ISO-8859-1（通常也被称为 Latin-1），或者你需要使用的其他字符集。

如果你对字符编码不太熟悉，或者使用 ISO-8859-1 没有什么问题，不用担心。我们将在这一章的后面讨论这个问题（看后面的章节："字符编码：无趣、无趣、真的无趣"）。现在，你所需要知道的只是：有三种方法告诉浏览器你使用的是什么标记语言，但只有一种方法能可靠的工作，而且不是 W3C 特别推荐的。

XML Prolog（以及如何跳过它）

很多 XHTML 页面用一个可选的 XML prolog 开始，也是一个 XML 声明。当使用 prolog 时，它放在 DOCTYPE 和命名空间声明之前，它的作用是指定 XML 的版本和声明这一页所使用的字符编码的类型。

W3C 推荐在所有 XML 文档（包括 XHTML 文档）开头都使用 XML Prolog。例如：要指定字符编码为 ISO-8859-1，你可以使用下面的 XMLprolog：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

没有任何复杂的东西，该标签告诉浏览器，这里使用的 XML 版本是 1.0，字符编码是 ISO-8859-1。有关这个 prolog 的唯一的新知识或不同就是使用了一个"？"号来标记开始和结束。

不幸的是，好几个浏览器，甚至一些有名的浏览器都不能很好地处理 XML prolog。当读取这个 XML 元素后，它们就不知所措或者胡乱处理了。

事实上，当站点不能正常工作的时候，浏览器本身并未受到处罚，忍受痛苦的是你网站的访客。使用 XML prolog，在一些情况下，你的整个站点对用户也许是看

不见的，甚至会使浏览器出错。另外一些例子中，站点没有垮掉，但不能正确显示（这就是当 IE6 遇上 XML prolog 会发生的事情）。IE7 的情况稍微好些，它忽略了 XML prolog，但允许使用正确的 DOCTYPE 校验 XHTML 站点并正确显示。（换句话说，严格些总比 Quirks 模式要好。参见第 11 章："DOCTYPE 切换和标准模式"。）

目前，IE 6 和更老版本的 IE 还是唯一一个阻止或忽略 XML prolog 的浏览器。如果你的用户里几乎没有使用 IE6 的，那就放心用 Prolog 好了，否则，你应该跳过 W3C 的推荐，别用它，而是在文档中的<head>标签中插入一个类型为 Content-Type 的 meta 元素来指定字符编码。例如指定使用 ISO-8859-1 编码，可以这样写：

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

现在你的 XHTML 文档的整个开头就像这样：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
<title>Transitional Industries: Working for Change</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
  </head>
```

如果你制作的是一个国际化站点，包含很多非 ASCII 字符，可以使用 Unicode 并在你的代码标记中，插入如下的 Content-Type 元素。

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

软件开发工程师/博客作者 Joel Spolsky 推荐的方式是把 meta 元素放在 title 之前，要不，浏览器将解析整个页面，然后再根据告知的编码重新解析一遍（www.joelonsoftware.com/ articles/unicode.html）。如果你的 Web 服务器能正确发送 Content-type 的 header 的话，倒不用担心这一点。

你可能想忘记前面讨论的详细技术细节，因为许多设计师一点都不知道这些问题，也不知道从哪一个标签开始复制到他们模板的顶部。但他们看起来也非常快乐地活着，创作着。

除了接下来几页之后还有一些更专业的主题之外，你已经完成了这一章里面容

易糊涂以及附有防错说明的部分。恭喜你！现在可以休息一会儿，吃点蛋糕吧。

6.1.6 用小写字母书写所有的标签

XML 不像 HTML，它对大小写是敏感的，所以，XHTML 也是大小写有区别的。所有的 XHTML 元素和属性的名字都必须使用小写，否则你的文档将是无效的（如果你忘记了 W3C 和 Web 设计团体提供的免费标记校验服务，请看第 5 章“现代标记”）。

让我们来看一个典型的 HTML 元素：

```
<TITLE>Transitional Industries: Our Privacy Policy</TITLE>
```

你认得这是 TITLE 元素，你也知道除了你们的法律部门之外没有一个人能够阅读到这份隐私保密页面。将这个元素转换到 XHTML 非常容易，只需要切换大小写就可以了：

```
<title>Transitional Industries: Our Privacy Policy</title>
```

同样地，<P>变为<p>，<BODY>变成<body>，等等。

当然，如果你原始的 HTML 元素和属性名字已经全部使用了小写，就不需要改变它们。但大部分人写 HTML 元素和属性名时都是大小写混杂的，所以当我们向 XHTML 转换时必须把它们变成小写。

一些流行的 HTML 编辑器，比如 Panic Coda、BareBones BBEdit 和 Optima-Systems PageSpinner，以及 AdobeDreamweave，可以帮助你将标签和属性名字自动转换为小写。免费的工具 HTML Tidy（参看下面的阴影部分：Tidy 时代）也能帮你做得很好。

不要担心属性值或内容的大小写

在前面的例子中，我们注意到只有元素名称 title 转换为小写，元素内容“Transitional Industries: Our Privacy Policy”依旧保留不变。当然你完全可以全部用大写（TRANSITIONAL INDUSTRIES: OUR PRIVACY POLICY），对于 XHTML 来说也是有效的，不过看上去不太方便。

元素和属性名字必须小写，属性值和内容则不一定。下面所有的例子都是有效的 XHTML：

```
<img src="/images/whopper.jpg" alt="Big John catches a whopper." />

<img src="/images/WHOPPER.JPG" alt="Big John catches a whopper." />

<img src="/images/whopper.jpg" alt="Big John catches a Whopper and fries." />
```

注意：根据你的服务器软件设置，src 属性中的“文件名称”可能是大小写敏感的，但 XHTML 不会在意。另一方面，Class 和 ID 值对大小写敏感。

Tidy 时代

到目前为止，建立有效 XHTML 页面最简单的方法是手工直接写代码。但是大多数的 Web 设计是对原有的网站进行重新设计，而你可能经常被赋予升级这些老页面的重任。重新设计意味着：这正是一个合适的机会来进行 XHTML 的升级，你用不着手工移植，由标准迷 Dave Raggetl 创建和由 SourceForge 维护为开源软件的免费工具 HTML Tidy（http://tidy.sourceforge.net/如图 6.1 所示）能够帮助你快速地将 HTML 转换到有效的 XHTML。

Tidy 有在线版本和针对 Windows、UNIX、各种 Linux 版本、Mac（OS 9 和 OS X）及其他平台可以下载的二进制版本一样，都能很好地工作。一些版本被当做插件来增强一些现有的 Web 软件。例如，BBTidy 就是 BareBones 软件公司 BBEdit（X）HTML 编辑器的一个插件。每个版本提供不同的功能，所以可能包含差别很大的文档说明。Tidy 看上去简单，但它的功能强大，仔细阅读用户手册能节约你许多的时间。

Macromates TextMate（www.macromates.com），Mac OS X 版本，包含了 Tidy，在 HTML 里也捆绑了在线校验服务。最近，一个专门命名为“Jeffrey Zeldman 的 Web 标准顾问”（www.zeldman.com/x/9）的工具，也可以在检测标识，校验服务时提供语义和结构上的建议。

要小心使用大小写混杂的属性名字，如果你使用像 Adobe Dreamweaver 那样的 WYSIWYG 工具，或者使用像 Adobe Fireworks 或 Adobe photoshop 的 ImageReady 的图片编辑器来自动产生 JavaScript 鼠标滑过的效果，就需要改变大小写混杂的 onMouseOver 为小写的 onmouseover，是的，这是真的。

下面的代码将使你遇到麻烦：

```
onMouseOver="changeImages
```

如果这样就完美了：

```
onmouseover="changeImages
```

（尽管这个例子可能从 XHTML 校验工具看来是可以的，但是如果要把你的网站结构和行为分离开来的话，这样做不行，你需要编写不冲突的脚本并逐步增强，就像第 5 章描述的那样。）

6.1.7 给所有属性值加引号

在 HTML 中，你不需要给属性值加引号，但是在 XHTML 中，它们必须加引号（例如，height="55"，而不能是 height=55）。

还有一些事值得提及。如果属性值已经有了引号怎么办？例如，你的 alt 属性值必须写成这样："The Happy Town Reader's Theater Presents 'A Christmas Carol.'"，你怎么处理它们？当然，你可以这样做：

```
<img src="/images/carol.jpg" alt="The Happy Town Reader's Theater presents 'A Christmas Carol.'" />
```

如果你想更完美一点，使用转义字符来避免双引号和单引号次序混乱，还可以这样做：

```
<img src="/images/carol.jpg" alt="The Happy Town Reader&#8217;s Theater presents &#8216;A Christmas Carol.&#8217;" />
```

现在你已经给属性值加了引号，还必须用空格分开属性，下面的代码是错误的：

```
<hr width="75%"size="7" />
```

正确的写法应该是：

```
<hr width="75%" size="7" />
```

注意：W3C 校验器会清楚地、方便地标记出发生错误的信息：“XML 解析错误：属性结构错误。”他们也许奇怪为什么会出现这样简单的错误。

如果有一些奇怪的理由，你需要在属性值里使用双引号，你可以用"，就像下面这样：

```
<img src="/images/hello.jpg" alt="Mrs. O'&amp;#8217; Hara says, &amp;quot;Hello&amp;quot;
to us." />
```

给属性值加引号在 HTML 里是可选的，但是我们很多人还是使用它，所以在转换到 XHTML 时候不需要再做处理。为了减少带宽，一些商业网站去掉了属性值的引号，如果要转换到 XHTML，这些站点将不得不加上这些引号。

HTML Tidy 能够自动给你的属性值加上引号。事实上，它可以自动完成我们这章提到的所有转换任务。

6.1.8　所有属性都需要值

所有属性必须有一个值，就像下面的 HTML 中的属性：

```
<td nowrap>
<hr noshade>
<input type="checkbox" name="shirt" value="medium" checked>
```

…必须被赋予一个值，属性值必须和属性名称相对应。

```
<td nowrap="nowrap">
<hr noshade="noshade" />
<input type="checkbox" name="shirt" value="medium" checked="checked" />
```

我们知道，虽然看起来怪怪的，感觉也怪怪的，和精益、结构化、节省带宽的标记原则完全相反，但还是养成习惯去做吧。第 7 章要讨论的 HTML5，就更精益一点，例如下面这个标记在 HTML5 里面是合法的：

```
<input disabled>
```

但是，当然，它在 XHTML 里不合法。

6.1.9 关闭所有的标签

在 HTML 中，你可以选择打开许多标签，如<p>和<li>，而不需要关闭它们。下面的代码对 HTML 来说是完全可以接受的，但对 XHTML 就是很糟糕的：

```
<p>This is acceptable HTML but would be invalid XHTML.
<p>I forgot to close my Paragraph tags!
<p>But HTML doesn't care. Why did they ever change these darned rules?
```

在 XHTML 中，每一个打开的标签都必须关闭：

```
<p>This is acceptable HTML and it is also valid XHTML.</p>
<p>I close my tags after opening them.</p>
<p>I am a special person and feel good about myself.</p>
```

“每个打开的标签都必须关闭”规则，比 HTML 的混乱和矛盾的方法更加理性。它可以避免许多麻烦。举例来说，如果你不关闭段落标签，在一些浏览器中就可能出现 CSS 显示问题。XHTML 强迫你关闭标签，用这种方法确保页面能像你意想中一样地工作。

“空”标签也要关闭

在 XHTML 中，甚至如
和<img>的“空”标签也要关闭。在标签尾部使用一个向前的斜杠“/>”来关闭它们自己：

```
<br />
<img src="zeldman.jpg" />
```

注意不要漏失在 br 标签结尾的/>和在 img 标签结尾的/>。 同时在/>之前要用空格分开，以避免浏览器不识别的错误。很简单吧，没有那么深奥。

这并不需要很大的工作量，自 2001 年以来的编辑器，像 BBEdit、PageSpinner 和 HomeSite，如果告诉它们你将在 XHTML 中工作，编辑器会自动在空标签后面加上所需的空格和“/”。Coda.TextMate 和可视化 Web 编辑工具（如 Adobe 的 Dreamweaver 和微软的 Expression Web）也同样这样做。

很自然地，为了保持有效性和可访问性，第二个例子中的图片也应该包含一个 alt 属性，以及一个可选的 title 属性：

```
<img src="zeldman.jpg" alt Jeffrey zeldman, author of Designing with Web Standards. "Jeffrey Zeldman, debonair web designer and billionaire author of Designing with Web Standards, now in its 400th printing。" />
```

现在，这就是一个良好的 XHTML 例子。

6.1.10　不要在注释内容中使用"--"

"--"只能出现在 XHTML 注释的开头和结束，也就是说，在内容中它们不再有效。下面两行代码都是无效的：

```
<!--Invalid - - and so is the classic "separator" below. - ->
<!---------------------------------------------------------->
```

用等号替换内部的虚线或者在虚线之间增加空格：

```
<!-- Valid - - and so is the new separator below -->
<!--==========================================-->
```

6.1.11　将所有的<和&符号编码

任何小于号（<），不是标签的一部分，都必须被编码转化为<，任何与号（&），不是实体的一部分，都必须被编码转化为&。因此：

```
<p>She & he say that x < y when z = 3.</p>
```

必须标记为这样：

```
<p>She &amp; he say that x &lt; y when z = 3.</p>
```

注意：即使你从来没有这样编码过，我们也要推荐用>来对>进行编码（除了在某个高度深奥的环境中），这里是为了对称，也可以使其他人更容易阅读。

下面让我们复习一下已经学过的 XHTML 规则。

6.1.12 执行概要：XHTML 的规则

- 以正确的 DOCTYPE 和命名空间开始文档。
- 使用 META 内容元素声明你的字符编码。
- 用小写字母写所有元素和属性名称。
- 给所有属性值加引号。
- 给所有属性赋一个值。
- 关闭所有标签。
- 用空格和斜杠关闭“空”标签。
- 不要在注释内容中使用“--”。
- 确保使用<和& 表示小于号和与号。

作为这么一个小型列表，XHTML 的规则看起来没几条，而且很简单。事实上也是这样。在我们介绍好东西前，必须额外说明一件无趣而令人悲伤的事情。

6.1.13 字符编码：无趣，很无趣，真的很无趣

在阅读前面章节关于 XHTML 第二个规则（“声明你的内容类型”）时，你也许会问：“为什么我要声明自己的内容类型？”你也可能会问：“什么是内容类型？”我们接下来将回答这些无趣的问题。也许你正在问自己“我真的需要读这么单调乏味的内容吗？”答案是：“当然要看”。如果我们不得不写，你就不得不看。这是最公平的，而且还可以学到一些东西。

Unicode 和其他字符设置

XML、XHTML 及 HTML 4.0 文档默认的字符集是 Unicode（http://www.w3.org/International/O-unicode.html），一个标准的定义，足够古怪，是由 Unicode 协会（www.unicode.org）制定的。Unicode 是一个全面的字符集，为每一个字符提供了独特的编码，“不论什么平台，不论什么程序，不论什么语言”。Unicode 是最接近最通用的一个字母表，尽管事实上它不是一个字母表，而是一个数字映射表。

虽然 Unicode 是 Web 文档的默认字符集，但是开发者也可以自由地选择其他的字符集，也许那更加符合他们的需求。例如，美国和西欧的网站经常使用 ISO-8859-1（Latin-1）编码。你也许会问什么是 Latin-1 编码，或者它来自哪里？说实话，你并没有问这种问题，但我需要一个过渡，那是我们进行简单介绍的最好方法。

什么是 ISO 8859

ISO 8859 是一系列标准的、多语言的、单字节编码（8 位）的图形字符集。其中第一个字符集就是 ISO-8859-1（通常也叫 Latin-1），用来映射西欧字符到 Unicode。ISO 8859 字符集还包括 Latin-2（东欧）、土耳其语、希腊语、希伯来语、日尔曼语及其他。

ISO 8859 标准是欧洲电脑制造商联盟（ECMA）在 20 世纪 80 年代中期建立的，并被国际标准组织（ISO）认可。现在你明白了吧！

声明你自己的字符集

不管你选择什么字符编码，都必须声明你的字符编码，就像在前面第二个 XHTML 规则中描述的。站点可以用三种方法声明他们的字符编码：

- 一个服务器管理员可以通过 Web 服务器上返回的 HTTP 头设置编码语言。W3C 推荐这种方法，但是很少有人这样做，因为也许服务器管理员宁愿玩网络游戏，也不愿意围绕 HTTP 头多做一点思考。

- 对于 XML 文档（包括 XHTML），设计师或开发者可以使用可选的 XML prolog 来指定编码语言。这个方法也是 W3C 推荐的，但是需要等到更多浏览器学会和正确掌握处理，我们才能推荐它。

- 在 HTML 或者 XHTML 文档中，设计师和开发者也可以通过“Content-Type”元元素（meta element）来指定编码语言。当服务器设置的方法被管理员忘记，XML prolog 方法被浏览器拒绝的时候，“Content-Type”方法能继续工作。这就是我们在本章开头推荐的方法，现在你知道为什么了吧。

恭喜！你现在已经阅读完本书最无趣的章节，至少是本章或者本页最无趣的。让我们恢复精神吧！从这里起，我们将开始有趣的工作，重新思考我们设计和建造网站的方法。

6.2 结构健康——对我有益

XHTML 的发展已经超越了大小写转换，以及在
标签尾部加上空格和斜杠的时代。如果 Web 标准都像改变“标签风尚”这样简单，就没有人会烦恼了。这本书也将要改为写“豆腐烹饪方法”了。哦！我们跑题了。对 XHTML 带来的好处，你需要比视觉效果花更多的时间来思考你的标记结构。

页面结构化，内容语义化的变化将使得 Google 以及访客访问你网站丰富内容时，页面大小大大缩小。事实上，如果从这本书上你只学到了结构化，语义标注 CSS，那么你就学成了。你肯定会比其他的网页设计师更有优势，因为你设计的每一个网站都是轻量化、易发现、易访问的。

好东西来了！！

6.2.1 用理性代替样式来标记你的文档

记住：你应该使用 CSS 设计布局。在 Web 标准世界中，XHTML 标记不是为了表现而设计的，它是以文档结构和单个元素的语义为核心的。具有良好结构的文档对那些使用 Palm Pilot 或者屏幕阅读器的用户更加友好，如同他们用 Google 的 Chrome

在 32 英寸的监视器上看网页一样。良好的文档结构对那些不支持 CSS 的老浏览器或者关闭 CSS 的现代浏览器的用户也有更好的视觉效果。

什么是我们所说的语义？我们的意思是，我们选择一个 XHTML 元素，以标识它包含的内容的意思——如果内容是一个列表项目就用 list，如果内容是一个段落就用 P，如果是页面里最重要的标题就用 H1，等等。我们所指的结构化是什么？我们的意思是，在一个有序的逻辑结构中用语义元素书写 XHTML（相对表结构布局、FONT 标签，以及其他垃圾标签）。

这只是一个起步，在我写本书第 1 版时，它还被认为是超前的，远离规范的。唉，这仍然是不规范，你接着看下去吧。

下面是一些技巧，帮助你开始在结构上做更多的思考。

大纲里的颜色

在学校的语法课上，我们大部分人都被迫用一种标准大纲式的格式写短文。当我们成为一个设计师后，噢，感觉多么自由，我们可以抛弃大纲那种呆板的条条框框限制，大胆投入到一个个性表达的领域。（也许我们的 brochure（企业用来宣传的网站）和商业站点不是那样独特和个性化。但我们至少不用担心大纲或者其他，不是吗？）

实际上，依照 HTML，我们应该总是按一定层次（大纲）来组织我们文本内容的结构。在浏览器支持 CSS 以前，我们不能提交这样受欢迎的布局，但是今天，已经完全可以提交基于文档结构的代码了，而且不用支付额外的设计费用。

当我们编写页面文本，或者将已经存在的文本文档转换到网页时，思考一下传统的大纲：

```
<h1>My Topic</h1>
<p>Introductory text</p>
<h2>Subsidiary Point</h2>
<p>Relevant text</p
```

避免使用不合理的 HTML 元素，类似<font>标签或者无意义的元素（比如
）来冒充一个貌似合理但实际上不能使用的逻辑结构。举例来说，不要像下面这样：

```
<font size="7">My Topic</font><br />
Introductory text <br /><br />
<font size="6">Subsidiary Point</font><br />
Relevant text <br />
```

根据元素的含义使用元素，而不是因为它们看起来像什么

一些人在写 Web 代码标记时，已经养成习惯，只是希望字号大点就用<h1>，希望在前面加个点符号，就用<li>。就像在第 1 部分中讨论的，浏览器会在 HTML 元素上强加默认的设计属性。我们总是想<h1>的意思是大的，<li>的意思是圆点，<blockquote>的意思是“缩进排列文本”。我们大部分 Web 设计师都乱写 HTML，强迫用结构元素实现表现效果。

按照这个想法，如果一个设计师想让所有的标题都有同样的尺寸，她可能将所有的标题都用<h1>设置，即便这样做也没有什么结构性的意义：

```
<h1>This is the primary headline, or would be if I had
organized my textual material in outline form.</h1>
<h1>This isn't the primary headline but I wanted it to be the
same size as the previous headline and I don't know how to use
CSS.</h1>
<h1>This isn't a headline at all! But I really wanted all the
text on this page to be the same size because it's important
to my creative vision. If I knew about CSS, I could achieve my
design without sacrificing document structure.</h1>
```

我们必须抛弃这些幼稚的思维，因为它们的含义，不是因为它们“看上去”是什么而开始真正使用元素。事实上，<h1>能变成你想要的任何样子。通过 CSS，<h1>能变成小的和罗马字号（标准粗细），<p>文本能够变成巨大的、粗体的，<li>能够变成没有圆点（或者用一个 PNG、GIF 或者 JPEG 的狗、猫或者公司 logo 图片代替圆点），等等。我的同事 Eric Meyer 还重写一个浏览器的内部样式表，并使用 CSS 来定义一个 HTML 页面的“invisible”元素，如 HTML 的 tittle 和 meta 数据。虽然浏览器都有默认支持，但 HTML 没有。在 HTML 或 XHTML 中，没有说<h2>必须要大显示，或者是可见的。这些仅仅是惯例，这样的观点就好像意大利面必须要放番茄酱一样。

从今天开始，我们要使用 CSS 来确定 XHTML 元素的外观。我们甚至能够根据它们在页面或者站点中出现的位置来确定它们不同的外观。<li>除了可以用在一个列表的每一项前面，也可以用在任何地方，这已经不再是一个需求，即使曾经是。除了它已经创建了，任何原因都可以使用<li>（以表明该元素是几个项目清单之一）。

把 CSS 从结构中完全抽象出来表示，允许你按希望的定义任何元素的样式。在支持 CSS 的浏览器中，如果设计师需要这样设计的话，所有标题的 6 个级别（h1~h6）可以看起来一样：

```
h1, h2, h3, h4, h5, h6      {
   font-family: georgia, palatino, "New Century Schoolbook",
times, serif;
   font-weight: normal;
   font-size: 2em;
   margin-top: 1em;
   margin-bottom: 0;
   }
```

为什么可以这样做？你可以用它在图形浏览器中表现出一个品牌的外表和感觉，而在文本浏览器、无线设备和 Opt-in HTML 邮件和新闻组中保留文档结构。

在 XHTML 章节中展示的 CSS 技术，并非想说明我们跳到了后面的内容，只不过是简单地展示，文档结构和视觉表达是清楚的两个方面。元素的语义值应该只用于传达内容的逻辑结构，而不要强制用于显示。

你必须抵抗诱惑

黑帽（Black-hat）搜索引擎优化建议利用 CSS 的优势挑战视觉公约，即：用 h1 来标记关键字（甚至整个页面），并用 CSS 来使 h1 标记的内容依然像普通文本一样。这样做的人们在将来可能需要符合更高的标准。别向不良倾向屈服。

宁愿用语义化元素代替无意义的垃圾

因为已经忘记，或者从来就不知道 HTML 和 XHTML 元素是特意用来传达结构的意义的，所以很多人已经习惯写标记时不含语义或结构。例如，许多 HTML 争论者在他们页面插入一个列表的时候会使用下面这样的标记：

```
item <br />
another item <br />
a third item <br />
```

我们可以考虑使用一个有序或无序列表代替：

```
<ul>
    <li>item</li>
    <li>another item</li>
    <li>a third item</li>
</ul>
```

你或许会说“但是<li>显示的是一个圆点，我不想用圆点”。在前面的章节中提到过，CSS 没有设定元素看起来是什么样子。它等着你告诉它们应该是什么样子。关掉圆点是 CSS 最基本的能力。它能使一个列表看上去像一段普通文本，或者像一个图形导航条，包含完整的滚动效果，如图 6.1 所示。

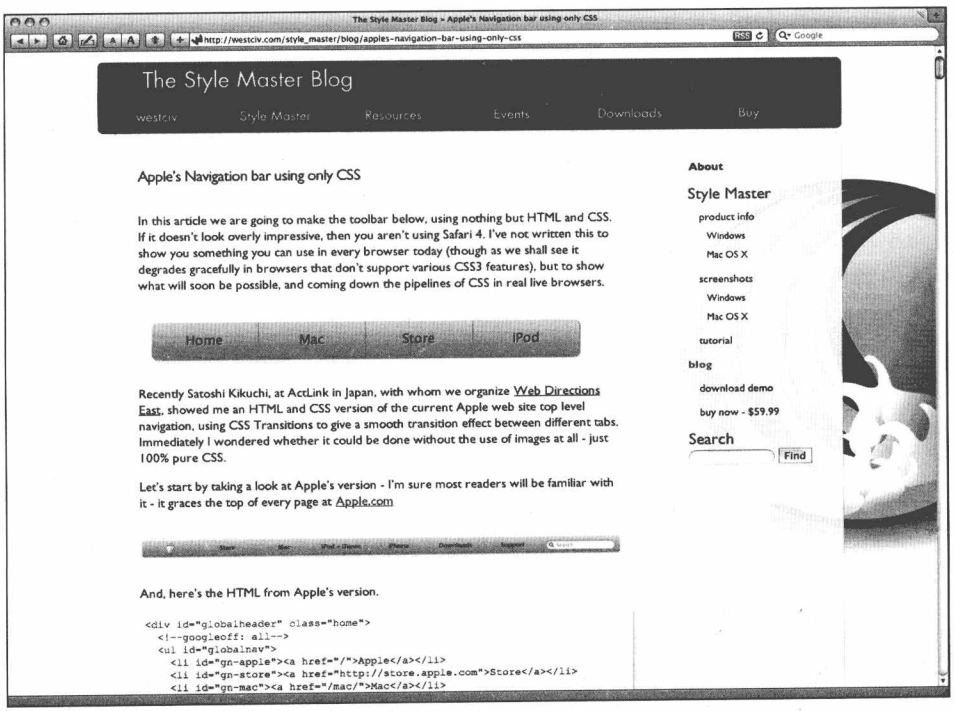


图 6.1　John Allsopp 只用 CSS 和语义结构的 list 不是图片，复制了 Apple 网站的导航条效果（www.westciv.com/style_master/blog/apples-navigation-bar-using-only-css）

所以，使用 list 元素来作为列表的标记。同样，最好用代替<b>，用代替<i>等。默认情况下，多数桌面浏览器将把当作<b>显示，把当作<i>显示。创建视觉效果不需要破坏你的文档结构。

虽然 CSS 发明者没有为任何元素假设显示方式，但浏览器做了大量的假设，我们从来没有遇到一个浏览器显示为<b>以外的其他方式（除了设计师使用 CSS 指定的）。如果你担心一些陌生的浏览器不会把当做<b>显示，就可以写这样的 CSS：

```
strong     {
   font-weight: bold;
   font-style: normal;
   }
```

使用类似的结构化标记，也保护了人们在他们的浏览环境中使用文本浏览器和其他的可选设备下载无意义的标记（对于盲人读者来说<b>意味什么呢？）。

当然，也有对每条规则的例外情况。你经常会在用<b>代替以及用<i>代替时遇到下列情况：

- 你正在做一个没有排序的列表，而且会在之后的 CSS 中对它进行样式化以使得它在一个支持 CSS 的图形浏览器中看起来像个导航条。你希望能够对首页这个“按钮”进行高亮显示，从而能够指示当前用户他们正处于网站的首页，同时你也希望“你正在这里”的提示信息能够在整个浏览过程中显示，即使是在一个非 CSS 的环境中。用<b>标记封装首页链接能够完美做到这点，并能够防止错误的语义。这也能节省一些代码。这种 CSS 技巧（hooks）可以增加语义，这就是为什么有些人还会使用 b 和 i 来代替 Span 的原因。我们会在其他章节介绍更多类似的技巧。

- 而<i>和则在 Paul Ford 的著名例子（www.ftrain.com/ProcessingProcessing.html）中提到：“当我发布一个 1901 年的信息，并使用斜体和非强调方式的话，这个版式便有个语义，说明这个信息是微妙的、变化的、且具有深远历史意义的”。

更深入的关于<b>和的思考和对比，可以参考 Matthew Paul Thomas

（http://mpt.net.nz/archive/2004/05/02/b-and-i），Joe Clark（http://blog.fawny.org/2004/05/16/ubu），以及 James Craig（http://cookiecrook.com/2004/05/#cc108424449201351713）的 blog。还有，在 HTML5 中用非视觉术语重新定义了<b>和<i>，这样或多或少重新体现了 Strong 和 em 的语义。

6.2.2 视觉元素和结构

“Web 标准”坚持的不仅仅是我们使用的技术，而且也包括我们使用它们的方法。用 XHTML 写标记，使用 CSS 控制一些或者所有布局，并不是使一个站点更加易用、更加方便，最少带宽所必需的。XHTML 和 CSS 可能也像早期的 Web 技术一样被滥用或者被抛弃。冗长的 XHTML 标记也会像 HTML 曾经做过的一样，浪费带宽和用户时间。啰唆一句，用过度的 CSS 替换 HTML 表现是不恰当的，这样做，只是用另外一种垃圾代替 HTML 表现代码垃圾。

在之前的“结构健康-对我有益”一节中的指导能帮助避免结构过度复杂。但是我们对关于品牌的视觉元素该做些什么呢，例如站点导航条，典型情况下它们包含一个 logo 标志。这些元素能进行语义化的处理吗？

还有，页面结构又怎么样呢？<P>和<h1>是没有什么问题的，但是如果你已经有了一个性感复杂的视觉设计需要渲染，如果不用表格，就需要上百个 DIV，代码会变得和曾经的表格布局一样的臃肿，在语义上毫无意义，是这样的吗？

我们会在第 8 章讨论这些问题，但是先让我们来考虑一下 HTML 的替代版本——它的名字是 HTML5。

第 7 章

HTML5：新的希望

HTML5 是有史以来 Web 标记语言首次发生的重大改变。一方面它是第一个由浏览器厂商制定的而不是科学家闭门造车的标记语言。正如第 4 章说的，Web Hypertext Application Technology Working Group" (WHATWG) (www.whatwg.org)是由 Apple，Opera 和 Mozilla 的雇员发起的，在 Google 的 Ian Hickson 领导下创建了规范。他们现在正和由软件开发者 Sam Ruby 和微软 IE 发展部 Chris Wilson 领导的 W3C HTML Working Group (www.w3.org/html) 共同制定规范。在任何一组你都很难发现一个学院派或者纯粹的理论家，同时你会发现两个组都有 Hickson 先生在场。

HTML5 也是第一个在 Web 标准认同时期创造的标记语言，浏览器制造商和许多设计者一样，终于关注标准了。同时，矛盾的是，它是 XML 和 XHTML 得到广泛接受之后第一个脱离规范化的第一个标记语言标准——就是 W3C 推崇，Web 标准传道者十年以来普及的一丝不苟的规范。

7.1 HTML5 和 Web 应用程序：风险很大

最重要的是，HTML5 是第一个在 Web 应用程序时代构想出来的网页标记语言。早在网页标记语言解析文档结构之前，科学家就设计出来了。浏览器制造商当然也可以设计 HTML5 来做这些，它们甚至给了我们更多的语义元素以帮助我们书写更智能更复杂的文档，但是它们也（基本上）把 HTML5 当成了一个推动 Web 应用程序发展的平台。

Web 应用程序的存在已经成为了无法改变的事实——它强大到足以替换桌面日历、联系人和项目管理软件，并威胁到微软 Office 和 Adobe Photoshop 的统治地位——并且，专用平台如微软的 Silverlight（www.microsoft.com/silverlight）以及 Adobe 的 Flex（www.adobe.com/ products/flex）都在想办法夺回 Ajax 和 Web 标准占领的富网络应用空间，这个赌注不可谓不高。

你会认为这是关于标记的另一个沉闷的章节。

HTML5 有什么新东西

HTML5 能通过以下几个方面加速 Web 应用程序的发展：

- 引入了崭新的应用程序元素，比如 canvas，实时的界面绘制，以及 video，插件的替代者；
- 可以直接和 API 一起工作；
- 建议了标准的错误信息处理方式，而且这种处理方式在所有浏览器中都一样；
- 通过 W3C GeoLocation API 支持本地位置；
- 引入多线程 JavaScript，因此能让浏览器还在处理之前触发的脚本时，Web 应用程序能响应用户的行为；
- 通过 contenteditable、draggable 和 spellcheck 等属性，方便用户编辑页面。

甚至它的 DOCTYPE 也精简了：

```
<!DOCTYPE HTML>
```

这就是全部的，也是你需要写的全部 DOCTYPE 内容。如果不需要兼容 Quirks 模式下的浏览器，HTML5 的编写者可能会完全忽略掉 DOCTYPE（第 11 章会讨论这个问题）。顺便说一下，字符串不区分大小写，你也可以写成<!doctype html>，<!DOCTYPE html>等等。

但它可以在浏览器里工作么？

尽管 HTML5 出来不久，但大多数浏览器已经尽可能地支持它的一些高级新特征了。Opera、Firefox 和 Safari，包括 iPhone 的手机版 Safari，都支持地理位置（GeoLocation）和 Canvas，而且 Firefox 已经开始支持 video 标签——目前只支持一种解码格式，将来会更多。事实上，浏览器厂商全方位的参与，意味着 HTML5 不会像 XHTML2.0 一样凋谢，而会很快得到正确而完整的应用。根据目前的时间表，HTML5 要在 2011 年才能完成，到 2022 年所有浏览器才会完美支持。听起来还有很长的路要走，但是在未来的几年里，大多数浏览器会预先支持 HTML5 的所有元素。

事实上，包括 IE8 在内的所有现代浏览器都可以让我们使用 HTML5 页面结构元素，像 nav，section（第 8 章讨论），并可以在 CSS 中定义它们。HTML5 默认不能在 IE6 或者 IE7 下工作，但 Remy Sharp 会告诉你怎么使用 JavaScript 来骗过老版本的 IE 来支持 HTML5 (www.slideshare.net/remy.sharp/html5-js-apis)，同时 Dean Edwards 创建了一个 JavaScript 库来支持浏览器的 HTML5(www.blip.tv/file/2299313)。

要详细介绍 HTML5 带来的编程机会超出了本书讨论基于标准设计的范围，但是勿庸置疑，在下一代那些最好的网络开发和网络编程书中都会有它们的位置。对网络开发者们来说，这真是一个激动人心的时刻。在 www.whatwg.org/specs/web-apps/current-work/multipage/semantics.html#semantics 有 HTML5 元素的完整列表以及怎样使用它们的介绍。在本章的剩余部分，我们将看看和 Web 设计师切身相关的那些元素——特别是新的页面结构元素。我们首先要看看 HTML5 和我们学过的语言有何不同。

7.2　HTML5 和 XHTML

如果你要一个冷冰冰似是而非的 HTML5 和 XHTML 的区别概要，你可以说，HTML5 有点像 XHTML 1.0，但是标签的关闭规则更宽松，并且新元素的设计促进了应用程序的发展，如果你喜欢，你可以创建一个语义化的页面布局，或者深化页面结构的语义化。你也可以说，XHTML1.0 是稳定的，浏览器支持很好（在 XHTML1.1，MIME 有点小问题），并且已经工作了 10 年，而 HTML5，在我写这些话时，尽管接近最终版，但还在发展和变化中。你可能通过观察最终发现，包括 IE8 在内的浏览器只是支持一部分，他们还没有支持 HTML5 的所有新元素。

还可以说出很多区别，不过主要就这些了。

该死的命名法

XHTML，在 Web 标记上脱胎于有规范化、严格化优点的 XML，而 HTML5 能包容 HTML 一样的松散问题，比如是否每一个元素打开都必须关闭。这个松散问题会激怒那些知道这些的开发者，并会被不知道这些的开发者忽视——这个自相矛盾的观点，我叫它 Hickson 不确定定律，以向 HTML5 规范的 WHATWG 编辑和 W3C HTML 小组的合作编辑 Hickson 先生致敬。

幸运的是对我们这样酷爱代码整洁的人来说，HTML5 提供两种类型的语法（或按 WHATWG 说的，2 种序列化方式）：HTML5，即简单又能像 HTML 一样使用，XHTML5 提供类似 XHTML 的语法。换句话说，如果你想保持严格的 XHTML 语法，也可以。

目前，验证服务会告诉你，你的 HTML 代码是不是有效—— validator.nu 已纳入 W3C 的验证服务，但它们不能告诉你，你的 HTML 语法（或者 XHTML 语法）是否正确无误。因此，你可以同时使用 HTML 和 XHTML 语法书写为 text/html 服务的有效 HTML5 代码，而如果你写错了什么，没有校验器会告诉你。好的方面是浏览器会毫不含糊地处理这种混合语法的 HTML5。它们这么做让你能去包含第三方的内容，比如，不用担心不同的书写风格导致产生校验错误或者导致浏览器崩溃。但是校验服务也为高级的 Web 设计和开发者提供了检查他们 HTML 和 XHTML 语法正确的选项。一些同事和我已经向 Henri Sivonen，HTML5 校验服务的设计者（about.validator.nu），提出加入这个特征。在你读到这里的时候，他可能已经这么做了。

撇开新元素不说，HTML5 在它的 HTML5 化身里，可以兼容旧的浏览器。按照 HTML5 创建者的意图，浏览器会将 MIME 类型是 text/html 的文档作为 HTML5 一样处理。同样适用 XML 语法的 XHTML5，在浏览器里一个 MIME 类型是 application/xhtml+xml 的网页会作为 XHTML5 处理。如果你用 XHTML5 的语法来书写有效的 HTML5，但是以 text/html 形式来提供给浏览器，浏览器也会显示得不错。

无疑，规范倾向于这样的语法规则：把 HTML5 标记为 text/html 类型，同时在标签的关闭上更宽松一些：

> 有人提醒我们，XML 和 HTML 处理起来不一样；特别是，即使很少的语法错误都将阻止 XML 文档的完全展现，而在“HTML5”的语法里，它们会被忽略掉。
>
> www.w3.org/TR/html5/introduction.html#html-vs-xhtml

这是对 Hickson 先生把 XHTML 用在 text/html 类型上的做法明显的反对（本书别的地方讨论过），这也是一个关于浏览器宽容语法错误的公平辩护，因为网上到处都有这样的错误。毕竟最严格，最语义化结构的网站也会由于读者的评论中未关闭 em 标签，因为一个 CMS 的疏忽，一个很差的广告软件，一个应用程序或文档片段引进父站点的内容，或者人为的错误而产生校验错误。如果浏览器因为简单常见的校验错误而退出或者卡死，那是错误的，会导致 99.9%的网站被关闭。相比之下，在 HTML5 以 application/xhtml+xml 执行时，浏览器厂商被鼓励用 Draconian 错误处理方式处理。当网页设计师可以使用 XHTML 和 HTML 语法书写 text/html 类型的网页，而浏览器可以忽略 text/html 类型的页面产生的错误，很难想象，除了 text/html 之外你的网站还会想用什么别的类型去进行服务。

7.3　HTML5 元素大检阅

除了比 XHTML 更宽松的规则之外，HTML5 最大的特色就是新元素的创造。其中一些是姗姗来迟，专门用于应用开发的亮点，如 video 和 audio 标签。正如本章前面提到的，这些元素标签可以允许浏览器直接运行视频和音频，不需要插件，没有之前不合法的 embed 元素，也没有嵌套的 object 元素以及脚本和退回（fallback）混合在一起。在“A Preview of HTML5”(www.alistapart.com/articles/previewofhtml5)，Lachlan Hunt 提供了以下代码：

```
<video src="video.ogv" controls poster="poster.jpg"
width="320" height="240">
    <a href="video.ogv">Download movie</a>
</video>
<audio src="music.oga" controls>
    <a href="music.oga">Download song</a>
</audio>
```

注意这些属性的 controls 元素，它告诉浏览器创建一个播放控制，以及 poster 元素，它告诉浏览器把图片作为一个占位符使用，直到用户决定播放视频为止。同时也要注意，如果浏览器不支持视频和音频标签时，它还有一个简单的退回（fallback）链接。

复杂的网页标记不会比这个简单多少，我们也不会去怀念那些嵌套的标签、脚本以及工作环境了。Opera、Firefox 和 Webkit（Apple Safari，Google Chrome）的实验版已经开始支持这些新的 HTML5 多媒体元素，因此，为了你自己，练习它们吧。在这本书付印时，Firefox 只会支持 Ogg 解码格式（www.xiph.org/downloads），不过没关系，你能在 www.vorbis.com/music 找到适合 Firefox 的 Ogg 音乐格式文件。

7.3.1 页面结构的语义化

更有趣（也更有争议的）是 HTML5 新的页面结构元素，section、article、header、nav、footer、figure 和 aside。它们都做些什么？很高兴你这么问。

- Section：像 div 一样，一个 Section 包含并组织相关的内容，包括：标题、段落、图片等等。它通常有一个 header，也可能有一个 footer，section 没有代替 div，div 仍然保留块的特性。一个 div 包含的内容之间可能有也可能没有关系。一个 Section，恰恰相反，只包含主题相关的内容，比如一个 h1 标题标明一栋要出售的房子，这个房子的一张照片，几个描述房子的文本段落，一个联系信息的列表，等等。这个封装的 section 传达了这几个项目内容相关联的语义信息。在一个基于标准的设计师手中，这种关系会在 XHTML 的一个 div 所包含的项目中暗示。但是 HTML5 让这个关联很明确，并在使用 div 的时候，去除了这种关联。在 HTML5 里，div 变成了一个废品抽屉（或者说是“流动内容”的容器，WHATWG 的叫法），并鼓励网页设计师用 section 来替换 div。

- Article：自包含的内容，像博客文章、新闻文章，或者读者评论，像 section 一样，它可以包含一个 header 和一个 footer。尽管规范里没有明确的这样定义它，一个 article 可以当做 section 的一种特殊形式。一篇博客文章或者新闻文章会被打上 article 的标签，而不是 section。无论 section 还是 article 是同级的。一个 section 可以包含很多 article，一个 article 也可以包含很多 section。像任何新事物一样，article 和 section 可能最开始都会让你觉得多余，不习惯。但

对创建页面结构语义化，它是一个好想法。使用它能很容易地创建页面大纲，远远超过传统的 h1～h6，同时可以消除 HTML4 遗留给我们的各种格式和校验问题。例如，在 HTML4 和 XHTML1.0 中，当复制一篇文章所有的内容到我的网页上的时候，需要分离和重新格式化这些标记，避免在一个页面重复出现很多 h1。但是在 HTML5 里，我可以用一个 article 容器来包含内容。接下来的一小节是 header 元素的描述，解释为什么要这么做。

- Header：有意(但不强求)去包含 section 的标题(h1～h6 的元素标签)。“header 标签同样可以封装 Section 的表格内容，搜索表单，或任何相关的 LOGO”。在 HTML5 中，首次我们可以有超过 6 级的标题，因为一个页面可能有一个以 h1 开始的 section，并且封装部分也可以以 h1 开始，因此我们可以在一个简单的例子里创建 12 级的标题。理论上讲 section 里标题的嵌套是无限级的。

- Nav：“一个链接到其他文档或者自身文档某部分的 section，更确切的说，是一个导航链接的 section”，是主导航模块，不是页面上的任何链接组。在书写时，面包屑式导航和底部式链接都不能在 nav 中使用。

- Aside：侧边栏（杂志使用的那种，不是三栏布局的那种）和导读。

- Figure：在这本书中，figure 元素是一张图表、一部电影、一张图片、一幅照片、一段代码实例，或者是对理解文章或段落有帮助的，但移除也不影响原文的其他信息。Figure 元素也给了 legend 元素另外一个场所(表单除外)。对于 HTML5 来说，这不是什么新元素，W3C 团队成员 Dave Raggett 在 HTML3 时就引入过 fig 元素，但是在 HTML3.2 时，这个元素被删除了。我们永远也不会知道如果 Raggett 语义化的 HTML3.0 得到广泛支持，Web 历史会是什么样的。

- Footer：“页脚通常包括的信息有谁写的、版权信息、相关链接等等，可能不包含标题和分区元素”，2009 年 8 月的规范这么写道。看上去和我们没有太大关系，我们的特色是在页脚加上导航元素，比如“关于我们”，“联系方式”。但是同样在这里，HTML5 的制定者用了一个熟悉的词语“footer”，给百万级的网络开发者传达了一个特殊的含义，以及与我们期望不相同的用法。在 HTML5 中，footer 标签过去主要是一个 metadata 的容器。HTML5 的超级爱好者发现了问题 (www.zeldman.com/superfriends)，于是 HTML5 的制定者取消

了对标题和分区元素的限制。footer 的内容模式现在和 header 相匹配了，能够让设计者顾名思义地理解它的意思。

值得一提的是：HTML5 的元素名是从成百万的网页文档中分析得来。理论上说，这是很伟大的，熟悉的名称能让这些新元素更容易被开发者理解和使用。但当熟悉的名字对应一个不熟悉的用法时，问题就出现了，比如 aside 的例子，尤其是 header 和 footer 的例子。

关于标记的全部

在讨论为什么有人对使用这些新的页面结构元素不满之前，让我们先看看他们想达成什么样的目标。在“The Power of HTML5 and CSS 3”中（www.perishablepress.com/press/2009/07/19/power-of-html5-css3），设计者兼开发者，Jeff Starr 将 XHTML 的“div soup”（div 堆砌）和 HTML5 的清晰结构做了对比。这里是 Starr 的(X)HTML 代码：

```
<div id="news">
  <div class="section">
    <div class="article">
      <div class="header">
        <h1>Div Soup Demonstration</h1>
        <p>Posted on July 11th, 2009</p>
      </div>
      <div class="content">
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
      </div>
      <div class="footer">
        <p>Tags: HMTL, code, demo</p>
      </div>
    </div>
    <div class="aside">
      <div class="header">
        <h1>Tangential Information</h1>
      </div>
      <div class="content">
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
      </div>
      <div class="footer">
```

```
            <p>Tags: HMTL, code, demo</p>
          </div>
        </div>
      </div>
    </div>
```

噢，好多的 div。在试图编辑这个文件时，设计师的眼睛很容易交叉，当少了或多了 标签，都会导致错误，打乱页面布局。

接下来 Starr 给我们展示了在 HTML5 里的样子，使用这个语言里新的页面结构元素：

```
<section>
  <section>
    <article>
      <header>
        <h1>Div Soup Demonstration</h1>
        <p>Posted on July 11th, 2009</p>
      </header>
      <section>
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
      </section>
      <footer>
        <p>Tags: HMTL, code, demo</p>
      </footer>
    </article>
    <aside>
      <header>
        <h1>Tangential Information</h1>
      </header>
      <section>
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
        <p>Lorem ipsum text blah blah blah.</p>
      </section>
      <footer>
        <p>Tags: HMTL, code, demo</p>
      </footer>
    </aside>
  </section>
</section>
```

第二个例子肯定更容易阅读和理解，且不太可能导致错误出现。

元素太少，还是太多？

关于 HTML5，人们要说的很多。John Allsopp，本书第 3 版的合著者，Developing with Web Standards（New Riders 出版，2009）的作者，对新元素本身没有任何问题。但他在它们缺乏向后兼容方面有一个问题，就是 HTML5 只给我们呈现了有限的新元素，但没有在将来增加新元素的机制。详见本书第 4 章中的“新语义学”。

我和我那些聪明的同事认为，给 HTML 增加有限的新元素是正确的。少量，有限的新元素有利于学习和使用，相比一大堆新元素可能更容易直观正确地使用。考虑一下那些你最喜欢的网站和应用的简洁性，再想想你必须使用（但你并不希望）的所有应用，因为它们带来了很多你很少使用的让人混淆的特性。HTML5 的挑战不是带来了很多新的元素，而是要确定，这些新元素是合适的，它们的名字也很直观，能告诉我们怎么使用它们。

7.3.2 HTML5：只是个规范

HTML5 规范的标准草案可以在 www.whatwg.org/html5 找到。请记住，那是一个正在进展中的工作，在用 HTML5 编码之前，记得去这个 URL 查看一下是否自本书付印之后有任何新的变化。

这个规范的开始是一些简单的原则声明，(www.whatwg.org/specs/web-apps/current-work/multipage/introduction.html#background)：

> 万维网的标识语言一直是 HTML。HTML 首先是设计作为一种对科学文档进行语义描述的语言，尽管它的整体设计和多年的适应性已经能用来描述一定数量的其他类型文档了。
>
> HTML 没有充分表述的主要领域是 Web 应用程序。本规范试图纠正这一点，同时更新 HTML 规范，以解决一些过去几年出现的问题。

阐述完目标之后，接下来很快就是规范的细节。

在 www.whatwg.org/html5 的大多数 HTML5 规范细节和你的 HTML 经验以及本书谈论的 XHTML 都很接近。HTML 的元素一般都有一个开始标签如<body>和一个结束标签如</body>。HTML 里，在某些情况下开始和结束标签可以被省略，但是在 XHTML 里是不行的。你甚至还可以在一些例子里省略掉 html 和 body 元素，这是我们一些人梦寐以求的。

正如在所有网页标记语言，标签（tag）必须嵌套正确：

```
<p>This is <em>so <strong>wrong</em>!</strong></p>
<p>This <em>is <strong>right</strong>.</em></p>
```

这些元素可以有属性，比如 a 元素有 href 属性：

```
<a href="#">
```

属性放置在开始的标签里，由名字和值组成，用“=”字符来分割，如果属性值不含特殊字符，可以不加引号。（在 XHTML 里，所有属性都必须加引号。）

在 HTML5 里，这样是有效的：

```
<input name=address maxlength=200>
```

而

```
<input name=address maxlength="200">
```

在 XHTML 里是无效的。

当我们用这些元素时（（www.whatwg.org/specs/web-apps/current-work/multipage/dom.html#elements）），会遇到一些熟悉的实体，诸如 id、title、class 属性等，也有一些古老的元素被重新定义成不那么有视觉含义，而更加表现中立。例如，i 现在表示“stressed”而不是 italic，斜体，让它像适应可视网页一样适合语音浏览器。还有一些特殊的属性，比如 dir，“指定元素的文本方向”。它的值可以为 ltr（从左到右）和 rtl（你猜对了，从右到左）。你可以在书写从右到左的希伯来语编码时用到它。（旧的 HTML 元素 dir，已经被删除了）。

在“嵌入式内容”一节中，我们遇到了新老朋友，如 svg、audio、canvas 和 embed

（最终合法了!，只花了 15 年。）

在“交互内容”下，我们发现了一些传统的元素，如 input、button、textarea，以及 video 和 datagrid（www.whatwg.org/specs/web-apps/current-work/multipage/ interactive-elements.html #datagrid），这是一个展示树状结构、列表、或表列数据交互效果的新元素，可以协助表示可排序的数据表格和画布等。

在规范的其他地方，脚本编写者将发现他们熟悉的构件，如 innerHTML()和 document.write()，也支持微数据，microdata（www.whatwg.org/specs/web-apps/current-work/multipage/microdata-0.html#microdata-0），如电子名片 vCard（www.whatwg.org/specs/web-apps/current-work/multipage/microdata-0.html#vcard），以及把 HTML 转换成其他的格式（www.whatwg.org/specs/web-apps/current-work/multipage/microdata-0.html# converting-html-to-other-formats），包括 JSON、RDF、vCard、iCalendar 和 Atom——但很奇怪，不是 Atom 的老兄弟 RSS。

除了 XHTML2 之外，HTML5 不同于以往任何网页标记语言，几乎所有的元素或者元素组都能打包成一个链接。支持的链接类型（使用 rel 属性）包括 nofollow、license、archives、author、bookmark、external、icon，以及更多。总之，各处的图书管理员和信息专家都很开心，简陋的链接现在包含了元素据，能让你的内容和脚本和应用程序一样对机器和搜索引擎更友好，并能额外提供 CSS 的简单 hooks。只是不要指望 CSS2 的属性选择器能在 IE6 下正常工作。

7.3.3 学习更多

这份非常简单的介绍表明，HTML5 提供了很多令人兴奋的东西，包括增加丰富的语义性，依附 CSS 的新元素，更强大的表单控制，更快速的网络应用原型，更强更快的（因为是多线程）的脚本执行，等等。这个规范的长度有 600 多个 Web 页面，PDF 有 900 多页，以至于在一本设计书籍中，想在一章中去描绘其中一部分内容都是一种冒犯。

毫无疑问，当标准完成时，这本书的下一个版本（或者是一本新书），关于 HTML5 将有更多东西可以说。在此之前，请看...

- Web 标准的发展，第 5 章(“HTML5”)；
- HTML5 Doctor (www.html5doctor.com)，Bruce Lawson 的见解；
- “Canvas 教程”(developer.mozilla.org/en/Canvas_tutorial)；
- HTML5 Gallery (www.html5gallery.com)；
- HTML5 Reset Stylesheet (www.html5doctor.com/html-5-reset-stylesheet)，Richard Clark；
- 主题：代码：HTML 和 XHTML（www.alistapart.com/topics/code/htmlxhtml）；
- “A Preview of HTML5”（www.alistapart.com/articles/previewofhtml5）；
- “Semantics in HTML5”（www.alistapart.com/articles/semanticsinhtml5）；
- “HTML5：能杀死 Flash 和 Silverlight 么?”（www.infoworld.com/d/developer-world/html-5-could-it-kill-flash-and-silverlight-291）；
- “Google Bets Big on HTML5”（radar.oreilly.com/2009/05/google-bets-big-on-html-5.html）；
- “HTML5: A Story in Progress” by Burningbird (www.burningbird.net/node/28)
- “Accessible Drag and Drop with WAI-Aria” (dev.opera.com/articles/view/accessible-drag-and-drop)；
- “Can HTML5 Make Accessibility Usable?” (my.opera.com/jax/blog/can-html5-make-accessibility-usable)；
- “HTML5: Features You Want Desperately But Still Can’t Use” (www.youtube.com/watch?v=xIxDJof7xxQ)。

第 8 章

更严格稳健的页面保证：结构和语义

无论如何，请不要跳过这个章节。这个章节的内容会大大提高你的技能，我将在这一章讲述怎么减少你的 Web 页面中不需要的冗余代码，以及加深你对标记和设计之间差异性的理解。本章的思想非常容易理解，同时可以帮助你提高站点的性能，并且简化网站的设计、开发和升级。

在本章，我们会展示怎么书写符合逻辑的、紧凑的代码，从而用来节省 50%或者更多的网络带宽，并且能通过减少服务器的负载而减少站点的加载时间以恢复站点的活力。我们将通过把表现层的代码从（X）HTML 页面中去除来达到这个目的，并且将学习怎么避免很多常见的坏习惯。

即使是那些已经大部分或者完全从表格布局转换到了 CSS 布局的网站，依然被这些不好的习惯所困扰。随着本书第 3 版出版，CSS 布局，曾经被视同咖啡馆诗歌一样高雅的技术，现在已经在企业建站中普及和规范。但即便是设计师在其他所有方面都很熟练了，但依然有很多 CSS 布局很浪费，很笨拙。这是一个机会均等的问题，可能出现在 Ajax 驱动的 Web 应用里，出现在类似 MySpace 那样复杂的社区网站中，也可能出现在孩子制作的邻家夫妻店网站中。（事实上，今天的孩子，比那些已经设计网站许多年的设计师更有可能使用结构化、语义化的标记。对于这个问题，无意冒犯 MySpace，MySpace 拥有一些最好的 Web 开发者，但他们似乎不太在乎用户的页面看起来是什么样子）。这个问题的根源不在 CSS，而在于页面的标记上。

在本章，我们将会对这些普遍的错误做法进行命名，从而使你能够认识并且防止它们，同时我们将学习替代这些错误做法的方法。我们还将和唯一标识符（id）属性交上朋友，它可以让你书写极度紧凑的，语义化的标记，让你为特定的页面类型定义正确的布局而不需要使用后台脚本；可以显示用户在网站的哪个层次而不需要 JavaScript；我们也将通过增加语义丰富的 class 属性，并避免 id 的滥用造成许多现代网站的语义混乱。

8.1　div、id 及其他

在本章和接下来的章节会多次提及 div 元素和 id 属性。如果正确使用，div 就是结构化代码的好帮手。而 id 是一个令人惊讶的小工具，可以用来书写紧凑性强的(X)HTML，更明智地使用 CSS，也可以通过标准 Javascript 为你的站点添加复杂的交互行为。W3C 在“一个 HTML 文档的全局结构”中是这样定义这个元素和其他两个重要的 HTML/XHTML 组件的：

> DIV 和 SPAN 元素，再加上 id 和 class 属性，提供了一个把结构添加到文档的通用机制。这些元素定义了内联的内容（SPAN）或者块级别（DIV）的内容，而没有在内容中引入其他表现层的东西。因此，设计师可以结合样式表和这些元素剪裁 HTML，以符合他们自己的需求和品味（http://www.w3.org/TR/REC-html40/struct/ global. html#h-7.5.4）。

8.1.1　为什么叫 div

有一个很好的解释，因为“div”比“division”短。当你把一组链接放在一起，这就是一块分隔（division）的文档，正文可以看做一块，页面底部的法律声明也是一块，诸如此类等等。在 XHTML 中，它是将相关元素放在一起的最好做法，可以通过给 div 元素一个语义值做相应的关联操作。如我们在第 7 章已经了解的，HTML5 取消了语义属性，用新的语义元素例如“section”、“article”来代替。

特殊结构的通用机制

所有的 HTML 开发者都对一些常用的标记很熟悉，比如段落标记 p 或者标题标记 h1，但是有些人对于 div 可能就不太熟悉了。若要理解 div 元素，W3C 的一个短

语，即“一个用来添加结构的通用机制”是关键。

当我用 HTML5 标记来写一个页面时，将使用 nav 元素来建立导航（www.zeldman.com/ x/36）。而使用 HTML4.01 或者 XHTML 时，导航则通常会使用 div class=”navigation”或者 div id=”navigation”来实现。当导航部分含有多个元素，例如第一个 ul 用做主导航，第二个 ul 用做站点工具等，我会用一个 div 来包含所有这些元素，同时传达它们共同的语义值，例如 navigation。不论是电脑还是人们阅读代码的时候，都能够明白这个 div 块是一个导航。

如果我的导航足够简单，没有子元素，我会采用 ul class=”navigation”或者 ul id=”navigation”来替代 div 元素（一些非常强调语义的开发者甚至在有很多元素的时候也这样做。）选择用 class 还是 id 取决与网页上还有什么内容。如果页面上只有一个导航，那么 id 是合适的选择；如果有多个导航，class 更加合适。我将在后面的章节更仔细说明 class 和 id 的区别。

我们必须把 id 或者 class 标记为“navigation”吗？你完全可以使用任何其他名字。“Gladys”或者“orangebox”等，都是很好的名字，是 XHTML 规则所允许的。但是有意义或元结构的名字（比如表达所包含元素的功能）是最好的。当你的客户想要蓝色的时候，而你却把这部分命名为“orangebox”是不是很奇怪？举个例子，如果从现在开始，只有 6 个月时间重新修订样式表了，而你却要拼命地回忆“Gladys”到底是代表导航区、一个缩进菜单，还是一个检索表单，这时，就会感到原来的做法很愚蠢。在互联网发展初期，我们乱写标签，在代码里插入搞笑的注释，并以此为乐趣。但是现在已经有了成熟的 Web 团队和流程，对分组内容的合理、结构化的标记，将帮助你和你的队友节约时间，提高效率。

另外，精心构思的结构化 id 标签，如“menu”或者“content”，或者“searchform”可以帮助你记住这些标记还没有布局，而一个结构良好的页面可以设计成任何你想要的外表。不管使用纯 CSS 布局还是混合的 CSS/表格布局，如果养成了给一些页面核心组件（比如导航、内容和检索区域）命名的习惯，你就会抛弃从页面表现层进行思考和开发的习惯了。

8.1.2　id 与 class

id 属性对 XHTML 来说并不陌生，同样，class 属性以及 div 和 span 元素也不是。他们都是 HTML 原来的元素。对一个元素进行 id 赋值必须提供唯一的名字，在一个页面中任何名字都只能使用一次。比如，如果你的页面包含了一个 div，它的 id 值为“content”，就不可能有另外的 div 或者其他元素拥有同样为“content”的名字。总之，ID 是用来标识唯一身份的，你不会分享你的护照或社会安全号码——它们被用来标识你，仅仅是你。ID 也一样。

相反，class 属性可以在一个页面中一次又一次地使用（比如，在一个页面中可以有 5 个段落共享一个名为“small”或者“footnote”名字的 class 属性）。下面的代码能够帮助你弄清楚 id 和 class 的区别：

```
<div id="search">
<!-- Search form components go here. This section of the page
is unique. -->
. . .
</div>
<div class="blog_entry">
 <h2>Today's blog post</h2>
 <p>Blog content goes here.</p>
<p>Here is another paragraph of blog content.</p>
 <p>Just as there can be many paragraphs on a page, so too there may be many
entries in a blog. A blog page could use multiple instances of the class "blog_entry"
(or any other class).</p>
</div>
<div class="blog_entry">
 <h2>Yesterday's blog post</h2>
 <p>In fact, here we are inside another div of class "blog-
entry".</p>
 <p>They reproduce like rabbits.</p>
 <p>If there are ten blog posts on this page, there might be
ten divs of class "blog_entry" as well.</p>
</dv>
```

在以上例子中，div id="search"用来标出页面中包含检索表单的区域模块，而 div class="blog_entry"则用来指出网站日志的每一个项目（我们所熟知的，每一篇 blog）。在这个页面中，只有一个检索表单，所以 id 用来表示这个唯一的检索表单实例。但是一个网站日志可能会有很多条，因此用 class 属性来标记。同样地，一个新闻站点也可

能用很多 div 的“news_entry”（或者“new_item”以及“news_story”）的 class 定义。

并不是任何站点都需要这些 div 元素。一个 blog 站点可以仅仅依靠 h1、h2 和 h3 以及<p>段落标记而不需要其他元素就能够处理得很好。一个新闻页面同样可以这么做。我们在上面列举的“blog_entry”的 class 属性的 div 例子不是让大家把自己的页面用 div 堆满，而是利用它来展示一个理念，那就是多个内容项目可以使用同一个类属性，而每个页面的一个内容项只能使用一个 id 属性值。(没有哪两个内容项能共用同一个 id 值)。

即时帖理论

把 id 属性设想成一个即时帖，有助于我们对它的理解。我们在冰箱上贴上一个即时帖来提醒我们要去买牛奶。一个放置在电话上的即时帖会提示我们要给一个还没有付款的客户通电话。另外，在账单的抽屉上贴一个标签，也会提醒我们必须在这个月的 15 号要支付这些账单（同样地，也会让我们记起还没有付款的客户）。

id 属性类似于在你代码中特殊区域的标签，会提示你这个区域的意图是什么，也帮助你记住这个区域在接下来的设计阶段通过CSS和JavaScript实现表现和行为的时候可能需要特殊的考虑。例如，你的 CSS 文件可能有特殊的规则来对那些 id 为“search”的 div 中的元素进行处理。或者你的样式表文件中包含只对 id 为“navigation”的 div 处理的规则。

当一个 id 属性值用来标记一组具体的 CSS 规则时，它称为 CSS 选择器。当然还有很多其他的方法能够创建一个选择器（看第 9 章“CSS 入门”），但 id 是最容易和最通用的方法。

id 的强大功能

id 属性有着令人难以置信的强大功能，它可以完成下列功能：

- 作为一个样式表选择器（看前面一节），可以让我们创建结构紧凑的、最小化的（X）HTML 页面文件；
- 作为一个超文本链接的目标 anchor，用来替代过时的“name”属性（或者为了向后兼容而和它共存）；
- 作为从基于 DOM 的脚本中特殊元素的引用方法；
- 作为一个声明的对象元素名字；

- 作为通常目的处理的工具（在 W3C 的例子中：“当从 HTML 页面提取数据到一个数据库，或者把 HTML 文档转化到另外一个格式等，这个时候用来标明某个区域”）。

id 的规则

在 CSS 里，标识符（包括元素名称，CLASS 名，选择器的 ID），只能包含字符 a~z、A~Z、0-9，以及 ISO 10646 字符 U +00A1 以上，加上连接符（-）和下画线（_），它不能以数字进行开头或者连接符(-)接着数字。标识符还可以包含转义字符和任何 ISO 10646 作为一个数字代码字符（见下项）。例如，标识符“B&W?”，可以写成“B\&W\?”或者“B\26 W\3F”。

此外，如果打算在表单 document.idname.value 的 JavaScript 中使用 id，你必须将它命名为一个有效的 javaScript 变量，也就是说，它必须以字母或下画线开始，后面跟字母、数字及下画线。不能有空格，尤其是不能有连接符，空格和连接符都是不允许的。（虽然只有在你使用 document.idname 时，无连接符规则才有效，但那是不推荐的，不管怎样，用 document.getElementById()替换它）。

8.2　让你的内容容易找到，容易使用

在这一小节讨论的元素和属性经常被误用和滥用，使网页的代码量成倍增加而不是减少，是在消除所有结构痕迹而不是为(X)HTML 的通用结构元素增加一个符合逻辑的大纲结构。

一个符合逻辑的大纲结构，正确应用时，能帮助搜索引擎很好地找到你的内容，否则会反过来帮助搜索引擎找到其他人的内容。在这本书里我反复强调这点也不为过。如果你希望用户找到你的内容，首先你要有优质的内容，其次你必须要做的是把内容按语义标记好。没有其他捷径可走，即使是在网页标题上做文章也没有用。

同样，一个大纲能帮助使用屏幕阅读器的用户浏览你的网站，如果没有它，你的网站将很难使用；视障人士是屏幕阅读器的主要用户群体；屏幕阅读软件能帮助他们“浏览”你网站上的内容，例如你网页上的 h2 标题，同样，“正常”视力的用户也能用他们的眼睛浏览到这些大标题。当你使用结构化的 HTML 元素，比如:h1，

h2，p，li 属性，再加上支持具有语义化的 class 和 id 属性的元素时，你就从一个关键词堆积，无可访问性的旧世界到了一个可发现、可用和可访问的新世界。

读完这一章，在去往新世界的路上你会感觉很好，你也会认出并避免那些让页面臃肿、无可发现性、对残障人士不友好的无语义堆积。

8.2.1 语义化标记和可重用性

既然我们已经讨论了通用的(X)HTML 元素（特别是 div 和 id），那就让我们看一下代码：

```
<div id="masthead">
<h1>Feed the Hungry Project</h1>
 <ul class="navigation">
  <li>Home</li>
  <li>About Us</li>
  <li>Get Involved</li>
</ul>
</div>
```

任何一个神志清醒的读者都能解释它。这个“masthead”（或“header”）部分是假想的 the Hungry Project 网站的 Feed。这个简单网站导航包含 3 个主要部分：Home、About us 和 Get Involved。这种绝对低带宽标准的网站对用户和对 Google 和 Bing 一样友好。

相反，一个老派的网站设计师会用 Photoshop 或者 Fireworks 做视觉设计，切割成许多的 GIF 或者 JPG 图片，用表格在网页上定位这些图片，通过 JavaScript（可能是内嵌）来做图片效果，比如图片交替显示。最终效果可能看起来一样，但需要很长时间来下载图片，对用户的可访问性不好，搜索引擎也找不到，因为图片文件替代了结构化的数据。

老式的表格布局浪费了带宽并且没有语义含义。（非语义“CSS 布局”，我们将在稍后解释，几乎一样糟糕。）上述标记呈现了一个我们努力就能很容易做到的理想。我们注意到上面的代码中没有任何 img 标记，也没有相关的宽度、高度、背景以及边框的属性代码。它完全不同于附带高度、宽度、背景颜色、行间距和列间距属性的表格代码。它像口哨般清脆，像上个世纪 80 年代的 bass line 一样单纯，并且仍旧能够提供所有的客户端或者其他相关所必需的信息。

十有八九，支持这个标记的 CSS 会通过图片替换和 CSS 的 rollover 技术实现一个能点击的公司 logo 图片来显示 h1 标题。一个用来标识“导航”的无序列表 ul，毫无疑问看上去就是一个导航条，再次感谢 CSS 的魔力。作为奖励，这个网页的结构很容易解析，即使是在有限的环境下，比如在不支持 CSS 的老式手机上，或者是因为网速慢，又或者某些国家互联网访问的限制而关闭 CSS 和图片显示功能的浏览器上。就算没有 CSS 和图片显示，网站仍然是清晰的，Feed the Hungry Project 是这个网站的标题，Home、About Us 和 Get Involved 是这个网站的导航连接，即使没有其他的支持，结构化标记也能传达信息。

删除标记里所有无语义的 img 元素不但能减少带宽成本，还能让内容可发现，可访问。因为我们把表现和页面结构分开了（把内容从外观和感觉中分离了出来），所以我们可以用 JavaScript 或者后端脚本鼓励访问者自定义阅读体验——例如：放大或者缩小文本尺寸，或者增加页面对比度来提高可读性。 我不是说上个世纪 90 年代末式的“个性化”的热潮，那时候没有人关心这些，只会在短命的门户和无用的内部使用软件中暴露出设计承诺的缺乏。我说的是有用的网页布局调整，比如 A List Apart （www.alistapart.com/articles/alternate）曾经引导的，现在已经成了一个好新闻网站的未来标准（如图 8.1 所示）。

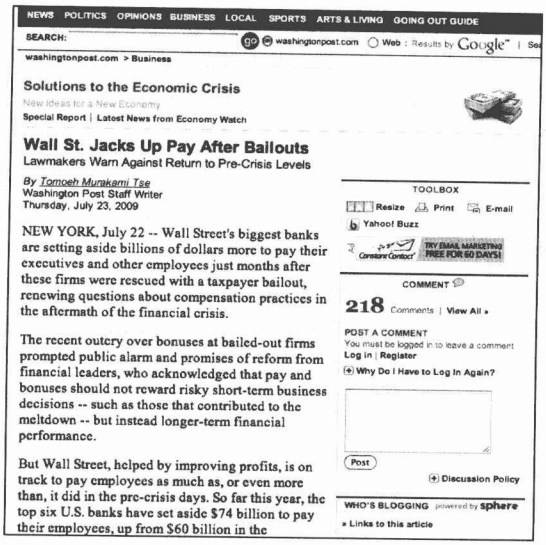


图 8.1　读者能改变文本的大小（并可以通过 cookie 记住他们的设置）是未来大多数现代新闻网站的特征之一，通过 Web 标准将结构从表现中分离出来鼓励了这种以读者为中心的考虑（www.washingtonpost.com）

标记是没有图片和表格单元的——因为它们纯粹就是页面的结构元素——只要这个网站的结构不变化，下一个小组就可以在一个全新完整的布局上以同样的标记为基础重新设计这个网站。精简，重用，继续这样循环下去。

就是这样，同样视觉设计效果下，基于标准的布局优于基于表格的布局。

8.2.2 现代标记的常见错误

唉，不是每一个使用结构 DIV 和 CSS 布局的网站都能达到那样的表现，事实上，更多的是恰恰相反。在本节中，我们要看看很多网站建造时的错误方法，并且解释为什么这些方法是适得其反的。我们也将给几个经常使用的，特别令人讨厌的方法贴上黑标签。阅读这一节不仅能改变你原来的坏习惯，还能帮助你发现同事或厂商工作中的错误。

在你已经阅读本章，并深印于心后，当同事或者厂商侥幸使用几个很傻的标记成功的时候，你可以用本章出现的恰当的标记告诉他们，你的同事和厂商会进步，你会得到尊重，最重要的是，无论何时你在他们身边，他们就会深感不安。

其实我接下来要说的这些不良书写习惯不是取笑任何人，只是为了鉴别和消除我自己工作中的这些习惯。自本书第 1 版出版以来，它们已经进入了 Web 设计的词典，并帮助许多设计师和开发者清理了他们的行为和标识，我希望它们也能帮上你的忙。

Classitis：代码标记的麻疹

时下的设计师在使用 CSS 布局替换表格布局，可是他们还在用过时的标识如 bgcolor 和 font。今天，即使生活在西伯利亚岩石下的 Web 设计师也知道我们的页面里不要出现那些东西。

```
<td align="left" valign="top" bordercolor="black"
bgcolor="white">
<font family="verdana, arial" size="2">Shoot me</font></td>
```

如果为了快速修复而放弃结构和语义，结果往往不太好，即使是如”Facebook”这样著名的网站，我们也看到太多这样的内容：

```
<div class="leftandtopalign"><p class="small"><span
class="blackborder">
Shoot the developer, then shoot me</span></p></div>
```

我们称这些代码的标记为 classitis。在那些被 classitis 折磨的站点中，所有要处理的元素都被它自己膨胀的、垃圾 class 所包围。看看这个代码：

```
<p class="bodycopy">Text goes here.</p>
<p class="bodycopy">More text goes here.</p>
<p class="bodycopy">Still more goes here.</p>
```

我们的标记要怎么替换呢？

```
<p>Text goes here.</p>
<p>More text goes here.</p>
<p>Still more goes here.</p>
```

如果需要用一个 class 定义页面特定部分的每一个段落，可能需要做的是将我们的段落封装在一个 div 里来传递页面这部分的结构，并且用一个包含块（block）作为目标机制来定义此部分所有的段落的样式，本书稍后会有技术解释 CSS 的细节，下面是标记看起来的样子：

```
<div id="maincontent">
 <p>Text goes here.</p>
 <p>More text goes here.</p>
 <p>Still more goes here.</p>
</div>
```

Classitis 在每个页面增加了不必要的代码，从而让含义变得含糊不清。这种困惑的源头可追溯到半支持 CSS 的早期浏览器时代以及很多设计师最初对 CSS 如何工作的幼稚理解。

哎，许多设计师还没有成长起来，对 CSS 不理解，这可能是因为他们转去学习其他技术，或者他们的可视化编辑工具对每个它产生的标记都添加了一个 class，并且他们“学习”CSS 是通过这个工具所产生的代码。Classitis 就和我们原有的<font>标记方式一样糟糕，好的标记代码是很少需要使用它的。

可视化网页编辑器和 Classitis

即使是最好的、最成熟的可视化 Web 编辑器，也一样会产生一些不需要的 class 样式，主要原因是它们毕竟是个可视化工具，而不是人。人们可以举一反三，可以对特殊和通用的东西进行抽象。而当你使用 CSS 对一个段落进行排版的时候，能知道你想要所有的段落都可以通过同一种方式来处理。但是对于一个可视化编辑器，它不知道你想要什么。假如，当使用这种编辑器工作的时候，你要创建 5 个都使用 0.75em Verdana 字号的段落，这可能会对每个段落都分配了一个 class 样式。最新版本的 Dreamweaver 和 Microsoft Expression 已经非常成熟、功能强大，并且对标准是兼容的。但是你仍然可能需要对它们产生的代码进行手工编辑而避免产生 classitis 和 divitis 错误。

Divitis 的失望

在好一点的情况下，classitis 会被复制到其他结构化标记中：

```
<p class="noindent">This is a bad way to design Web pages.</p>
<p class="indentnomargintop">There is no need for all these
classes.</p>
<p class="indentnomargintop">Classy designers avoid this
problem.</p>
<p class="indentnomargintop">Class dismissed.</p>
```

上面的例子是一些很成熟并且与标准兼容的可视化编辑程序可能产生的代码（参见“可视化网页编辑器和 classists”）。有时候，会出现更严重的情形，类似的 classistis 会更恶化，我们把这种情况称为 divitis：

```
<div class="primarycontent"><div class="yellowbox"><div class=
"heading"><span class="biggertext">Welcome</span> to the
Member page!</div><div class="bodytext">Welcome returning
members.</div><div class="warning1">If you are not a member <a
href="/gohere/" class="warning2">go here</a>!</div></div></div>
```

这里，我们没有看到任何结构，只有很多非结构化的代码和大量没必要的垃圾代码。用 Palm 或者文本浏览器访问这样的站点，你就不知道这些元素是怎么和其他元素关联在一起的。事实上，它们不关联其他任何元素。

Divitis 用空的垃圾代码代替了结构化的代码，它们就像花园中的杂草。

从你的代码中无情地切除那些 classitis 后，我们将看到一个臃肿的 Web 页面缩减了一半的大小。避免了 divitis，你会发现你能够书写干净的、紧凑的、结构化的代码，它们能够在一个文本浏览器中正常地工作，就像在你喜爱的桌面浏览器里一样。严格按照这种方式去做，你就会得到自己创建更灵巧、更兼容的 Web 页面的方法。就像 Smokey 说的，只有你才能防止 divitis。

8.2.3　divs 刚刚好

你们当中的一些人可能会想："嘿，写了这个奇怪标准书籍的家伙，你说 div 元素是不好的，然而你自己却在本章的第一个例子中使用了 div 元素。我已经抓住你的大谎言了，你这个坏家伙"。

事实上，我们没有说过 div 元素是不好的，同样创造过所有这些元素的 W3C 也没有说过（如果你跳过了那一节的话，请回去参考本章前面的章节"div、id 和其他助手"）。Div 是一个非常适合模块化站点的结构区域的标记。

当你使用 div 来替换那些好的（或者说更适当的）元素的时候，Divitis 就出现了。如果你写了一个段落，它应该用<p>标记，而不是 。你的最高级别的标题应该为<h1>，而不是 。看到区别了吗？我想你肯定看到了。

为什么用 div

许多设计师使用非结构化的 div 来代替任何元素，从标题到段落。大多数习惯在 4.0 浏览器出现时候就形成了，特别是 Netscape，默认的样式会破坏布局，比如：不必要的围绕在如<h1>的元素周围的空白，但是它们并不破坏使用非结构性 div 的布局。

为了那些快速消失的 4.0 浏览器用户而保护近乎完美的布局，许多设计师坚持使用非结构化的 div 元素而不是使用标准的段落和标题等，这么做的成本非常高，也无法让智能电话、无线设备和屏幕阅读机的用户访问你的站点。不管他们用的是什么样的浏览器和设备，都会增加 2 至 3 倍的带宽，这根本就不值得。

另外一个典型的 divitis 例子是，设计师中了“表格都是不好的，CSS 都是好的”的毒，所以他们用大量嵌套的 div 代替表格。然而除了自鸣得意外，他们得不到任何好处，而且使得文档更加难以编辑。

8.2.4 热爱 id

如果表现层的 HTML 代码被淘汰了，classitis 和 divitis 也被我们放弃了，那我们怎么去表达一个元素是“特别”的，需要特别处理呢？我们可以通过 id 的魔力来实现它。给包含菜单的表格赋予一个唯一的、元结构化的标记，如 ul id="navigation" 或者 div id="search"，接下来，在书写样式表的时候，你就可以创建一个选择器，如 ul#navigation 或者 div#search 以及关联的一些 CSS 规则，控制元素及其子集按你希望的效果显示，然后可以使用同样的选择器在你的 JavaScript 里去控制这些元素及其子集的行为。

8.2.5 清除（或最小化）内嵌 CSS 和脚本

如果要建立精简、语义化的网页，最终我们必须要清除内嵌 CSS 和脚本。类似以下的代码标记会增加不必要的带宽，同时会给我们更新设计或者移植到另外一个 CMS 系统带来困难。

```
<div style="float: left; width: 500px;"><p style="font-family:
'Helvetica Neue', Arial, Helvetica, sans-serif;">Got some
Helvetica, y'all!</p></div>
```

如果你留意一下，就会知道这个 CSS 应该归入样式表而不应该出现在页面标记里面。

```
<span id="somebehavior" onclick="SomeFunction();">Some text</span>
```

虽然我们都是这样学习 JavaScript 代码的（虽然这本书第 1 版中的 JavaScript 正是这样写的），但在今天，这是错误的。第 15 章将阐述更多的细节，但内嵌 JavaScript 中的问题包括：

- 和内嵌 CSS、divitis、classitis，以及老式的表格布局技术一样，它会增加不必要的带宽，和等待时间；

- 和内嵌 CSS 一样，内嵌脚本没有缓存，强制用户每次都要下载并增加了进一步处理的负担；

- 和内嵌 CSS 一样，和内嵌 JavaScript 很难维护，因为你的代码分散在各个地方，而不是存储在一个单一合理的地方；

- 最后，内嵌 JavaScript 没有退回交互处理机制，对一些用户来说不可访问。与此相反，第 15 章会介绍的不唐突的脚本，即使在不支持 JavaScript 的环境下也能工作。

8.2.6　暂停并回顾

我们现在已经学会了所有需要知道的知识来创建简洁、健壮的标记——纯结构化和语义化的标记（至少，和 SGML 子集一样语义化），现在来到了有趣的部分：通过漂亮优雅的 CSS 来增加设计效果，如果你准备好了，请翻到下一页。

第 9 章

CSS 入门

在本章我们将快速学习 CSS 的基础语法，然后了解一些不那么基本的概念，最后还会介绍一种 CSS 设计方法。即使你对 CSS 非常熟悉，也建议你浏览一下本章的知识。

9.1 CSS 概述

W3C 把 CSS 简易定义为“一种对 Web 文档添加样式（例如：字体、颜色、间距）的简单机制”（www.w3.org/Style/css/）。CSS 概述中一些省略掉的细节也值得我们注意：

- CSS 是 Web 标准化布局语言，它可以控制颜色、字体，以及元素、图像的尺寸和布局；
- 正如本章所示，尽管 CSS 精确而又强大，但是它手工书写起来很简单。CSS 创始人 Håkon Lie 和 Bert Bos 舍弃深奥复杂，设计出了一个容易理解、容易学习、容易教授、容易实现、容易维护和容易扩展的 Web 表现层语言 (www.zeldman.com/2009/07/24/why-standards-fail)；
- CSS 对带宽的节省：只要一个 CSS 文件就能控制包含上千个页面和几百兆字节的整个站点的外观；

- 它的创建者（W3C）的意图是以 CSS 取代 HTML 表格式布局、帧和其他外观技巧，还原 HTML 标记的本来目的：按语义来描述我们的内容；

- 9.2 节“CSS 的优点”将会描述，纯 CSS 布局与结构化 XHTML 相结合能帮助设计师把表现从结构中分离出来，使站点更具有可访问性以及更容易维护。

9.2　CSS 的优点

有句俄罗斯谚语说道：“重复乃学习之母”。原谅我们的罗嗦和一再提醒：同其他的 Web 标准一样，创造 CSS 并不是为了一个抽象的目的和遥远的目标，而是针对当前，现在 CSS 就能提供以下这些实用性方面的好处（但不仅限于此）：

- 节省用户的带宽，加速网页加载时间，特别是当使用拨号上网时；

- 减少服务器和带宽的费用，以节约资金。（参见第 1 章）；

- 一旦掌握以后，就能缩短设计和开发的时间，便于动态原型开发。这使得 CSS 特别适合敏捷开发和 Web 应用的设计。（当然这种时间上的节省仅限于开发，用户体验设计、内容规划、界面设计和艺术设计需要的时间仍然和以前一样）；

- 缩短更新和维护的时间；

- 当文档更新时，人们无须再担心复杂的表格、字体标签以及其他旧式的布局组件。因为不存在这样的元素（或极少），所以几乎没有这方面的问题；

- 设计师、开发者和代理商无须再担心用户会破坏站点；

- 所有修改能在几分钟内完成。如果文本颜色太暗，只要修改 CSS 文件里的一到两个规则，整个站点将立刻呈现修改后的效果；

- 遵守 W3C 推荐的 Web 标准以增强交互性；

- 从标记代码里清除表现元素以提高可访问性。

9.3 样式解析

这一节介绍 CSS 的各个细节部分。这本书并不是完整的 CSS 指导手册，CSS 指导手册中的内容要比本书多得多，虽然我们希望 CSS 指导手册能小到可以放进口袋——请看“CSS 相关书籍介绍”。但从另一方面来说，又有多少 CSS 指导手册可以写得像本书这么详细？这本书绝对是物有所值的。我们先来看看这些书，然后再看详细的样式解析。

CSS 相关书籍介绍

The CSS Pocket Reference，《CSS 袖珍指导》（Eric Meyer:O'Reilly&Associates, Inc., 2001），如书名所示，你可以把它放在口袋或小包里。别以为书小就用处不大，实际上它是 CSS1 最清晰完整的指导书。

另外也推荐以下几本书。

- Bulletproof Web Design (New Riders: Dan Cederholm)
- Handcrafted CSS (New Riders: Dan Cederholm and Ethan Marcotte)
- Transcending CSS: The Fine Art of Web Design (New Riders: Andy Clarke)
- CSS Mastery: Advanced Web Standards Solutions (Friends of Ed: Andy Budd)
- Web Standards Creativity (Friends of Ed: Budd, Weychert, Lloyd, Rubin, et. al)
- Stylin' with CSS: A Designer's Guide (New Riders: Charles Wyke-Smith)
- The Zen of CSS Design (New Riders: Dave Shea, Molly E. Holzschlag)

9.3.1 选择器、声明、属性和值

样式表由一个或多个规则定义组成，这些定义控制所选择的元素将如何显示。一个 CSS 定义由两部分构成：选择器和声明。

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

p 是选择器，而在花括号内的{color:red;}就是声明。相应地，声明也由两部分组成：属性和值。在上面那个声明里，color 是属性，red 就是属性值。

选择和选项

除了英文单词 red 之外，我们还可以写成红色的十六进制值#ff0000：

```
p { color: #ff0000; }
```

十六进制的颜色由三对数值组成（“FF”表示红色的值，“00”表示蓝色和绿色的值），若是用 CSS 简写规则可以节省一些字节，而效果完全一样：

```
p { color: #f00; }
```

我们也可以用 RGB 值，下面两种方法都可以：

```
p { color: rgb(255,0,0); }
p { color: rgb(100%,0%,0%); }
```

关于零值

注意使用 RGB 百分比值时，甚至当值为零时也要带百分比符号，在其他 CSS 情况里则不需要带单位符号。例如，指定像素大小时，0 后面可以不跟 px。

很多人认为写出 0px、0in、0pt 或 0cm 没有必要，零就是零。值为零时谁会关心后面的单位是什么呢，但指定 RGB 百分比时，零值要求带百分比符号。不要问我为什么，反正这样写就对了。

多重声明

只指定文字颜色而不指定背景的颜色，是一种不好的做法，反之亦然：

```
p { color: #f00; background-color: white; }
```

注意一个定义可以由不止一个声明构成，可以用分号来区分不同的声明。

分号的作用

这里我们再次谈到矛盾和例外情况。声明语句中最后一个规则并不需要以分号结尾，一些设计师会在一系列的声明后省略最后一个分号（因为在英文中，分号的作用是分隔而不是终结）。

但是大多数有经验的 CSS 编写者在每个声明后都加上了分号，他们这样做在一定程度上是为了保持连贯性，但主要还是为了避免从现有的规则中增加或减少声明时的麻烦。如果每对属性/值都以分号结尾，移动声明位置的时候就不用担心会产生问题。

空白和不区分大小写

大多数样式表本身含有不止一个定义，而多数定义又含有不止一个声明。使用空白将使多重声明更容易理解，编辑样式表也更容易：

```
body {
    color: #000;
    background: #fff;
    margin: 0;
    padding: 0;
    font-family: Georgia, Palatino, serif;
}
p {
    font-size: small;
}
```

用不用空白对 CSS 在浏览器上的显示没有影响，与 XHTML 不同，CSS 选择器是不区分大小写的，例如上面的选择器也可以写成 BODY 和 P，对浏览器来说是一样有效的。当然也有例外，当与一个 HTML 文件联用时，class 和 id 名称是区分大小写的，myText 和 mytext 在这种情况下是不匹配的。CSS 本身不区分大小写，但文档语言也许就区分了。

9.3.2 可选值和默认值

网页设计师可以为整个站点指定字体，如下：

```
body {
    font-family: "Lucida Grande", Verdana, Lucida, Arial,Helvetica, sans-serif;
}
```

注意这个由两个单词组成的字体名称（Lucida Grande）必须用 ASCII 引号框起来，逗号要在后面的那个引号之后而不是在它之前，美国的设计师对此有点烦恼，因为他们习惯了把逗号放在引号的里面。

字体会按照所列出的顺序选用。如果用户的计算机含有 Lucida Grande 字体，文档将被指定为 Lucida Grande。没有的话，就被指定为 Verdana 字体。如果也没有 Verdana，就指定为 Lucida 字体，依此类推。为什么这些字体要按照这个顺序排列呢？

排序是为了适应不同的平台

顺序很重要。Lucida Grande 字体在 Mac OSX 里可以找到，而所有现代的 Windows 系统、Mac OSX 和旧版本的 Mac 操作系统（“Classic”）里都有 Verdana 字体。所以如果把 Verdana 字体列在第一位，OSX Mac 就会显示 Verdana 字体而不是 Lucida Grande 字体。

使用列在最前面的 Lucida Grande 和 Verdana 两种字体，设计师能满足几乎所有用户的要求（Windows 和 Macintosh 用户）。Lucida 适用于 UNIX 用户，Arial 适用于旧版本的 Windows 用户，Helvetica 适用于旧版本的 UNIX 系统。如果所列出的字体都不能用，那么默认的 sans-serif 字体能保证调用。万一用户的计算机没有 sans-serif 字体，将使用浏览器默认字体。如果浏览器没有默认字体，那就用大猩猩的牙齿来代替。最后一句是开玩笑。

并非完美的科学

没人会认为 Lucida、Verdana、Arial、Helvetica 在美观、优雅、宽度、x 高度（desktoppub.about.com/od/glossary/g/xheight.htm）和屏幕的适用性方面都一样（即使 Helvetica 各方面都同样出色）。我们的目的不是为了给所有的用户创造相同的视觉效果；平台、浏览器、显示器尺寸、显示器分辨率、显示质量、gamma 值，以及操作系统的抗锯齿设置都有可能不同，因此保持相同的视觉效果是完全不可能的。我们只是希望在他们的条件允许的情况下尽可能保证最佳视觉效果，让不同的用户看到

的效果比较一致。值得一提的是，可以调节 font-size-adjust 属性的值来保持字体的 x 高度获得一致的表现。技术编辑 Aaron Gustafson 曾在编写由 Happy Cog 设计的 Brighter Planet 网站时采用了这种方法（www.brighterplanet.com）。

CSS 的@font-face 属性可以嵌入一个字体到页面中，而不再遵循上面的通用规则，对此我们将会用一个独立的章节来谈论。

群组选择器

当几个元素共享样式属性时，用逗号分隔，可以使多个选择器调用一个声明：

```
p, td, ul, ol, ul, li, dl, dt, dd {
    font-size: 1.2em;
}
```

当结合使用下节将要讨论的样式继承特性时，群组选择器将发挥真正强大的作用。

9.3.3 继承和它的不足之处

根据 CSS 规范，子元素可以从父元素继承特性，但不总是如此，见下面的定义：

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

学到第 9 章，你应该能理解这个定义表达的含义了。它是指在用户系统里发现 Verdana 字体时，站点的 body 元素将被指定为 Verdana 字体，否则指定为通用的 sans-serif 字体。

所有子元素

每个 CSS 定义都有继承性，对最上级元素的定义（这里是指 body 元素）对其子元素（p、td、ul、ol、ul、li、dl、dt 和 dd）也起作用。不必重复为每个子元素添加定义，body 的所有子元素都将显示为 Verdana 或者通用的 sans-serif 字体，每一级的子元素也将显示为这样的字体，大多数现代的浏览器都支持这项功能。

（由于 Netscape 4 这类古老的浏览器不支持继承，所以在本书以前版本的

“Netscape 4 友好”一节中推荐了给 body 的每个子元素都进行明确定义的方法。而现在我们已经不再需要这么做了，如果你正在清理多年前写的 CSS 代码，那么可以删除这些多余的代码了，相信现在的浏览器已经能够很好地处理继承了。）

继承会是一场灾难吗？

如果你不希望每个子元素都继承为 Verdana、sans-serif 字体，比如，你想要用 Times 字体显示段落，怎么办呢？没问题，为 p 创建一个特定的定义（在下列代码中用粗体表示），它将覆盖掉父元素的定义。

```
body {
    font-family: Verdana, sans-serif;
}
p {
    font-family: Times, "Times New Roman", serif;
}
```

有效的 Times 字体也能被所有浏览器接受，在此不再赘述。

9.3.4　派生选择器

你可以根据元素出现的上下文来决定元素的样式，从而避免使用不必要的 classitis，并保持标记的简洁整齐。采用这个方法的选择器在 CSS1 里被称为“上下文选择器”，因为元素只有在指定的上下文里才会应用特定的样式。在 CSS2 里称为派生选择器，无论名称如何，作用都是一样的。（在这本书里我们还是称之为派生选择器。）

若要指定当出现在列表项里时，标记为 strong 的文档显示为斜体而不是粗体，应该这样定义：

```
li strong {
  font-style: italic;
  font-weight: normal;
}
```

这样的规则能做些什么呢？

```
<p><strong>I am bold and not italic because I do not appear in
```

```
a list item. The rule has no effect on me.</strong></p>
<ol>
<li><strong>I am italic and of Roman (normal) weight because I
occur within a list item.</strong></li>
<li>I am ordinary text in this list.</li>
</ol>
```

或者考虑下面的 CSS：

```
strong {
    color: red;
}
h2 {
    color: red;
}
h2 strong {
    color: blue;
}
```

以及它们对标记的影响：

```
<p>The strongly emphasized word in this paragraph is
<strong>red</strong>.</p>
<h2>This subhead is also red.</h2>
<h2>The strongly emphasized word in this subhead is
<strong>blue</strong>.</h2>
```

你可能不会用派生选择器来把列表中的粗体字文本指定为斜体。但你也许会用这种方法来实现复杂的设计，例如给一个普通的(X)HTML 元素添加背景图像和大量的空白（边框）以防止文档和图像重叠。但是更有可能的是用 id 或 class 选择器来实现这些效果。

id 选择器和 id 派生选择器

在现代的布局中，第 6 章介绍的 id 选择器经常会和派生选择器一起使用：

```
#sidebar p {
  font-style: italic;
  text-align: right;
  margin-top: 0.5em;
}
```

上面的样式只用于指定 id 属性为 sidebar 的元素里的段落。该元素极可能是一个 div 元素，也可能是一个表格或是其他块级元素，甚至可能是一个内联元素，如或。这样的用法有些古怪，也是不合法的——不能把 p 元素套入 span 元素中。不管是什么元素，它必须是该页中唯一使用 id 属性为“sidebar”的元素。如果不记得原因，可以看看第 6 章。

一个选择器，多种用法

即使标记为“sidebar”的元素在页面中只出现一次，id 选择器在需要时也可以被多次用作派生选择器：

```
#sidebar p {
    font-style: italic;
    text-align: right;
    margin-top: 0.5em;
  }
#sidebar h2 {
    font-size: 1em;
    font-weight: normal;
    font-style: italic;
    margin: 0;
    line-height: 1.5;
    text-align: right;
  }
```

这里 sidebar 属性里的 p、h2 元素不同于该页中的其他 p、h2 元素，两者均被特别定义。

选择器可以单独使用

id 选择器并不是必须跟派生选择器一起使用，它可以单独使用：

```
#sidebar {
  border: 1px dotted #000;
  padding: 10px;
}
```

根据这个定义，id 属性为 sidebar 的页面元素将有黑色（#000）点状粗细为 1 像素的边框和 10 像素的内边距（内部空白）。

9.3.5 class 选择器

在生活中以类取物是很糟糕的事情，但在 CSS 中却是一件好事。在 CSS 里，一个点表示类别选择器：

```
.fancy {
  color: #f60;
  background: #666;
}
```

任何 class 名称为“fancy”的元素会是橙色（#f60），背景是灰色（#666）。因此<h1 class="fancy">Boy Howdy!</h1>和 Yee haw! 都会应用这个颜色方案。

如 id 一样，class 选择器也可跟派生选择器一起使用：

```
.fancy td {
  color: #f60;
  background: #666;
}
```

在上面的例子中，那些包含在 class 名称为“fancy”的元素里的表格单元文本都显示为橙色，背景显示为灰色（class 名称为“fancy”的较大元素可能是表格的一行，也可能是一个 div）。

也可以根据它们的 class 名来选择元素：

```
td.fancy {
  color: #f60;
  background: #666;
}
```

上例中 class 名称为“fancy”的表格单元将显示为橙色，背景为灰色。

```
<td class="fancy">
```

你可以把 class 名称“fancy”仅仅指定给一个表格单元，也可以指定给很多表格单元。这些表格单元将显示为橙色，背景为灰色。而没被指定 class 名称为“fancy”的表格单元将不受这个定义的影响。使用 class 名称为“fancy”的段落和其他元素的文字都不会显示成橙色，也不会有灰色背景。这种效果仅限于 class 名称为“fancy”的表格单元，因为我们就是这样定义的（用 td 元素来选择 class 名称“fancy”）。

组合使用选择器创造精致的设计效果

class、id 和派生选择器可以组合使用，从而创造出精致或突出的视觉效果。Housing Works 是一个致力于帮助流浪者的非营利组织。由 Happy Cog 制作的 Housing Works 站点上，用一张取自品牌图片中的屋顶状图片（bullet-std.png）来代替大部分无序列表前乏味的黑点（如图 9.1 所示）。

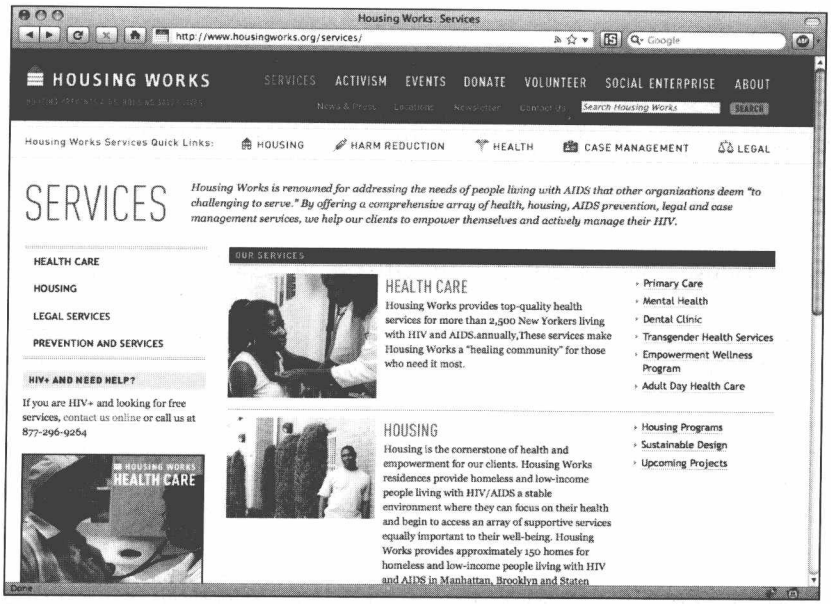


图 9.1　class、id 和派生选择器可以组合使用制造精致或突出的视觉效果。在这个站点的某些部分（http://www.housingworks.org/services）用取自商标形象里的粉色三角形代替了大多数无序列表前乏味的黑点

下面的 CSS 规则告诉 class 名称为“std”的无序列表，显示图片而不是黑点。

```
ul.std {
    list-style: none;
}
ul.std li {
    background: url(../i/screen/bullet-std.png) no-repeat 0.5em;
    margin: 0;
    padding: 0 0 5px 10px;
}
```

为了节省书本页面，这里是部分调用此样式的标记代码：

```
<ul class="microlist std">
<li><a href="/services/health-care/1-primary-care/">PrimaryCare</a></li>
<li><a href="/services/health-care/2-mental-health/">Mental
Health</a></li>
<li><a href="/services/health-care/3-dental/">Dental Clinic</
a></li>
</ul>
```

如果按进度学习，你就会明白，使用其他 class 名称的无序列表是不会显示粉色三角形图片的，而且如果这个 class 名称被用于其他元素而非无序列表，那么也不会出现粉色三角形的图片。

另外，在 Housing Works 站点上，还应用了群组选择器：

```
.secondary-content ul li,
ul.std li,
.full-col ul.std li li,
.home .col ul.std li {
    background: url(../i/screen/bullet-std.png) no-repeat 0.5em;
    margin: 0;
    padding: 0 0 5px 10px;
}
```

为什么要把 CSS 规则用这种方式凑在一起？用一条规则把多个选择器群组到一起，我们就可以把这个样式用到多个散落在各处的 class 名称为“std”的元素上。这意味着我们的标记代码可以保持精简，而让 CSS 去处理繁重的工作。

继续学习

在下一章节，我们将学习更多的 CSS 语法，但是现在，我们必须停下来考虑一个问题：我们应该把 CSS 写在哪儿？它能嵌入(X)HTML 中吗？它是一个单独的文件吗？或者是其他的什么方式？（提示：继续往下看。）

9.3.6 外联、嵌入、内联样式

有三种方式把样式表应用到页面上：外联、嵌入或内联，我们从最好的方式说起。

外联样式表

外联样式表（CSS 文件）是一个和它控制的(X)HTML 页面分别存在不同的地方的文本文档。(X)HTML 页面通过文档顶端的一个链接引用那个 CSS 文件，或者在 style 元素中导入 CSS 文件（也位于文档的顶端）。样式表的链接看起来像这样：

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

@import 命令用于导入样式表，就像这样：

```
<style type="text/css" media="screen">
@import "/styles/mystylesheet.css";
</style>
```

或者这样写：

```
<style type="text/css" media="screen">
@import url("/styles/mystylesheet.css");
</style>
```

无论是用链接还是导入方式，外联样式表都能以最低的成本提供最强大的功能。当用户把一个外联样式表下载到本地缓存后，它就能持续发挥作用，它能控制站点中一个、数十个、数百个、乃至数万个页面，而不需要再另外下载，极为方便又功能强大。

使用外联样式表还可以获得一个很大的好处，就是能大大地节省用户和服务器的带宽。

嵌入样式表

除了链接和导入一个或多个独立的样式表文件外，设计师还可在 XHTML1.0 页面的 head 位置嵌入样式表，使用的 style 元素（用粗体突出显示）如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-type" content="text/html;charset=utf-8" />
<title>Site Title</title>
<style type="text/css">
```

```
<!--
body {
  background: #FFF;
  color: #000;
  }
-->
</style>
</head>
```

与链接和导入样式不同的是，嵌入样式并不节约带宽，因为用户每次打开新网页时必须下载一个新的嵌入式样式表。即使每页的样式表都相同，用户也必须重新下载。为什么设计师还要用嵌入式样式表呢？几点原因如下：

- 站点可能只由一个页面构成。我并不认为这是适当的理由，只是说有这种可能性；
- 用户还在使用 IE3 访问站点。IE3 是最早开始支持 CSS 的浏览器，但是它不支持外联样式表。好吧，仔细考虑一下，这好像也不是很适当的理由；
- 设计师已经用外联样式表控制整个站点了，但仅仅有一个页面需要额外写定义。那么用嵌入方式写样式表就有很好的理由了；
- 设计师在不断地修改样式表，并且需要立刻看到修改后的效果。这是另一个很好的理由。

设计站点时，在你要设计的页面<head>中嵌入样式表是非常方便的，当你对你的设计满意以后，就把样式复制到外联 CSS 文件中，删除标记中的嵌入式样式。我每次设计站点时都是这么做的。（注意：许多全职专业的 Web 前端开发者会建立一个本地的服务器，然后用绝对路径调用 CSS 文件来代替嵌入样式表的方式。）

用@import 导入多个文件

这里还有另外的方法可以使用：就是在 CSS 文件中可以用@import 导入其他的 CSS 文件。（我们马上可以看到。）我们有一个链接到 main.css 文件的链接：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
<title>Your Title Here</title>
<link rel="stylesheet" href="/css/main.css" type=
"text/css" media="screen, projection" />
</head>
```

如果打开 main.css 文件，我们可以看到如下内容：

```
@import url("reset.css");
@import url("core.css");
@import url("lib/sIFR.css");
@import url("lib/lightbox.css");
@import url("2009-fall-colors.css");
```

通过 main.css 这个“容器”文件，那些导入的样式表被链接到我们的 XHTML 页面，并且依次执行每一条样式规则。这种方式有什么好处呢？

- 如果你开发的是一个有一定规模的站点，或者是一整套的应用，你也许会从 CSS 文件中提出可共享的部分。

- 一些第三方的 UI 组件，像 Lightbox(www.huddletogether.com/projects/lightbox) 和 sIFR(wiki.novemberborn.net/sifr3)就需要你包含一个 CSS 文件才能使用。在这种情况下，把这些样式表跟你自己的样式表分开管理，对将来的更新维护会有很大的帮助。

- 也许你仅仅是为了设计的目的，把你的 CSS 分成多个文件，给它们取不同的名称——typography.css，colors.css 等等。

用一个样式表链接导入多个样式表，这种方法通常是有用的。但是，需要注意的是，每个引用到文档中的样式表文件都需要额外请求一次服务器，对于一个高访问量的网站，这可能会影响一点性能。另外，Microsoft 的浏览器通常不会对超过一定深度导入的样式表进行缓存。而且，建立一大堆 CSS 文件对于维护可能也会很头疼。另一方面，那些精于后端开发的专家有一个两全其美的办法，就是当服务器端

接受到请求时把多个 CSS 文件合并成一个。这些技术的运用，需要根据你和用户的最优选择来取舍。

内联样式

通过加在元素上的样式属性，CSS 可以应用在单个元素上。举例如下：

```
<h1 style="font-family: Verdana, Arial, sans-serif;">Headline</h1>
<img style="margin-top: 25px;">
```

正如你所知，使用内联样式不会节省带宽，每个使用它们的页面都会增加带宽，这跟使用 font 标签一样浪费带宽。

我们不应该把内联 CSS 作为主要的方式来定义网页的样式，那就像用小瓶的修正液来粉刷房子一样荒唐。但是如果谨慎地使用内联 CSS，那会是一个有用的工具。内联样式是 CSS 的修饰和补充。比如，如果你必须在某页上改变某一个经常使用的元素的位置，就可以使用内联 CSS 来节省带宽，而不是在元素上设定一个只会引用一次的 class 名称，然后用嵌入的方式把该 class 的样式写在页面上。

9.4 “最合适方案”的设计方法

以前，当我们几乎全部使用表现标记创建布局时，要在硬盘上最老最劣质的浏览器上测试成果。为了使它在老式浏览器上看上去显得更好，我们创建嵌套表格，用非结构化 div 代替像 h1、h2、li 和 p 之类的结构元素，以及其他一些我们再也不想使用的方法。

如果站点在老浏览器上看起来不错，再拿到新浏览器上测试，看上去效果可能也会相当好，可是代价就是带宽和语义。很多网页设计师会这样做——用他们手头上最差的浏览器上测试。但是这个方法的代价太高，现在已经很少使用了。

替代的方法是用嵌入方式书写 CSS 样式表，然后在可靠的浏览器上预览，如最新版的 Firefox、Opera，或 Safari。用这个方法，你可以制作出可访问性高、占用带宽少、遵循标准的网页。

当你对设计满意后，就可以在其他优良且遵循标准的浏览器上进行测试，比如 IE8。在这些浏览器上显示的效果都应该一样，如果不一样，你就要再做些修改。可能是某个 CSS 定义书写得不正确，而你最喜爱的浏览器却能理解你的意图，就像一个好朋友，即使你嘴里塞满食物，他也明白你在说什么。这时 W3C 的标记及 CSS 校验就可以帮助你在测试之前找到这些小失误。

当然，不是所有的浏览器都是 100%遵循标准的——就像我们前面所提的那些浏览器，它们都不是完全遵循标准的。那么，当不好的浏览器遇上好的代码会发生什么呢？如果你的站点在优良的浏览器上表现得既正确又完美，但是在一些对标准遵循得不怎么好的浏览器上却出现了问题，那么，你需要考虑使用一点特殊的技巧和变通的办法。这类问题我们会在第 10 章里解释，请继续阅读。

第 10 章

CSS 布局：标记、盒模型和浮动

在这一章，我们将把前面所学的标准知识运用到实践中，用符合标准的方法来设计一个页面布局。我们将把所有的技能都集中体现在一个独立页面上，就好象某种白热化的激光。也许我的比喻不怎么样，但你一定会成功的。我们将运用在第 9 章学习的知识、理论和标准方法在这一章做一个简单练习。相信我，你就看着自己是如何成功的吧。

首先需要更好地理解 CSS 的细节，让我们来了解一下页面布局的机制，以及样式表是如何影响它们的。

10.1 页面流之道

页面上的每个元素都根据文档流来占据一个位置，而且会影响它周围元素的位置。不管是什么标记，每个元素都会在文档流中建立一个“盒”，这个盒的属性能够由 CSS 来控制。

有两种类型的盒。

1. 块盒，用它们包含的元素水平向占据整行宽度，并把随后的元素挤到下一行中。段落元素、列表元素、table、div 和标题类元素，比如 h1 和 h2 等，都是具有块行为的例子。

2. 与此相对应的是内联盒——比如由 a、img、em、span 和 strong 元素产生的——它们包含的元素会水平向排列，并和它们邻近的元素排在同一行中。如果内联元素的盒没有足够的宽度容纳，那么它们就会折到下一行中。

如果没有应用 CSS 样式，页面上的每个 HTML 元素都会直接表现为两种盒模型中的一种。这就是为什么段落元素和标题元素都会形成一行行的，一个 h2 元素会排在 h1 元素的下一行，而在 h1 元素中却可以排列着多个链接元素。然而，这些行为仅仅是一个默认的表现，这在本章的其余部分以及你运用 Web 标准来设计页面的时候到处都可以看到。而使用 CSS 是可以完全覆盖掉这些默认表现的。

让我们仔细地研究一下这些盒和它们的组成部分。它们对于本章及后面章节中的布局学习至关重要，对于你将来从事 Web 标准的职业生涯也同样极其重要。

10.2　盒模型

图 10.1 说明了 CSS 盒模型的四个区域。每一个盒，不管是内联的还是块的，在最中间的都是内容区域，之外包裹着内边距（padding）和边框（border）。在这个盒的四周还有一圈外边距（margin），以此来确定与其他元素的间隔距离。

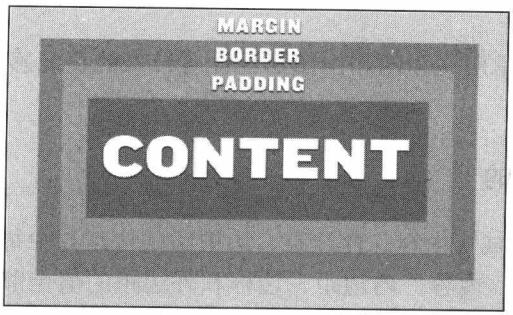


图 10.1　CSS 盒模型的四个区域：内容、内边距、边框、外边距（为了看得更清楚，外边距故意使用了暗色填充。）

我们使用几行 CSS 就可以很容易地改变这四个属性的值：

```
#content {
  border: 5px solid #000;
  height: 200px;
```

```
  margin: 10px;
  padding: 20px;
  width: 400px;
}
```

在上面这条 CSS 规则里，我们选择出一个 id 属性为“content”的元素，然后对它的内容区指定了宽度和高度，又指定了一个 20 像素的内边距和 5 像素的黑色边框。注意：我们并不是写成这样：

```
content: 400px;
```

也不是写成这样：

```
content-area-width: 400px;
content-area-height: 75px;
```

在 CSS 中没有这样的属性。边框、外边距和内边距都是直接用名字指定它们的值，但是内容区却不是。当刚开始学习使用 CSS 时，你也许会认为 width:400px 是应用于整个盒的（除了外边距）。毕竟，页面看上去就是这样布局的，设计师也是这样想的，包括用户也是这样理解布局的。如果你建立了一个 CSS 布局，它包含了两个相挨着的 div 元素，然后对每个 div 指定了浏览器窗口 50%的宽度，你期望当增加内边距和边框时仍能保持这两个宽度。但是，CSS 却不是这样工作的。

事实上，相比通常所认为的和由图形设计所带来的直观感觉，CSS 的盒模型要更复杂一些。

盒模型是如何工作的

CSS 中，四个区域（内容、内边距、边框和外边距）中每一个都可以指定值，而且这些值是累加的。内容、内边距、边框和外边距的值全部加起来就是盒的总宽度（如图 10.2 所示）。如果内容的宽度是 400px，每边的内边距是 50px，每边的边框是 2px，那么，盒的总宽度就是 504px（400 内容+2 左内边框+50 左内边距+50 右内边距+2 右边框=504 总宽度）。

CSS 并不关心你混合使用了哪种单位的值。举个例子，也许你指定了内容的宽度为访问者浏览页面时的浏览器窗口的 67%宽度，5em 的内边距和 1px 的边框。总的宽度是多少呢？呃…好像很难算出来，这要根据用户浏览器窗口的宽度和文字默

认（1em）的尺寸来定。

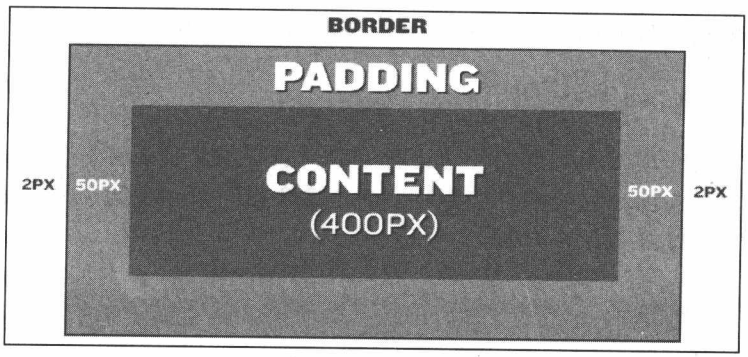


图 10.2　盒模型实例。内容、内边距和边框的值全部加起来就是盒的总宽度

当指定盒子的外边距或内边距的值时，我们可以使用 CSS 的简写法。比如，我们能够这样写一条规则：

```
margin: 25px 0;
```

这条简短的规则对一个元素的垂直方向的外边距指定了 25px 的值，而水平方向的外边距设置为了 0。它是下面这条规则的简写版本：

```
margin: 25px 0 25px 0;
```

而前面两条规则又是下面这些规则的简写版本：

```
margin-top: 25px;
margin-right: 0;
margin-bottom: 25px;
margin-left: 0;
```

在 CSS 中，赋值时是根据顺时针方向的：12 点的位置（顶部的外边距），3 点的位置（右部的外边距），6 点的位置（底部的外边距）和 9 点的位置（左部的外边距）。如果我们想让页面有 25px 的顶部外边距，5px 的右部外边距，10px 的底部外边距和 3em 的左部外边距，那么可以这样写：

```
margin-top: 25px;
margin-right: 5px;
margin-bottom: 10px;
```

```
margin-left: 3em;
```

但是，效率更高的方法是使用 margin 属性的简写法，简单地依次设置四个值就行：

```
margin: 25px 5px 10px 3em;
```

当垂直方向的顶部和底部的外边距相同时（就像我们第一个例子里——25px），以及水平方向的左部和右部的外边距相同时（就像我们第一个例子里——0），那么我们能够这样写，以节省几个字节：

```
margin: 25px 0;
```

注意：在第 9 章里已经提过，0 值是不需要单位的，0px 和 0cm，0in 或者 0 bazillionmiles 都是一样的。（在 CSS 里是没有“bazillion miles”这个单位的，但是如果有的话，当值是 0 时，我们也不需要写。）

当然，如果四边的外边距都是同样的——比方说，每边都是 25px——那么，我们可以压缩为一个值：

```
margin: 25px;
```

简写属性让我们的代码更简洁高效，这意味着我们的用户将下载更小的文件。也就是当用户浏览我们的站点时使用更少的带宽，因此也为我们减少了服务器的开销。每个人都有好处。

现在，CSS 中的简写属性并不只有 margin 属性，上面的例子很容易就应用到 padding 和它的组成部分（padding-top，padding-right，padding-bottom，padding-left）上。Border 也可以同样地扩展为 border-top、border-right，等等。这些简写属性并不会对盒模型产生不同的影响，我们将在本章及后续章节中广泛使用。

我们已经探讨了不少盒模型的问题，但是还有更多内容值得探讨。比如，关于相邻的垂直方向的外边距相互抵消的问题。（Andy Budd，英国的设计师和杰出的标准化实践者，他写过一篇非常棒的论文来解释这个问题，请参阅 www.andybudd.com/archives/2003/11/no_margin_for_error/）

但是现在，让我们先把理论放在一边，来看看这整个的“盒模型”在实践中是

如何工作的。

10.3　实用布局 101

看下面这个设计稿，它由非常简洁的图形和大量的文字组成（如图 10.3 所示）。设计者已经给出了一份页面图样，而由你来负责把它转换成符合标准的代码。

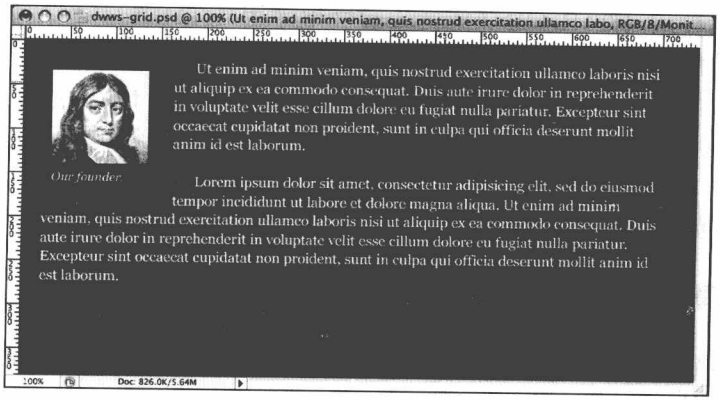


图 10.3　一份简单的页面图样，我们将把它转换成符合标准的 XHTML 和 CSS（很明显，设计师为了这个杰作已经花费了数周时间。）

如果拿到了这份设计稿，你也许会马上开始取样颜色的值，或者分析文字的字体和大小。但是，在我们考虑如何去实现这个设计之前，需要对我们所看到的内容进行一下整理分类，而不去管它看起来是什么样的。以此为出发点，我们首先选择语义上最合适的标记，然后在此基础上用 CSS 来处理表现。

我们将用静态的 XHTML1.0 过渡模版来构建这个基础，如下：

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

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

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

```
    <link  rel="stylesheet"  href="styles.css"  type="text/css"  media="screen,
projection" />
    </head>

    <body>

    </body>
    </html>
```

你也可以使用 HTML4.01（或者实验性地使用 HTML5），但是我们更喜欢跟 XHTML1.0 的标记需求保持一致。无论选择的是哪个 DOCTYPE，基本原则都是相同的：我们要在文档的 body 里书写合法而且语义化的 HTML 代码，把 CSS 写在名为 style.css 的独立样式表中，然后通过 head 中的 link 标记链入我们的页面中。

让我们开始吧。

10.3.1 从基础开始

设计图样包括两个段落和一个左边插有图片的标题。我们将简单地用三个 p 元素来标示我们的内容：

```
<p><img src="j-milt.jpg" alt="" /> Our founder.</p>
<p>Ut enim ad minim veniam, quis…</p>
<p>Lorem ipsum dolor sit amet, consectetur...</p>
```

三个 p 元素简单地包含在 body 元素里面，效果就如图 10.4 所示。没有专门指定样式，页面流中的三个段落都作为块级的盒模型，各自占据一整行。第一个段落中的 img 元素作为内联的元素后面紧跟着文字。当然，目前的样子肯定不能让设计师满意：端庄的衬线在哪里，迷人的色彩又在哪里？

用几行简短的 CSS 来调整是很容易的。回顾一下我们的设计稿，页面的背景色是深蓝色的(用十六进制表示为 #48505A)，文字是白色的 16px 大小的 Georgia 字体。让我们把下面这几行加到 style.css 文件中：

```
body {
  background: #48505A;
  color: #FFF;
  font: normal 16px/1.375 Georgia, Times, serif;
}
```

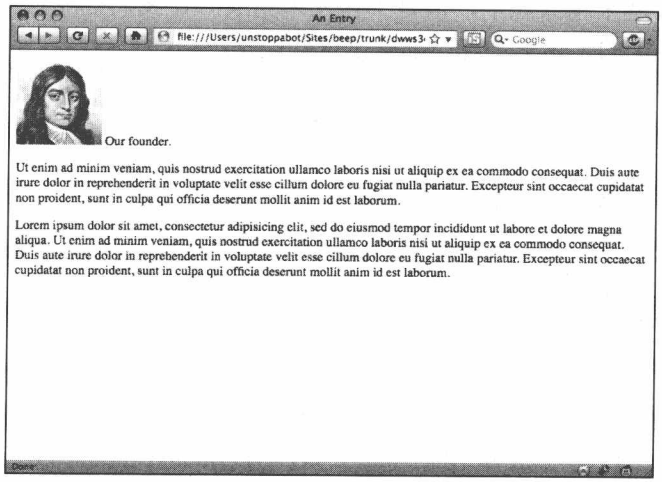


图 10.4　没有应用样式，我们的段落是最原始的样子。但是，简朴的基础可以创造出令人激动的效果

在第 9 章中我们已经学过，每一条样式表规则都通过一个选择器来挑选出所要应用到的元素（或者元素们）。因此，这里的第一条规则是应用到 body 元素的，根据设计，它的背景色设为#48505A，页面文字颜色设为白色（color:#FFF），字体属性也设置了相应的类型。我们将在第 13 章中介绍 Web 字体的细微差别，但是现在让我们看一下图 10.5 的成果。

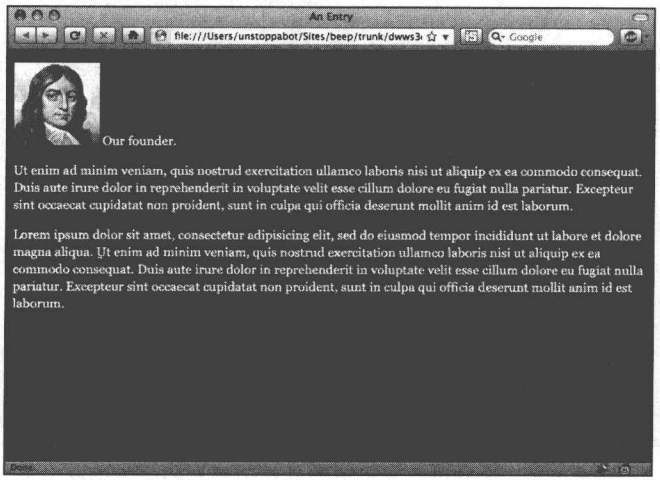


图 10.5　仅用一条样式规则，我们就设置了恰当的颜色和基本的版式。接招吧，font 标签们

我们的模版越来越漂亮了，但是工作还未完成。让我们把注意力从那些默认的样式属性转到设计的主体上来吧，那就是这些段落：

```
body {
background: #48505A;
color: #FFF;
font: normal 16px/1.375 Georgia, Times, serif;
}
p {
margin: 0 0 1em;
text-indent: 1.5em;
}
```

第一条规则只会应用到文档中的 body 元素，而第二条规则则会应用到页面中所有的段落——三个段落（如图 10.6 所示）。我们使用 text-indent 属性（www.w3.org/TR/CSS21/text.html#indentation-prop）来把每一段的起始地方缩进 1.5em。margin 属性用简写法指定了段落的顶部、右部和左部的外边距为 0，底部的外边距为 1em。（你应该注意到我们并不需要重新为段落指定颜色和字体。那是因为这些属性都会从 body 元素中继承，节省了我们的时间和带宽。请到第 9 章的“所有子元素”部分查阅更多关于 CSS 继承的信息，默默地感谢这个特性吧。）

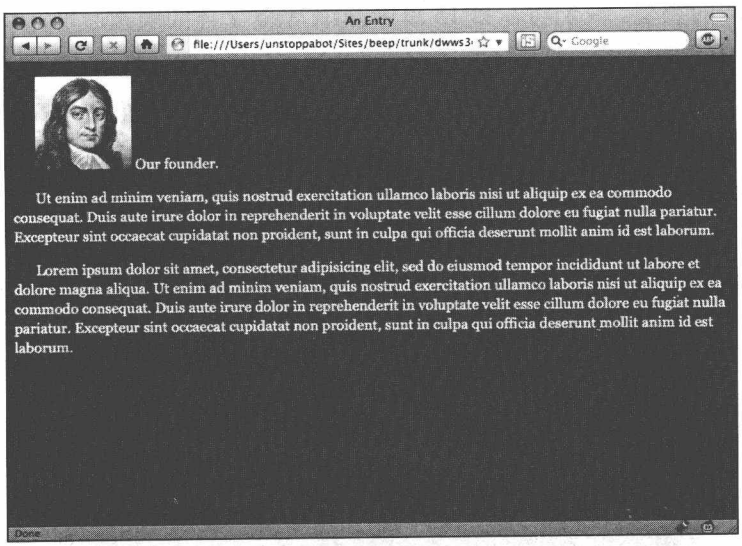


图 10.6　又应用了一条规则，我们就对文档中所有的段落进行了版式调整

简写法小提示

你也许已经注意到，我们是用简写的十六进制来表示字体的 color 属性的——用#FFF 来代替#FFFFFF。第 9 章中已经提及，简写法只能用来替换配对的字符：#FC0 跟#FFCC00 是一样的。颜色值的简写不能用在不配对的字符上。比如像我们的 body 元素的颜色#48505A 就不能用简写形式。另外，除非颜色值中的三对值都能配上对，否则也不能用简写。比如#FFCC09 不能简写成#FC09。

关于 Web 字体的更多细节将在第 13 章讨论，但是你也许已经注意到，在 body 规则中，font 属性要比其他属性复杂得多。那是另一种简写形式，就跟把四个 CSS 属性合为一个属性的 margin 或 padding 属性一样。我们可以用下面的写法代替：

```
body {
background: #48505A;
color: #FFF;
font-weight: normal;
font-size: 16px;
line-height: 1.375;
font-family: Georgia, Times,serif;
}
```

然而，这将令我们的 CSS 出现不必要的散乱，而且浪费了用户的带宽。那四个属性（font-weight, font-size,line-height 和 font-family）可以合成为一个 font 属性。有一句谚语说的好：少就是多。

略微调整了一些元素的盒模型后，我们的模版很快就跟着起了变化。然而，设计的细节部分仍然需要注意：怎样才能让段落围绕着图片呢？

10.3.2　使用 class

包含了图片的那个 p 元素跟其他段落在样式上有很大的不同，所以需要一种“钩子”来把这段特别的段落标记和 CSS 标识挂钩起来。因此我们加了一个 class 属性到这个元素上，并起了一个很有描述性的名字“figure”：

```
<p class="figure"><img src="j-milt.jpg" alt="" /> Ourfounder.</p>
```

（如果能确定这个元素将是页面上唯一设计成这种风格的元素，那么我们也可以用 id 属性来代替。但是有可能这个样式是可以重用的，所以我们还是倾向于采用 class 属性。）

添加了新的 class 属性后，让我们把下面的规则加到样式表里：

```
p {
  margin: 0 0 1em;
  text-indent: 1.5em;
}
p.figure {
  background: #2B3036;
  font-size: 14px;
  font-style: italic;
  margin-right: 1em;
  padding: 10px;
  text-indent: 0;
}
```

就像图 10.7 所显示的，新的规则大大地改变了第一个段落的样子。我们设置了一个新的背景色（#2B3036），设置了一个 14px 的斜体字体，还把这个段落的缩进去掉了。

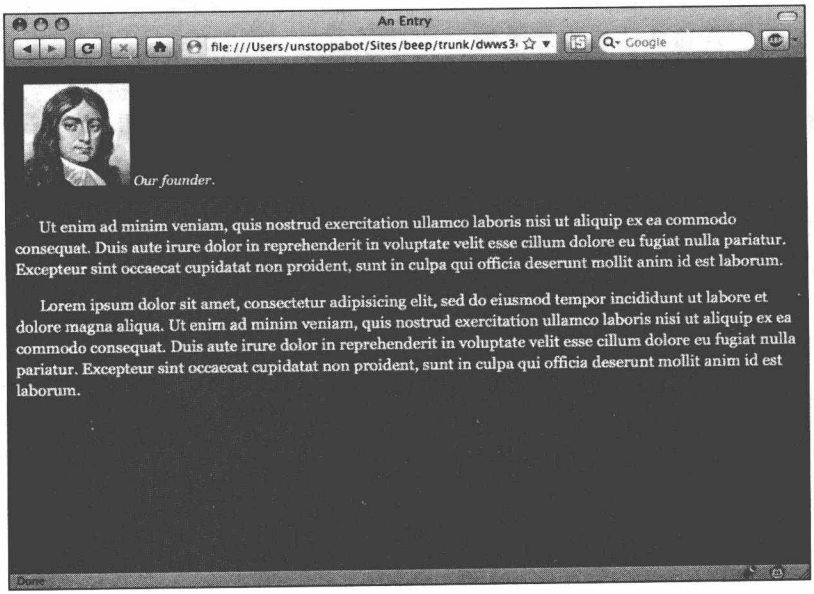


图 10.7　包含图片的标题现在有了一个新的样子

我们为这个段落的周围加了 10px 的内边距，在右边加了 1em 的外边距。这第二条规则只对 class 为“figure”的 p 元素起作用，对其他两个段落是不影响的。

接下来，让我们把标题放到图片的下面，就像设计稿上的一样。图 10.7 中，图片和文字在标记中都在同一行，这是因为在页面流中 img 元素建立了内联盒。然而，用一条新的规则就可以轻松地覆载掉默认的样式：

```
p.figure {
  background: #2B3036;
  font-size: 14px;
  font-style: italic;
  margin-right: 1em;
  padding: 10px;
  text-indent: 0;
}
p.figure img {
  display: block;
  margin-bottom: 0.4em;
}
```

这看上去很简单，你可能都不会仔细考虑就直接跳过去了，但是显式地把一个元素指定为 block 或 inline 是一个非常强大的工具。就这么一条规则，就能把内联的图片转变为块级盒模型，使它像 p 或 div 元素一样，单独成为一行了（如图 10.8 所示）。（我们也在 img 元素下面增加了一点底边距，使该图片跟相关的标题之间留有一点空隙，但是 display:block 在这里是最重要的，你觉得呢？）

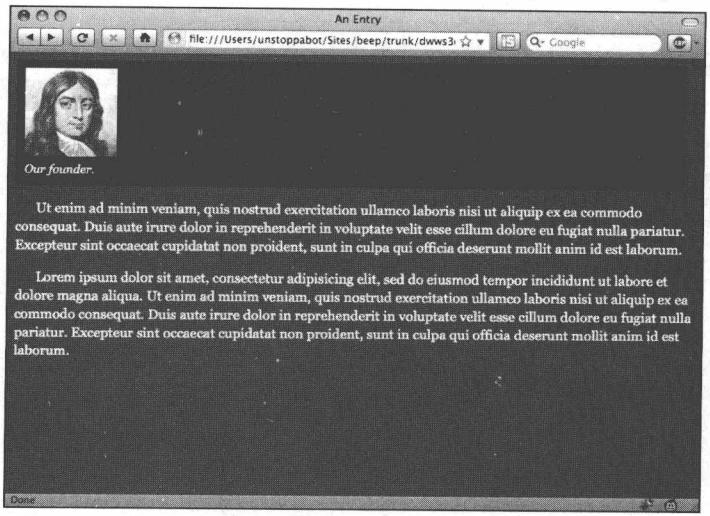


图 10.8　display 属性是非常强大的，它把内联的 img 转变成了块级盒模型

display 属性可以接受的值有很多，都列在这里会非常多（完整的列表请参考这里：www.w3.org/TR/CSS21/visuren.html#display-prop）。

色彩、字体都已经设置好了，图片也已经放到了标题的上面了。现在剩下的事情就是把图片插到整个文本的左边。我们用 float 属性来实现：

```
p.figure {
  background: #2B3036;
  float: left;
  font-size: 0.875em;
  font-style: italic;
  margin: 0 1em 1em 0;
  padding: 10px;
  text-indent: 0;
}
```

就像这个名称所暗指的，应用 float:left 将会大大地改变“figure”段落的版式（如图 10.9 所示）。首先，它会使元素整个“浮动”到它的容器（body 元素）的左边。浮动一个元素也产生一种收缩的效果，即把自身的宽度减少到所包含元素的宽度。这就是为什么浮动后，元素的宽度减少到了图片宽度加上左右 10 像素的内边距的宽度了。

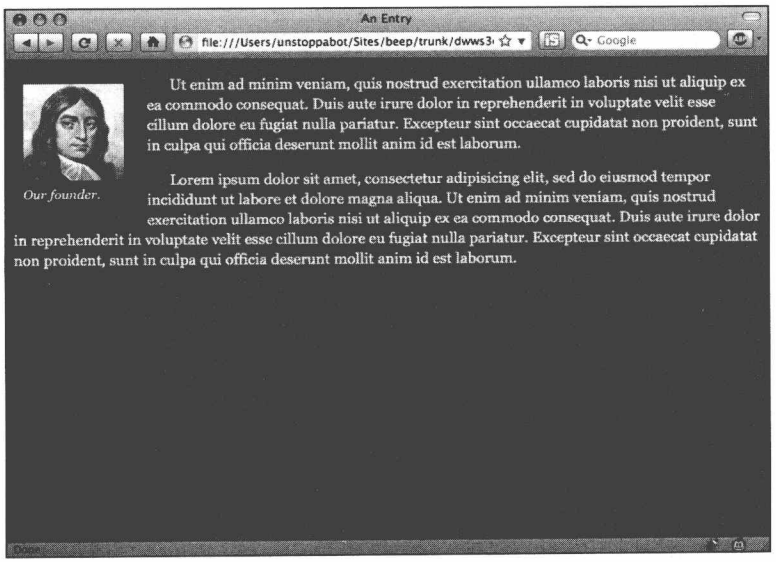


图 10.9　采用 float 来改变页面流

有一点值得重视的是，浮动的元素部分地从文档流中被剔除了。随后没有浮动的段落则完全忽略了浮动元素之前在页面流中所占据的位置。如果我们为没有浮动的段落设一个背景色和外框，那么你就能看到它们和被浮动的段落重叠在一起了（如图 10.10 所示）。然而，段落中的内容却知道浮动段落所占据的位置，段落文本很好地围绕在它的周围。

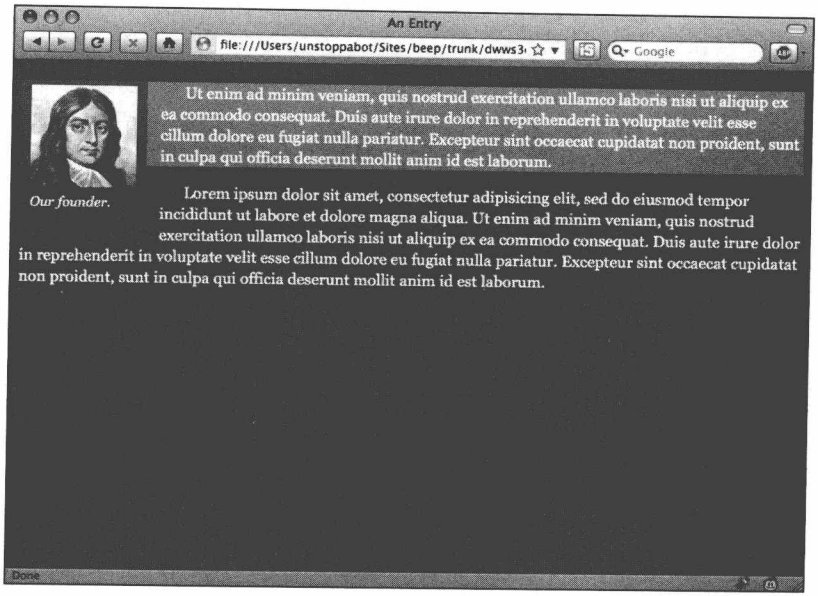


图 10.10　我们设置临时的背景色来观察浮动是怎样部分地从文档流中被删除的。临近的未浮动的元素提升并占据浮动元素空出的位置，而它们的文本围绕着那个浮动的元素

现在，我们完成了我们的模板。用一个语义性的 XHTML 和一些必要的 CSS 作为基础，我们完美地实现了设计稿。二十来行的 CSS 看上去好像很多，但是请记住，这是一个外部独立的样式表文件，可以被包含进无数的模板。对比一下用表格切割的页面吧。

至此，我们的任务完成了，可以喝杯咖啡休息一下了，不是吗？

10.4　重新进行布局

我们不应该就此止步。这里就有一个小要求：你的设计师（和你的老板）已经

对你如何快速地实现了上次的模板印象深刻，所以他们需要做一些修改了。实际上，他们给出了一份新的设计稿（如图 10.11 所示），在已有的基础上增加一些新的模块。

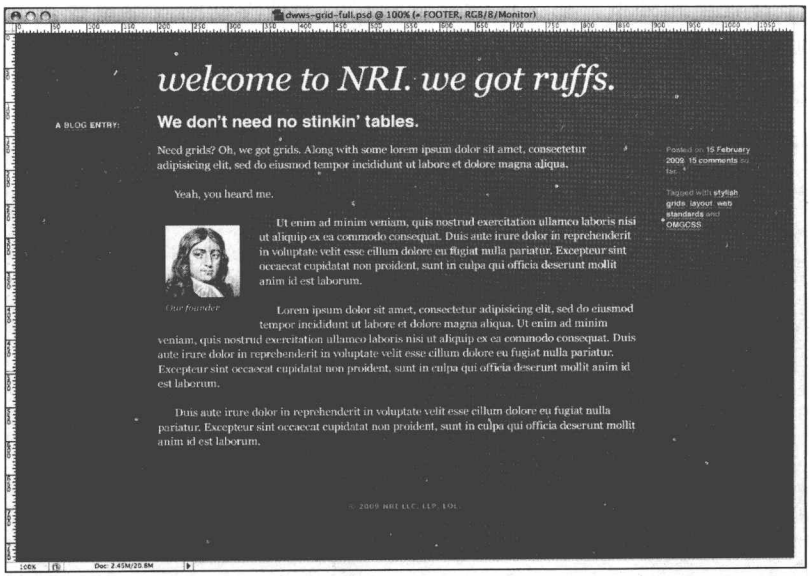


图 10.11　修改后的设计。中间部分跟原来的很接近，但是增加了一些元素

没问题，这根本不需要着急（或跟你的设计师拼命）。当然，这需要在页面上增加一些元素，但是你在前面处理简单段落时积累的工作方法对此可以帮上很大的忙。当我们处理前面三个段落模板时，是按照下面三个简单的步骤进行的：

1. 我们把设计中的美学部分放在一边，先把页面的内容进行仔细的分类整理；

2. 当我们分清了那些内容的类型之后，就把它们转换为简洁的语义性标记；

3. 完成标记的构造之后，我们书写所需的 CSS，对设计中特别的部分使用 class 和 id 选择器。我们还适当地使用了浮动模型来实现设计中不同的栏目表现。

听上去好像很熟悉？这是因为你已经掌握了工作方法。所有的基于标准的设计都是使用跟上面差不多的工作方法，所以你已经有了一个很好的起点了。

那么，让我们开始吧。

10.4.1　内容清单，终极版

第一步，我们回顾一下页面上的内容种类——自问一下，我们真正查看到的是什么。我看到的是，设计师把内容分成了几个不同的部分，如图 10.12 所示。仔细观察，我们得到如下结果。

1．一个非常醒目的整页标题。

2．跟在主标题下面的是 blog 的文章，它包括：

1）文章标题的左边有一个标为“intro”的导言部分。

2）有两个栏目嵌套在文章中：

（1）左边是这篇 blog 文章的基本内容；

（2）右边是一些相关信息。

3．在 blog 文章的底部，横跨着一个通栏的注脚部分。

图 10.12　在考虑怎样用 CSS 实现设计之前，我们仔细区分不同的内容区域

所有的内容都包含在宽度为 988 像素的栅栏中，并且水平居中在页面上。

嘿、嘿、嘿——我们不是说不用着急吗？我发现现在好像跟原来的模板有很大的不同，而且比原来多了很多内容哦。但是，我们可以重用前面的工作成果来避免从头开始。

让我们从一小段段落开始：

```
<p class="figure"><img src="j-milt.jpg" alt="" /> Ourfounder.</p>
<p>Ut enim ad minim veniam, quis…</p>
<p>Lorem ipsum dolor sit amet, consectetur…</p>
```

这些段落仍然是页面的主体，但是在它们周围包裹着额外的内容层。实际上，如果我们插入根据内容来命名的 div 元素，我们就能勾勒出模板的结构来，如下：

```
<div id="page">
  <div class="entry">
    <div class="content">
      <div class="main">
        <p class="figure"><img src="j-milt.jpg" alt="" />
        Our founder.</p>
        <p>Ut enim ad minim veniam, quis…</p>
        <p>Lorem ipsum dolor sit amet, consectetur…</p>
      </div><!-- /end .main -->
      <div class="meta">
      </div><!-- /end .meta -->
    </div><!-- /end .content -->
  </div><!-- /end .entry -->
  <div id="footer">
  </div><!-- /end #footer -->
</div><!-- /end #page -->
```

我已经把原有的三个段落用粗体标识出来了，所以你可以清楚地看到所做的改变。使用 class 和 id 属性作为描述性的标识，不同的内容类型在标记中很好地表现出来：blog 文章的 class 为“entry”，它的相关信息也标识为相应的 class，还有底部的注脚部分也有相应的 id. . .所有这些部分都包含在 id 为“page”的容器中，它把我们的内容基本上全部包在了里面。

你可以看到，这些 div 里面都是空的。它们已经被很好地命名了，但是却没有内容在里面，这使得我们的标签非常简洁。让我们把内容填充进去吧。

首先，让我们看一下页面中的标题（如图 10.13 所示）。我们有三种标题：页面

顶部醒目的主标题、blog 文章的标题和介绍右面内容的导言标题。三种标题按照它们在页面中的主次来分级。标准化的代码能做什么呢？

```
<div id="page">
  <div id="content">
    <h1>Welcome to <abbr title="Neck Ruffs International">NRI</abbr>. We Got
Ruffs.</h1>
    <div class="entry">
      <h2>We don&#8217;t need no stinkin&#8217; tables.</h2>
      <h3 class="info">A <a href="#">Blog</a> Entry:</h3>
      <div class="content">...
```

图 10.13　在设计稿的顶部有三种不同的标题。在视觉上，它们是有很大区别的——但是先忽略美学上的差别，让我们考虑一下它们在语义上的层次

如果你猜测应该是 h1、h2 和 h3，那么你答对了。（如果你不介意的话请留下你的名字和地址，我们很乐意送一份奖品给你。）请记住：标题元素并不就意味着“又粗、又大、又难看”。它们有语义上的作用：根据标题在页面中的重要性来命名和加重。我们在新添加的 h3 元素上加上了值为“info”的 class 属性，用来区分页面中可能出现的其他 h3 元素。

相对于标题的分类，剩下的内容更容易转换成 HTML 了。实际上，我们只需要添加一些 p 元素就可以了。

```
      <div class="main">
        <p>Need grids? Oh, we got grids…</p>
      </div><!-- /end .main -->
      <div class="meta">
        <p>Posted on 15 February 2009. <a href="#">15 comments</a> so far.</p>
        <p>Tagged with <a href="#">stylish</a>, <a href="#">grids</a>, <a
href="#"> layout</a>, <a href="#">web standards</a> and <a href="#">OMGCSS</a>.</p>
      </div><!-- /end .meta -->
    </div><!-- /end .content -->
  </div><!-- /end .entry -->
```

```
    </div><!-- /end #content -->
    <div id="footer">
      <p>&copy; 2009 <abbr title="Neck Ruffs International">NRI</abbr> <abbr
title= "Limited Liability Corporation">LLC</abbr>, <abbr title="Limited Liability
Partnership">LLP</abbr>, <abbr title="Laugh Out Loud">LOL</abbr>.</p>
    </div><!-- /end #footer -->
  </div><!-- /end #page -->
```

blog 文章的相关信息部分（包含在 class="meta"的 div 元素里的）和页面的注脚部分（包含在 id="footer"的 div 元素里面的）都只是文本段落，所以用 p 元素是最合适不过了。好吧，结束段落的讨论，让我们继续。

10.4.2 应用样式

我们现在的模板已经充满了适当的内容，你可以看到，页面流中所有的元素都一行挨着一行地排着（如图 10.14 所示）。当然，除了那个含有图片的 class="figure"的 p 元素，它浮在其他段落的左边。但除此之外，还有一大堆东西需要合理地布局。我们从哪里开始呢？

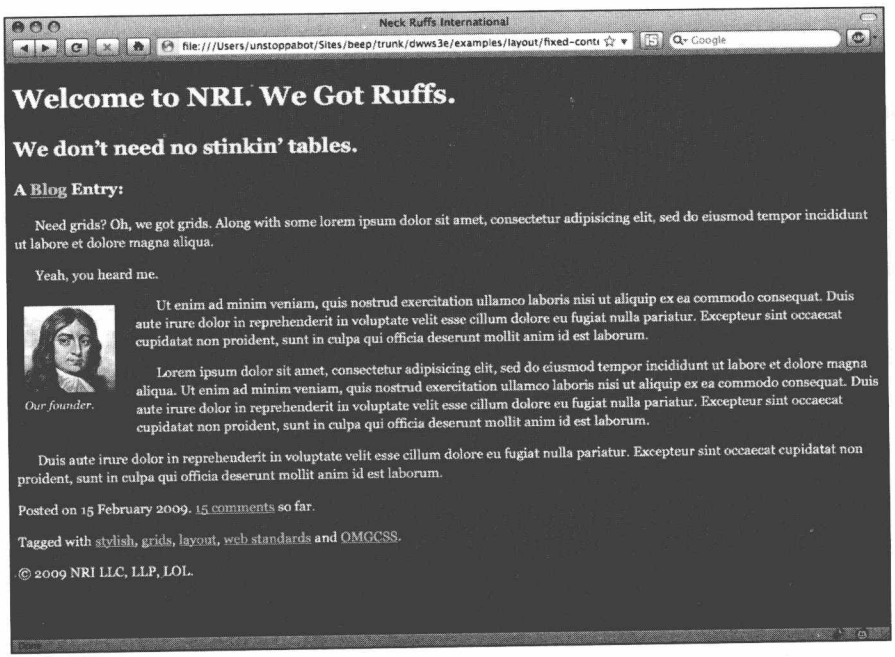


图 10.14　页面中的标签部分已经完成了。但是，目前还都处于默认的页面流中，需要调整至合适的位置

在先前的模板样式基础上，我们只需要稍做修改。原有的三个段落现在是包含在“entry”块中——而且是包含在其内部的 class=“main”的 div 中——我们的 CSS 将反映这一点，我们不需要把这些规则应用到文档中的每个段落上。让我们把这段规则应用到这个新的标记结构中，使其更特殊化一点：

```
.entry .main p {
    margin: 0 0 1em;
    text-indent: 1.5em;
}
.entry .main p.figure {
    background: #2B3036;
    float: left;
    font-size: 0.875em;
    font-style: italic;
    margin: 0 1em 1em 0;
    padding: 10px;
    text-indent: 0;
}
.entry .main p.figure img {
    display: block;
    margin-bottom: 0.4em;
}
```

我们把一些 class 选择器放在一起组合成一个新的派生选择器，这样就把这些规则的应用范围缩小到新的标记结构上。先前这些规则是应用到文档中所有的段落元素上的，而现在仅仅应用到 class 为“entry”的元素内部的 class 为“main”的元素中的段落上。打个赌，你肯定无法快速地念上五次。

上面我们说过，我们设计了 988 个像素宽的范围，并且是基于浏览器窗口水平居中。我们把所有的内容都包在 id 为“page”的 div 中，我们把样式直接应用到这个 div 上：

```
#page {
    margin: 0 auto 40px;
    width: 988px;
}
```

这 width 属性的值是非常明显的，只要从设计稿中直接量取像素值就可以。至于 margin 属性的值，我们也讨论过，这是一种简写属性，用一条简明的语句就可以设定四边的外边距的值。这里，我们为这个 div 的顶部和底部的外边距分别设置了 0 和 40px 的值，而把它的左右两边的外边距都设为 auto。把一个块级元素的左右两边的

外边距设置为 auto，将使这个元素水平居中在它的容器中（请参阅 www.w3.org/TR/CSS21/visudet. html#blockwidth）。不管浏览器的窗口有多宽，我们的内容将始终固定在正中间（如图 10.15 所示）。

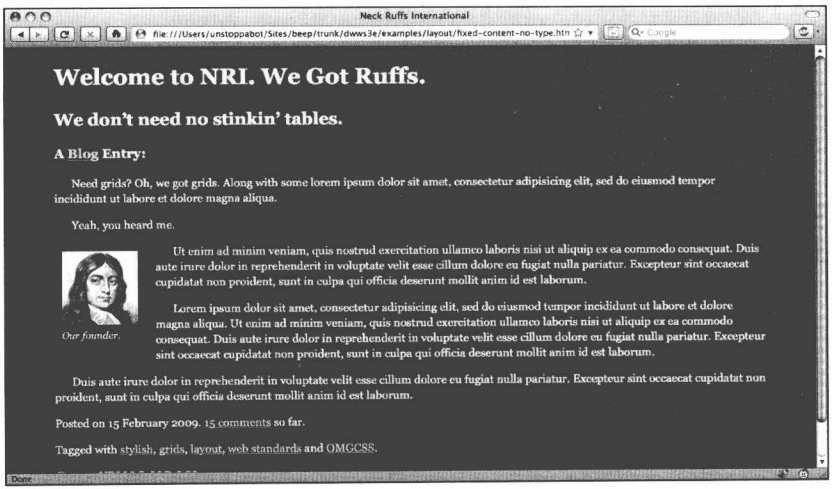


图 10.15　把水平方向的外边距设为 auto 将使这个元素水平居中于容器中

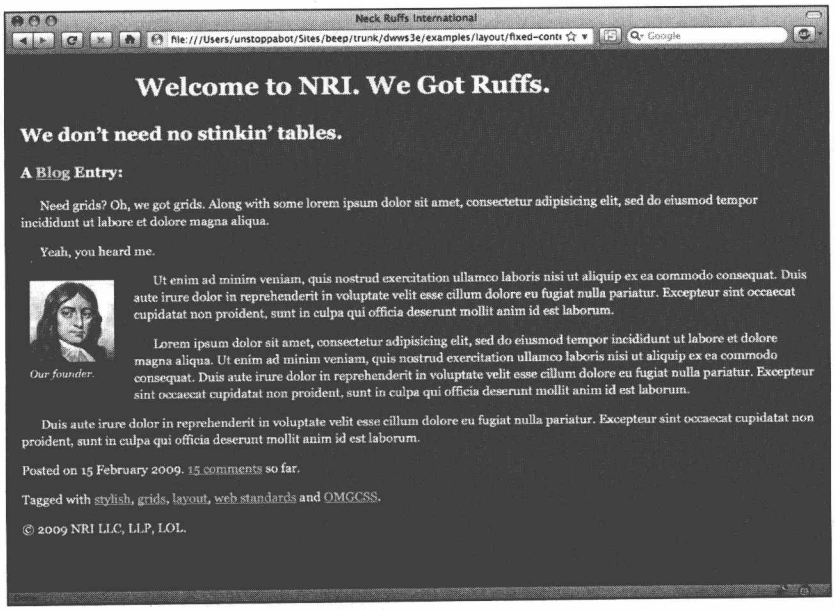


图 10.16　为 h1 设一个宽度和左边的外边距，把它调整至合适的位置

接下来，我们来把主标题调整到合适的位置上。从设计稿上看（如图 10.16 所示），标题有 700px 宽，且距离内容部分的左边界有 144px。我们用 margin-left 和 width 属性很快地就实现了。

```
h1 {
  margin-left: 144px;
  width: 700px;
}
```

两条简单的 CSS 规则就使我们的布局开始有模有样了，我真是喜爱干这活。

10.4.3　再次运用浮动

对于其余的内容部分，我们将略微改变一下我们的任务。除了 h1 元素和 class="figure"的 p 元素外，其他内容仍然处于默认的文档流中：段落和 div 元素简单地堆叠在一起，就像真的块状盒子一样。

上一次，为了改变默认的页面流，我们简单地把一个段落浮动到了左边。我们能对 h2 和 h3 元素进行同样的操作吗？

```
.entry h2 {
  float: right;
  width: 844px;
}
.entry .info {
  float: left;
  width: 124px;
}
```

接着有趣的事情发生了，如图 10.17 所示，两个标题在主要的文章内容上面形成了一个水平的行。而且，由于 h3 的标记是出现在 h2 标签之后的，把它浮动到左边后使得它显示在了 CSS 样式页面的前面。很灵巧，不是吗？这是完成我们页面布局的关键：我们可以用 float 属性来把包含有不同内容的 div 元素从文档流中提出来，允许它们根据设计要求来水平排列成栏。

首先，我们可以把 class="content" 的 div 元素进行浮动，它包含了我们所有的文章部分：

```
.entry .content {
```

```
    float: right;
    width: 844px;
}
```

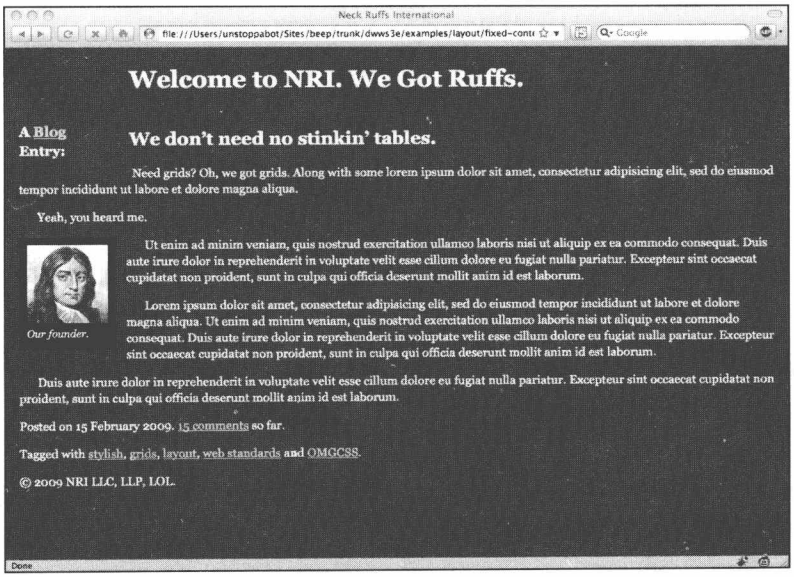


图 10.17　把两个标题进行浮动，导致它们排成了一行，这是完成多栏布局的关键

浮动是诡异的 hack 技巧还是优雅的布局技术？

某些标准化的支持者批评使用 float 来布局的方法，把它比作为滥用间隔图片来布局的老方法。严格地说，他们没错：float 原本就不是用来控制页面布局，而是用来控制文本流围绕某个对象的，就像用 align="left"或者 align="right"一样是用来让文本围绕图片的。因此，尽管是用一个优雅的方法来实现完全不同的目的，float 方法确实可以被看作是一种 hack 技巧。但是，这仍然是实现 CSS 驱动的页面布局的最可靠的方法，它提供了比表格布局更大的灵活性。

W3C 已经宣称 CSS 需要一个更健壮的布局模型，计划在 CSS3 规范中提供支持。网格定位模型（www.w3.org/TR/css3-grid）的开发目前还在进行中，这是一个有希望替代 float 布局风格的方法。但是目前来说，float 是最满意的方法。

我们把它浮动到了右面，并且根据设计稿赋予了一个宽度，让它跟 h3 处于同一行中（如图 10.18 所示）。

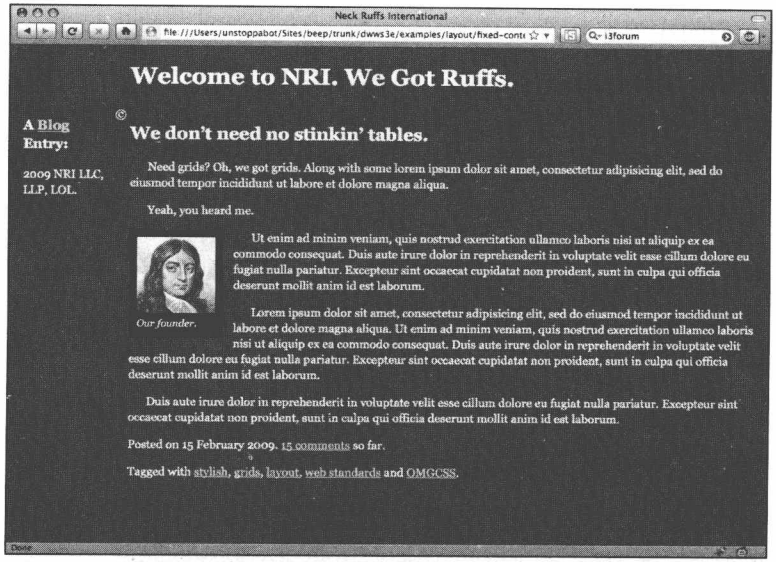


图 10.18　我们把内容区域浮动到右面，建立了一个基本的两栏结构

另外，可以把 float 应用到已经浮动的元素上，我们将对“entry”块中的“main”和“meta”两个 div 元素进行浮动：

```
.entry .main {
  float: left;
  width: 700px;
}
.entry .meta {
  float: right;
  width: 124px;
}
```

把“main”浮动到左边，把“meta”浮动到右边，这样我们把两栏放到了一行中。根据设计的布局要求，我们把这两个 div 元素放到了合适的位置上：“main”占据了 blog 内容区域 700px 的宽度，“meta”分配了 124px（如图 10.19 所示）。现在，我们完成了最后的工作——真的完成了吗？

10.4.4　处理细节

你也许已经注意到页面最后的几个数字显示得稍微有点奇怪。事实上，你可能已经在急切地指出它们有很奇怪的错误，希望我停止罗嗦“浮动”和“分栏”，而先

来修复一下我们的注脚部分。

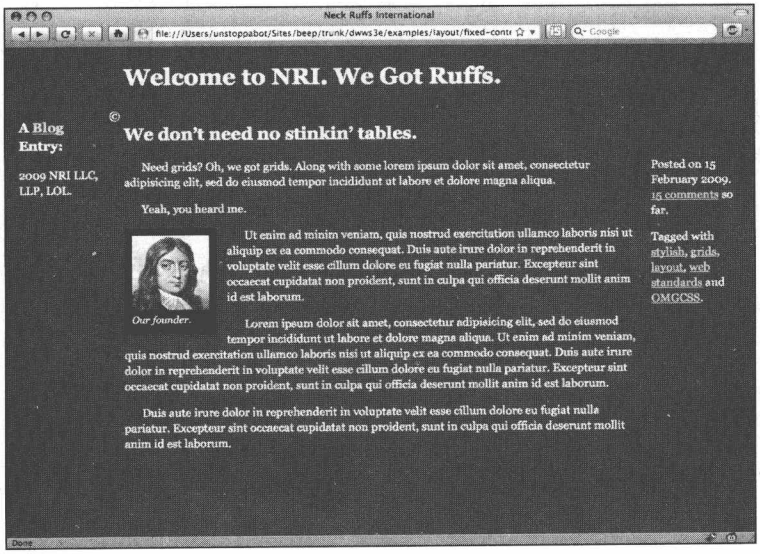


图 10.19　最里面的栏也被适当地浮动到位了，我们的页面布局（差不多）完成了

你说得很对。页面最后的那个 div 显示得很奇怪（如图 10.20 所示），它的内容出现到了 h3 的下面。而且，不知道为什么，©符号从原来的文本序列中跳了出来，悬停在文章标题的 h2 的前面。怎么回事？

图 10.20　我们的注脚部分错位了（有点名不副其实了）

要回答第一个问题，我们需要回顾一下先前的第一个模板中插入图片的那个部分（你可以翻回去看一下，我们在这里等你）。当我们把 class="figure"的 p 元素浮动之后，它就从原来的文档流中提了出来（如图 10.9、图 10.10 所示）。结果使得没有

被浮动的 p 元素们代替了原来那个已经浮动掉的元素的空间，就像我们没有把图片插入进去一样。但是那些段落中的文本却让开了浮动元素，并且围绕着它。

这里也是这种情况。我们的注脚部分是页面上唯一一个没有浮动的 div——事实上，除了位于顶端的 h1 元素，主要的结构标签元素都已经被浮动了。对于注脚部分而言，它认为 h1 是页面上唯一的元素了，这就是为什么它会显示在页面上部的原因。然而，就像先前那个模板中的情况一样，注脚部分中的文本却围绕着浮动元素流动。所以，我们需要用一种方法告诉 CSS，注脚部分应该显示在浮动元素的下面。

clear 属性正好派上用场：

```
#footer {
  clear: both;
}
```

clear 属性接受四个值：none、left、right 和 both。通过 CSS，clear 属性告诉浏览器在定位元素时要把它放到所有浮动到左边（clear:left）、右边（clear:right）或者两边（clear:both）的元素的下面。在我们的注脚部分的前面，左右两边都有浮动的元素，因此，我们用样式表定位注脚部分到所有类型的浮动元素的下面，也就是页面的底部（如图 10.21 所示）。

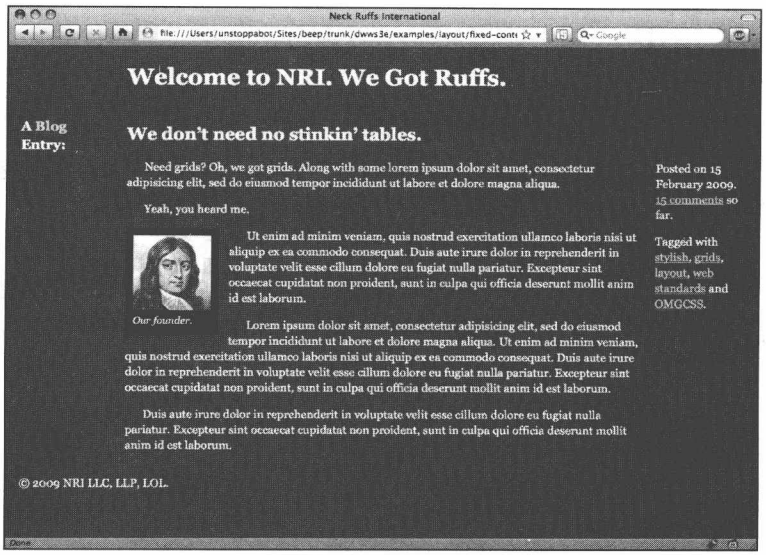


图 10.21　应用 clear:both，注脚部分调整到正确的位置上了

至此，页面布局全部完成了。再调整一下修饰和色彩，我们的页面就可以准备交付给设计师检阅了（如图 10.22 所示）。

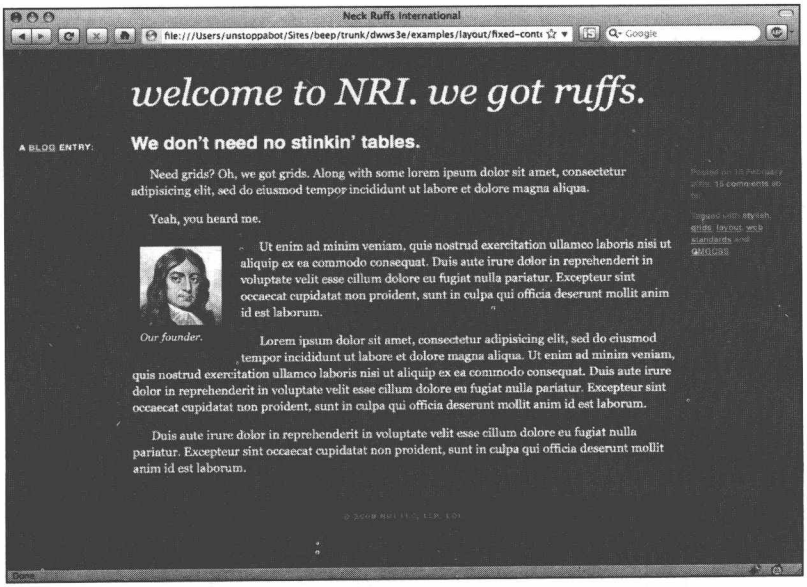


图 10.22　布局完成后，再处理一下设计的细节，我们的工作就完成了

10.5　小结一下

你终于可以把 Web 标准应用到一个充满灵气的设计中了。更重要的是，你已经清楚地了解了构建一个遵循标准的页面的流程，在以后的章节中，这个流程会在更多的 CSS 布局中碰到。有了这些基础，以后的学习将会更有乐趣。

当然，偶尔会碰到挫折。即使是对标准支持得最好的浏览器，有时也会出现“殊异”的行为。但是不用担心：下面的几个章节中，我们将会学到更多的方法来对付这些问题。

第 11 章

深入浏览器之一：DOCTYPE 切换和标准模式

现代浏览器是怎么“知道”你创建的是一个向前兼容的站点呢？它在支持那些使用过时方法创建的站点的同时，怎么能正确地显示你的标准化站点呢？

答案就是，大多数现代浏览器使用了我们的老朋友 DOCTYPE。我们在第 6 章中已经介绍过它了。DOCTYPE 是把标准模式（符合 W3C 规范的网站就是标准模式）和向后兼容的怪异模式 Quirks 模式（这样命名是因为那些用老方法制作的站点使用了各种各样不可兼容的 Quirks 模式）联系起来的纽带。本章将教你怎样控制站点的外观和行为。

本章叙述了这些模式是怎样工作的，并提供一个简单的方法来使你的站点实现所期望的视觉效果，尽管每个优良的浏览器理解 CSS 和其他规则的方式都不尽相同。

11.1 DOCTYPE 切换的传奇故事

20 世纪 90 年代后期，处于领先地位的浏览器制造商认识到，对于设计师和创建网站的开发人员来说，完全精确地支持 Web 标准尤为重要，但他们希望，在升级到正确的支持网络标准的新式浏览器的过程中，不要损坏原先的非标准网站。

毕竟，3.0 和 4.0 版本的浏览器已经说服了大批设计师学习他们特有的 Quirks 模式，包括对 HTML 的专有扩展，以及不完全和不正确的 CSS 实现，还有浏览器制造商们自己定制的脚本语言。微软和 Netscape——那时侯他们对 Web 未来的影响力势均力敌——愿意对 Web 标准提供更好的支持，但前提是不损坏现存的、价值上亿的网站，否则对于浏览器制造商来说，那样做等于是自杀。

下面的一个例子可以说明浏览器制造商的尴尬处境。

20 世纪 90 年代中期，Internet Explorer 的早期版本已经开始部分支持 CSS1 了，但他们弄错了一些地方，如盒模型（第 12 章中将详细说明）。尽管第一稿总是有点粗糙，但值得称赞的是，在 1997 年初微软公司终于开始支持 CSS 了。

不知是功还是过，微软早期对 CSS 盒模型的错误理解带来了一个问题。数以万计的设计师已经“学会”了 IE4.X 和 IE5.X 中所使用的错误的盒模型，并调整了他们的 CSS，以求在这些版本的 IE 上正确显示。如果 IE 的后续版本和其他制造商的浏览器更准确地支持了盒模型，那么那些已有的设计肯定要土崩瓦解了。怎么办呢？

11.1.1 用来切换标准的开关

在微软和 Netscape 同意制造准确且全面支持标准的浏览器之前，一个无名英雄解决了如何优雅地处理用非标准方法建立网站的问题。那个英雄就是用户界面技术专家 Todd Fahrner （style.cleverchimp.com），他是 W3C 的 CSS 和 HTML 工作组的参与者，还是 Web 标准项目的共同创立者。1998 年初，在一份鲜为人知的 W3C 邮件列表上（W3C 的邮件列表都是鲜为人知的，这也让这个故事多了点神秘感），Fahrner 建议浏览器制造厂商提供一种切换机制，可以用来开关符合标准的显示模式。他建议根据 DOCTYPE 的存在与否来实现这种切换。

Fahrner 的想法是，如果网页是以一个 DOCTYPE 标签开始的，这意味着设计师了解 Web 标准，并打算遵循标准。浏览器就应该按照 W3C 的规范来解析这样的网页。相反，没有 DOCTYPE 的网页不会通过 W3C 的校验测试（http://validator.w3.org/），而这样的网页极有可能是使用老方法制作的。浏览器对此应该区别对待；也就是说，它们应该用老式未遵循标准的浏览器的方式来处理这类网页。

但还存在一个问题：现在还没有完全遵循标准的浏览器。如果 DOCTYPE 切换是向前向后兼容的关键所在，那么全世界的用户就还要等上两年。（而你不必再等上两小段。）

11.1.2　切换开关浮出水面

2000 年 3 月，微软公司发行了 IE5 Macintosh 版，它的 Tasman 渲染引擎是由微软工程师和 W3C 标准的发烧友 Tantek Celik 开发的，它对 Web 标准，包括 CSS1、XHTML（text/html）和 DOM，都提供了相当准确和近乎全面的支持。IE5/Macintosh 还采用了文本缩放以提高可访问性，它是第一个用 DOCTYPE 切换来实现 Quirk 模式和标准模式之间转换的浏览器。

基于 Gecko 浏览器进行开发的工程师们，马上就明白这是两个非常有用的功能（DOCTYPE 切换和文本缩放）。因此继 IE5/Mac 之后不久推出的 Netscape 6+、Mozilla 和 Chimera 的 Gecko 浏览器，都包含了 DOCTYPE 和文本缩放功能，通过 Gecko/Mozilla 渲染引擎，使得支持严密详细的 Web 标准成为可能（http://www.Mozilla.org/newlayout/）。时过境迁，Mozilla 孕育出了今天的 Firefox，Chimera 孕育出了 Camino。当 IE6/Windows 加入遵循标准的行列后，也同样支持 DOCTYPE 切换，并增加了一个根据 Web 文档来切换是否使用标准模式的 DOM 特性。

11.2　DOCTYPE 切换基础

今天，几乎所有浏览器的 DOCTYPE 切换功能都是按照 Todd Fahrner 在 1998 年提出的工作方式进行实现的。当时符合标准的浏览器还未出现在开发者的面前。

- 在标准模式里，浏览器假定你明白自己在做什么。一个包含完整 URI（完整的网址）的 XHTML DOCTYPE 告诉浏览器以标准模式渲染网页，根据相应的 W3C 规范来执行 CSS、XHTML 和 JavaScript。一些完整的 HTML 4 DOCTYPE 也会触发标准模式，从现在起，我们将要讨论的一些网页就是这样的。而 HTML5 的 DOCTYPE 不再需要一个 URI。

- 使用一个不完整或过期的 DOCTYPE 或根本不用 DOCTYPE 时，浏览器就会应用 Qucrks 模式，这种模式假定你的（也许是正确的）书写是过时的，含有

不合法标签和某种浏览器专属的非标准代码。在这种情况下，浏览器尝试用向后兼容的方式来解析你的网页，应用你的 CSS，表现得就像在 IE5 上一样，并使用此浏览器专属的 DOM。

要控制浏览器采用哪种模式，我们要做的就是包含或者省略一个完整的 DOCTYPE，就这么简单。

11.2.1 切换有多准确

在 2000 年，当浏览器第一次开始实现 DOCTYPE 切换的时候，它被作为一个可靠的指标，来衡量网站开发者是否了解涉及 Web 标准的工作。毕竟，当时只有少数人了解基于标准的设计，并开始使用符合 XHTML DOCTYPE 规范的标记。

后来，Web 标准组织说服了一些 Web 开发工具厂商，比如 Adobe Dreamweaver，默认包含一个合法的 DOCTYPE，并在所见即所得的开发模式下提供合法的 XHTML 标记。这加快了那些使用 Dreamweaver 来获得 Web 标准的开发者的工作流，但这也导致了 DOCTYPE 开始出现在一些由并不理解标准的开发者所创建的 Web 文档中。

有些时候，那些开发者是幸运的，他们的开发工具纠正了他们的错误。但是在其他情况下，不了解标准的开发者也许书写了不合法的标记、CSS 和仅对 IE 有效的脚本，但是浏览器将被 DOCTYPE 所愚弄，它会试着用基于标准的方式来表现这些非标准的站点。特别是脚本，那些不可预料的结果可能会很严重。

11.2.2 Web 标准和 IE8

基于上述原因，微软这个最后加入标准行列的浏览器制造厂商（以 IE8），在 2008 年提出，DOCTYPE 不再是衡量是否符合标准的可靠指标了，而需要一个额外的触发器。关于标准协会对此看法的细节，可以翻看第 4 章“IE 和 Web 标准”。这个额外的触发器是一个 meta 元素，如下：

```
<meta http-equiv="X-UA-Compatible" content="IE=8" />
```

（另一种方式是，一个 X-UA-Compatible 的 HTTP 头部的设置可以代替这个 meta 元素。）

让面向标准的 Web 设计师和开发者们开心的是，微软最终决定这个额外的触发器采用一种“选择退出”模式。这意味着，如果你想让 IE8 认出非标准的站点，你必须在文档的头部插入 meta 元素来退出标准模式。如果你想获得标准支持，你就应该像在其他浏览器上所做的那样：书写合法的标记，以及用一个现代的，包含完整 URI 的 DOCTYPE（HTML4.01 以上）。

这个可选的触发器服务于一个标准的目的：就是用来把 IE8 锁定在标准模式，覆盖掉让浏览器运行在其他模式下的设置。这里说的其他模式是什么？问得好。

IE8 的四种模式

为了兼容的目的，IE8 有四种渲染模式。

使用合法的标记和一个现代的包含完整 URI 的 DOCTYPE（HTML4.01 以上），可以使 IE8 进入标准模式，它会使用面向标准的开发者所期望的方式和优秀的 DOM 支持来渲染你的站点。这是浏览器的默认模式；也是推出这款浏览器的初衷。

使用不合法的文档并且没有 DOCTYPE 的话，就会进入 IE5.5 的 Quirks 模式，IE8 会用 IE5.5 的方式来渲染页面。

在 IE7 标准模式下，也称为 IE8 的兼容视图，IE8 的行为差不多就跟 IE7 一样了。当用户希望浏览器用 IE7 的方式来渲染页面时，就可以触发这种模式。特别是站点在默认的标准模式下显示不正常的时候，用户就只能这么做了。微软增加这个功能来兼容那些使用 IE7 的方式制作的站点——这是一个聪明的解决方法。如果有一定数量的用户在站点上选择通知微软需要 IE7 模式的话，IE8 将默认用 IE7 的标准模式来渲染站点。这听上去像是一个黑名单（或者是白名单，看你怎么认为了），但实际上并不是——它是有意义的。当一个站点被加到了 IE7 的兼容列表里后，微软就会试着联系这个站点的主人，提出一个问题列表或者了解一下准备何时抛弃这个兼容模式。而且，微软一直在不断地跟踪和更新这个列表。当站点从原先的 IE7 标准模式转变为使用纯粹的标准，那么用户会要求浏览器用 IE8 的模式来渲染站点。通过人们所分享出来的使用数据，获得足够数量的这类要求后，微软将会把站点的默认渲染模式转变回 IE8 的标准模式。

IE8 的准标准（是指过渡模式吗？）模式，跟下面要讨论的 Mozilla/Firefox 的准标准模式差不多。

11.2.3 Web 标准和 Gecko

不仅仅只有微软提供多种模式。现代的 Gecko 浏览器，像 Firefox，可以在三种模式下切换：怪异（Quirks）模式（前面提到的）、准标准模式（上面讨论的）和标准模式（完全严格且精确地支持标准）。早期的 Gecko 浏览器比如 Netscape 6 和 7，以及 Mozilla 1.0，所提供的切换标准模式的 DOCTYPE，已经被新近的 Gecko 浏览器的切换准标准模式的 DOCTYPE 所代替。随着那些早期的浏览器被淘汰，而且 Firefox/Mozilla 的粉丝们只要一有新版本的出现就马上升级他们的浏览器，那些兼容的问题似乎不再困扰我们。但是理解准标准模式仍然是很重要的。

Mozilla/Firefox 的准标准模式

为了更好地帮助正在过渡到标准的设计师，Mozilla 的工程师们建立了准标准模式，这是一种类似 IE6/7 的标准模式。比如，在标准模式下，Gecko 把图片看作是内联元素，除非你用 CSS 特别地把它们变成块级元素。所有的内联元素，比如文本，都是根据基线来调整小写字母如 y、g、j 的下沉位置的。而基线的尺寸和位置是依据它所在的容器元素的字体类型和大小来决定的——比如，包含内联图片的段落的字体类型和大小。

一个正在过渡到标准的设计师也许建立了一个混合型的布局（部分用表格布局，部分用 CSS 布局），期望图片能够在 Firefox 和 IE6 中都适当地显示。但是在标准模式下，文本的容器元素的基线和行高会使图片错位，破坏设计者的表格布局。

为了避免出现这种情况，Mozilla 的工程师们决定让 XHTML 过渡型 DOCTYPE 调用 Gecko 的准标准模式，让严格型 DOCTYPE 来触发更严格的标准模式。毕竟，如果书写的是严谨的没有表现性的标记，你可能也不会把图片放到表格里。

相对于触发 Gecko 准标准模式的过渡型 DOCTYPE，正在创建基于标准的 CSS 布局的本书读者们，可能更愿意使用严格型的 XHTML——如果你能保证你的客户和他们的 CMS（内容管理系统）能原样保持你的标记和 CSS 代码的话，这当然是一个明智的选择。但是，那些宁愿使用过渡型的 XHML 的设计师可以调用 Gecko 的准标

准模式，你需要做的就是按此模式来工作，用 CSS 声明来把图片变成块级元素。

11.2.4　完整的和不完整的 DOCTYPE

使用 DOCTYPE 切换的浏览器会寻找完整的 DOCTYPE，也就是说，它们会寻找包含完整网址的 DOCTYPE，如 http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd，很多现代浏览器是用下面这个标准模式的 DOCTYPE 触发器来进行 DOCTYPE 切换的：

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

这是一个严格的 DOCTYPE，尽管到目前为止，我们一直推荐采用 XHTML1.0 的过渡模式。在这里我们用严格的 DOCTYPE，是为了避免过多的解释，这会在以后讨论。现在要说的是：使用一个就像上面这样完整的 DOCTYPE 将能触发标准模式。

在我们编写开发的很多站点中，一些开发工具默认地插入了不完整的 DOCTYPE，从而触发了向后兼容的 Quirks 模式而不是触发我们想要的标准模式。例如，很多开发工具插入的 DOCTYPE 是从 W3C 网站上获得的，如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "DTD/xhtml1-strict.dtd">
```

这条规则和前面的那条有什么不同呢？

如果你仔细看前面那个 DOCTYPE 的最后一部分（"DTD/xhtml1-strict.dtd"），就会发现它实际上是一个相对链接而不是一个完整的 URI。其实，在 W3C 网站上，它是链接到 DTD 文档的一个相对链接。除非你已经把 W3C 的 DTD 文件复制到自己的服务器目录——没人会这样做——这个相对链接将链接不到任何东西。因为 DTD 文件存在于 W3C 网站而不是在你的网站上，URI 被认为是不完整的，所以浏览器只能以 Quirks 模式显示你的网站。

（现今的浏览器会尝试查找和载入 DTD 文件吗？不会。它们仅仅是对照它们的数据库，然后简单地认为这是不完整的 DOCTYPE，而进入 Quirks 模式。这意味着 w3.org 上用这个相对的 URI DOCTYPE 的网页也会以 Quirks 模式呈现吗？看起来是这样的。）

现在让我们再看一下能触发标准模式的完整版本：

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

请注意，标签的最后包含了一个完整的 URI。可以复制粘贴这个 URI 到浏览器的地址栏里，这样你就能阅读到充满了诡异代码的 XHTML1.0 Strick DTD 的文档了。因为 DOCTYPE 末尾的这个 URI 指向了一个网络上的有效位置，支持 DOCTYPE 切换的浏览器认为这个 DOCTYPE 是完整的，就会以标准模式渲染你的网页。

另外有一个额外的问题，请注意在 IE8 之前的浏览器中，只要有 XHTML DOCTYPE 存在，就会切换到标准模式，而不管这个 DOCTYPE 是否包含一个完整的 URI。但是，一个不包含完整 URI 的 XHTML DOCTYPE 在技术上讲是无效的。因此，在下面的讨论里，我们提倡你在 XHTML DOCTYPE 里用一个完整的 URI 地址。（毕竟，如果你的网页代码不是合法的，那为什么还要转变成标准模式呢？）

这些复杂性看上去违背了标准所崇尚的简单理念，不过你会很高兴地了解到，让浏览器们（包括 IE8）进入 HTML5 DOCTYPE 的标准模式是不需要 URI 的：

```
<!DOCTYPE html>
```

IE6 中，头部声明的怪异问题

每条规则执行起来都会有例外。即使是使用完整的 XHTML DOCTYPE，但如果包含可选的 XML 头部声明，那么 IE6 就会返回到 Quirks 模式。Opera7 也有这个 bug。实际上，不光是 xml 的声明，只要是在 DOCTYPE 的声明之前写入任何代码，都会把 IE6 带入怪异模式。多么古怪，不是吗？这就是我们在第 6 章里建议你省略 XML 头部声明的原因。

11.2.5 完整的 XHTML DOCTYPE 列表

下面是完整 XHTML DOCTYPE 的列表，它们都可以在支持 DOCTYPE 切换的浏览器上触发标准模式或 Mozilla 的准标准模式：

XHTML 1.0 DTD

XHTML1.0 Strict（**严格方式**）——在所有支持 DOCTYPE 切换的浏览器上都触

发标准模式，但对于 Opera 7.0 以前的版本和 IE6.0 以前的版本没有效果。

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

XHTML 1.0 Transitional（过渡方式）——在遵循标准的 IE 浏览器（IE6+/Windows、IE5+ /Macintosh）上都触发标准模式。在第一代基于 Gecko 浏览器上（Mozilla 1.0、Netscape 6）也触发标准模式，在新版本的基于 Gecko 的浏览器（Mozilla 1.0 1 、Netscape7+、Firefox 1.0+、Camino）上触发准标准模式。但对于 Opera 7.0 以前的版本和 IE/Window 6.0 以前的版本没有效果。

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

XHTML 1.0 Frameset（框架方式）——在遵循标准的 IE 浏览器（IE6+/Windows、IE5+/ Macintosh）上都触发标准模式。在第一代基于 Gecko 的浏览器上（Mozilla 1.0、Netscape 6）也触发标准模式，在新版本的基于 Gecko 的浏览器上（Mozilla 1.01、Netscape7+、Chimera0.6+）触发准标准模式。但对于 Opera 7.0 以前的版本和 IE/Window 6.0 以前的版本没有效果。DOCTYPE 转换也许会影响框架的呈现方式，就算真的有影响，看上去也没有浏览器厂商会把这些差异文档化。

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

XHTML 1.1 DTD

XHTML 1.1（严格定义）——在所有支持 DOCTYPE 切换的浏览器上都触发标准模式，但对于 Opera 7.0 以前的版本和 IE6.0 以前的版本没有效果。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/
DTD/xhtml11.dtd">
```

保护你漂亮的手指

如果你不喜欢输入书里的这些定义代码（谁会喜欢呢？），可以去 A List Apart 网站上的文章“Fix Your Site with the Right DOCTYPE”里复制粘贴那些代码（http://www. alistapart.com/stories/doctype/）。

DOCTYPE 转换：细节是魔鬼

DOCTYPE 切换不仅限于 XHTML 或 HTML5。前面提到过，一些完整的 HTML 4 的 DOCTYPE 也触发标准模式。例如，IE 中完整的 HTML 4.01 Strict DOCTYPE 也可以触发标准模式（21 世纪推出的基于 Gecko 的浏览器，则触发准标准模式）。代码如下：

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

但是，不管在 IE 浏览器上，还是在基于 Gecko 的浏览器上，一个完整的 HTML 4.0 DOCTYPE 却会触发向后兼容的 Quirks 模式。这 0.01 的版本更新到底有何不同？因此，我们一直都建议你最好使用 XHTML 建立你的站点，而不要用 HTML 4，以避免出现这些问题。

11.2.6 保持简单

基于标准的设计需要在标准模式的支持下才能蓬勃发展。可以使用一个包含完整 URI 的 XHTML 1.0 的严格型或过渡型 DOCTYPE，或者用一个 HTML 5 DOCTYPE 来触发标准模式。在 Gecko 的准标准模式下，可以在 CSS 里，明确地把图片转变为块级元素。如果你想把 IE8 的行为锁定在标准模式，即使用户要求 IE7 模式，你可以把 X-UA-Compatible 包含进去。

第 12 章

深入浏览器之二：Bug、变通办法和 CSS3 带来的一线希望

“一次创建，随处发布”是基于标准设计和开发的长期努力目标。我们学习书写正确的（X）HTML 代码并不是为了去赢得什么荣誉，而是为了让我们的站点能在今天或明天，甚至往后的十年里，都能在桌面浏览器、文本浏览器、屏幕阅读器及手持设备上很好地运行。同样地，我们使用 CSS 不是专门为了短期目的，比如节省带宽以减少本月在服务器上的花费。我们主要的目标是为了保证我们的站点将来在 IE 14.0 上的表现能跟今天的浏览器里看上去一样，而不必要的表现类标记将不会在没有应用 CSS 的环境里妨碍到用户体验。为了描述简便，我给基于标准设计的目的贴上了“向前兼容（Forward compatibility）”的标签。（我还把我的内衣贴上了“Jeffrey”的标签，当然这是基于另外不同的目的。）

遵循标准的浏览器开发商开始向前兼容，从原先怀着侥幸的想法，到转为理性的、可持续发展的设计策略。如果网站主要还是用那些上个年代的老旧浏览器来访问，那么多数站点就不可能获得向前兼容。如果在 4.0 版本的浏览器上，成功的主流浏览器开发商以牺牲基本标准为代价，继续推广他们专有的技术，那么网络作为一个开放式平台的前景就不怎么乐观了。幸好，在过去几年里推出的主流浏览器和一些小浏览器都还能被称为是“遵循标准”的。只是一些浏览器遵循得比另外一些要好一些。解决遵循标准中的缺陷是本章要讨论的内容。

12.1 细述 CSS 的 Bug

想象一下，你正在制作一个简单的内容模块。设计师交给了你一份设计稿（如图 12.1 所示），需要你制作一个提示框。因此，你回忆起在第 10 章中学到的标准工作流方法，你不是一开始就编写代码，而是先关注内容，列出设计中的不同内容类型。当我们完成了内容分析，我们就开始把它们转化为恰当的语义化标记。

这个任务看上去可以非常快速地完成。很快就可以大体分出两种类型的内容：

1. 模块头部的一个标题；

2. 紧跟在下面的一个段落。

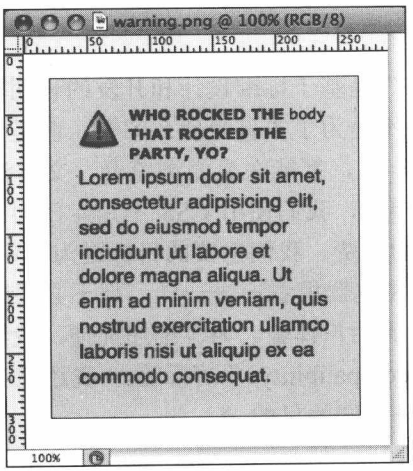


图 12.1　一个简单的小提示框需要制作模板。让我们来完成它

根据上面的分析，我们可以在一个空的（X）HTML 模板里书写如下代码：

```
<div id="warning">
<h2>Who rocked the <code>body</code> that rocked the party,
yo?</h2>
<p>Lorem ipsum dolor sit amet…</p>
</div><!-- /end #warning -->
```

简单的标记负责简单的任务：我们的段落使用标记 p，我们的标题使用标记 h2。我还使用了一个 code 元素来表示出标题中的一个有特定意义的小片段。另外用一个 div 元素把这些标记包在里面，并适当地把 id 属性设为“warning”，以便我们能够在 CSS 中指定元素。

注意：我为什么使用 h2 元素呢？这个假定的模块会是一个更大的页面的一部分，我设想这个模块的标题并不是一个页面上最重要的标题——为了这个练习的目的，我把它的重要等级放在 h1 之下。如果这是一个真实的模块，需要根据页面内容的相关标题的重要性来评估它，选择优先等级最适当的 HTML 元素来表示。现在，就让我们使用 h2 吧。

初看起来，这些基础的标记代码所表现出来的样子似乎没有什么吸引力（如图 12.2 所示）。但是，你现在已经知道，这个未经修饰的 HTML 需要应用一层样式：

```
body {
  background: #FFF;
  color: #333;
  font: normal 100%/1.3 Helvetica, Arial, sans-serif;
  margin: 0;
  padding: 0;
}
code {
  font: normal 1em/1.3 Monaco, "Lucida Console", "Bitstream
  Vera Sans Mono", monospace;
  text-transform: none;
}
#warning {
  background: #FCC;
  border: 1px solid #C00;
  float: left;
  padding: 1.4em;
  margin: 20px;
  width: 200px;
}
#warning h2 {
  background: url("alert.png") no-repeat 0 0.3em;
  color: #000;
  font: bold 0.75em/1.3 Verdana, sans-serif; /* 12px / 16px =
  0.75em/1.3 */
  margin: 0 0 0.3em;
  min-height: 40px;
```

```
  padding-left: 40px;
  text-transform: uppercase;
}
#warning p {
  margin: 0;
}
```

图 12.2　未经样式化的标记，不过，很快就会改观

可能有一些属性对你来说是新接触的，但是经过我们第 10 章的学习，这些通用的语法应该会感觉很熟悉。我会帮你把整个样式表完全分解（毕竟，我们在这里是为了展示 bug），但是，这里有几个需要注意的地方：

- 现在，你可能已经变成一个擅长发现的老手了，但是，我们的首要任务是在 body 元素上设定默认的版式属性。在随后的规则里，我们同样为文档中的 code 元素设定了页面范围内的参数；

- 我们针对“warning”元素自身的大部分工作都已经完成了：设定了它的 width 为 200px，为四周的 margin 值设定了 20px，还把它的 background-color 设为#FCC（使用更简洁的简写属性 background）。我们把它浮动到左边，这是假定我们的设计师是这样要求的；显然它将被插到页面的内容中，根据需要在它的右边围绕文本；

- 在下一条规则中，我们使用 background 属性来给标题（#warning h2）附上图标（alert.png）。就像 margin、padding 和 font 一样，background 也是一个 CSS 的简写属性，它包含了 background-image 的值（为我们的图片提供一个 URL）、background-repeat 的值（我们并不想让图片重复显示，所以我们把它设为 no-repeat）和 background-position 的值（两个值，0 和 0.3em，分别代表图片在元素内部的 x 和 y 方向的偏移值）。也可以在其开头处声明 background-color 的值（background：#FCC url（“alert.png”）no-repeat 0 0.3em），但在这里，我们保持它的 background-color 默认值为 transparent；

- 为了配合 background 属性，我们标题的 padding-left 的值设为 40px。没有这个值的话，标题的文本就会覆盖掉背景图片，因为背景图片——就像我们的小图标——在页面流中是不占据空间的；

- 相类似地，我们使用 min-height 属性来保证，即使标题缩得非常短时也能有一个最小的高度 40px。如果没有这个保证，当标题只有一个单词的时候，元素的高度将会大大地减小，以至切掉底部的图片。没有人希望这样；（相信我。）

- 最后，你可能注意到了，我们的标题在浏览器里显示的是大写字母，而在 HTML 源文件中却不是。我们把 text-transform 属性设为 uppercase，使标题显示为大写，而不用修改标记。当然，我们也可以把 code 元素设为 text-transform：lowercase，来把看不顺眼的大写字母转成小写字母。

当我们把样式表链接到未经修饰的标记后，Web 标准就这样极富魅力地起作用了（如图 12.3 所示）。你的标记是合法的，你的 CSS 代码是规范的，而且在各种浏览器中，比如 Safari、Firefox、Opera，甚至在 IE7 和 8 中，看起来都非常完美——总之，干得漂亮！你已经完成了另一个基于标准的设计，而且仅用了一点点的 CSS 和（X）HTML 代码。我们能够就此满足吗？

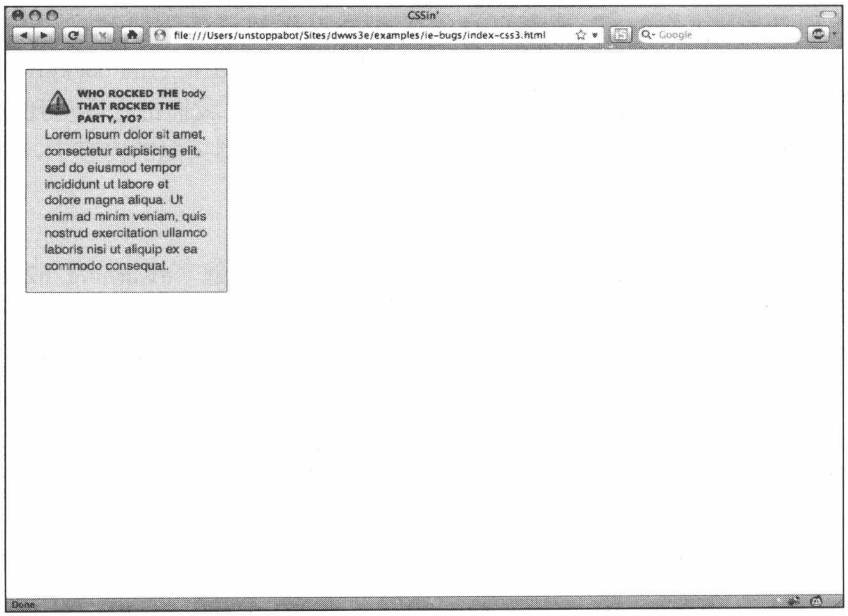


图 12.3　我们小小的“warning”div 在大多数现代浏览器中看起来都非常棒。但是，我们的工作还没做完

不幸的是，工作还没结束。虽然在这些现代的主流浏览器上没问题，但是还不是发布代码的时候，还需要在另一些浏览器上做进一步的测试。可能有点枯燥，但是全面地进行各种浏览器的测试是保证项目成功发布的关键，跟书写正确的代码同样重要。虽然你的 CSS 和（X）HTML 也许是精确地遵循规范的，但是有些浏览器可能会用一些不同的方法实现某个特性——或者，至少是用了不是你所预期的方法来实现，根据含糊不清的规范，使用它自己独特的解决方案（可称之为：不兼容）。

有时，某个浏览器就是一个失败的产品。如果把我们刚做的那个简单的小小的模板在 IE6 中打开——啊，心都碎了，简直就是布局破坏者——我们看到的就是这样。让我们来看一看破坏这可怜模块的三个小 bug。

嘿嘿，我再也不用安装那么多版本的浏览器了

在互联网早期的那些混乱岁月，针对不同的浏览器/平台进行兼容性检查往往是令人丧气的。如果那些浏览器是跟操作系统捆绑在一起的，像 IE/Windows 或者

Safari，那要测试它们经常是不可能的。切换浏览器需要频繁地切换它们所依赖的操作系统，所以，在老版本的浏览器上检查代码，需要在不同的机器上，或者是不同的硬盘分区，每个版本一个。

幸运的是，这一切都已经改变。一些独立开发者已经为我们提供了非官方的方法来突破这些限制，比如，提供多版本共存的 Multi-Safari(www.michelf.com/projects/multi-safari)和 Internet Explorer Collection(www.finalbuilds.edskes.net/iecollection.htm)。而如果要和一些资源结合起来一起使用的话，例如 evolt.org 网站上的 Browser Archive（Browsers.evolt.org），那搭建起一个合适的测试环境还真不是一件容易的事情。

12.1.1　浮动元素的双倍边距 Bug

实际上，乍一眼看，我们的布局看上去并不是太坏（如图 12.4 所示）。我们的颜色是对的，字体也是规范的，“warning”块看上去也很好。毕竟，也许这个“标准”的浏览器就是这样工作的。温暖的感觉油然而升。

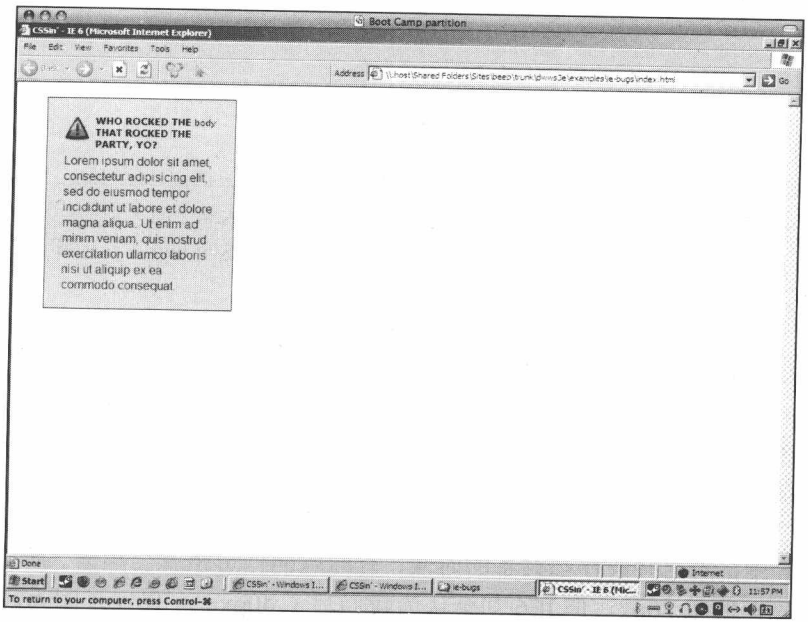


图 12.4　这是“warning”div 在 IE6 中显示的样子。看上去不太坏，但是我们能做得更好

但是，如果你看一下模块的左边，这温暖的感觉肯定就没了。你将会注意到它的外边距在 IE6 中大得有点古怪——事实上，它把我们在 CSS 中所设定的值加倍了，是其他遵循标准的浏览器里显示的两倍（如图 12.5 所示）。根据网上的一些研究文章，我们意识到我们在无意中碰到了 IE6 中的浮动元素双倍边距的 Bug 了（www.positioniseverything.net/explorer/doubled-margin.html），下面的两种情况反映了这个问题：

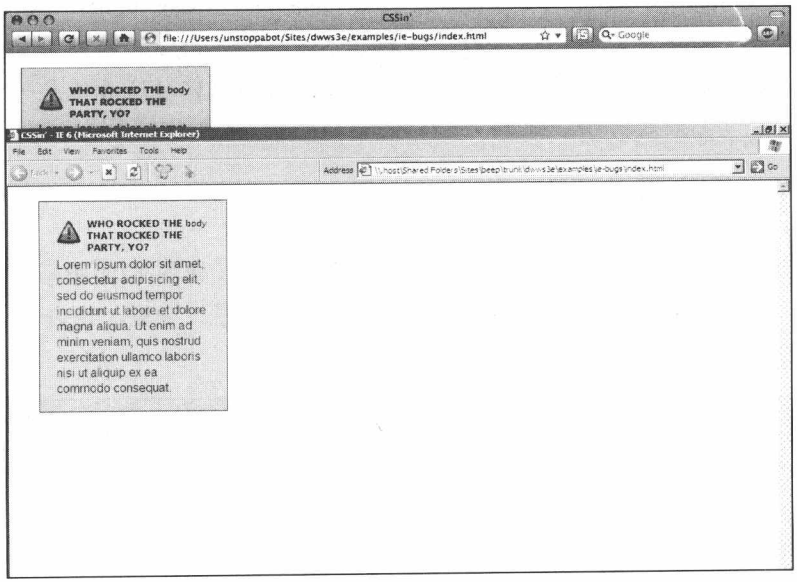


图 12.5　看出在 Firefox（上面的）和 IE6（下面的）中的不同了吗？碰到浮动元素双倍边距的 Bug 了。我真想跟你说你再也不会碰到这个 bug 了，但那是骗你的

1. 这个元素是浮动的，不管是向左浮动还是向右浮动；

2. 在浮动的相同方向设有一个外边距（比如，float：left 的元素设了 margin-left，或者 float：right 的元素设了 margin-right）。

在 IE6 中，当这些条件都具备的时候，有问题的外边距就会翻倍。所以，尽管在 CSS 中明确声明了 20px，但 IE6 却会认为我们写了 40px——仅仅是因为我们在 float 的相同方向声明了一个 margin。（如果你能解释为什么会这样的话，估计 Web 设计师社区都可能捐钱给你，或者送你一匹小马了。）

祝贺你，你已经揭露了第一个浏览器的 bug 了。但是，还没完——IE6 还有很多的问题。

12.1.2　PNG 图片的透明背景问题

接着看模块的中间，你会注意到 alert.png 的图标看上去不太对。在我们的图标底下，IE6 并没有透出“warning”元素的粉色背景（#FCC），而是出现了一种神秘的灰色（如图 12.6 所示）。不幸的是，问题出在文件名扩展上：我们并不是用一张简单的 GIF 图片作图标，而是用了一张带透明底的 24 位 PNG 图片——这样，我们在改变模块的背景颜色时就不须要重新做一张新的图片来匹配了。（更多的关于 PNG 的信息请查看 www.mywebsite.force9.co.uk/png。）

虽然使用 PNG 图片具有非常棒的灵活性，但是在 IE6 及以前的版本中，都不支持它们的 alpha 透明层。取而代之的是，IE6 把透明的像素转换成了实体的灰色像素，也许它喜欢听到我们尖叫的声音。

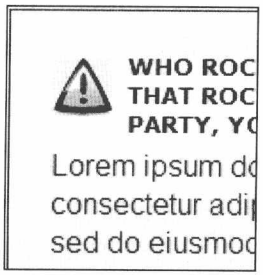


图 12.6　我们的 PNG 图标肯定是少了透明度。天哪，真是要谢谢 IE6 了

12.1.3　前进之路

我们这个简单的模块，在所有现代的桌面浏览器里都能恰当地显示，只是在 IE6 中有两个引人注意的 bug，而 IE6 是目前使用的最古老最普及的遵循部分标准的浏览器。那么，IE6 所拥有的大量市场份额（同时也拥有的大量 bug）是否意味着 Web 标准只是一个理论上的空想，是一个无法适用于现实设计的学术上的灵光一现？

不全是。为了对付各个浏览器之间表现手法的差异或者 CSS 实现的不完整，这里有一些变通办法。在这些浏览器中有着比这章中所提到的多得多的 CSS 错误，数以十计（大多数出现在 IE6 及更早的版本上，微软对此无疑应该感到羞愧），下面列

出的一些站点或许可以帮助你快速了解：

- Position is Everything（www.positioniseverything.net）。“详细解释现代浏览器的 bug。”这是一个值得加入书签的站点；

- CSS-Discuss（www.css-discuss.org）。这是由 CSS 专家 Eric Meyer 创办的，一个很受欢迎的邮件列表，专注于“讨论 CSS 在现实工作中的使用方法。”。CSS-Discuss 对于初学标准设计的新手和有经验的老手都是不可或缺的，它能在你最无望的时候提供帮助；

- Browser Bugs at CSS-Discuss（css-discuss.incutio.com/?page=BrowserBugs）。这里简直就是一个富矿。你需要了解的所有主流浏览器中的 CSS 缺陷都可以在这里了解到，是使用 CSS-Discuss 时的好帮手；

- WordPress CSS Codex—Fixing Browser Bugs （codex.wordpress.org/CSS_Fixing_Browser_Bugs）。WordPress 的 CSS 使用宝典之修正浏览器的 bug；

- IE Blog（blogs.msdn.com/ie/default.aspx）和 Surfin'Safari（www.webkit.org/blog）。苹果公司让 Safari 浏览器的首席工程师 Dave Hyatt 在博客里讨论 Safari 的 bug 及修复状况，以此来培养一种开放的感觉——以便说服 Mac 用户转而使用 Safari。这是很有效果的。微软也正在做同样的事情。尽管它们本身是“bug 博客”，但却都是一个跟这两个重要的浏览器开发者进行交流的绝好地方。

幸运的是，大多数 IE 的 bug 在 IE7 中已经解决了，而发行于 2009 年上半年的 IE8 也是非常稳定的。但是，并不是每一个 IE 用户都会升级原来的 IE6 浏览器——或许还是会升级的，所以你仍然需要了解和解决本章讨论的这些 bug。有了这些资源做后盾，我们就可以开始解决这些小小的 CSS 问题了。

12.1.4 发现问题仅仅是战斗的一半

首先，让我们来仔细看一下 IE6 中双倍 margin-left 的问题。根据 Position Is Everything 网站上有关浮动元素双倍边距 bug 的详尽文章（www.positioniseverything.net/explorer/doubled-margin.html），在 CSS 里加上 display：inline 将使这个夸张的边距

消失。（我们和他们都无法解释为什么会这样。）在这个例子中，我们是否这样写就行了呢？

```
#warning {
    background: #FCC;
    border: 1px solid #C00;
    display: inline;
    float: left;
    padding: 1.4em;
    margin: 20px;
    width: 200px;
}
```

它的效果看起来像这样（如图 12.7 所示）。增加了这一句之后，IE6 正确地把我们的 margin-left 设为了 20px。更重要的是，把“warning”转变为一个内联盒没有对布局产生不利的影响。CSS 规范明确地说明浮动总是加在块级元素上的，把 display：inline 应用到浮动的元素上，它的效果跟 display：block 是一样的——也就是说，没有影响。所以，我们用 display：inline 来修复 IE6 的特殊问题，是不会让我们的 CSS 失效或者影响到其他浏览器的显示。

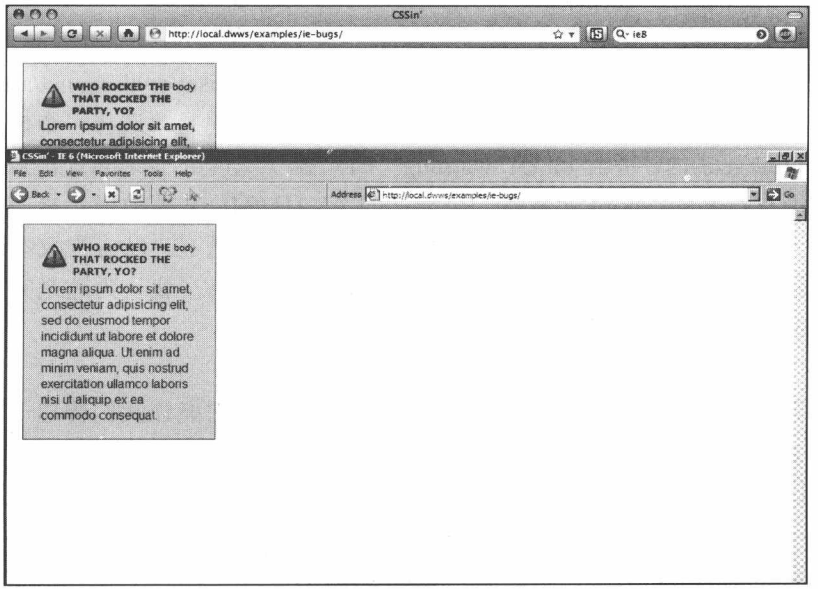


图 12.7　加上 display：inline 后，我们的边距恢复正常了。但这是最好的修复方法吗？

然而，某些你将来会碰到的 CSS bug 需要更小心地修复。假如我们需要针对某个特定的浏览器来修复我们的 CSS，那么，让我们看看能不能找到更多的修复策略。

Hack 浏览器

CSS 的 hack 技术在基于标准的设计中已经有一段传奇历史了，设计师们会利用某个浏览器在 CSS 实现上的 bug，来让适当的样式规则应用到“好的”浏览器，而同时给有问题的浏览器提供一个它的渲染引擎能理解的值。这里有一个例子：

```
#warning {
    background: #FCC;
    border: 1px solid #C00;
    float: left;
    padding: 1.4em;
    margin: 20px;
    width: 200px;
}

* html #warning {
    display: inline;
}
```

使用星号 HTML（* html）hack 技巧（参见 www.positioniseverything.net/articles/poll/star-html.php）。从字面上来说，第二个规则的选择器的意思是“选择一个位于任何元素降序的 html 元素内，并且 id 为‘warning’的元素”。聪明的读者可能注意到了，这条规则是没有意义的：html 元素是一个合法的（X）HTML 文档的根元素，它不能包含在其他元素中。所以，理论上第二条规则是无效的。但是 IE7 以前的版本认为 html 元素上面还有一类不可见的元素，即使它不在标记里显示。（我认为这是完全没有意义的。欢迎加入忍受着 IE6 痛苦的圈子里。）

这条无效的代码只对 IE6 及之前的版本有意义，因此它能让我们传递一个特殊的值来修补 IE 遗留版本的错误。正确的浏览器不会理会这个不存在的 HTML 元素，随同这条规则一起被忽略掉。还有额外的好处吗？虽然* html 是完全没有意义的，但是在 CSS 中，它的语法却是完全合法的。这样我们就可以把 display：inline 的声明隔离在一条独立的规则中，以便让我们的无 hack 的规则保持“干净”，不搀杂无效的样式表。使用 hack 技巧从来没有感觉这么好过。

拥有数学头脑的读者或许已经想到，修复双倍边距的问题还有一个完全不同的策略：

```
#warning {
  background: #FCC;
  border: 1px solid #C00;
  float: left;
  padding: 1.4em;
  margin: 20px;
  width: 200px;
}
* html #warning {
  margin-left: 10px;
}
```

我们仍然使用星号 HTML 的 hack 技巧，但是我们并不是去纠正 IE6 糟糕的算术，而是顺着它。我们告诉老版本的 IE 把 margin-left 的值设为 10px，这样双倍后就是 20px，正是我们期望的值。不过，这并不是一个非常有实践意义的方法，因为使用 display:inline 来修复这个问题总是正确的，而且还能让我们省去烦琐的“算术”工作。但是如果当你需要使用一个纠正后的值给某个特别的浏览器时，这便是另一个可以帮助你的办法。

注意：请查阅 www.centricle.com/ref/css/filters，获得关于 CSS 的 hack 技巧的详细列表。你也许需要准备一大杯饮料。

星号 HTML 的 hack 技巧是非常温和的方法。就像我上面提到的，它利用了某个浏览器的特性来解析错误，而不会使我们的 CSS 失效。但根据你所修补的浏览器，hack 技巧的使用不应该太随意。另外，星号 HTML 的方法也不是唯一的 CSS hack 技巧。根据你正在修复的 bug，你可能需要加入一些畸形（非法）属性，这些属性标签只对受影响的浏览器起作用。或者你在 CSS 中加入了更多合法的但没有意义的选择器，就像* html 之类的，那么，所增加的样式表尺寸也会增加你的维护工作。

我意识到自己正在描绘一幅暗淡的图画。事实上，恰恰相反。就像我在本书中一直说的，一个恰当构建的样式表——应用在一个合法的（X）HTML 文档上——能够在今天大多数主流的浏览器上顺利工作，虽然在早期版本的浏览器上可能会不太完美。一旦你熟悉了一个浏览器的特性，就会发现需要用到 hack 技巧的地方是难以

置信地少；取而代之的是，你将会用简单的代码来绕过问题，大大避免了凌乱的 hack 代码的使用。

保持分开：使用条件注释

让我们看一下让 IE 支持 24 位 PNG 的修复方法，以此来把标题图标的灰色背景去除掉。

```
#warning h2 {
  background: url("alert.png") no-repeat 0 0.3em;
  color: #000;
  font: bold 0.75em/1.3 Verdana, sans-serif; /* 12px / 16px = 0.75em/1.3 */
  margin: 0 0 0.3em;
  min-height: 40px;
  padding-left: 40px;
  text-transform: uppercase;
}
* html #warning h2 {
  background: none;
  filter: progid:DXImageTransform.Microsoft.AlphaImageLoader(src="alert.png",
  sizingMethod="image");
}
```

星号 HTML hack 技巧所使用的选择器也许是合法的 CSS，但包裹在它里面的代码却不一定是。仔细看 AlphaImageLoader，这是一个微软的 CSS 滤镜，可以用来修复在遗留的低版本 IE 中 PNG 图片的透明背景问题。更多的信息可以查阅微软的开发者网站（msdn.microsoft.com/en-us/library/ms532969（VS.85）.aspx），不过，这并不符合标准的精神：这个变通办法使用了不合法的 CSS，使你的代码散乱不堪，通常使你更痛苦。有时，当我们需要为有问题的浏览器使用变通办法时，如果能够避免使用含有不合法代码和使 CSS 代码量膨胀的那些 hack 技巧，那样会更好。那可选的方案是什么呢？

为 Happy Cog 项目构建 XHTML/CSS 时，我们的模板通常会在 head 里包含以下代码：

```
<link rel="stylesheet" href="css/screen/main.css" type="text/
css" media="screen, projection" />
<!--[if IE]>
<link rel="stylesheet" href="css/screen/patches/win-ie-all.css"
```

```
type="text/css" media="screen, projection" />
<![endif]-->
<!--[if IE 7]>
<link rel="stylesheet" href="css/screen/patches/win-ie7.css"
type="text/css" media="screen, projection" />
<![endif]-->
<!--[if lt IE 7]>
<link rel="stylesheet" href="css/screen/patches/win-ie6-below.
css" type="text/css" media="screen, projection" />
<![endif]-->
```

那些看上去奇怪的 HTML 注释实际上是条件注释（msdn.microsoft.com/en-us/library/ms537512（VS.85）.aspx），一个微软发明的属性，它可以让 IE 在解析 HTML 时使用一些逻辑条件。当 IE 碰到一个条件注释，它就会对方括号内的代码进行求值；如果条件符合，那么就解析其中的标记代码；如果不符合，就会跳到页面中的下一段代码，就像什么都没有发生。

在上面的例子中，所有的浏览器都会获得没有 hack 代码的基础样式表 main.css。而我们使用条件注释来让 win-ie-all.css 对 IE 类浏览器有效，win-ie7.css 只对 IE7 有效（[if IE 7]），win-ie6-below.css 只对 IE7 以下版本有效（[if lt IE 7]）。如果我们需要应用一个针对 IE8 的分支，那么我们可以用如下的代码增加一个额外的样式表：

```
<link rel="stylesheet" href="css/screen/main.css" type="text/
css" media="screen, projection" />
<!--[if IE]>
<link rel="stylesheet" href="css/screen/patches/win-ie-all.css"
type="text/css" media="screen, projection" />
<![endif]-->
<!--[if IE 8]>
<link rel="stylesheet" href="css/screen/patches/win-ie8.css"
type="text/css" media="screen, projection" />
<![endif]-->
<!--[if IE 7]>
...
```

目前，IE8 能很好地兼容标准，看上去似乎不需要特别的照顾。但是这表明了使用条件注释的好处：我们主要的 CSS 文件可以保持原样没有 hack 代码。而针对 IE6，我们能够不修改 main.css，而仅仅在 win-ie6-below.css 中书写代码来修复两个 CSS bug：

```
#warning {
display: inline;
}
#warning h2 {
  background: none;
  filter: progid:DXImageTransform.Microsoft.
  AlphaImageLoader(src="alert.png", sizingMethod="image");
}
```

我们仍然使用原来的修补方法（使用 display:inline 和 AlphaImageLoader 声明），但是使用了更简单的选择器。通过条件注释来区别对待我们的浏览器，我们就不需要老想着浏览器是如何解析那些 bug 的，也不用费力去记着哪些 hack 技巧是针对哪些浏览器的了。我们可以简单地书写正常的 CSS 选择器，然后使用条件注释来使样式表应用于所需要的浏览器。

我知道倡导使用一个“微软独有的发明”也许是令人痛苦和焦虑的。然而，天才的条件注释确实是合法的 HTML：所有非 IE 的浏览器都把条件注释看作是普通的 HTML 注释，从而跳过注释里面的代码。但是 IE 中的条件注释肯定是一种非标准的行为，因而导致一些崇尚标准的发烧友反对使用它们，认为依靠这样一个特有属性是违背基于标准来设计的精神的。

我理解他们的异议。我不能夸大用条件注释来修复 IE bug 的价值。而且，为了修复 bug 而使用的那些 IE 特有代码是规避了验证程序的，安全地藏在一个看上去无害的 HTML 注释里。

但是，我们不会使用条件注释来故意骗过验证程序，而把不合法的 CSS 藏在后面。用只针对 IE 的文件来把 CSS 隔离成独立的分支，使我们能够减少大多数遵循标准的浏览器访问时的带宽，而仅仅把分支文件应用在需要它们那些有缺陷的浏览器上，并不是把很多针对不同浏览器的 hack 代码散落在我们的基础样式表里，因此减轻了我们的维护工作。如果我们需要修复一个 IE7 的问题，就不须要在上千行的基础样式表里查找了，而只要在那个特别针对这个浏览器的文件里修改就可以了。

使用脚本来修复 PNG 图片的问题

用了一阵 AlphaImageLoader 的方法，你可能很快就发现了它的缺点。首先最重

要的就是，每个使用 PNG 图片作背景的元素都需要写一个相应的 AlphaImageLoader 滤镜。你会发现你写了一大堆这类规则，如下：

```
#brand h1 a {
  background: none;
  filter: progid:DXImageTransform.Microsoft.
  AlphaImageLoader(src="/-/img/logo.png", sizingMethod="crop");
}
#brand h1 b {
  background: none;
  filter: progid:DXImageTransform.Microsoft.
  AlphaImageLoader(src="/-/img/08.png", sizingMethod="crop");
}
li.alert {
  background: none;
  filter: progid:DXImageTransform.Microsoft.
  AlphaImageLoader(src="/-/img/alert.png", sizingMethod="crop");
}
```

让我们面对现实：没有人愿意看到这样的 CSS。（好吧，或许你愿意，这里没有判决。）

此外，当我们应用 AlphaImageLoader 到一个元素时，它的背景图片就失去了作为背景图片的一些行为，忽略了 CSS 属性中的 background-position 和 background-repeat。这意味着如果你用 AlphaImageLoader 修补了一个背景图片，那么它就会固定到元素的左上角，并且也不能平铺它。事实上，这给我们的这个小模块带来了一点麻烦（如图 12.8 所示）；放大头部，可以看到 alert.png 固定到了左边的最上角，而忽略了 background-position：0 0.3em 的属性。

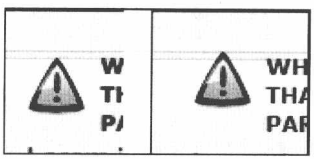


图 12.8　左图是支持 PNG 透明背景的兼容标准的浏览器中正确应用了 background-position 的效果，IE6 中的 AlphaImageLoader 的应用正确地显示了透明背景，但是偏移了位置

宽慰的是，一个相对而言比较新的 JavaScript 库叫做 DD_belatedPNG（www.dillerdesign.com/experiment/DD_belatedPNG）提供了比 AlphaImageLoader 更灵活的方法，能够把图片转换成 VML 对象，不仅使其在 IE6 及以前的版本中显示正确，而且

同样可以定位和平铺。只要下载这个文件，然后把它加到文档的 head 里便可：

```
<!--[if IE 6]>
<script src="js/DD_belatedPNG.js" type="text/javascript"></
script>
<![endif]-->
<script src="js/scripts.js" type="text/javascript"></script>
```

这两个 script 元素的功能跟 link 元素一样：引用了两个外部的在 js/目录下的 JavaScript 文件，文件名为 DD_belatedPNG.js 和 script.js。前面的一个是我们刚刚下载的库文件，我们用 IE6 规范的条件注释来分开；后面的一个是空的文件，我们将把自己的 JavaScript 代码写在里面。打开 script.js，写入如下代码：

```
window.onload = function() {
  if (typeof DD_belatedPNG == "function") {
    DD_belatedPNG.fix("#warning h2");
  }
}
```

我们将在本书的最后更多地讨论 JavaScript 的细节，但上面的片段中有一些值得一提。第一行（window.onload ...）指示浏览器在页面下载完成以后执行花括号里面的代码。当页面下载完时，JavaScript 会查看 DD_belatedPNG 函数是否已经下载完，而这仅会发生在用户使用的是 IE6 浏览器时，感谢我们的条件注释。如果是的话，针对 PNG 的修复将会被应用，我们的小模块在 IE6 中最终被恰当地修补好了（如图 12.9 所示）。（使用 DD_belatedPNG，当图片过多要处理时，是存在性能问题的，作者没有提到）

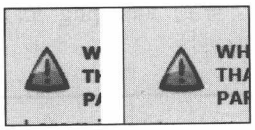


图 12.9　我们的 PNG 图片回到了原位，真是值得庆祝

12.2　CSS3：新的热点

CSS 规范的第二个版本 CSS2 是由 W3C 标准组织于 1998 年 5 月完成的。是的，就是在 1998 年产生了我们今天还在使用的 Web 标准。花了十多年的时间最终看到在大多数的浏览器中广泛地支持了这个规范，但是 Web 设计师们却没花多长时间就开始提出这个规范的局限性了。毕竟，我们有很强的好奇心，我们开始询问在这个语

言应用了十年以后，将会如何发展。

感谢 CSS3，这个规范的下一个主要版本，目前正在由 W3C 编写。草案的工作还远远没有完成，但是你现在可以在 CSS 工作组网站（www.w3.org/Style/CSS/current-work）上查阅不同组件模块的状态。目前你们也许还在犹豫是否要去阅读这些沉闷枯燥的规范。（不要说我责备你。）但是，那里有一大堆有趣的特性，远远不止我们在这里列出的这些。事实上，CSS3 规范中的许多可爱的特性都是设计师们呼吁了好多年的。让我们来看几个。

12.2.1　关于 Alpha 通道

Wilson Miner 是一个位于旧金山的设计师，他有一个可爱的小网站——他的设计是低调而到位的，一个良好定义的字体使得阅读起来更舒服（如图 12.10 所示）。然而，如果查看他的样式表的话，你很快就会发现一些有趣的事情。Wilson Miner 到处书写了类似如下的规则：

```
body {
  background: #9178C4;
  color: rgba(30, 40, 30, 0.9);
}
```

图 12.10　Wilson Miner 的可爱的小站点（www.wilsonminer.com）

遍布他的整个样式表，Miner 选用了 rgba()来声明他的颜色值，这是一个新的 CSS3 颜色模型的特性（www.w3.org/TR/css3-color/#rgba-color）。如果你曾经在 CSS 中用 rgb()来设置过颜色值，那么这种标记法你会觉得很熟悉。括号里前面的三个数字是红色、绿色、蓝色的值，可以设在 0 到 255 之间。
这三个值可以分别转变为三个十六进制的色彩值，在这个例子中是#1E281E，一个深灰色。

而有趣的是第四个值，它允许用户设置色彩的 alpha 通道的值，就是指定色彩的透明度。这个值可以设在 0.0（完全透明）到 1.0（完全不透明）的范围。Miner 使用这第四个值来调出惊人的效果，他设置站点的字体颜色为 rgb(30，40，30)（就是十六进制值#1E281E），但是加上了 90%的透明度（0.9）。结果就是，如果他改变了背景颜色，他就无须调整字体的颜色就可以保持文本相应的对比效果（如图 12.11 所示）。更重要的是，rgba()并不局限于用在字体颜色上。它同样可以应用到边框和背景颜色，允许设计师灵活地建立一个微妙的色彩覆盖层，来增加设计的效果（如图 12.12 所示）。

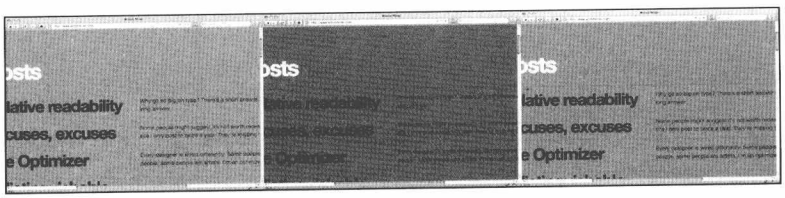


图 12.11　如果改变 Wilson 的背景颜色，rgba()设置的透明度将保持同样的效果

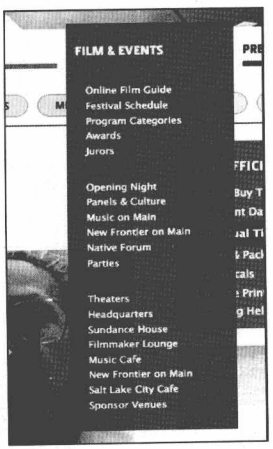


图 12.12　这里是一个下拉菜单的例子，可以看到使用 rgba()的好处

跟 rgba()相对应的是 opacity。两者都影响 alpha 通道，但是它们之间有一点微妙而重要的差别。rgba()改变的是色彩自身的透明度，而 opacity 改变的是整个元素的透明度，包括它的子元素。看实际例子，让我们在“warning”元素上加上一个 opacity 属性：

```
#warning {
  background: #FCC;
  border: 1px solid #C00;
  float: left;
  padding: 1.4em;
  margin: 20px;
  opacity: 0.7;
  width: 200px;
}
```

跟 rgba()一样，opacity 也可以设置为 0.0 到 1.0 之间，完全隐藏或者完全显示这个元素。但是在图 12.13 中可以看出，改变的是整个元素的透明度，包括里面的文本和图标。关于透明度，最强大，最棒的一个方面是它得到了广泛地实现：在本书编写的时候，Firefox 3+、Safari 3+、Opera 10+和 Google Chrome 都支持 rgba()和 opacity。因此，alpha 通道可以很好地使用了。

图 12.13　设置 opacity 来改变“warning”的透明度，同样影响到它的子元素

12.2.2 突破四四方方的样子

在采用 Web 标准的早期，有一个通常的批评是认为基于 CSS 的设计感觉上都是方方正正的。在前面的例子中，我们依靠盒模型来构建设计，主要是设置边框、边距和背景颜色，因此上面的批评可能不算是离谱的。但是后来我们发现了一些方法来实现诸如圆角效果和阴影效果，希望以此来增强页面的立体感。

但是，我们中的一些人可能已经有点吃不消了。因为，实现这些效果需要大量的冗余标记代码，大部分是为我们的 CSS 提供非语义性的标记，来实现一种圆角方块或者阴影的错觉。我们选择的这些方法基本上都缺乏灵活性。为了建立令人着迷的阴影效果，我们设置一个块的宽度来匹配阴影的图片；而改变这个块的宽度通常需要编辑样式表和重做一张新的图片。

注意：如果有兴趣学习使用 CSS3 之前的技术来建立圆角的话，CSS-Discuss wiki 通常是一个很棒的资源：css-discuss.incutio.com/?page=RoundedCorners 和 css-discuss.incutio.com/?page=DropShadows 上有很多链接。

尽管这些技术都是经过时间考验的，也很可靠，但是 CSS3 许诺了我们三个听上去绝妙的可选方案：text-shadow、box-shadow 和 border-radius 属性。www.10to1.be 网站（如图 12.14 所示）就是一个应用了这三个属性的很好的例子，这个网站是由 www.madebyelephant.com 的 Tim Van 设计的。感谢 box-shadow 和 border-radius 属性，头标记（header）下面淡淡的阴影和重要链接上面的圆角都是纯粹用 CSS 实现的（如图 12.15 所示）。Tim 还对整站的文字使用了 text-shadow 属性，以实现一种凸起的亮光效果（如图 12.16 所示），代码如下：

```
body {
  background: #222 url(../img/body.gif) repeat-y fixed 50% top;
  color: #86521E;
  font: normal 13px/20px "Lucida Sans", "Lucida Grande",
  "Lucida Sans Unicode", Verdana, sans-serif;
  text-shadow: #f1e4bc 0 1px 0;
}
```

没有切割图片来实现这些效果，也没有增加额外的标记。CSS 的远景可见一斑。

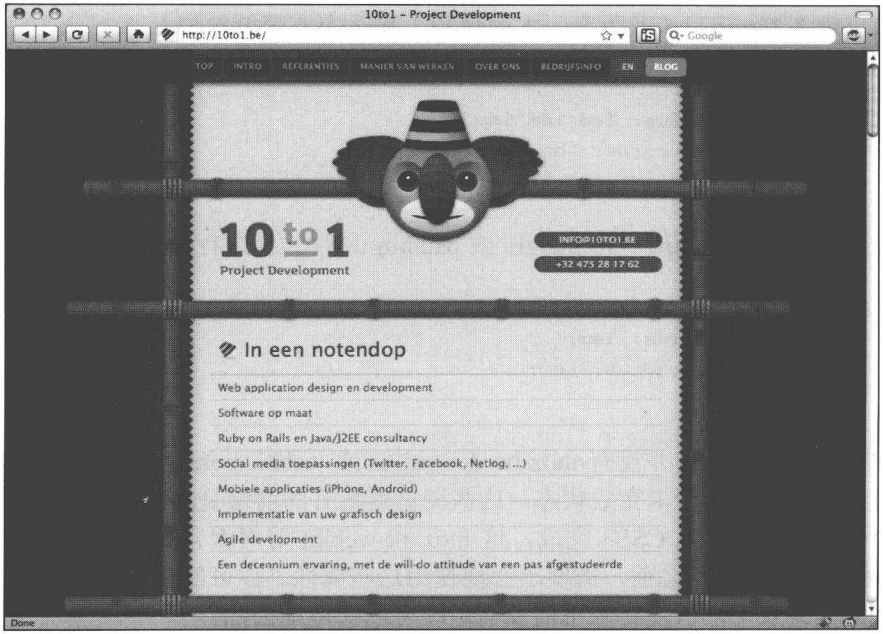


图 12.14　10to1.be 网站上雅致地应用了 CSS3 的属性，实现了显著的效果

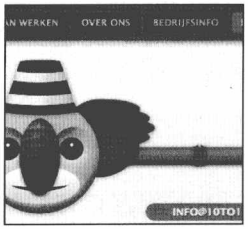


图 12.15　没有使用 PNG 图片就实现了这些效果

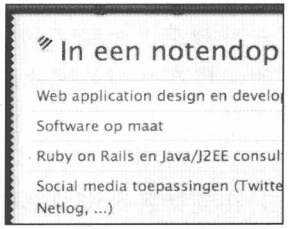


图 12.16　body 元素上的一个小小的 text-shadow 属性令所有的文本都有了一种凸起的效果

12.2.3　让编码器注意

值得强调的是 CSS3 规范的很大一部分尚未完成。一些浏览器厂商正等待规范更成熟一点后再去实现它，但是也有一些厂商已经小心翼翼地迅速包含进了一些 CSS3 的特性。比如，如果想在 Safari 3 及更高版本中使用 border-radius 属性，那么应该这样写：

```
#warning {
    -webkit-border-radius: 1em 1em 1em 1em;
}
```

如果想在 Firefox3.5 及更高版本中使用，那么应该简单地加上：

```
#warning {
  -moz-border-radius: 1em 1em 1em 1em;
  -webkit-border-radius: 1em 1em 1em 1em;
}
```

我们也能进一步地像对待 margin 或 padding 属性一样进行缩写：

```
#warning {
  -moz-border-radius: 1em;
  -webkit-border-radius: 1em;
}
```

你会发现在规范中并没有-moz-border-radius 或-webkit-border-radius，而且将来也不会有。由于规范处于未完成状态，这些浏览器特有的前缀是被 Apple 和 Mozilla 作为一个预防步骤；一旦 CSS3 完成，这-moz 和-webkit 前缀就会被去掉，而变为恰当的 border-radius。在这之前，需要小心地使用这些属性。你可能会想到把它们放到一个独立的样式表中，以便将来不需要前缀时可以简单地去除。

注意：在本书编写时，Opera 和 IE 还没有实现 border-raduis。（不过 Opera 10 已经实现了一些规范上更稳定的特性，比如 rgba()、text-shadow 等。）更重要的是，Mozilla 和 Safari 的实现在一些关键地方有不一致。所以，使用这些实验性的属性，需要你自己去把握风险。

此外，在支持各种浏览器中 CSS3 的不同模型时，对不兼容 CSS3 的浏览器进行支持是值得的。比如，当使用 rgba()来设置颜色值时，增加一个备用的颜色值是很简单的：

```
#warning {
  background: #FCC;
  background: rgba(255, 0, 0, 0.2);
}
```

我们重复声明 background 属性的理由在于：没有实现 rgba()的浏览器将使用十六进制的值，而认为第二个 background 是畸形的，并且忽略它。事实上，Wilson Miner 在他的个人网站上就是这样做的。认识 rgba()的浏览器在背景上会有透明度，而在 IE 和 Opera 10 以前的版本上，使用的是 Miner 用 rgb()提供的不透明的颜色（如图 12.17 所示）。

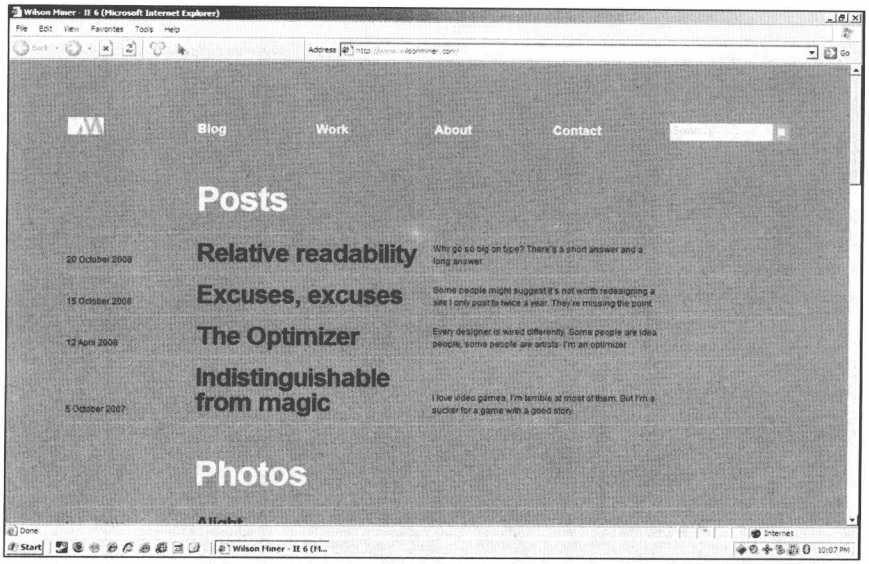


图 12.17　在 IE6 和老版本的 Opera 中，颜色明显地缺少 alpha 值

因此，适当地使用这些回退设置，就能逐步确保我们的内容尽可能地被最广泛的受众访问到。但是，根据对 CSS3 的支持程度，你的设计在不同的浏览器之间将会出现明显的差别。这是一件坏事吗？

12.3.4　重新思考“支持”

可以预见的是，新的标准快速地发展，使得一些浏览器已经不能适应了。当人们为 CSS 的将来激动不已时，围绕着那个古老的主题，人们也已经展开了大规模的讨论：那就是“浏览器的支持”在今天到底意味着什么？一个新的令人兴奋的 CSS 规范正在积极地开发，浏览器厂商们快速地采用了其中他们最感兴趣的部分；某些情况下，有些浏览器厂商还在创建一些自定义的扩展，以期望 CSS3 能够采纳（如图 12.18 所示）。那么，那些不能支持这些特性的遗留的老旧浏览器该怎么对待呢？

当然，当支持问题提出时，IE6 通常就是讨论的中心。它的吱吱作响的，有缺陷的渲染引擎已经折磨了设计师们数十年。不过现在，随着 CSS3 和 HTML5 的出现，规范与现实之间的关系从来没有这么紧张过：一方面，我们激动于一个新的语言从 W3C 中出现；另一方面，我们还有一些发行于 Spice Girl 年代的浏览器。

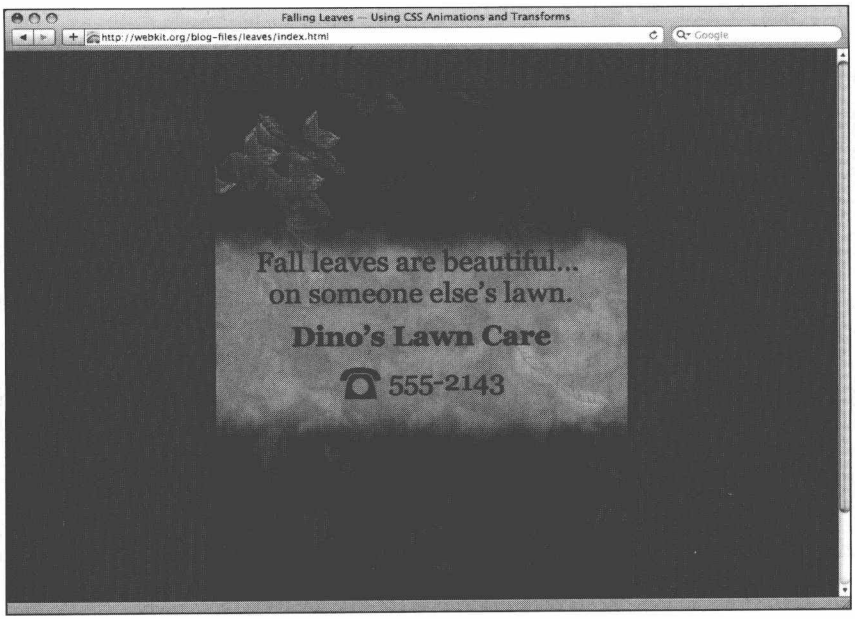


图 12.18　用最新版本的 Safari 访问 www.webkit.org/blog/324/css-animation-2，上面有 Apple 提出的 CSS 动画模型的信息（对，就是动画。）

即使你不用 CSS3 来做实验，IE6 仍然令人沮丧地需要进行支持。如果遗留版本的 IE 是你考虑要进行支持的浏览器，那么有以下几个不同的方法可以选择：

- **为遗留版本的 IE 提供一个可选的样式表。**通过条件注释的应用，为 IE 提供一个不同的完整的 CSS 文件是可能的，提供给用户一个不同的（降低了的）体验。就像在第 1 章中所提到的，Andy Clarke 已经开发了一套称之为“万能的 IE6 CSS”的样式表，它为 IE6 的用户提供了一个精美的版式体验，尽管它没有一些布局或装饰（可以在 www.stuffandnonsense.co.uk/content/demo/2009/05/21/forabeautifulweb.html 上查看一个例子）；

- **为 IE6 隐藏所有的样式表。**和 Andy 的方法相反的是，使用条件注释把我们的 CSS 针对 IE6 隐藏起来——不让样式应用到老旧的浏览器。CSS 专家 Dan Cederholm 最近发表了一篇博客来讨论这个方法（www.simplebits.com/notebook/2009/02/13/iegone.html），根据这个方法，可以设置如下代码：

```
<!--[if gte IE 7]><!-->
<link rel="stylesheet" type="text/css" href="screen.css"
media="screen, projection" />
<!--<![endif]-->
```

这段代码比平常的条件注释多了一点点，但结果是 IE6 及其以下版本会把它解析为一整段的 HTML 注释，从而跳过这个 link 元素；

- **用 JavaScript 来修复 IE6 中的 CSS 问题。**由 Dean Edwards 开发的，现在命名为容易误解的“ie7”的 JavaScript 库（code.google.com/p/ie7-js/）是在 IE7 浏览器之前发布的，它设计用来帮助老版本的 IE 成为恰当地兼容 CSS 的浏览器。简单地包含进这个 JS 文件，那些 IE5 和 IE6 中的 CSS 布局 bug 就会自动修复好；

- **什么都不做。**是的，什么都不做。不去管什么 IE6 的 hack 代码、条件注释，而是简单地把没有 hack 的代码提供给所有的浏览器。当然，在那些老旧的浏览器中会显示得不怎么好，因此，你也许需要确定你的客户同意这个策略。

你选择的策略应该是建立在数据和研究基础上的——最重要的是——要跟你的用户沟通。如果你的受众中有很大的比率通过 IE6 和 IE7 来访问，那么这个事实将帮助你决定采用哪个策略。（很有可能不是“什么都不做”。）最近的一篇博客文章中（blog.digg.com/?p=878），Digg 网站的用户体验小组列出了 IE6 不可忽视的使用率上的变化进程。IE6 的使用率虽然在下降，但是在网站的读者中仍然占有很大的部分，同样地，有很大一部分人仍然维持着他们的低预算（而不去升级更好的版本）。结果，他们通过反复考虑决定在 Digg 网站上发布一条升级浏览器的信息，建议那些 IE6 的用户考虑安装更好的浏览器。

然而，在发布这条信息前，Digg 网站决定询问他们的 IE 用户使用的是哪个版本的浏览器，以及为何使用。结果令调查者惊讶：仅仅有四分之一的用户是因为对 IE6 满意而选择使用它的。Digg 网站上的四分之三的 IE6 用户说，他们不能升级浏览器；总之，大多数 Digg 的 IE6 用户是因为他们别无选择而使用这个浏览器的。如果可以的话，他们会升级的。但是有些用户无法在他们的计算机上安装软件，可能是限于操作系统，也可能是受工作场所的约束。因此，再次重申：了解你的受众，然后进行相应的设计。

12.3 Flash 和 QuickTime：期望的对象

很多人在网站中插入 Flash 和 Quick Time 这样的多媒体对象。目前还没有一个遵循标准的方法来使这些对象在各种浏览器上可靠地工作。要理解这是为什么，我必须讲一个就像莎士比亚或意大利歌剧一样极富传奇剧情的傲慢与复仇的故事。好吧，就算不是传奇剧，但也很接近了。

12.3.1 可嵌入的对象：一个傲慢与复仇的故事

当 Mosaic 和 Netscape 浏览器的制造商最初想到可以让设计师在网页里插入图片时，他们为浏览器特别创建了一个 img 标签，借此“扩展”HTML。而 W3C 组织并不赞成这种做法。他们建议网页设计师采用 object 元素。但后来上百万的网站使用 img 标签，却没有网站支持 W3C 的 object 元素。

随后，FutureSplash 插件程序（后来被重新命名为 Flash）和其他诸如 Real、QuickTime 电影等多媒体元素也被插入网页。W3C 再次建议使用 object 标签把这些内容嵌入 Web 网页。但是 Netscape 又发明了 embed 元素——而当其他浏览器进入竞争舞台后，他们几乎都支持 Netscape 的 embed 元素。

在 Netscape 和微软看来，他们的顾客希望网络能成为一个丰富的多媒体空间，浏览器制造商在获得市场份额的过程中不是靠运气，而是必须靠革新来满足顾客的需求。

在 W3C 看来，浏览器制造商忙于创建他们自己的元素而不去使用已经非常完善的标准。如果 W3C 的会员公司都不理会开放的标准，那创建它们还有什么意义呢？在随后的几年里，W3C 组织对那些忽略其美好标准的人进行了一场血腥的双重报复。

（好吧，好吧，W3C 并没有进行一场血腥的双重报复或者做过类似的事。他们只是按照他们的章程来做了：建立有意义的标记标准。前面那种说法只是为了更好玩一点。）

12.3.2　W3C 的双重报复

W3C 的第一个复仇举动就是避免在任何官方的 HTML 规范中包含 embed 元素，尽管数十万设计师在数百万网站上都使用着 embed 元素。Embed 元素没有包含进 HTML3.2。它也没有被加进 HTML 4.0——或者加进 HTML 4.01（这个版本只有一个目的，就是增加一些 HTML4 中漏掉的常用标记）。由于 XHTML1.0 是建立在 HTML4.01 基础上的，所以 embed 仍然没有成为 XHTML 的一部分。因此，使用 embed 的网站就无法通过 HTML 或 XHTML 的校验。

是的，就是这样，你没有看错。数百万嵌入了媒体的网站都不能通过 W3C 的合法性校验，因为 W3C 从来就没想过要屈尊承认 embed 元素。

大概因为伤口还在疼痛，W3C 当初在现已废除的 XHTML2.0 规范中把那个可怜的 img 元素也清除出去了。你想要插入图片吗？请使用 object。你想要插入 Flash 吗？请使用 object。你想要插入 QuickTime 吗？请使用 object。W3C 就是这么说的。

问题是，有些现代浏览器对 object 的支持是不完全的，那些老式的浏览器甚至完全不支持。而设计师们却不能仅仅因为 W3C 的反对就不嵌入 Flash 或其他的多媒体内容。那么当一个信任 Web 标准的设计师需要嵌入多媒体的内容时该怎么办呢？好吧，她可以转换到 HTML5，那是支持 embed 的。或者如果她仍然在使用 HTML 4.01 或 XHTML 1.0，那么可以试试下面的方法。

12.3.3　折中方法：在支持标准的同时嵌入多媒体

在 2002 年 11 月，allinthehead.com 网站的 Drew McLellan 和 Web 标准项目组进行了一个试验。在给 A List Apart 在线杂志的一篇文章里，Drew 摒弃了在 Web 网页里嵌入 Flash 时常用的臃肿的不合法标记代码，取而代之的是简洁且遵循标准的 XHTML（http://www. alistapart.com/articles/flashsatay/）。他摒弃了所有不合法的 embed 元素，并把下面这个笨重的 IE 式的 html 代码

```
classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
```

改写成了兼容标记的 HTML，如下：

```
type="application/x-shockwave-flash"
```

你在 Flash 里发布一个影片时所生成的 HTML 代码，典型地会像下面这样：

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=6,0,0,0" width="400" height="300"
id="movie" align="">
<param name="movie" value="movie.swf">
<embed src="movie.swf" quality="high" width="400"
height="300" name="movie" align="" type="application/xshockwave-
flash" pluginspage="http://www.macromedia.com/go/
getflashplayer">
</object>
```

这个 HTML 非但臃肿而且不合法，但在那些安装了 Flash 插件程序的浏览器上都能工作。Drew 将它重新书写成简洁合法的 XHTML：

```
<object type="application/x-shockwave-flash" data="movie.swf"
width="400" height="300">
<param name="movie" value="movie.swf" />
</object>
```

Drew 的写法不但简洁合法，而且确实能在现代浏览器上播放 Flash 电影。但是由于一个众所周知的 bug，在 IE/Windows 上却不能连续地播放。如果 Flash 文件只有几 KB 的话，不能连续播放倒不是个大问题。但要是 Flash 文件很大，那问题就大了。

Drew 解决 IE/Windows 的连续播放问题办法是，用一个小 Flash 电影去下载一个更大电影。看来嵌入多媒体的适应标准的方法最终被发现了。Drew 的“Flash Satay”方法在每个浏览器和平台上都通过测试之后，A List Apart 发表了那篇文章。

12.3.4 美中不足：对象失效

遗憾的是，这篇文章发布以后，一个新的问题冒出来了。一些用户看不到嵌入的 Flash 内容，而是看到一个空白的文字区域。它主要出现在 IE/Windows 浏览器中，尽管大部分 IE 用户可以正常看到 Flash。类似的问题出现在 Linux 版本的 Konqueror（Safari 的前身）和 Mozilla 上。跟本章开头描述的 IE6 浮动边距双倍 bug 一样，object 的 bug 也影响了无数的用户。

12.3.5　使用一点 JavaScript

在实现嵌入 Flash 内容而又能通过 (X)HTML 合法性校验的努力中，一些设计师又试着使用 JavaScript 来侦测浏览器，然后用 document.write 方法偷偷地插入不合法的 embed 元素，从而通过 W3C 的合法性校验：

```
<!-- used to create valid xhtml with embed tag -->
<script type="text/javascript">
//<![CDATA[
if (navigator.mimeTypes && navigator.mimeTypes["application/
x-shockwave-flash"]){
document.write('<embed src="/media/yourflashmovie.swf" ...
```

这个技巧是有效的；所有支持 JavaScript 的浏览器都能正确显示 Flash（除非用户设置 JavaScript 为高风险并关闭了 JavaScript），W3C 的校验被愚弄了，认为你的页面标记都是合法的。但是，正如本章开头所言，我们不需要为了通过 XHTML 校验来赢得一枚金质勋章。欺骗校验服务和支持标准不是一回事。

开发社区已经有了一些既遵循标准又能可靠工作的方法。这些方法通常结合了 Drew 的 object 方式和一些基于标准的最佳实践，比如不唐突的 JavaScript（unobtrusive JavaScript，wwzw.onlinetools.org/articles/unobtrusivejavascript）和渐进增强（progressive Enhancement，www.hesketh.com/publications/ progressive_enhancement_ paving_way_for_ future.html）。

SWFObject 项目（code.google.com/p/swfobject）就是一个典范，用一段 DOM 脚本来提供多种对标准友好的方法嵌入 Flash，而不需要求助于过时的标记和属性。我们已经在大项目上使用这个方法，可以保证它的有效性和可靠性。除非所有的浏览器都支持 object，就像 W3C 所期望的那样——或者等到所有的浏览器都完全支持 HTML 5——那么我们就只能使用这类变通方法。

12.4　一个枯燥的变通办法世界

像 SWFObject 和条件注释这类变通办法帮助我们开始实现标准的理想（“一次创建，到处发布”），尽管实际上并没有浏览器完美地实现了标准，当然也有一些浏览

器更接近完美一点。这些变通办法使我们能够自由地改善网站的内容、设计和可用性，不用再把宝贵的时间浪费在那些已经过时的属性和技术上了。

但不是每个人都赞同这些变通办法，无论从实践上还是从它所带来的好处上。这些观点认为，如果浏览器不能完全地或正确地支持 W3C 规范，那么对于用户或者标准来说，就是最糟糕的，就应该简单地避免使用这个浏览器。这种观点有很多问题，包括以下这些：

- 如果你限定自己只使用那些所有浏览器都完全准确支持的规范，那根本就无法创建一个网页。即使是 HTML 3.2 也没有得到完全准确的支持；

- 实现标准需要花费时间——而时间在竞争激烈的市场上是很宝贵的。有时商业压力迫使工程师们在推敲每个规范的细微差别之前就将其发行了。偶尔他们不得不在明知有缺陷的情况下发行软件。（在本章中提及的那些 IE6 的诸如浮动和盒模型的 bug，其实它的工程师们都是知道的。）如果我们太追求纯粹，以至于不去使用变通办法，那么就会不必要地降低了我们用户的体验；

- 有时规范在某些地方也是含糊的，甚至在发布之后又被更改了。CSS2 就是一个例子，在 2002 年 CSS2 被修订为 CSS2.1，因为在原来的版本中有一些含糊的或是无法实现的规则。目标不断变化，团队就会迷茫。同样地，在 CSS3 中，我们已经看到浏览器厂商如何小心翼翼地实现那些不稳定的新特性。他们清楚地标明这些令人激动的新特性是实验性的，而并不承诺规范将会如此更改。毕竟，这规范也是在不断地改进的；

- 我们必须对那些老式浏览器的缺陷进行弥补，特别是如果这些有缺陷的浏览器仍然被广泛地使用着的情况下。

这些技巧本身是没有错的，甚至是恰当的；它们令我们现在就能使用标准，而不是只能等待将来完美浏览器的出现。随着浏览器的改善，以及老式浏览器的逐渐废弃，变通办法将会需要得越来越少，尽管我们可能还会用它们来保证网站在向前和向后兼容时看上去效果一样好。在第 13 章中，我们将结束这场针对浏览器的短暂且充满悲喜的讨论，而检查一个设计中最基本的方面——文字版式的控制。

第 13 章

深入浏览器之三：文字版式

和位置、色彩一样，文字版式也一直是最核心的设计工具。印刷设计师花了多年的时间来研究字体的历史和应用。他们学会了区分在外行看来是差不多相同的字体，比如字体 Arial 和 Helvetica（如图 13.1 所示）。他们花费数小时来调整标题字母的字距和对文本进行统稿，甚至跟作者一起为了适应布局而修改单词。当这些受过传统教育的设计师转到 Web 设计时，有限的工具和不可预测的输出效果，使得他们经常不如那些没有传统设计教育背景的设计师做得好。

在这一章，我们将讨论把字体应用在 Web 上的工作原则，并且仔细考虑文字版式在印刷上和在 Web 上的重要区别。我们将检视 CSS 中控制文字尺寸的工具，和它们在网站上控制文字版式层次的常见手段。我们将研究每一种设计师们苦苦寻找的 CSS 折衷方法，努力提交一个优美的布局，同时尊重用户的自由和避免可访问性的问题。我们还将看一下真正的 Web 字体所做的承诺。

图 13.1　Arial 和 Helvetica（细节对比）（www.ilovetypography.com/2007/10/06/arial-versushelvetica）

13.1 关于文字版式

不管是不是受过正规训练的设计师，任何人想要创建一个可用并有利于阅读的网页布局都应该遵循下面的这些指南。

1. 限制你自己只用两种（最多三种）字体，避免“分类广告”式的效果。除非你是一个字体库网站，否则不要在同一个页面上出现各种各样的字体。

2. 根据一个快速引导用户发现的机制来限制色彩的使用。比如，如果#c30 红色是未访问过的链接的颜色，那么就不该把这个颜色用到非链接的文字上，也不要再设定一个绿色的未访问过的链接，阅读者很可能会被迷惑，从而错过了重要的链接——或者令人沮丧地点到一个看起来像链接的文字。从美学上讲似乎也不能忍受大量的色彩。很少有设计师能够违背指南 1 和 2（如图 13.2 所示），除非你具有他们的技巧，并且小心处理。

3. 字体大小要匹配分栏的宽度，建立一个适合阅读的尺度（www.webtypography.net/Rhythm_and_Proportion/Horizontal_Motion/2.1.2）。这个尺度就是每行文本所容纳的字母的数量。太少了，眼睛会疲劳；太多了，读者可能会看丢了（www.markboulton.co.uk/journal/comments/five_simple_steps_to_better_typography）。用 serif 类字体的话，单个的文本栏有 45 到 75 个字母是比较合适的；多栏的话，40 到 50 个字母更合适。建立一个合适的尺度，在固定宽度的布局上比较简单，而在一个自适应的布局上很棘手（www.maxdesign.com.au/presentation/liquid）。给内容栏设置一个 min-width 和 max-width 会很有用。关于自适应设计的技巧，请查阅 www.alistapart.com/articles/fluidgrids。

4. 当设计多栏页面时，可以使用网格来实现美观和视觉上的连贯和多样化（www.markboulton.co.uk/articles/detail/five_simple_steps_to_designing_grid_systems）。基于网格的布局设计案例有很多，包括 NYTimes(www.nytimes.com)、Silnt(www.silnt.com/v4)、设计师 Jason Santa Maria 的网站(www.jasonsantamaria.com)、Times Online(www.timesonline.co.uk/tol/news)、An Event Apart(www.aneventapart.com) 和 Khoi Vinh 的 Subtraction 站点(www.subtraction.com)，还有很多安全地使用网格来布局的站点。帮助网站进行基

于网格布局的要素是幕后一个叫做“CSS 框架”的技术，比如放在 Google 站点上的 Blueprint 项目（code.google.com/p/blueprintcss）。不过，我在这里只是随便说说，那些不用第三方的框架的 Web 设计师们和前端开发者们包括 Eric Meyer、Dan Cederholm、Ethan Marcotte、Aaron Gustafson、Jason Santa Maria、Cameron Moll，还有我。这并不是因为我们炫耀，这是因为最好的 CSS 经常是产生于你自己的设计——就像最好的设计来自于你的内容、产品和用途。

图 13.2　Jason Santa Maria 在 Happy Cog 为 AIGA 设计的站点，打破了字体的限制和色彩运用的简洁机制。如果你是天才，那么也可以这么做，否则就不要这么做（www.aiga.org）

5. 为鼓励阅读而进行设计（www.alistapart.com/articles/indefenseofreaders）。扩大段落之间的距离、摘出引文、首字母大写和另外一些书籍杂志的设计技巧都可以很容易地加到你的站点中，只需要很少的 CSS 和语义性标记。提供一个熟悉友好的阅读环境，就可以引导视线，顺应思路和帮助理解。

6. 最后也是最少的，为字体设置基线网格可以为你的 Web 布局带来额外的节奏和协调（www.alistapart.com/articles/settingtypeontheweb）。

学习资源

本书无法教你用字体传达设计的精妙艺术和微妙科学——仅仅只能感受一下——但是这里有两个经典的宝贵学习工具：

Ellen Lupton 写的“*Thinking with Type:A Critical Guide for Designers*”（Princeton Architectural 出版社 2004 年出版）（www.papress.com/thinkingwithtype）

Robert Bringhurst 写的“*The Elements of Typegraphic Style，2nd Edition*”（Hartley & Marks 出版社 2002 年出版）。

每一个设计师都应该读读 Lupton 和 Bringhurst 的著作，但是我们如何把他们所教的技巧应用到 Web 设计上呢？The Elements of Typographic Style Applied to the Web 网站（www.webtypography.net）和 A List Apart 网站（www.alistapart.com/topics/design/typography）可以帮助你。也可以看看 Mark Boulton 的 A Practical Guide to Designing for the Web（www.fivesimplesteps.co.uk）。

如果想进一步学习字体知识，可以看看 Ilene Strizver 的“Top Ten Type Resources Online”网站上推荐的一些站点（www.creativepro.com/blog/typetalk-top-ten-type-resources-online），还有我推荐的一些：

- ilovetypography (www.ilovetypography.com)；
- Typophile (www.typophile.com)；
- Typographica (www.typographica.org)；
- TypeCulture (www.typeculture.com)；
- Type Directors Club Books on Typography (www.tdc.org/tdc/resources)；
- TheFontFeed (www.fontfeed.com)；
- The Elements of Typographic Style Applied to the Web (www.webtypography.net)；
- TypeRadio (www.typeradio.org)；
- A List Apart (www.alistapart.com/articles)；
- Twitter (www.twitter.com) (在上面寻找新的字体资源)。

13.2　Web 字体的 A-B-C

在开始崭新的令人兴奋的 Web 文字版式设计之前，让我们先了解一下基础知识。

平台和浏览器，光栅和行参差

Windows、UNIX 和 Macintosh 操作系统上安装着不同的字体集，有着不同的分辨率，还常常有不同的默认渲染样式——传统 PC 上的光栅化是使用亚像素抗锯齿的方式，Mac OS X 上默认的是 Quartz 图形模式（如图 13.3 所示），而 Windows XP+可以让用户切换到 Cleartype（清晰）选项。

字体设计师们一直在争论到底是 Quartz 模式还是 Cleartype 模式在屏幕上的效果更好，更接近印刷字体的效果。大家一致认为 Mac OS 的 Quartz 光栅模式令人信服地更接近印刷文本的效果，因为它在字体的间距和上下突出部分的处理上，以及其他地方都更好一点，但是 Windows 的 Cleartype 字体集在文字很小的时候更容易阅读。字体设计师们，比如 Font Bureau 公司的 David Berlow，认为要在 Web 上产生合适的字体只有一个办法，就是根据传统的字体进行重新设计，以便更好地用屏幕的像素点来校正它们的边缘——他和 Roger Black 已经把这个理论应用到 Palm Pre 操作系统和一些网站（www.mit.edu）的设计实践中了。

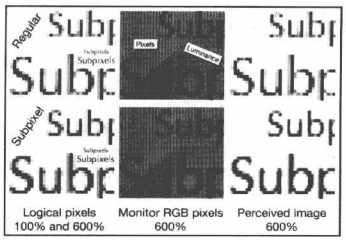


图 13.3　Mac OS Quartz 模式下的亚像素抗锯齿效果，放大到 600%，显示出边缘的色彩是如何偏移，来实现一种平滑的类似印刷字体的效果（commons.wikimedia.org/wiki/File:Subpixel_demonstration_(Quartz).png）

不管哪个字体集和光栅化的方法是“最好的”，它们都影响着页面布局，并且设计师们无法进行完全控制。这才是要点。

另外，Windows 上的 Lucida Sans 字体和 Mac OS 上的 Lucida Grande 字体不仅不相同——当然 Mac 上的很多字体都不能移植到 Windows 上，反之亦然——而且两者处理字体的方式也不同。因此，以前的 font size=3 在不同的操作系统里指定的尺寸和外观都不一样。而且，即使在同一个平台上，每个浏览器对字母间距的处理都有差别。Safari 会根据连字符折行，而 Firefox 则忽略它（如图 13.4、图 13.5 所示）。所以相同平台相同布局中的相同文本在不同的浏览器中会有不同的“行参差”。（“行参差”是指不整齐的文本右侧边，就是打印时设置为“左对齐”的效果，这是大多数 Web 上的对齐方式）。虽然，有些人迷恋于为了布局整齐而安排合适的单词，但是我们却无法保证在所有的环境下都能精确地跟印刷一样——来满足用户的喜好。

"We've got mail!" the old site cheerfully announces, complete with a meaningless header image. The image, like the header and navigation typography, is pixellated to convey "webbiness"—in case you forgot that you were looking at a website in a browser, I guess. "Got mail" is a play on America On-line (kids, ask your parents). "We" is the royal first person plural with which I used to write this site, despite being its sole author. I'd gotten into the habit of "we" from writing copy on entertainment sites for clients like Warner Bros. It made their sites, and mine, seem bigger. It was also an ongoing, self-deprecating joke, although not everyone got it.

图 13.4　Safari 会根据连字符折行（www.zeldman.com/2009/08/05/past-blast）

"We've got mail!" the old site cheerfully announces, complete with a meaningless header image. The image, like the header and navigation typography, is pixellated to convey "webbiness"—in case you forgot that you were looking at a website in a browser, I guess. "Got mail" is a play on America On-line (kids, ask your parents). "We" is the royal first person plural with which I used to write this site, despite being its sole author. I'd gotten into the habit of "we" from writing copy on entertainment sites for clients like Warner Bros. It made their sites, and mine, seem bigger. It was also an ongoing, self-deprecating joke, although not everyone got it.

图 13.5　Firefox 却不会折行。因此即使在相同的平台上，这篇博客文章的右边行参差在不同浏览器中都不同。因此，上帝戏弄了那些为了布局整齐而安排合适单词的传统艺术家

用户控制中的不能承受之轻

除了平台和浏览器之间的不同，Web 设计和印刷设计之间还有一个区别，跟书籍杂志不同的是，一个网页在视觉上的效果可以由用户浏览器的组件、偏好设置和

插件来调整。做出适应不同用户的偏好设置，且保持一致的设计是非常棘手的。对于受过传统设计训练的设计师来说，要接受用户控制这个前提很困难。有些站点的构建者可能并没有意识到普通用户的喜好变化。因此有着印刷设计背景的设计师常常把字体设得太小，阅读起来不舒服。（你可以在前面提到的“Top Ten Type Resources Online”上看到有几个关于这方面的资源。）在本章的后面，我们将涉及字体尺寸的方面，考虑传统及新出现的最佳实践。

13.2.1　Web 字体的简短历史

互联网的发明者和 W3C 组织的创始人 Tim Berners-Lee，认为他所发明的是一种进行简便文档交流的媒介；因此他在 HTML 语言结构里没有包含文字版式的控制。就像在第 2 章里所说的，Web 设计师们起初是用 tt、pre、blockquote 和一些非语义性的段落标签来模拟各种文字版式、实现定位效果的。后来他们还使用 Photoshop 制作 GIF 图片来实现各种 Photoshop 中的字体集，这种方法现在仍在延续使用。

到了 1995 年，随着一些商业站点雨后春笋般地到处出现，设计师们使用各种 HTML 的 hack 技巧，因此必须要提供一些至少是基本的文字版式控制工具。Netscape 给我们提供了 font 元素，它具有 size 属性。你可以指定一个特定的值（font size=2），或者指定一个基于用户默认值的相对值（font size=+1）。设计师们很快就放弃使用段落和其他的结构元素，而转变为结合使用 font size 和换行元素（br）来控制他们的布局。为了不被淘汰，Microsoft 给我们提供了 font face。

本书的一些读者可能还记得那些日子，那时的主要问题就是这些平台的差异。Windows 假定在 96ppi（每英寸的像素数）下默认的基本尺寸是 16px。Mac OS 更接近印刷设计，基于 PostScript 标准，假定在 72ppi 下默认尺寸是 12px。因此，字体尺寸在一个平台上正好，但在另一个平台上就显得太大或太小。

“Windows 太笨了。”基于 Mac 的设计师说。

“Macintosh 太笨了。”基于 Windows 的设计师说。

差异点

接着，早期的 CSS 实现比如 IE3 出现了，把这个跨平台问题进一步扩大了。磅

值（pts）是一个印刷单位，而不是屏幕单位，但是设计师们都熟悉磅值，很多人使用这个单位来设置他们 Web 上的文字大小。在 Windows 世界，7pt 字型意味着有 9px 高，它是指文字易读部分的高度。而在 Mac 上，7pt 字型就是 7px 高，这就使得这个设定变得含糊不清，从而没法用了。

在 1997 年，Microsoft.com 网站使用了 7pt 的字型来宣扬他们新的 Windows / Macintosh IE3 浏览器的 CSS 功能。这就像邀请人们参加电影首映，以使人们在上映前聚焦在影片上。这个字型在 Macintosh 上的 IE 和 Netscape 中同样是含糊不清的，这不是浏览器的问题，而是由于平台的差异。后来提出 DOCTYPE 转换方案（查阅第 11 章）的 Todd Fahrner 曾在个人网站上发布了一张图示（如图 13.6 所示），上面所标示的注解指出了在 CSS 中使用磅值单位在屏幕的设计上是没有效果的，虽然在打印样式表里使用效果是不错的。

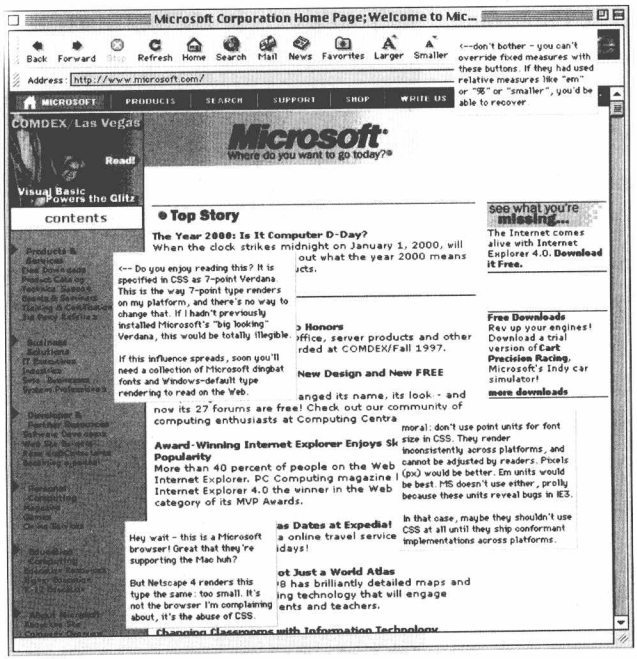


图 13.6 在上世纪 90 年代中期，Todd Fahrner 指出了在截图中 CSS 使用字体磅值时，屏幕显示上出现的问题（style.cleverchimp.com/font_size/points/font_wars.GIF）

Fahrner 还指出（在当时）文字的磅值和像素值无法通过浏览器内置的字体大小

调节组件来调整。这并不是因为 CSS1 采取了固定的磅值和像素单位。（CSS1 并没有定义一些不能被调整的字体尺寸设置方法。）而是一个首次支持 CSS 的浏览器制造商所做出的简单决定。用户可以用 em 和百分比来重新设置字体，但是 IE3、IE4 和 Netscape 4 在使用 em 或百分比设置时都碰到了 bug，这使得在当时用 em 和百分比来设计是不可靠的。Fahrner 后来对此也提出了一个方法，解决了部分问题。

当时 Fahrner 提出，唯一能够在跨浏览器和跨平台中都保持一致的尺寸单位，就是不能被浏览器调整大小的像素单位。10 年之后，这唯一完全可靠的单位仍然是像素。（IE/Windows 仍然阻止用户通过文本缩放功能来调整文字的尺寸。然而普通的页面缩放功能是可以达到类似效果的，在本章的后面，我们将会讨论文字的像素问题。）

13.2.2　最终的标准尺寸

为了克服平台差异带来的影响，浏览器厂商 Netscape、Microsoft 和 Mozilla 在 1999 年的晚些时候，一起决定把默认的字体大小跨平台地标准化为 16px/96ppi。把这个标准应用到同一页面的所有的平台上以后，浏览器厂商和用户就可以避免字体尺寸及文字清晰度上的跨平台问题。

字体尺寸标准 16px/96ppi

在 1998 年的 W3C 邮件列表中，勇敢的 Todd Fahrner 提出应该把 Windows 的默认值 16px 作为标准。所有领先的浏览器厂商都在 2000 年采用了他的建议。虽然 Fahrner 所关注的是现实性（他想保证字型在网上阅读时的效果），但是在随后的引述中，他提到的一些事情可能会让你觉得奇怪。

CSS1 的制定者抽象出一个概念，认为 Web 用户手臂的“平均”长度对于定义尺寸的像素值是很重要的，而实际上并不是这样的。虽然 W3C 为我们带来了很多有用的 Web 标准，但是它的成员公司们有时却遵循着自己的学术理念，从而产生了偏差。不管怎么说，Fahrner 关于字体尺寸的跨平台标准化的理念是先进的，他小心翼翼地把更普遍的可用性和可辨别性的要求，连同手臂长度问题一起进行了综合考虑。他写道：

在 Mosaic 之前，所有主流的浏览器中默认的字体大小都是 12pt。我建议把默认值重新定义为 16px…当前的默认值 12pt 在不同平台上光栅化有很大的差异。在 Mac

上，它被光栅化为 12px（按照 72ppi 来修正）。而在 Wintel 类的 PC 上，它被光栅化为 16px（按照 96ppi 来修正）…所有可缩放的字体大小的值…都根据其不一致光栅效果来操作。对于一个设计师来说，这意味着为了保持字体大小在跨平台时的一致性，唯一可用的方法就是在 CSS 中使用像素单位，而这样用户就不能进行缩放了，因此也就无法对用户的友好性和便利性进行优化了……

这合适的纠正度量…是打破 Mac（X11 版？）浏览器的传统，而用“中等” 字体集的尺寸值 16px 来代替 12pt。当然应该保留用户调节的需求，但是一个一致的初始值至少能为设计师在使用可缩放字体尺寸值时减少一点问题，一些从默认值衍生出来的变化将是由于用户偏好的设置、而不是变化无常的操作系统差异引起的。

如果设计师们觉得把 16px 作为基准太大了，那么为什么还把它作为默认值呢？

原因之一是，这纯粹是权宜之计。Mac 相对来说使用的人还是比较少的，虽然它在 Web 设计领域用得非常广泛。（我用的就是 Mac！）而期望让 Windows/X11 的浏览器改变其默认值来匹配 Mac 那有局限的 72ppi 的逻辑分辨率，是不现实的。

1996 年 CSS1 标准建议使用 1/90 英寸的值来作为一个“像素的参考值”，这是根据手臂长度所决定的 0.0227 度视角来推断出来的值。如果物理分辨率能在这个值上出现显著差异的话，就能期望浏览器绘制出合适的像素。然而，一个 1/90 英寸的参考像素是建议把 12pt 光栅化为 15px、而不是 16px。当然，15px 比 12px 更接近 16px。因为 OS/UA 当前并没有假定为 90ppi 的逻辑分辨率（也没有实现像素的缩放）…这个参考像素的值应该被改为 1/96 英寸。这是简单地参考了一个更长的手臂长度，来保持 0.0227 的视角。

设计师们简单地认为 16px 太大了，这是由于他们使用的是 Mac 上 12px 的基本尺寸。而易读性是用户友好性中的重要部分。

lists.w3.org/Archives/Public/www-style/1998Dec/0030.html

13.2.3 手臂和像素

两年后，第一代兼容标准的浏览器相当大的程度上接受了 Fahrner 的建议。在

Windows 和 Macintosh 平台上的 Internet Explorer、Netscape 和 Mozilla 中，默认的文字尺寸都变为了 16px/96ppi，增强了易辨性，受到了全世界的喜欢——除了一些用户自己把它改回去之外，这将在后面讨论。

Fahrner 的努力最终导致 W3C 同意在 CSS2.1 中把标准的像素参考尺寸定为 16px/96ppi。在下面的摘录中，你将注意到 W3C 仍然保留着用户平均臂长的概念：

> 一个参考像素的建议大小应该是根据一个具有 96dpi 像素密度的装置与其距离读者的手臂长度所形成的视角推算出来的。一个正常的臂长是 28 英寸，因此视角大约在 0.0213 度。
>
> —CSS 2.1 Working Draft, www.w3.org/TR/CSS21/syndata.html#length-units

在 W3C 标准中，用户默认的像素大小似乎已经清楚了，但是第二点没有涉及到。W3C 说了第一个问题，但是没有说第二个问题。

Netscape 6+/Mozilla（Firefox 的前身）、IE5+/Macintosh 和 IE/Windows 给所有的用户提供了相同的默认字体大小。磅值、em 值、百分比值和一些字体尺寸的关键词在不同平台上都将以相同的方式工作（只要用户没有改变她的偏好设置），再也没有用户会由于设计师或开发者对平台差异的无知而产生不必要的痛苦。在一些浏览器中，对于百分比和 em 值的实现仍然有 bug 和继承问题，但是主要的可访问性和可用性的障碍已经清除了。

有标准的像素参考，却没有标准的磅值参考

浏览器厂商无法解释他们为什么要这么做，用户为什么不去研究这个设计问题（但他们为什么要研究呢？）。标准尺寸的好处暂时让一些用户（和一些设计师）忽略了文字的磅值。

举个例子，一个不知道百分比的 Macintosh 用户，他厌恶所看到的文字尺寸，觉得又大又丑，他马上把浏览器切换为原来的 12px/72ppi 的设置，因此标准化在这里失败了，而再次使他们自己无法看清许多网站上的文字。在 Web 上，用户被假定为是有自主权的，即使他自己并不知道什么是他的最佳利益。

当这些“切换回 12px”的用户正巧是设计师的时候，他们也许会把站点制作得在 Macintosh 上看着很棒，而在处于主流地位的 Windows 上却显得不怎么样。许多基于 Macintosh 的设计师被错误地引导至使用 12px/72ppi，他们的工作（或他们的受众）遭受着这个结果所带来的痛苦。感谢本书的第 1 版和第 2 版的广泛传播，目前只有很少的基于 Mac 的设计师会犯这个错误了。（这也是本书继续讨论这些 Web 历史的原因之一。）

当然，一些 Windows 用户也会发现默认的 16px 太大了，因此把他们浏览器的文字尺寸设置得比标准的小，这在浏览所有网站的时候都生效。当 Mac 和 Windows 用户这样做时，这不会影响那些用像素来设定文字尺寸的网站，但是它会使那些使用 em 值、百分比值和（在较低程度上使用的）CSS 字体尺寸关键词的网站产生问题。我们将在本章后面研究这些难题。

13.2.4 无声无息地被遗忘

人们切换到比标准字体更小的尺寸仅仅是为了看起来更舒适。他们忽略了标准尺寸的磅值，而且我们也不应该期望他们能理解它。很多从事 Web 职业的人也忽略了磅值，部分的原因是 Netscape 和 Microsoft 没有宣传它。

特别是，许多开发者（我们称他们为“浏览器怪异军团”）开始相信浏览器在表现和行为上总是有很大不同的，解决这些差异的办法就是运用代码和标签技巧来进行浏览器侦测。他们迅速地按照所训练的方法去做：浏览器检测代码分支。

在 2000 年之前，这些开发者们使用了磅值，而不是为了文本在不同浏览器和平台上的尺寸都相同而使用像素值。因为（a）磅值就屏幕而言是无意义的，（b）两大主流的计算机平台对磅值的实现方法有着很大的不同，这些开发者建立了多个磅值驱动的样式表，使用 JavaScript 来做平台侦测，使一个磅值样式表应用于 Windows 用户，另一个应用于 Macintosh 用户。但是，这经常使事情变得更加复杂了。

贪吃蛇吐出了它自己吞下去的尾巴：有条件的 CSS

到了 2000 年，随着遵循标准的浏览器的发行，它们都是跨平台支持相同的默认字体尺寸和分辨率的，Web 设计师们终于有机会摈弃浏览器侦测和磅值驱动的有条件的 CSS 了。但是，“浏览器怪异军团”反而升级了他们的脚本和额外增加了有条件的

CSS 文件。“额外的条件”听起来好像很有趣，不是吗？但是，这结果却不一定有趣。

事与愿违的是，这些扩充的脚本和条件文件往往不能实现他们的目的，反而造成含糊不清或版式奇怪的页面。创造出这些违背可用性的梦魇的开发者并不愚蠢。通常，他们都是一些有着很高技巧的专业开发者。他们的客户们为了这些并无必要如此复杂的网站花费很多的钱，却得不到正确的结果，而且需要持续昂贵的维护，这些更复杂的却不一定成功的方法带来了进一步的花费。这听起来似乎是很不理智的愿望，但是这却是正常情况（一些大公司的网站仍然是这样的）。当然，代理商可以要求不知情的客户为了这额外的持续昂贵的维护费用买单，但这可不是一个好的赚钱方法。

而这是无法持续下去的。最终，咖啡喝完了，预算耗尽了，网站所有人和受众都将面对一个过期的网站，就像在第 1 章中所描述的那样。为了预先防止上面所描述的不可避免的结果，有些设计师在他们的页面上方放置了一个横幅“最好使用某某浏览器浏览”。这就像在香烟的包装上贴着卫生局的警告，期望它能使吸烟的人转而食用一罐维生素一样。

如果我们知道浏览器是支持通用标准的，就可以阻止这种混乱的发生，往往最简单的方法就是最好的方法。（那就是对所有的浏览器都应用同一个样式表，并避免使用磅值，除了为打印准备的样式。）

在离开这一小节以前，让我们注意一下，一个不同类型的条件 CSS 有时不仅仅是可接受的，而且还是必须的，特别是当你需要支持过时版本 IE 的诡异布局和行为时，例子请查看 17 章。

13.3　字体尺寸的冒险之旅

在本书第 1 版的第 13 章，我花费了大量的篇幅来抱怨非标准尺寸和不可缩放的像素之间糟糕透顶的相互作用。

回顾以前，有一个浏览器厂商（其实就是 Apple）故意地偏离新的 16px/96ppi 的标准，它认为这又大又难看。由于大多数用户从来不会去改变默认的字体大小，

因此，如果 em 值或百分比值被用在字体尺寸上，那么 Apple 浏览器的用户就会跟别人看到的文字大小不一样。当看到这种情况发生时，设计师和客户们都会紧张不安。而且，那时几乎所有的浏览器对百分比值和 em 值的实现通常都比较拙劣。

相比而言，用像素来指定文字大小是比较直接的——可以简单直观地反映出数字大小之间的关系，比如，用 12px 指定 body 的内容，用 18px 指定行高，用 24px 指定一个标题。这就是创建垂直网格来匹配水平网格布局的做法。当然这种比例关系也能通过 em 值和百分比值来建立，但是算起来就比较麻烦了，而且结果也不容易预测。

还有，除非你也使用 em 值来定义图片的尺寸，否则将无法在视觉上让图片尺寸、标题尺寸和分栏尺寸等协调一致。你可以用 em 值来设置图片大小，而且使用 em 值来设定每个元素的大小，这是得到可伸缩页面的黄金通道，关于这点请查阅文章“every element scales along with the user’ stext size”(www.alistapart.com/articles/elastic)。Patrick Griffiths 在 CSS Zen Garden(www.csszengarden.com/?cssfile=/063/063.css&page=0)网站上发表了一篇文章叫“Elastic Lawn”，来宣传这种可伸缩的布局；America On-Line 网站是首先实现这种技术的主流网站。虽然一些不太好的浏览器可以单独处理，但是，并不是所有的浏览器和操作系统都能很好地缩放图片尺寸的（参见 www.unstoppablerobotninja.com/entry/fluid-images）。不管怎么样，“整体页面缩放”功能可以实现类似效果，它把所有的布局都看作是可缩放的。我们马上就会进一步讨论这一点。

像素带来的麻烦

使用像素单位所带来的麻烦传统上是由这个问题引起的：在 IE/Windows 中用户不能缩放设置为像素值的文字。如果你把 body 的文字设定为 9px，那么很多浏览者都会马上按浏览器的后退按钮。即使是用 11px，有时也显得太小，根据所选用的字体（比如，11px 的 Verdana 字体和 Georgia 字体比 11px 的 Timer 字体会显得好一点）、显示器的大小和分辨率、浏览者的视力、前景和背景的对比，以及是否存在混乱的背景。对于视力低于 20/20 的人，这个问题也许只是有点困扰，而对于视力严重受损的人来说，就要糟糕得多。Wilson Miner 有一篇很好的文章，关于适当大小的 Web 字体的重要性：“www.wilsonminer.com/posts/2008/oct/20/relative-readability”。

偶尔还会有 CAD 工程师在他的 4000x3000 分辨率的 32 英寸工作站显示器上浏览网页，那些蝇头小字惹得他怒气冲冲地写信给 Web 设计师。实际上如果他希望解决这个问题的话，可以使用浏览器提供的文本缩放或页面缩放功能。如果他喜欢使用 IE6，他可以打开忽略字体尺寸的选项，甚至还可以写一个自己的样式表，就像这样：

```
html,
body {
    font-size: 1em !important;
}
```

但是他宁愿抱怨，也不愿意用这些方法去解决问题，因为对他来说，这整个事情就是一个宗教问题。（他也抱怨所订的杂志上充满了空洞的名人八卦和他们的婚姻问题。他也不喜欢花生，但是他还是在吃它们。）作为一个设计师，你就需要对用户的问题负责——即使是这样一个家伙。

这就是我们的窘境。作为一个跨平台保持设计实现的一致性和准确性的传递机制来说，使用像素值要比 em 值和百分比值更好一些。它们也使得在计算上及视觉上和谐布局变得更容易。但是在 Internet Explorer 浏览器中，无法对基于像素的文字进行缩放，会对很多读者造成很多的问题。为此，在关心可访问性的基于标准的设计师中，有很长一段时间的最佳实践是，使用 em 值或百分比值而不是用像素值来设定字体尺寸。这个最佳实践会像钻石一样永恒吗？

长久以来，就像永恒不变的法律一样，世界已经接受了 Microsoft 的决定，不允许用户缩放用像素设定的文字。我的一些 WaSP（Web 标准项目组）的同事和我却有不同的感触。成功地说服浏览器厂商支持标准的字体尺寸，先是 IE5/Mac，再是 Mozilla/Firefox，然后是 Safari，欢呼它们通过文本缩放功能来提供给用户缩放文字的能力（包括像素设定的文字），我们中的一些人还艰难地游说了近十年，让 IE/Windows 也实现文本缩放功能。可是令人叹息的是，它并没有成为现实。

13.3.1　页面缩放功能：使像素更安全

但是有些事情确实发生了。Internet Explorer 7 已经包含了页面缩放功能（一个 Opera 开创了很久的功能）。使用页面缩放功能，用户不是放大和缩小文字的尺寸，而是放大和缩小整个布局（如图 13.7、图 13.8、图 13.9 所示）。或许这是一个潮流，

也或许是那些高级的标准专家们已经厌烦了绞尽脑汁地为了保持复杂页面布局而阻止文本缩放。不管是什么原因，在 2009 年，Firefox 3.5 和 Safari 4.0 都更偏爱页面缩放功能（如图 13.10、图 13.11 所示）。

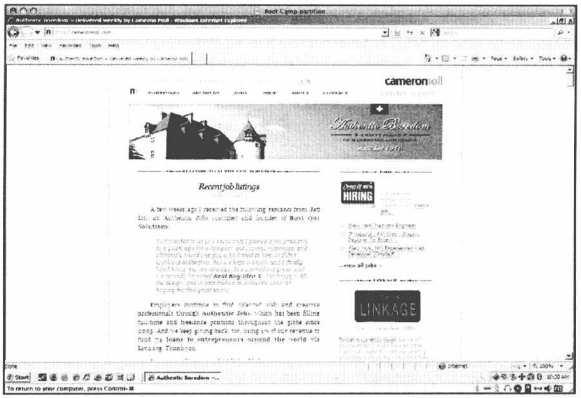


图 13.7　设计者 Cameron Moll 的网站在 IE8 中的效果（www.cameronmoll.com）

图 13.8　相同的站点使用了页面缩放功能

当用户调整文字尺寸是破坏页面设计的主要原因时，整体页面缩放功能能够大大简化书写稳固布局的复杂度。使用页面缩放，可以保持你的复杂布局——包括图片尺寸、标题尺寸、分栏宽度和行高之间的精妙数字关系——把最终的文字尺寸控制交给你的用户。尽管浏览器厂商在实现页面缩放功能时，可能并不是为了缩放像素值文字，可是随着 IE6 的日渐黯淡，而页面缩放功能却变成了标准功能，设计师们通过 px 来设定文字大小也不再会被系上羞耻的石头了。

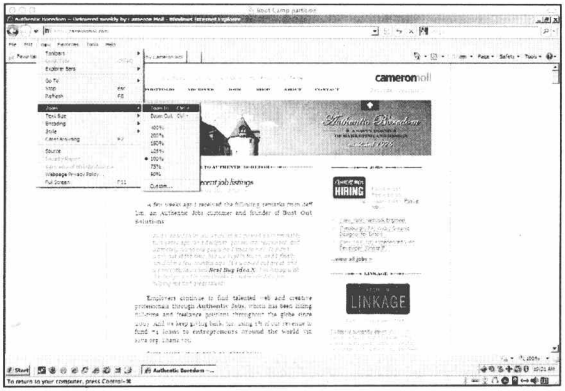


图 13.9　IE8 中的页面缩放功能组件

图 13.10　Happycog.com 网站在 Safari 4 中的效果

图 13.11　相同的站点使用了页面缩放功能

多久以后你才能转换到 px 方式上呢？据我看来，当你的参考日志告诉你，你的多数受众都开始使用 IE7 或更高版本时，就可以使用这个方式了。你不需要关心有多少用户在使用 Safari、Firefox 和 Opera，因为像素值文字在那些浏览器中总是能被用户用某种方式缩放的。IE，经常就是我们唯一的问题所在；而 IE6 当我写本书时，已经开始慢慢走向灭亡了。

13.3.2 使用 em 值来设定尺寸：欢笑和眼泪

由于在 IE7 之前的版本中像素值文字有缩放的问题，所以可访问性的拥护者和基于标准的设计师们——更不用说 CSS1 的创建者们——长久以来一直认为 em 值就是未来之路。通常表达出来的观点是沿着这些思路的："对于可访问性，所有的 font-size 都应该被表示为相对的单位，而不是像素。"设计师们表达这类观点和建议请求并不罕见。

62.5%的解决方案

在 2004 年，开发者 Richard Rutter 提出一个 62.5%的解决方案，来处理基于 em 值设计中复杂的数学继承问题（www.clagnut.com/blog/348）。默认尺寸 16px 的 62.5%是 10px。设定 body 为 62.5%的话，其他的计算就容易了（因为是基于 10 的）：

```
body { font-size: 62.5%; }
p { font-size: 1.2em; line-height: 1.5; }
h1 { font-size: 2.4em; }
```

在上面的例子中，p 的文字是 12px 高（1.2 x 10px），它的行高是 15px（1.5 x 10px），h1 标题的文字高 24px（2.4 x 10px）。非常容易计算！

使用 Rutter 的方法，你可以获得完美的接近像素值的控制，而且避免了 IE 中不能缩放像素值的问题。Web 设计师们马上采用了 Rutter 这个杰出的主意，它变成了一个最佳实践。

但是 62.5%的方法并不完美。如果用户缩小了默认尺寸，那就没那么幸运了——那是 em 值产生的问题——通常是基于设计上的。当用户缩小了浏览器的默认尺寸，或者当 IE/Windows 的用户设置他的浏览器为：文字大小菜单下的"小"，而不是"中"

的时候，这个方法就会失效。这样的改变使得一些低于 1em 大小的文字比期望的还要小，以至于不适合阅读（甚至是看不清的。）Rutter 后来改进了他的技术，近乎完美地协调了各方面的需求：用户按需要调节文字的需求、设计师追求准确性的需求，以及由用户对默认字体大小的控制所带来的风险（如图 13.12 所示）。我建议你去读一下他写的这篇文章“How to Size Text in CSS”（www.alistapart.com/d/ howtosizetextincss），然后自己决定下次页面布局时是采用 Rutter 改进后的基于 em 值的技术，还是用像素值布局的方式。

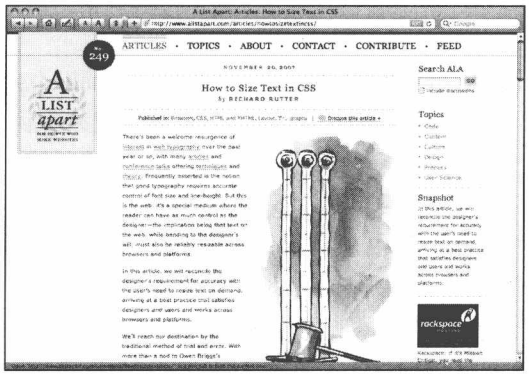


图 13.12　在文章“How to Size Text in CSS”（www.alistapart.com/articles/howtosizetextincss）中，Richard Rutter 改进了他原先的开拓工作。具有讽刺意味的是，一些 A List Apart 站点的读者批评作者没有拥护 62.5%规则——其实他们没有意识到他就是这条规则的发起人

另一种控制文字尺寸的方法值得在这里讨论一下。

13.3.3　使用字体尺寸关键词的方法

不怎么知名，也很少被使用的是，CSS1（和后来的 CSS2）提供了七个字体尺寸的关键词来控制文字大小，期望能解决像素值的绝对性问题、继承问题、跨平台的问题和使用 em 值和百分比值所带来的用户控制的风险。这七个关键词依次出现，买过 T 恤的人都能理解它们的意思（www.w3.org/TR/CSS21/fonts.html#font-size-props）：

```
xx-small
x-small
small
medium
large
x-large
xx-large
```

可爱的关键词

当你使用 em 值或百分比值时，总会有产生数值累乘的危险，结果就是文字太小了或者太大了。相对而言，关键词的值不会累乘，即使元素是嵌套着的。如果 body、div 和 p 元素都设置为 small，而 p 在 div 内，div 又在 body 内，那么这三个 small 并不会累乘（而 em 值和百分比值就会累乘），因此，最终的文本仍然是清晰可阅读的。而且结果仍然是 small（而不是 x-small 或者 xx-small）。百分比值和 em 值是累乘的。关键词的值不是累乘的。

此外，至少在 Firefox 和现代的 IE 浏览器中，xx-small 不会小于 9px，因此不会看不清楚。虽然 9px 的文字可能对某些用户来说阅读起来很困难，但是却并不是看不清的。

跟 em 值一样，关键词的值也是基于用户默认字体尺寸的。不像 em 值那样（也不像像素值在页面缩放时缩小的那样），关键词的值不会减小到合适分辨率以下。如果用户的默认尺寸是 10px（不太可能，但还是有可能的），那么 x-small 就是 9px，而 xx-small 也是 9px。在这种情况下，你显然失去了 x-small 和 xx-small 之间的差异，因此模糊了字体的层次。但是你却不会失去你的读者，因为你给了他们能看得清的文字。

虽然现在使用字体关键词的需求比本书的前两版出版的时候更少了，但是它还是可以作为某些页面布局时的有价值的工具。在本书以前版本中推荐的，使用“盒模型 hack 技巧”来解决关键词尺寸问题，现在已经不再需要了，除非你使用一个 DOCTYPE 来把 IE6 带入 Quirk（怪异）模式（如第 11 章所示）。

13.4 我想要我的“Franklin Gothic”字体

能够对文字的主体、边距和大小进行控制是非常好的，但是怎样控制字体呢？我们难道只能局限于使用 Times、Verdana、Arial 和那些默认安装在大多数 Windows 和 Mac 系统上的字体吗？反问句通常意味着答案是否定的。事实上，CSS 提供了一套机制，能让我们把现实的字体嵌入到标题和主体的文本中（如图 13.13 所示）。

图 13.13　在 CSS 发布十周年之际，它的联合作者 Håkon Wium 为 A List Apart 的读者写了一篇文章，关于如何通过@font-face 引入实际的字体来使 Web 设计突破字体的限制

13.4.1　CSS@font-face：在 Web 上使用现实的字体

当然，标题通常是用 Photoshop 来制作出一张有着漂亮字体的图片，然后通过 img 元素插入到网页中。这是一个古老而经典的方法（你可以在 alt 属性中写入替换文字），但是很烦琐。

那些年，设计师们局限于插入字体图片到 Web 页面中，或者直接用一个图片文件...

```
<img src="headline.gif" alt="Headline text goes here for people who have images turned off." />
```

...或者使用在本书前两版中普及的 CSS 图片替换方法。（想了解更多的图片替换方法，包括 Phark 的-9999px 的方法，请查阅 www.mezzoblue.com/tests/revised-image-replacement。）

回到 1998 年，CSS2 提供了一种方法在样式表里链入现实的字体：

```
@font-face {
  font-family: "Watusi";
```

```
  src: url("http://www.example.com/fonts/watusi.ttf")
  format("truetype");
}
h1 { font-family: "Watusi", sans-serif }
```

代替静态的字体图片，链入的字体文件能从网络上找到，并用来显示 HTML 的文字。不仅仅是针对标题，而且也可以应用到主体文本上。太棒了！但是这里有两个问题：

1. 除非它们是特地许可为 Web 使用的（大多数字体都不是），否则，如果你在你的网页上嵌入字体的话，可能就会违反字体厂商的最终用户许可协议（EULA）。许可为可以用来嵌入网页的字体可以查阅这里 www.webfonts.info/wiki/index.php?title=Fonts_available_for_%40font-face_embedding

2. Safari（和其他的 WebKit 浏览器，包括 Google Chrome）、Opera 和 Firefox 都支持用@font-face 导入 TrueType（TTF）和 OpenType（OTF）字体，猜猜哪个浏览器不支持呢？对了，就是 Internet Explorer。这不是因为 IE 在技术上比其他浏览器差，而是因为 Microsoft 不希望提供这种可能会侵犯字体设计者权利的技术。取而代之的是，Microsoft 仅仅支持用@font-face 导入 Embedded OpenType（EOT）格式的字体——这是 Microsoft 自己发明的字体格式。EOT 通过加密、“子集”（仅使用所需的字母而不是完整的字体）和一些别的技术来防止字体文件被盗版。Microsoft 从 IE4 起就已经支持 EOT 了——而且建议它作为 W3C 的标准（www.w3.org/Submission/2008/01），不过还没有其他的浏览器厂商支持 EOT。长久以来，这一直是 Web 标准的一个问题，直到 Microsoft 加入了标准的阵营为止，Jon Tan 提供了一个值得赞赏的变通办法（www.jontangerine.com/log/2008/10/font-face-in-ie-making-webfonts-work）。

如果你访问 www.zeldman.com/dwws（本书的一个小站点），将会看到网页上使用了 Font Bureau 公司的 Franklin Pro Medium 字体的 Web 许可版本，通过@font-face 链入，而且还兼容 Internet Explorer。查看源代码还可以看到通过标准的 CSS 嵌入有许可的字体是多么的简单。

内嵌字体的标准：太多选择了

当我写这本书的时候，字体厂商正趋向于同意标准，这将保护他们的权利，也

能让设计师们通过标准的CSS把现实的字体嵌入到网页上(www.zeldman.com/x/39)。

他们还只是趋向于，仍然没有真正实现（www.zeldman.com/x/16）。竞争性的建议也来自 Erik van Blokland 和 Tal Leming。.webfont 是一个包含 XML 和字体数据的压缩格式（lists.w3.org/Archives/Public/www-font/2009JulSep/0440.html）；Asender 的 EOT Lite 摈弃了对 Microsoft EOT 格式的主要反对意见，而仍然在 IE 下工作（www.readableweb.com/jeffrey-zeldman-questions-the-eot-lite-web-fontformat）；David Berlow 的 OpenType 权限与推荐表是一种机制，显示了设计师们在特定领域可以使用的特定字体的权利（www.fontbureau.com/otpermtable）。这些方法现在都是可以工作的。举个例子，你可以用不含 Web 许可版本的 20%左右的价格来购买 Font Bureau 字体的 Web 许可版本，然后你可以通过@font-face 嵌入规定域的网页。这将是合法的授权，而且可以在 Safari、Firefox 和 Opera 上工作——但是在 IE 中不行，除非你使用 jon Tan 的变通办法。（我在 www.zeldman.com/dwws 站点上使用了一个 Berlow Font Bureau 的字体。）正如我们可以用 JavaScript 和超前的 CSS 在个人网站和练习项目上做试验那样，现在也可以在你的博客网站和一些非关键的项目上尝试使用@font-face（当然要使用有许可的字体）。而且，只要你获得了字体的授权，使用 Tan 的 IE 变通方法，那么你就可以在重要的项目上使用现实的字体了。希望在今年或者将来，字体设计者们能对标准形成一个一致意见，Microsoft 也能完善@font-face 的支持范围，或者丧失一些市场份额，分给 Chrome、Firefox、Opera 和 Safari。

让 Microsoft 采用其他浏览器的方法来支持@font-face 是很重要的，这不仅仅是因为在原则上（对它自己也很重要）浏览器应该正确且完整的支持 Web 标准，而且也因为Microsoft的EOT格式移除了一些修饰和一些使字体在屏幕上看起来更漂亮的细节。只要 Microsoft 还是坚持只支持 EOT 字体，那么 IE 用户在网页上看到的现实字体就是达不到标准的。

与此同时，如果你宁愿等待字体厂商们解决他们之间的差异，那么你可以考虑使用下面的临时解决方案。

13.4.2　sIFR——可访问的替代类型

在 2004 年，设计师兼开发者 Shaun Inman 设计了一个方法，他谦虚地称之为 Inman Flash Replacement，它允许设计师们通过 JavaScript、Flash 和 CSS 选择一种字

体来替换 XHTML 的文字标题（www.shauninman.com/plete/inman-flash-replacement）。他提交到 A List Apart 网站上，但是由于它不具有可访问性，所以我们拒绝发表它。之后不久，Inman 自己发表了这个技术，开发者 Tomas Jogin 和 Mark Wubben 对它进行了改进，使它的字体可以缩放，可以访问。

sIFR（www.mikeindustries.com/sifr），这项改进后的 Inman 方法，被称作 scalable Inman Flash Replacement——这个小写字母 s 是经过深思熟虑的。IFR 和 sIFR 都是使用 Flash 和 JavaScript，把一小段 HTML 文字替换为使用现实字体相同文本的 Flash 电影。这个 Flash 电影可以随意地创建。如果 JavaScript 或图片被禁用，那么用户将“看到”HTML 文本；sIFR 中的文字也可以被复制和粘贴。

很明显，sIFR 仅仅适用于标题和一小段文本。

sIFR 把丰富的可访问的字体带入了数以千计的网站。它是免费的，而且够酷，还得到了可访问性专家 Matt May 和 Joe Clark 的赞赏。1.0 版本和 2.0 版本实现起来很痛苦，而且如果用户安装了一个屏蔽 Flash 的插件的话就会失效。——这个缺点在更高的 3.0 版本中得到了纠正，这个版本是由 Mark Wubben 维护的（wiki.novemberborn.net/sifr3）。差不多每一个我认识的优秀设计师都至少使用过一次 sIFR 技术。而差不多每一个我认识的四十岁以上的人都有一张结肠镜检查报告。这两句话看上去好像有某种联系。

13.4.3 Cufón——“使用你想用的字体”

为了减轻 sIFR 1.0 和 2.0 实现起来的痛苦，Simo Kinnunen 提出了一个更简便的内嵌字体的方法（cufon.shoqolate.com/generate）。Cufón 看上去应该读作 Coo-PHONE，大多数人都是这么读的，但是它实际的意思是“Custom FONts”。（选出的字母仅仅是为了看起来酷一点。）关于它的文档可以参阅 wiki.github.com/sorccu/cufon。

Cufón 不需要 Flash 或者其他插件；浏览器本身就支持它。它是快速的，即使在处理大量文本时；它是很容易设置的；而且在所有主流的浏览器中都可以工作。Cufón 由两部分组成——一个字体生成器，它把字体转换成一种专有格式，以及一个用 JavaScript 写的渲染引擎。通过标准的 script 元素载入一种字体。Cufón 在下面的浏览器中都工作得很好：

- Internet Explorer 6, 7, and 8；
- Mozilla Firefox 1.5+；
- Safari 3+；
- Opera 9.5+；
- Google Chrome 1.0+。

在你兴奋不已之前，请注意目前在 Cufón 中，用户无法选择文本。这限制了它的可用性和可访问性，也许这就是一个不使用它的充分理由。同样 Cufón 也有许可的概念，所用字体的 EULA（最终用户许可协议）需要有对网页内嵌的特别允许。请查阅 www.cameronmoll.com/archives/2009/03/cufon_font_embedding 上的节选文章“EULA Licensing and Security”。让 Cufón 文本可选择的改进工作正在进行。我们拭目以待。

13.4.4　Typekit 和它的兄弟们

在字体设计师们形成一个标准和所有的浏览器都支持@font-face 之前，“中间人”平台，诸如 Typekit（如图 13.14、图 13.15 所示）和 Typotheque（如图 13.16、图 13.17 所示）处理了许可证和技术上的麻烦，使得使用真正的 Web 字体成为可能。

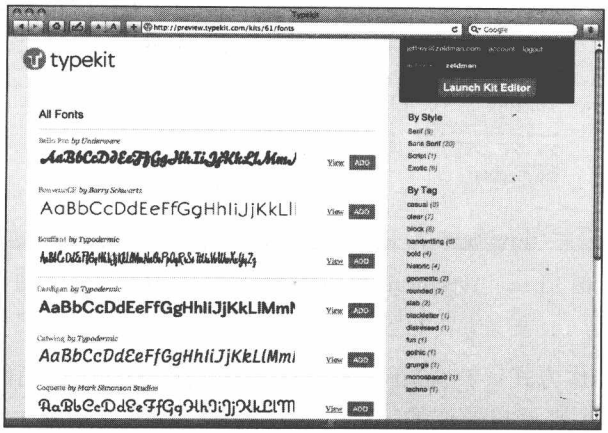


图 13.14　Typekit（www.typekit.com）让设计师们把现实字体嵌入到他们的网页上，而不会违反最终用户许可协议，它们在所有的浏览器中都可以工作

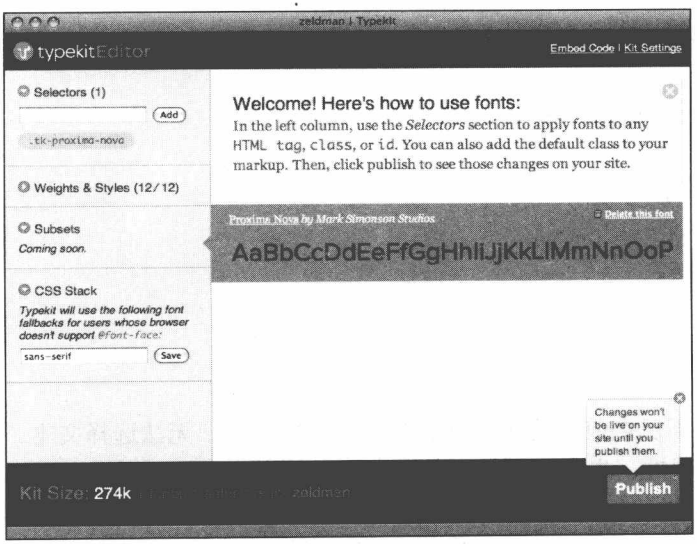


图 13.15　Typekit 网站的界面非常容易使用，直观和雅致

图 13.16　Typotheque（www.typotheque.com）是一个图形设计工作室、字体厂商和内嵌字体的平台

下面是它的工作方式：Typekit 这类公司首先获得字体供应商的签约。字体供应商同意通过 Typekit 来授权他们的字体。设计师们支付月（或年）费给 Typekit 来获得字体的授权和存放。Typekit 也提供一个技术解决方案，来确保字体能够在那些支持@font-face 标准字体格式的浏览器上显示出来（Safari、Firefox、Opera），同样也

要在不支持的浏览器上显示出来（Internet Explorer）。值得一提的是，Typekit 是字体厂商中立的，欢迎所有的厂商，而 Typotheque（至少在开始时）是一个特定的字体厂商。某些高手们在工作中还有他们自己的中间人平台。所有这些解决方案目前都还是测试版。在本书撰写的时候，除了 Typekit，所有平台的定价模式都还不知道——定价肯定会影响认可和接受程度。

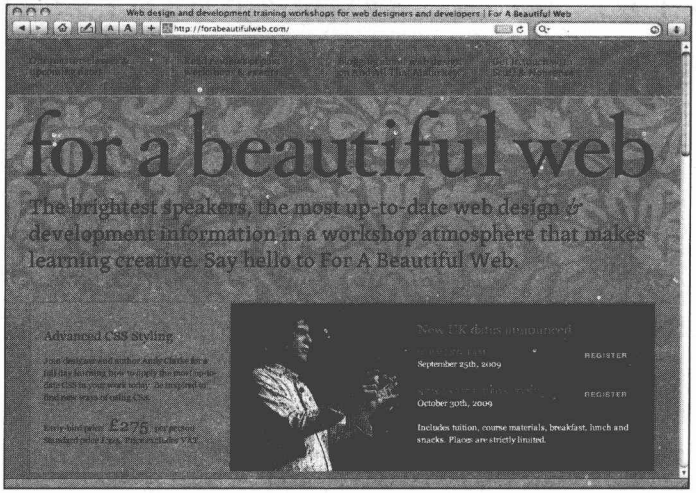


图 13.17　引入了 Typotheque 的页面（www.forabeautifulweb.com/s/660）。

此外，还没有哪个 Web 字体中间平台能提供你所期望的每一种字体，最终，设计师们只会选择那个提供了他们所期望使用的字体的平台。也不知道谁的技术解决方案最好，谁的字体文件下载得最快，当使用量上升时哪个平台最可靠，等等。对于设计师们和这些刚刚起步的平台来说，一个真正令人兴奋的时代来临了。

还有待观察的是，一个字体许可的标准和浏览器对于@font-face 的普遍支持是否会扼杀中间平台，或者中间平台非常成功，以至于他们会延迟或遏制一个字体许可标准的采用，而允许 Microsoft 无限期地让@font-face 只支持它私有的 EOT 文件格式。当然也有可能中间平台使得 Web 字体逐渐被接受，从而加快了许可标准的实现——相应地，促使 Microsoft 全面正确地支持@font-face。

第 14 章

可访问性：Web 标准的灵魂

这一章的标题并不是空想或夸大其词，可访问性就是基于标准设计的核心。和语义性标记以及 CSS 布局一样，可访问性（或者叫通用性）设计的目标就是保证你站点的内容可以被每个人阅读，你的站点可以被每个人使用——不管他们用什么设备浏览，也不管他们的身体机能怎样。尽管可访问性的前景曾被低估，但它实际上已经成为今天许多革新的方向。举个例子，如果你喜欢在 iPhone 上浏览网页，那么就要感谢使这种界面成为可能的关键技术：屏幕放大，它最早是为了帮助可访问性而开发的。

可访问性与本书中讨论的其他标准是联系得非常紧密的，20 世纪 90 年代，W3C 发布了 Web Accessibility Initiative （WAI，Web 可访问性倡议），给网站建造者提供实现可访问性的方法和策略（http://www.w3.org/WAI/GL/）。WAI 的首要任务是制定让残障人士能访问 Web 内容的实现指南。因此毫不奇怪，这些指南被称为“Web 内容可访问性指南”，简称 WCAG。它发布于 1999 年的 WCAG 1.0 (www.w3.org/TR/WCAG10)，提出了 14 条关于可访问性设计原则的指导方针。当前的标准 WCAG 2.0（www.w3. org/TR/2008/REC-WCAG20-20081211），强调了“可测试”的原则，包括动态的富媒体环境，比如 Flash 和 Ajax，这些都是在 WCAG 1.0 中要么回避、要么根本没有设想到的方面。

很多国家，特别是欧盟国家，都使用 WCAG 1.0 或 2.0 作为法律标准来强制和鉴定网站的可访问性。这样很不错，因为它能使整个世界使用同一个标准，可是整个

世界并不总能利用这样的机遇。例如美国，就单方面制定了 Section 508 法案来取代 WCAG 1.0，可能是为了避免跟法国重复吧。

14.1　建立可访问性网站的五个技巧

如果对让你的站点具有可访问性有很多顾虑，或者阅读 WCAG2.0 的文档使你头晕，那么请先在每个网站上简单地应用这五个技巧。

14.1.1　开始

就像当初对非表格布局的恐惧阻碍了数以千计的开发者发现基于标准的设计一样，现在，许多 Web 工作者也害怕可访问性设计掌握起来太复杂，或者需要完成的任务太庞大。（“我们的网站太混乱了，都不知道从哪里着手。”）最长的旅程开始于最短的 alt 文字。使一个网页具有可访问性的方法，跟使它具有语义性的方法是一样的：从一个 p、一个 div、一个 form 字段的使用开始。

14.1.2　使用有逻辑的页面结构

遵照本书的核心建议，按语义来构造你的页面，使用合适的 h1、h2、p 等等。这不仅可以让你的内容更容易被搜索引擎找到，也能让盲人可以使用屏幕阅读器在你站点内容中游历。

屏幕阅读器用户访问你的页面时，它是一个 h2 接着一个 h2，一个段落接着一个段落往下进行的，就像平常用户扫视页面一样。非语义性的元素比如 div class= ”headline”，对于屏幕阅读器和 Google 搜索引擎来说是没有帮助的。用 h1-h6 来代替，能使你的内容让搜索引擎找到，并能为盲人提供导航。

当开始根据语义来构建你的站点时，特别要注意表单和表格的使用，并会运用本章后面将讨论的一些技术——在本书的姐妹卷“Developing with Web Standards”中有更详细的讨论。

14.1.3　提供键盘访问的方法

鼠标对于有“正常”机能的人来说很好用，但是有些人却无法使用鼠标。键盘

和辅助设备是他们上网的工具。让你的站点能够用键盘访问的技术会在本章后面讨论，如果不支持键盘访问，那么你就把数百万人挡在了门外。当 Ajax 成为用户体验流的一部分时，提供键盘访问的能力就变得特别重要了。如果你使用了一些幻灯片效果、弹出式效果和其他一些 Ajax 技巧的话，就必须提供键盘的访问方式。

14.1.4 提供可选方案

在通过一种单一感觉来传达信息时，比如音频，就需要一个可选的切换方案（例如文本）。

14.1.5 挑选一种标准，然后坚持下去

在黑暗中摸索建立不了一个满足可访问性要求的网站，应挑选一种标准，然后遵循它的指导。你应该选择哪一个标准呢？其实这是无关紧要的，只要你理解这个标准的指导方针，尽力去应用它。（不像 HTML 有合法或不合法，WCAG 指南是必须理解的。我的理解可能跟你的不同，但只要残障人士能够使用我们的网站，那我们就都是“正确的”。）

WCAG 1.0 就像黑暗里的一盏明灯，但它同时也是一个落伍的规范，而且并不总像一个标准所应该的那样明确或者详细。自 WCAG 1.0 出现以后，在 Web 开发中，为了清理含糊的部分和适应变化，WAI 发展出了 WCAG 2.0 规范(www.w3.org/TR/UNDERSTANDING-WCAG20)。

WCAG 1.0 提供了 3 级可访问性标准级别：从容易实现（优先级 1），到需要做更多的工作（优先级 2），到精通水平（优先级 3）。WCAG 2.0 同样提出了 3 级可访问性标准级别。这三层级别都是指向可访问性的，就像遵循本书中讨论的其他形式的标准一样，它是一个渐进的支持程度，而不是完全的支持或不支持。你的第一个 CSS 站点，可能不够语义化，可能使用了一个或者两个表格来做内容布局，但是你至少在尝试了。同样，用一些小小的合理的努力，任何人，即使是刚刚接触可访问性的人都能达到或者接近 WACG1.0 的优先级 1。这样，就可以开始让原先那些被我们挡在外面的用户也可以访问我们的站点了。

你必须选择 W3C 的标准吗？不能选择 Section 508 吗？相反，在美国，是强制要求网站和 Section 508 保持一致的。这里有一个小提示：如果达到了 WCAG 1.0 或

WCAG 2.0 的优先级 2 的可访问性要求，那么你也就自动地达到了 Section 508 的要求。如果你更愿意直接在 Section 508 的指导下工作，那当然也是可以的。就像我们在上面说的，挑选一种标准，然后开始应用它。

14.2　有关访问性的书籍

许多本意良好的关于可访问性的书籍都引发了战火。这硫磺的味道可无法鼓舞设计师。这些书里包含了很多外观粗糙，或完全不切实际的可访问性示例站点，还有一些诸如“不要指定字体大小”之类不实用的建议。这个领域里，一些作者对设计很不在乎，还有一些作者对开发商业性站点毫无经验。看了这些书的设计师们可能就会认为可访问性是无关紧要的。

另外一些书，则是深入研究和充满激情的，它们值得花时间阅读，但是它们没有被推荐给一般的 Web 专业人士，因为它们的读者定位在一些有残疾的人。为了满足这些读者，书里花费了很多篇幅来介绍替代的输入方法和评估替代设备的优缺点。而非残疾的设计师们可能由于对残疾本身的无意识焦虑，已经对可访问性感到不安了，因此这些书往往不是最好的选择。

我推荐下面这些书：

- *Universal Design for Web Applications*，作者 Wendy Chisholm 和 Matt May（由 O'Reilly 出版社在 2009 年出版）

  到目前为止，这是关于这个课题的最新书籍，由两位知名的可访问性专家和标准专家撰写，“通用设计”涵盖了所有可访问性 Web 标记的基本要求，比本章中的讨论要深入得多。这本书还解释了手机设备日益增长的需求，要求对可访问性设计给予更多细致地关注；讨论了法律环境的变化；甚至还诠释了 W3C 的文档 WCAG 2.0、ATAG（The Authoring Tool Accessibility Guidelines——www.w3.org/WAI/intro/atag.php）、UAWG（www.w3.org/TR/WAI-USERAGENT）和 WAI-ARIA（www.w3.org/WAI/intro/aria.php）。这些都是什么东西呢？ATAG 告诉 Web 开发软件的厂商比如 Dreamweaver，该如何保证他们的产品能够被残疾人使用。UAWG 为这些厂商提供类似指导。WAI-ARIA，“这一可访问的富媒体互联网的应用套件定义了一种方法，来使 Web 内容和 Web 应用让残疾

人更容易使用。这本书对动态内容和使用 Ajax、HTML、JavaScript 及相关技术开发的先进用户界面控件特别有帮助。”

Chisholm 和 May 在本书中花费了大量的篇幅来描述如何创建具有可访问性的表单，他们还对视频、脚本、富媒体互联网应用及其他方面进行了精彩的讨论。因此，这本书有阅读的必要；

- *Building Accessible Web Sites*，作者 Joe Clark（由 New Riders 出版社在 2002 年出版）

一半是指南，一半是宣言，Clark 的书是描写 Web 设计最引人入胜的书籍之一：诚实，有主见，妙趣横生。在我不寻常的工作历程中，我看过大多数设计方面的新书和许多计算机方面的新书。其中很少有完整而又清晰，能让我真正感觉到是在阅读一个大师的手笔的。可是我贪婪地阅读 Clark 的书，就像是在读最新版的哈里波特小说，一遍遍地读。你不仅能从这本书中学到关于可访问性实践中所需要了解的每一个常识，而且能真正愉悦地阅读这本书。

这本书涵盖了所有方面，从最基础的，书写一个有用的 alt 属性，到复杂的富媒体主题都有论述。被“大西洋月刊”杂志称为“隐藏式字幕（closed captions，CC 字幕）之王”的 Joe Clark，在写这本书之前已经在媒体访问和展示领域工作了二十年的时间。他用独特的立场来明确肯定地引导读者从全局到细节——提供分阶段的可访问性策略，以便适合不同的预算和时间的限制，直截了当，让读者理解得更加清晰。而且 Clark 同样也关注美学上的设计，并在书里展示了如何使它跟可访问性协调起来。我毫无保留地推荐这本书。可惜的是，这本书现在已经绝版。但是你可以去找一本廉价的二手书——或者阅读此书的免费在线版本：www.joeclark.org/book/sashay/serialization；

- *Web Accessibility：Web Standards and Regulatory Compliance*，作者 Jim Thatcheret 等（由 Friends of Ed 出版社在 2006 年出版）

这本书是由多个方面的专家所撰写，包括 Jim Thatcher、Shawn Lawton Henry（一个 WAI 的成员和 WCAG 的贡献者）、A List Apart 的 Andrew Kirkpatrick，以及 Bruce Lawson（HTML5 的专家），这本书的重点任务是告诉你要做出完全遵循可访问性法律规则的 Web 内容所需要知道的具体操作细

节，涉及了从具有可访问性的导航、数据的输入到具有可访问性的 JavaScript、PDF 和 Flash 的方方面面。你将会学习到屏幕阅读器和其他形式的辅助技术是如何帮助人们使用、导航和创建 Web 内容的。它的术语表深入涵盖了美国的可访问性法律，以及世界范围内的可访问性法律和政策等等。这本书适合每一个需要制作符合可访问性法律要求的站点的开发者和所有者。

读这三本书还有一些别的好处，可以帮助你纠正一些错误的认识和我们接下来要讨论的一些可悲而普遍的观点。

14.3　普遍的质疑

就可访问性的概念而言，许多设计师、开发者及网站所有者都倾向于滔滔不绝地说一些毫无意义的有关服务客户的格言警句。而当收到类似于美国康复法案的 508 条款关于可访问性法律的通知时，他们就会像是精神失控了一样。

天才的参与

在很多面向专业听众的演讲场合，我都会听到一个倍受尊重的 Web 设计师在回答一个观众提出的有关可访问性问题时，所说的一些毫无意义的话："我们建立领先品牌的工作是为了服务于那些我们的客户所想获得的精英顾客。可访问性的需求只占我们目标市场的很小一部分，我们的客户并不介意丧失这些顾客。我的意思是，我们的客户是制造高清宽屏电视机的。一般的盲人今年是不会购买的（窃笑...）。"

实际上，如果网站能让他们阅读产品规格说明和使用在线定购服务的话，盲人也许会为视力正常的朋友或者家人购买这样的电视机。此外，绝大部分视力受损的网络用户并不是完全失明或者接近失明的人。大多数要求提高可访问性的人是那些低视力、色盲和轻微近视的用户，他们很可能希望，也愿意购买高清的大画面电视机。

14.3.1　"盲人亿万富翁"

此外，网络爬虫事实上也是盲人用户。Google 搜索引擎是网络上最大的盲人用户。这个"用户"每天为成千上万的客户以搜索结果的形式提出建议。提供合适的内容，进行良好的书写及构建，你不仅为一部分盲人用户提供了服务，也能吸引数

以百万计的视力良好的用户。另一方面，Google 所聚集起来的访问量使得搜索引擎就像一个盲人亿万富翁。有多少网站会对一个有着上亿身价的潜在用户说“不”呢？考虑一下第三方面，光算算美国的残疾人数就有洛杉矶加上纽约的人口总数那么多。拒绝这么多人访问你的网站不是什么明智之举吧？

14.3.2 可访问性不只是视力受损用户的问题

可访问性不只是视力受损用户的问题。活动机能受损（部分或完全瘫痪）的用户也许想买一个好的电视机，而且更愿意在网上购物，而不愿去一趟实体店。很多可访问性的增强功能就是针对这部分用户的。可访问性增强功能也帮助那些使用智能手机浏览网站，并打算购买高品质电视机的非残疾用户。

简而言之，那些说“盲人顾客不会买我们产品”的设计师、开发者或网站所有者错过了这些用户。他们自己盲目地拒绝了本不需要拒绝的用户——包括数百万通过搜索引擎找到他们网站的非残疾用户，如果他们的网站遵从了可访问性指导，许多人可能就会在手机上订购他们的产品了。遗憾的是，他们并不是唯一错误地理解了可访问性所起的作用和所服务的群体的人。

一大堆糊涂的观念

这些年，我听到了很多网络专业人士说出的误导性言论，包括以下这些。

“这是我们客户的问题，如果他们不在 RFP（Request for Proposal，征求建议书）里提出这个问题，我们就不需要考虑这个问题。”一个全球最大的著名网络代理开发商之一如是说，不久之前，这家开发商破产了，公司的剩余部分卖给了中亚的一家小公司。顺便说一句，他所说的是错的。

“508 条款？这是为政府制定的。我们可不是政府职员。”我听到一个设计公司这样说道，而他的客户被法律要求需要提供具有可访问性的服务。

“我们委员会的一个成员正在做调查，关于这个问题我们迟早会发布白皮书的。”在 508 条款颁布一年之后，一位美国政府机构的资深项目管理人员跟我分享了这个看法。

“我们是一个 Dreamweaver 商店，”一个设计师告诉我们，“因此可访问性问题会在新版本的更新中解决，不是吗？”也是，也不是。最新版本的 Dreamweaver 提供了很多可访问性的增强功能，但是你必须知道怎样使用它们。仅仅打开 Dreamwave 是不能保证建立一个具有可访问性的站点的，只有多多使用尼康相机才能拍出好照片。

法律和布局

当 U.S.Rehabilitation Act（美国康复法案）的 508 条款通过时，人们更加疑惑了。（注意：这里我们用的是美国的例子，使用的是美国人的视角，但是相同的原则可以应用在任何地方，而不管地方法规是怎么样的。在大多数案例和大多数地方，地方法规将直接来源于 WCAG1.0 或 2.0。）

508 条款（www.section508.gov）要求很多网站能适应残障人士——从活动能力受限制的残疾人到广大的视力受损者——的需求，并阐述了可访问性的含义。（提示：仅仅给图片添加 alt 属性是不够的。）对于这样的一个任务，很多网络专业人士认为，可访问性就是指纯文本的网页或不具吸引力的，“低端”的设计。其实不是这样的。

目前所有主要的网页设计产品都能完全遵循 508 条款，只要稍加注意就行。随着本章的深入，我们将讨论有一些什么样的可访问性需求和遵循 508 条款有些什么特别的要求，我们还将讨论如何使用智能判断和可用工具来使你的网站完美地符合要求。

14.3.3　508 条款的解释

508 条款是 1973 年颁布的 Rehabilitation Act（康复法案）的一部分，旨在消除对残障人士的歧视。由美国国会于 1998 年 8 月 7 日颁布的公法 105－220 条（1998 年的康复法修正案）对 508 条款的技术访问要求做了很大的扩充。此项法律涉及计算机、传真机、复印机、电话、交易机、共用电话亭，以及其他用于传输、接收或存储信息的设备。当然也包括一些网站。（然而，似乎没有对收银台使用的键盘产生任何影响。即使他们使用的是真正的按钮，我们也没有找到一个符合可访问性的装置——这是很罕见的。）

508 条款于 2001 年 6 月 21 日正式成为了美国的法律。它直接影响了联邦部门和一些代理机构，还有为他们服务的网页设计师。这条法律也适用于政府投资的项目和一些采用该法律的州。许多人都这样做了。

概括地说，508 条款适用于下列情况：

- 联邦部门和代理机构（包括美国邮政服务系统）；
- 为这些部门服务的承包商所交付使用的产品；
- 联邦政府投资或主办的活动；
- 由采用该法律的州政府主办的活动。

“为每个人提供平等或同等的可访问性”

508 条款要求每个网站在其权限下“为每个用户提供平等或同等的可访问性”，包括视力听力受损的、身体有缺陷的和光敏性癫痫的人士。

这些 Web 用户所面临的问题是你想像不到的。例如，不可调整大小的小字体文档会使一些视力有限的用户不能阅读文档内容。（参见第 13 章中对像素问题的讨论。）只有很小“点击”区域的细小导航按钮，会使那些身体机能受损的用户难以使用。闪光或闪烁的页面（是的，仍然在创建大量的那种页面）会诱使患有癫痫症的用户出现威胁生命的发作。这个列表可以继续列下去。该法律列出了很多常见的可访问性问题，并建议了一些可行的解决方法。

508 条款没有禁止使用 CSS、JavaScript、图片或者老旧的表格布局。也没有阻止你在网页中包含如 Flash、QuickTime 这样的富媒体元素，只要你遵循本章后面会讨论的那些指导方针。当然，大多数遵循 508 条款的站点（就像大多数遵循标准的站点一样）在新版本的浏览器上都比在老版本浏览器上看起来更漂亮。这不是法律因素造成的，而是因为 Web 用户可以通过下载最新版本来升级浏览器，大多数浏览器都可以免费获得。

14.4 纠正关于可访问性的错误观点

到目前为止我们学到了什么？遵循可访问性准则不仅使数百万的残障用户能访问你的网站，而且能帮助你获得数百万甚至更多的，包括使用移动设备的顾客——通

过搜索引擎还能吸引更多的访问者。听起来非常棒。但是为什么那么多的设计师、开发者和站点所有者要拒绝或反对可访问性准则和要求呢？首先是因为他们对于可访问性有许多错误的观点。让我们试着清除这些错误的观点。

错误观点：可访问性迫使你要为网站创建两个版本

事实并非如此。如果你用 Web 标准进行设计，并遵循一定的方针，那么你的网站对于屏幕阅读器、Lynx（一种纯文本浏览器）、移动设备和老式浏览器，跟对于现代遵循标准的浏览器一样，都具有可访问性。标准和可访问性一致认为一个 Web 文档应当服务于所有的读者和用户。目前，甚至 Flash 和 PDF 都可以提供可访问性。欲知详情可参见其他书籍。（查阅 *Web Accessibility: Web Standards and Regulatory Compliance* 和在本章后面列出的 Adobe 网站上的内容。）

错误观点：纯文本的网站能给每个人提供平等或同等的可访问性

事实并非如此。适应性技术的发展经历了很长的一段时间，可以使常规网页上的大部分内容完全或部分地具有可访问性，而在布局和外观上不发生变化。（记住：这项工作发生在背后。）为残障访问者提供纯文本网站，就好比把色盲用户假定为看不到所有的东西，或者把活动受限的人假定为不喜欢图片，还假定这些用户没有兴趣在你的商业网站上购物，或者不喜欢参于一个在线论坛的讨论一样。总之，用过时了的纯文本方式对任何人都没有帮助。不仅如此，创建及维护纯文本网页比给网站简单地添加辅助标记和属性的花费要大得多（也是不可靠的）。

错误观点：可访问性的代价太高

事实并非如此。为你的 Web 表单增加一个 label 标签的代价是多少？或者书写一个表格摘要的代价是多少？为页面上每个图片加上一个短短的 alt 文本的代价又是多少呢？这些事都是可以在几分钟内完成的。实现更高级的适应性要求更细致的工作，当然也比完成简单的任务花费得更多，但这并不是必须的。例如，给 Web 视频制作隐藏式字幕或为实时传送的现场流媒体新闻添加字幕会需要花费多一些时间。但它并不必须是成吨的工作，或花费成堆的现金。如何实现这些呢，你可以阅读 Joe Clark 的“Best Practices in Online Captioning” (www.joeclark.org/access/captioning/bpoc)。

大型网站的内容更新工作都是由非开发人员（编辑人员）来做的，要给新的页

面增加可访问性，往往只需要更新内容管理系统，简单地为可访问性所需的属性增加提示。（比如，系统可以要求上传图片的用户，必须给图片加上 alt 文本才能上传。）对于动态网站，也许只需要简单地修改一下生成网页的模板就可以了。给表格添加可访问性的属性，给表格添加结构摘要，对整体模板做其他类似的调整都是一次性的投入，而获得的回报却是更多的新用户和更少的法律麻烦。

“这不在预算之内”

我有这样一个朋友，他总是买 CD 但是从没有时间听，总是买 DVD，但是从没有时间看。他租了一间画室为了想画画的时候用，但是他两年来没画过任何东西。他开通了所有的有线频道，但从不看电视，因为他总是去泡酒吧。这种使人毛骨悚然的消费的生活方式带来的问题就是当他左下侧的臼齿一阵阵的疼痛时，他忍了两个月的牙痛却没有钱看牙医。你也许会说我朋友对于事情的优先级别安排不当，但是很多公司其实和他的态度毫无差别。

那些抱怨可访问性花费的人（一般这些人也抱怨 Web 标准的花费）总是那些把几千美元花费在浏览器检测、有条件的脚本、有条件的 CSS，甚至第 13 章里描述的有条件的 HTML 上的人。他们不认为给 10 个不同的浏览器提供 10 个不同的样式表是在浪费金钱，而其实用一个 CSS 文件就能更好地处理此类问题。他们花费了数百万美元在一个产生不合法的标记代码，对用户不友好的 URL，并且需要请昂贵的专家维护的臃肿并专用的框架上，而不采用一个开源的内容管理系统，比如 Drupal (www.drupal.org) 或者一个低成本并支持标准的，比如 ExpressionEngine (www.pmachine.com)。

但是能为可访问性挤出一点点钱吗？“花费太高了”，他们认为。

许多声称负担不起即使是提供最基本的可访问性支持的公司，却在后端开发，流媒体和一些不必要的复杂 JavaScript 上花费颇多。如果这些花费都被考虑进商业预算中，那么提供可访问性的微小花费也必须同样考虑进去。

错误观点：可访问性迫使你创建原始的、低级的设计

事实并非如此。图片、CSS 布局、JavaScript、服务器端技术如 PHP 和其他现代 Web 设计元素与 WCAG 优先级 1 和美国的 508 条款都能完全地兼容，它们只需要多

一些关注和判别。只要遵循有关的指导方针（本章稍后会讨论），你也可以使用基于插件的技术，如 Flash 和 QuickTime。所有 Happy Cog 在过去九年里创建的网站都采用了基于表单的用户交互和基于 DOM 的脚本，CSS 布局，丰富的图片等技术，它们都是具有可访问性的，也都很容易地通过了在线访问适应性测试。那些在 Clearleft、Erskine Design 和其他数百个公司的同行们都是这样制作网站的。

注意：我们马上会发现，通过在线访问适应性测试，不能保证你的网站真正地具有可访问性或是适应性。软件能运行的测试有很多局限；必须用人的判断来调节评估所有这类测试。另一方面，不能通过 WCAG 优先级 1 或美国 508 条款的测试，就很明显地表明，你的网站不能适应可访问性的要求。

错误观点：根据 508 条款，网站必须在所有的浏览器和用户代理上看上去都一样

事实并非如此。怎么可能呢？当然，大多数适应 508 条款的网站在新式浏览器上的显示效果比在老式浏览器上好。这不是法律因素造成的，而是因为 Web 用户能简单地通过下载来免费更新浏览器。内容必须能在任何用户代理或设备上可使用和可访问。视觉设计在某些环境里根本无法表现，也不必要跟一个图形浏览器中显示的都完全一样。旧式的方法想要使站点在所有浏览器和平台上都看上去相同，这是造成 Web 上仍然有许多不可访问的、不合法的和难以维护的网站的一个原因。

错误观点：可访问性“只是为残障人士服务的”

事实并非如此。诚然，可访问性能改善（或者在某些方面，第一次提供）残障用户的访问。但它同时也帮助下面这些人群：

- 使用移动设备的用户，他们所占的市场份额正在指数级地增加，在亚洲和非洲已经使桌面用户黯然失色；
- 暂时有损伤或残障的人（例如，摔断了手腕的人）；
- 有可校正的轻微视力问题的人，包括年纪比较大的人（占人口很大的比例）；
- 临时使用那些非习惯的环境来访问网站的人——例如，通过机场或其他公共场所的公用电话亭或限定特性的浏览器浏览网站的人；

- 想从竞争者那里争取到这些用户的网站所有者；
- 想从最大的“盲人用户”搜索引擎那里获得好处的网站所有者（参见本章前面的“盲人亿万富翁”）。

错误观点：如果客户提出可以忽略可访问性，设计师们就可以不理睬可访问性条款了

这还有待观察。我所知道的是没有哪个 Web 设计师需要对所创建的不符合可访问性要求的网站负责——还没有。但是美国司法部门曾经处罚过违反美国残障法令的建筑师，而不管客户是否告诉建筑师“要这样建造”。Web 设计师也许某一天也会面临这样的处罚。当我们知道所做的事情是错误的时候，我们的责任是劝告客户（或老板），而不是归咎于他们。此外，即使免除了责任，你难道真的期望你的客户被用户起诉或被司法部门介入？

在 2006 年，盲人用户状告 Target 网站，宣称它们的站点对于盲人用户不可访问，因而违反了反歧视法律（www.news.com.com/Blind+patrons+sue+target+for+site+inaccessibility/2100-1030_3-6038123.html）。在拖延了几个月之后。Target 急急忙忙地修正了站点。我不想成为那个赞成之前不符合可访问性版本的副总裁。我也不想成为这个副总裁电话那头的设计师或代理商。

14.5 关于可访问性的小技巧，一个一个元素地介绍

下面提供了一些方法，使常用的网页元素与 WCAG，或政府的可访问性方针相适应。

14.5.1 图片

如果遗漏了 alt 属性的文本，那么使用 Lynx、屏幕阅读器和其他非主流浏览器及设备的用户就会听见或看见很多的“[IMAGE][IMAGE][IMAGE][IMAGE][IMAGE][IMAGE]”，或其他一些同样没有帮助的信息。缺乏 alt 属性文本将被标记为 WAI 访问错误或 XHTML 校验错误。给 img 元素添加 alt 属性来描述每个图片的内容（www.ws.org/WAI/GL/WCAG20/checkpoints.html）。我们的意思是内容——而不仅仅是外表。

你的朋友，无效的 alt 属性

对于一些无意义的图片，例如用做间隔的 GIF 图片（并不是说你仍在使用间隔图片），使用 alt=""，也就相当于使 alt 属性无效或使 alt 属性的文本无效。不要为一些无意义的图片添加诸如 alt="pixel spacer gif "或 alt="table cell background color gradient"，这会增加用户的问题。为那些纯粹为了视觉设计效果（无意义、非语义）的图片添加无效的 alt 属性。或者，更好的做法是把表现类的图片放到 CSS 中，那里才是它们该待的地方。

用 alt 属性给访问者传达意思

使用 alt 属性值给访问者而不是给你或你的同事传达意思。例如，在一个也用作链接回首页的标志图片上，使用 alt="Smith Company home page"，而不是使用 alt="smith_ logo_rev3"或 alt="Smith Company logo"。对于一个盲人用户来说，他对看不见的那个图片是不是一个“标志”不感兴趣。而告诉她单击这个图片就能返回首页，才更有意义。如果你觉得有必要的话，你可以写成如下这样：

```
alt="Smith Company home page [logo]"
```

由于 alt 文本的目标是当图片不能被访问到的时候提供一个可选的可读文本，因此你应该在链接上使用 Smith Company[logo]作为 alt 文本，使用 Smith Company home page 作为 title 文本。

不要使用自动生成 alt 属性的软件，这种软件很可能生成直接源自文件名的没有用处的 alt 文本：

```
alt="smith_logo_32x32"
```

总之一句话，不要让机器去做需要人工判断的工作。

不要相信软件能做人类的工作

即使你的网页通过了基于 Web 的可访问性测试，也不要认为 alt 属性就能正常工作了。每个图片都使用 alt="mickeymouse"（或是 alt=""）的网页也许也能很顺利地通过这些测试。没有软件能告诉你 alt 文本是否适合。坦白地说，你也不愿意生活在一个由软件决定的世界里。如果你不知道我在说什么，看看电影《2001》《刀锋战士》《黑客帝国》《少

数派报告》《终结者》或者《烤面包机》。好吧，《烤面包机》可能和这没什么关系。

alt 提示文本狂想曲

当访问者的鼠标悬停在图片上方时，一些主流的浏览器（换句话说，所有的老版本 Internet Explorer）错误地把 alt 属性显示为提示文本。虽然现在数百万的 Web 用户对此已经习以为常，但是出于很多原因，这样的显示是一个可怕的主意。主要原因是，alt 文本是一个可访问性的工具，不是什么提示文本（title 属性才是用作提示文本的）。在提出这两个提示属性之前，W3C 就清楚地解释了 alt 文本只有当图片不能显示的时候才能被看到：

> alt 属性指定了当图片不能显示的时候所显示的替换文本……当用户代理不支持图片、或不支持某种图片类型、或不适合显示图片时，必须显示替代的文本。
>
> —www.w3.org/TR/REC-html40/struct/objects.html#h-13.2

浏览器不需要冗余地显示 alt 描述的文本，因为视力正常的用户本来就能清楚看到它们。但是，举个例子，IE/Windows 在 IE8 出现之前都会这样显示。作为防守手段，Netscape/Windows 更是会首先显示 alt 文本，这两个浏览器同时锁定在 HTML 武器的竞赛上，总是试着超越或赶上另一个。这不是你的问题，除非它误导你去写一些“创意”alt 文本，这原本主要是用来为那些看不到图片的人做解释的。

背景图片没有 alt

应用可访问性的初学者们经常问是不是要给 CSS（或 HTML）背景图片书写一个 alt 文本。这是一个合情合理的问题，答案是不需要。事实上，即使你想这样做也不可能做到。在 CSS 里是没有 alt 属性（例如，没有 alt 属性可以指定给 body 的背景图片）。

如果背景图片表达的是页面文档里没有表达的重要意思——例如，如果页面上的文本只写了“他是一个正直的人”，而 CSS 背景图片是一张亚伯拉罕·林肯的画像——你可能想给 body 元素的 title 属性里添加一个包含“总统亚伯拉罕·林肯”的文本，或者，如果使用了表格的话，想添加一个 summary 属性。而更好的做法是包含一张具有合适的 alt 文本的图片作为内容（也就是一个 img 元素）。你总是可以用 CSS 来调整图片的位置，让它显示在文本的后面，看起来就像一个背景图片。

或许，更好的做法是，你应该抛弃上面整个的想法。在一张林肯图片的旁边写上这样的文字，“他诚实吗？”那是多么简洁的网页啊。

视频流媒体

需要用到插件的地方也要提供一个明确的链接。

如果你用一个图片链接到所需的插件程序，要确定图片有合适的 alt 文本。

为了提高 QuickTime 视频（或其他的多媒体格式）的可访问性，可以使用字幕工具或像 SMIL 这样的 Web 标准给音频提供描述文本和与音频轨道相同的字幕。查阅 http://www.alistapart.com/stories/smil/中关于 SMIL 的友好性设计概述，也可阅读前面提到的“Best Practices in Online Captioning（在线字幕最佳实现）”（www.joeclark.org/access/Captioning/bpoc）。

Flash

Flash CS 4 符合 508 条款的要求，包括以下这些方面：内容放大、不使用鼠标的导航、声音同步、等效文本等等。它可以用 Microsoft Active Accessibility 来创建符合可访问性的 Web 内容，使其可以在屏幕阅读器上传达。屏幕阅读器，有时也被错误地称为是“声音浏览器”或者“文本阅读器”，是能够把屏幕的内容翻译成音频的程序，为盲人用户提供一个听觉上的计算机交互界面。两大主流的屏幕阅读器，Windows－Eyes（www.gwmicro.com/Window-Eyes）和 JAWS（www.freedomscientific.com/jaws-hq.asp）都可以理解具有可访问性的 Flash 内容。关于如何用 Flash 来创建具有可访问性站点的教程，请参看 www.adobe.com/accessibility/products/flash/tutorial

额外的提示：

- 如果你用 JavaScript 来检测是否存在 Flash，最好为那些不能使用 JavaScript 的用户制定一个备用计划——就是使用一个清楚地指向目标内容的链接。另外，你在使用 JavaScript 检测是否存在 Flash 的时候，看在上帝的份上，必须清楚地知道你在做些什么。检测插件的脚本和浏览器的缺陷，以及 Web 用户的困惑，纠缠在一起可能会让事情变得一团糟。一个很好的选择是使用 SWFObject（code.google.com/p/swfobject）；

- 要知道不管你做得有多好，即使尽了最大的努力，仍然会有一些用户不能看到你的 Flash 内容。

颜色

如果你用颜色来传达信息，比如为了表明某些文本是一个链接，那么最好用其他一些方法来强调它——例如，粗体或者下画线链接，而不仅仅是依靠色彩。如果你通过 CSS 去掉了下画线，那么应该考虑使链接变成粗体字。同时还应避免在非链接的文本上使用粗体，以免患有色盲的用户分不清哪些粗体文本是链接，哪些是没有链接的普通文本。如果链接文本和普通文本之间的区别很明显，色盲用户都可以分辨的话（举个极端的例子，如果文本为黑色，链接为白色），就可以不使用粗体或其他的区分方案。

在文本中避免提到颜色，“单击黄色的盒子来获得帮助”这样的指令，对于那些看不见（或者色盲）的用户是没用的。

使用和谐的配色方案时要仔细小心，因为一些色盲用户分辨不出颜色间的细微差别。Joe Clark 对于这方面问题解释得很详细，他的书可以在线获得，也可以在很多书店买到。

你还可以浏览 www.vischeck.com 网站，看一下你的网页在不同类型的色盲用户看起来是怎样的。

CSS

在有样式表和没有样式表的两种情况下测试你的网站，以确保在每种情况下网页都是可读的。不用担心关闭样式表时图形设计发生的变化，除非这些变化使网站不可使用。

当去掉 CSS 时，用结构化标记传达意思

如果你使用结构良好的(X)HTML 制作，那么当样式表被关闭或不可用时，网页也能很好地显示。读者“获得”的是有或没有 CSS 的语义标记代码。我们不断地强调使用结构化的标记，避免使用无意义的 div。像下面给出的那些标记，当无法使用 CSS 时，对用户就没有什么意义了：

```
<div class="header">Headline</div>
<div class="copy">Text</div>
<div class="copy">Text</div>
<div class="copy">Text</div>
```

不要相信只在一个浏览器上显示的效果

书写合法的 CSS，并在多个浏览器上测试。不要书写碰巧在某个浏览器上能正常作用的非法 CSS。不合适的 CSS 可能会使得网页无法阅读。不是每个 Web 用户都知道怎样关闭 CSS，更别说怎样用自定义的样式表覆盖你的 CSS 了，而且目前也不是每个浏览器都支持自定义样式表的。

当设置文本大小时要仔细当心

如果你使用 em 值或百分比值来指定文本大小，那么用比默认字体更小的尺寸来检查你的站点（见第 13 章）。如果你的主要受众都是使用 IE6（见第 13 章）或者使用 DOM 或后台驱动的样式转换器来让用户改变文本大小的话，那么避免使用像素值来指定文本大小（见第 15 章）。

我再次重申，不要相信可访问性校验的测试结果

不要仅仅因为你的网页通过了在线可访问性测试就认为你的 CSS 是“安全”的。一个使用了无法阅读的 7px 字体的页面，仍然可能通过这些测试。

复杂交互效果和其他脚本行为

书写的代码应该保证即使在 JavaScript 禁用时，链接也能正常作用。禁用浏览器的 JavaScript，测试一下你所书写的代码。

为非鼠标用户提供一个选择

行动不便的用户可以使用支持 JavaScript 的浏览器，但可能无法单击和完成其他用鼠标操作的功能。给这些用户提供的一个可选的代码，如下：

```
<input type="button" onclick="setActiveStyleSheet('default');
return false;" onkeypress="setActiveStyleSheet('default');
return false;" />
```

在上面那个示例中，onkeypress 对于非鼠标用户来说等价于 onclick。两行代码和谐共存。可选代码对于鼠标用户是不可见的。是的，书写同一功能的两种代码会给网页增加一点字节。在这种情况下，虽然增加了少量带宽，但是吸引了更多残障用户的访问。

当然，实际上你很想把 JavaScript 的事件处理代码从标记中移走，使用不冲突的 DOM 脚本（unobtrusive DOM scripting）方式，但那是第 15 章的主题。

对于不能使用 JavaScript 的用户，使用 noscript。或者更好的方法是，使用渐进增强（progressive enhancement）的方式，摈弃 noscript。

脚本行为和可访问性之间的相互作用可以非常复杂，完整的讨论已经超出了本章的范围。想更多地了解 JavaScript，可以参见第 15 章中推荐的一些书。

表单

在本章的开始部分，我们极力推荐了三本书。每本书中都用了整整一章的篇幅来讨论具有可访问性的在线表单的创建。由此你可能会推断出创建具有可访问性的在线表单是有些复杂的。你的推断是正确的。

但是不用慌张。大多数要做的工作都比较简单直接，比如把表单字段和适当的标签关联起来（例如，把一个搜索表单中的文本区域和一个“Search”标签关联起来）。还有很多像这样的小工作，如果全都讨论的话就超出本章的范围了。

创建了你所期望的具有可访问性的表单后，把它放在 Lynx（lynx.browser.org）或 JAWS 上测试。老的 Mac 用户还需要虚拟 PC（www.microsoft.com/windows/virtualpc/default.mspx）或一个真实的 PC 来运行 JAWS。在基于 Intel 的 Mac 上，你将需要 Boot Camp（www.apple.com/ support/bootcamp）、Parallels（www.parallels.com）、Fusion（www.vmware.com/products/fusion）或 VirtualBox（www.virtualbox.org），外加一张 Windows CD 或一台虚拟机。你也可以运行 VoiceOver（www.apple.com/macosx/features/voiceover），一个内置于 Mac OS X（始于 Tiger 版本）的可访问性交互界面。对于 Linux 用户，可以选择一个包含在 SuSE Linux 7.0 发行版（www.hicom.net/~oedipus/vicug/SuSE_blinux.html）中的免费屏幕阅读器，或者选择 Speakup（www.linux

-speakup.org/speakup.html）。

图片热点图

尽可能避免使用图片热点图，一般都可以不用。一定要用的话，使用带有 alt 文本的客户端图片热点图，另外要提供冗余的文本链接。不要使用老式的服务器端图片热点图。

表格布局

别头疼这个。用第 8 章中所学的，书写一个简单的表格摘要，使用 CSS 来避免深层嵌套表格和 GIF 间隔图片，以及其他的垃圾，这些都在第 8 章到第 10 章中讨论过。

事实上，不管你听到过什么样的相反意见，使用简单的表格布局并不是造成可访问性问题的主要因素，在 WAI 或 508 条款下也不是不合法的，也不会谴责你的灵魂，使其永久受折磨。好吧，我不能确定关于永久折磨的部分，但是表格布局在 WCAG 或 508 条款下确实不是不合法的（虽然我们希望它们是不合法的）。

数据表格的使用

标识表格标题，用合适的标记把数据单元和标题单元关联起来，让表格含有两个或更多逻辑层次的行和列的标题。在一个列出电影“*The Music Man*”的演员表里，典型的表格标题应该是“演员”，与它关联的表格单元应该包括 Robert Preston、Shirley Jones、Buddy Hackett、Hermione Gingold 等。

使用图形浏览器的正常视力的用户可以看出“演员”及其下面的姓名栏之间的关联。但是对屏幕阅读器的用户来说，则需要额外的标记来关联表格标题和数据单元。查阅 www.w3.org/WAI/wcag_curric/sam45-0.htm 上的资源，了解 WAI 小组是如何阐明标题和相关数据单元之间的关联的。

帧（Frame），Applets

不要使用它们。

闪烁和闪光的元素

也不要用。自从使用 FrontPage 模板以来，你可能就再没有使用过 blink 或 marquee，但是记住，在 Flash 和 QuickTime 内容里也不要使用闪烁和闪光元素。

14.5.2 专业工具

如果你用可视化编辑软件制作网页的话，使用一些工具和插件程序可以更简单地实现可访问性。

Firefox 的 Web 开发工具条

难以置信的是，Firefox 的免费菜单和工具条提供了众多的 Web 开发工具，很多工具的想法都是为了实现快捷的可访问性测试，如 CSS 样式的开关、临时禁用 JavaScript 和页面颜色等等。工具条可以安装在 Firefox 或 Mozilla 上，在任何可以运行这些浏览器的平台上都可以使用，包括 Windows、Linux 和 Mac OS X（www.chrispederick.com/ work/webdeveloper）。

IE 的 Web 可访问性工具条

Vision Australia 的 Accessible Information Solutions（AIS）团队提供了这个工具条，这个免费的 IE/Windows 下的工具条（欢迎捐助）很容易对不同方面的可访问性进行检验。开关 CSS 样式表、用 alt 属性替换 img（图片）元素、测试动画 GIF 图片的闪烁是否在光敏性癫痫患者的“可接受范围”，等等（www.visionaustrahia.org.an/info.aspx?page=614）。

可爱的 Cynthia

无论你是使用前面提到的开发工具还是你自己手工书写站点的标记和代码，Cynthia Says Portal 都将是一个选择（www.contentquality.com）。

这个免费快捷的服务，能为你检测网页的可访问性，尽管细微的差别还是要靠你自己的判断和分析。WAI 和 508 条款都依赖一个手动填写的检查表来验证可访问性。不像 W3C 的标记和 CSS 校验服务，Cynthia Says Portal 的校验测试不能为你提供健康证明书或是一个需要修复的错误列表。相反，你得分析输出结果。这是其复

杂之处。但是，这也是它们可以成为教育工具的地方，你可以成为一个知识更丰富的设计师、开发者或有关的 Web 专家，获得更好的薪水。

在 Cynthia 之前，还有 Bobby，是一个以英国警官作为吉祥物的在线可访问性测试服务（Bobby 后来被 Watchfire 收购了，他们将程序修改了一下，并改叫 WebXACT，但它并不能很好地工作，最终退出了商业市场。）

14.5.3　保持有效的 Tab 键：我们的好朋友，tabindex 属性

tabindex 属性可以指定页面上按顺序用 tab 键定位选择元素。如果你没有创建标记的逻辑次序，依赖 tab 键切换（不用鼠标）的用户就会按(X)HTML 源代码里所列出的次序从一个链接选择到另一个链接。这不是引导用户浏览你网站的最有效方法，特别是当你的 body 文本包含很多链接或者标记的一开始就有冗长的导航的时候。

就像 Skip Navigation（跳跃式导航）一样，tabindex 属性可以帮助屏幕阅读器用户省去串联导航带来的麻烦，使他们能很迅速地跳到感兴趣的内容。跳跃式导航可以跳过长长的链接列表，tabindex 提供一种快捷方法，能快速地访问到页面的不同部分——但是不像 DVD 的章节索引那样可以直接往前往后跳到喜欢的段落。

在建立商业性网站时，应该在创建了 tab 次序后，让真正的用户来测试。在个人站点或非营利网站上，你也许没有条件去做这样的测试。如果不能让用户测试，可以建立一个用户情景，在此基础上创建一个 tab 顺序，然后看看使用 tabindex 的访问者的反应是否符合你的预期。

14.5.4　为可访问性做计划：你将受益良多

虽然现在有些网站还没有被法律要求必须提供可访问性支持，但是将来可能就会有这样的要求。我们知道法律总是在不断地修改中。我们还知道，法律的条款不管我们是否喜欢都得遵守。对你的站点进行可访问性方面的增强，即使目前的法律没有要求你这样做，那么也许当明年法律发生改变时就能让你省下昂贵的重建花费，还可能使你免除因反歧视法的起诉而产生的花费（和负面宣传）。

热爱你所做的事情

在那些突然对可访问性产生兴趣的站点所有人所提出的理由中，我还想加一句，因对法律管制的恐惧而把可访问性包含进你的设计实践中，这并不是一个正当的理由。这些增强能让一些网站获得新的访问者——谁的网站不想吸引更多的访问者呢？如果你对标记做些调整，欢迎那些被其他站点拒之门外的用户，他们就会成为你的忠实用户。如果其他在线商店拒绝了残障用户和非传统设备用户，而你的商店欢迎他们，那么谁还会去别的商店买东西呢？

别忘了，你的网站对残障和非传统设备用户的可访问性越高，就越容易在 Google、Bing 和其他一些爬虫驱动的搜索引擎上被搜索到，并指引用户到你的网站上。相反，可访问性越低就越不容易被 Google 和其他搜索引擎搜索到。啊，我想达到更高的道德层面，我还有一个纯粹的个人兴趣来实现可访问性。这里有两个额外的理由：

1. 提高可访问性可以加深你对设计的理解。考虑标签次序这样的问题可以使你超越只对表面外观（“look and feel”）做设计，从而进入用户体验设计、可能性设计和可用性设计的领域。这些都是 Web 设计师、用户体验设计师、信息架构工程师和可用性专家思考的问题。可访问性是关于如何构建最好的网站来满足不同群体的需求，所需要考虑的另一个方面。你对可访问性考虑得越多，你就将越深刻地理解用户体验和预见用户行为——让你成为一个更好的设计师；

2. 实现可访问性和研究适应性策略能提高你的开发技术，为你开拓一些你原先从来没有考虑过的新视野。学习 WAI 的方法和 508 条款的详细内容将提升你作为一个专业 Web 设计师的价值，提升你的 Web 设计代理公司的地位，使其比其他竞争者更聪明更博学，并且帮助你的站点获得更多的用户。这是每个网站所有者、设计师或开发者努力的目标。

是的，实现可访问性可以帮助你的访问者实现他们的目标，但是也能帮助你实现自己的目标。

第 15 章

使用基于 DOM 的脚本语言

最初，Netscape 发明了 JavaScript，它工作得非常好。后来，Microsoft 推出了 JScript，但它和 JavaScript 不太一样。DHTML 带来的巨大冲突几乎吞没了一切。而文档对象模型标准（DOM）的诞生则意味着一个拯救者的出现，它首次出现是 DOM Level 1 规范（http://www.w3.org/TR/1998/ REC-DOM- Level-1-19981001/）。它的确是个非常好的规范，W3C DOM 第一次给予了 Web 设计师和开发者一个标准的方法，来把站点的数据、脚本和表现层结合在一起。

之后的这些年中，W3C 持续更新 DOM 规范，并且在 Web 标准项目组（WaSP）的推动下，很多浏览器都开始支持绝大多数 DOM Level 1 规范内容，尽管他们有时采取了不同的方法去实现。在这个章节中，我们将接触到 DOM，并且研究一些简单的应用，来帮助创建一个具有可访问性的为用户考虑的站点。

这是一个简短的章节，并不是因为这个主题过于简单，而是由于它很有深度和广度，远远超出了本书的范围。幸好，有两本很好的书能够提供本章所无法展开的内容。

15.1 关于 DOM 的书

- “DOM Scripting: Web Design with JavaScript and the Document Object Model，作者为 Jeremy Keith（由 Friends of ED 出版社在 2005 年出版；中文书名为

《Javascript DOM 编程艺术》，人民邮电出版社 2007 出版）。这本书是我所期待的最具可读性并最适合设计师阅读的书籍。它从粗浅地介绍 JavaScript 和 DOM 开始，逐渐涵盖最佳实践，带领读者进入一系列令人激动的视觉效果和有实际应用的项目中，每个项目都会教给读者如何去思考和实现。一个在线的试读章节可以表现出本书讨人喜欢的写作风格（www.domscripting.com/book/sample）。本书的作者是 Clearleft 公司（www.clearleft.com）的创始人，该公司是一家位于英国的专注于基于标准设计的公司，并且是 Web 标准项目组（WaSP）的成员。

- “*The JavaScript Anthology: 101 Essential Tips, Tricks & Hacks*”，作者是 James Edwards 和 Cameron Adams（由 SitePoint 出版社在 2006 年出版；中文书名为《JavaScript 精粹》，人民邮电出版社 2007 年出版。）该书列出了几百种控制网页行为的 JavaScript 脚本（也就是使用 DOM）。本书的重点是关于近年来在基于标准的脚本社区中涌现的最佳实践——比如渐进增强（progressive enhancement，提供给那些没有 JavaScript 支持的用户）以及不唐突的脚本编程（unobtrusive scripting，也译作谨慎的编程，把行为从结构和表现中分离出来）。书里有个关于 JavaScript 和可访问性的章节非常有趣，还有一些关于应用 Ajax 来创建 Web 应用的好内容。James Edwards 和 Cameron Adams 是分别来自英国和澳大利亚的开源程序开发者。

脚本将如何在创建一个便于访问的基于标准的网站中扮演关键角色？这本书将为你展示这一点。不过本章下面的部分只是对没有脚本开发经验的设计师做一个快速概览。

不冲突的脚本编程（Unobtrusive Scripting）

本章关于 DOM 的书的部分我们提到了术语不唐突的脚本编程（Unobtrusive Scripting，也有叫 Unobtrusive JavaScript 和 Unobtrusive DOM Scripting），这已经成为基于标准开发的一种最佳实践。这个思想非常简单：把行为从 Web 开发的另外两个部分（结构和表现）中分离出来。不像以前的弹出式广告、强制新窗口、强制尺寸改变和其他“外观上”的 JavaScript 花招，不唐突脚本编程是以用户为中心的，也是内敛的。它设计用于增强具有语义性和可访问性的标记结构，并能够在用户或者设备不支持 JavaScript 的情况下提供回退支持。

虽然 Web 标准项目组（WaSP）的 DOM 脚本任务小组但它已经解散，但是他们指出了这种实践方法的好处：

- 可用性方面的好处：一个应用不唐突 DOM 脚本编程方法创建的效果不会打扰用户。它是如此好地附加在站点上，以至于访问的用户不假思索就能使用它；
- 平稳退化（graceful degradation）方面的好处：一个不唐突的 DOM 脚本在它出错的时候不会打扰用户。它不会产生错误信息，即使是在那些老版本的浏览器中。不唐突的脚本首先会询问，浏览器能否支持我想要使用的对象？如果浏览器回答不能，那么这个脚本就会默默地退出；
- 可访问性方面的好处：页面的基本功能不依赖于不唐突的 DOM 脚本。如果脚本无法运行，页面仍将通过标记、样式表以及/或者服务器端脚本技术来提供核心功能和信息。用户不会注意到少了什么东西；
- 分离的好处：不唐突的 DOM 脚本编程在 Web 开发工作中不会打扰其他方面的开发。所有的 JavaScript 代码都是单独维护的，不需要在 XHTML、PHP、JSP 或者其他语言代码中到处嵌入。

关于这个主题的更多信息可以查阅 Bobby van der Sluis 网站上的文章“Ten Good Practices for Writing JavaScript in 2005”（www.bobbyvandersluis.com/articles/goodpractices.php）。

15.2　DOM 是什么

那么 DOM 是什么呢？根据 W3C 的定义（www.w3.org/DOM），DOM 是一个浏览器独立、平台无关、语言无关的编程接口，它允许“程序和脚本动态地访问和更新文档中的内容、结构和样式。文档可以进一步处理，处理的结果可以混合回当前的页面。”

通俗地讲，DOM 使得另一些标准页面组件（样式表、(X)HTML 元素和脚本）更容易处理。如果把你的站点比作一部电影，那么，(X)HTML 就是剧本作者，CSS

就是艺术指导，脚本语言提供特效处理，而 DOM 就是监管整个电影的导演。

15.2.1 使网页变得像应用程序的标准化方法

尽管介绍这类编程的方法已经超出了本书的范畴，但是 DOM 驱动的交互开发中最令人激动的方面就是能够使网页模仿传统应用程序的行为。举个例子，用户可以通过单击表格头部行来改变表格中数据的排列顺序，就像一个 Excel 表格或者在 Macintosh 的 Finder 程序（这个程序可以让 Macintosh 用户对他们计算机上不同种类的文件和文件夹进行排序、复制、移动、更名、删除，或其他操作）中一样。

近些年来，这种能够模仿传统软件的 DOM 能力推动了基于标准的 Web 应用的开发，创造了一些令人激动的新产品。但这不是一朝一夕的事。

开发者最初使用 DOM 驱动的交互能力在客户端模仿软件行为——就是说在访问者的电脑上（如图 15.1、图 15.2、图 15.3 所示）。在客户端处理数据意味着即使互联网连接中断了，功能仍然可以正常运行；还有就是运行速度的优势，比大部分美国的 Web 用户所喜欢的快速不断线的速度都快。在用户的电脑上运行而不是通过网络还可以降低服务器的负载。更有意思的是，这些初期的 DOM 演示介绍了一种对于 HTML 来说是全新的领域：能够在不刷新页面的情况下响应用户的操作。这便是 Ajax 的领域了。

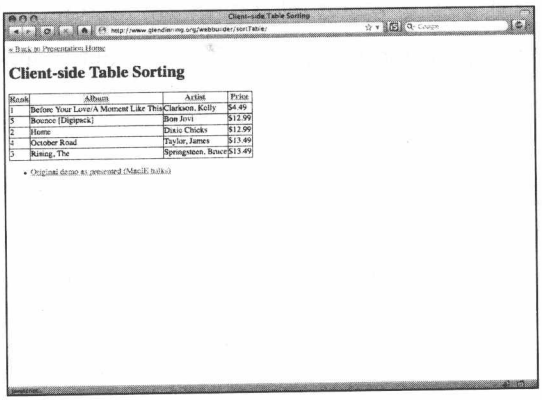


图 15.1 DOM 可以让网页的响应类似桌面程序。在这个早期（2002 年）Porter Glendinning 的 demo 中，当用户点击专辑的表头就能让专辑名称数据进行排序（www.glendinning.org/webbuilder/sortTable）

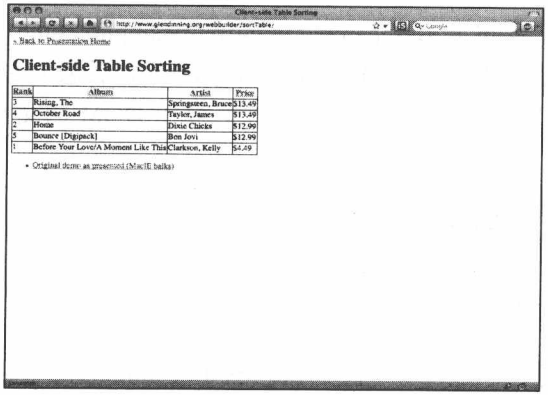


图 15.2　再次点击表头能够反向排序。改变后的新序列能够马上在原有的页面上即时显示。不需要一个额外的“结果”页面

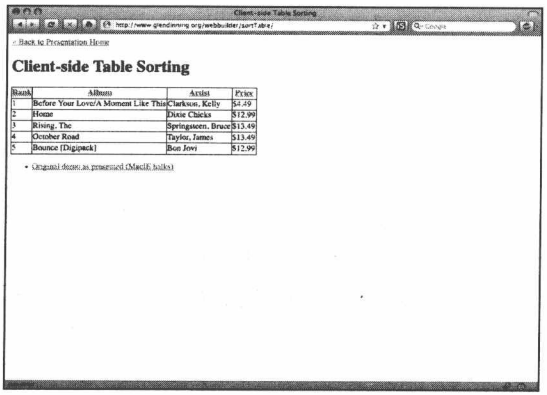


图 15.3　点击 Rank 的表头能够重新按数字排序，同样不需要加载一个新页面，也不需要服务器/客户端的处理或者通信

虽然图 15.1～图 15.3 中展示的 demo 比较简单，但是他们的意义却不简单，因为它们证明了，不需要 Flash 就可以在 Web 上创建出无缝的交互功能。在本书的第 1 个版本里，我曾经预言，随着浏览器最终支持 W3C 的 DOM，Web 标准将开创一个富应用和用户体验的新时代。可是几年过去了，事实并没有证明我是正确的。我的预言开始变得像上个星期的寿司一样，过期了。

随后 Google 开发了 Gmail（mail.google.com），一个在线邮件应用程序，它构建于 CSS、XHTML、JavaScript 和 DOM，以及一点点 Microsoft 技术的基础上。

后来，37signals 发布了 Basecamp（www.basecamphq.com），作为一个在线专业项目管理工具，它已经可以取代 Microsoft Project 了。后来 Garrett 先生发布了关于 Ajax 的文章（第 4 章）。自从 Gmail 面世以来，基于 DOM 的交互开发变成了最热门的技术。

15.2.2 那么，它在哪里运行呢

虽然实现上有些不同，不过下面的浏览器都支持 W3C 的 DOM：

- Mozilla 0.9 或者更高版本（包括 Firefox 和 Camino）；
- IE5/Windows 或者更高版本；
- Safari；
- Opera（自版本 7 开始）。

由于近年来对 DOM 支持的发展，这个列表不够彻底全面。现代浏览器也都支持 XMLHttpRequest（en.wikipedia.org/wiki/XMLHttpRequest），这是一项可以不刷新页面就能获得服务器响应的 DOM 技术。XMLHttpRequest 开始是作为 Microsoft 的一项发明而出现的，后来被 W3C 纳入了 Web 标准（www.w3.org/TR/XMLHttpRequest）。

除了一些废弃的浏览器，还有什么被排除在上面的列表之外呢？许多手持设备和 Web 电话还没有充分支持 DOM，一些文本浏览器比如 Lynx 也不能支持。在很多情况下，你需要像弥补不支持 JavaScript 的环境那样，对不支持 DOM 的用户代理进行弥补处理。（你总是要为那些不能运行 JavaScript 的环境进行弥补处理的，不是吗？）

为了支持不支持 DOM 的设备，可以进行如下操作：

- 使用 noscript 元素来提供替代的交互方式（比如用一个链接代替一个奇特的按钮）；
- 在 Lynx 中测试，就像我们在第 14 章“可访问性的灵魂”中对可访问性和 JavaScript 的讨论一样；

- 绝对不要使用像<a href="javascript: ... ">这样的方式来伪造链接。取而代之的是，通过操作 DOM 来把丰富的行为附加到链接上，或者如果非要在你的标记中使用 javascript：式的链接，那么也请你通过 DOM 来插入它们。这样才能保证只有支持 JavaScript 的设备能看到它们，而不会干扰不支持 JavaScript 的用户。

15.3　合理使用 DOM

通过使用不唐突脚本编程（unobtrusive scripting）的方式来支持不能运行 DOM 的设备和用户不仅仅是一个最佳实践，它还是在任何情况下都必须做的正确的事。它特别适用于那些内容站点、传统的交互站点和结合了交互和信息的站点。这已经涵盖了绝大多数的站点，可能也包括你的网站。

15.3.1　它是如何工作的

设想有这样一种情况：你是一个大站点的 Web 设计师，这个站点有一个独立的编辑部门负责内容。有一天，当你正在浏览 RSS 文章时，一个市场部门的同事来到你的座位旁。看上去这个家伙跑上楼来是有事情要说，他们想让你在网站上做一点小小的改动：就是在所有指向外部网站的链接后面加上一个小图标（如图 15.4 所示）。他们希望把这个加到站点上的每一个页面上，包括数以百计的以前发布的文章。然后市场人员离开了，顺便还表示感谢你的帮助。（当她离开时，她可能还说了一些关于这个图标将会带来多少多少市场价值之类的话，但是你也许完全没听到她在说什么。）

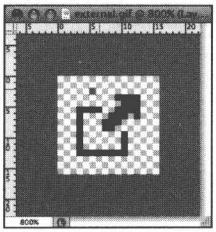


图 15.4　简单的小图标 external.gif，我们将把它放到每一个外部链接上

负责内容的团队无法帮助你完成这个任务——他们没有你所拥有的技术，而且他们也被截稿日期掐得紧紧的——看起来这只能看你的了。我们如何才能把这个小小的 GIF 图片插到数百个页面中的每一个外部链接上呢？

使用 DOM。假设你已经快速地弄好了一个 HTML 演示页，body 里面有一些测试用标记：

```
<p>Here we have a selection of links: <a href="/about/">an internal page</a>, <a href="http://yahoo.com/">an external site worth visiting</a>, <a href="http://google.com/">another external site</a>, and <a href="/contact/">one final internal page</a>.</p>
```

没什么特别的：这是一个包含了四个链接的段落，其中有两个是指向外部网站的。我们可以认出它们是外部链接是因为它们的 href 属性值是用 http 开头的（这一点在后面会很重要）。然后，让我们给它加上一点样式：

```
body {
  color: #303030;
  font: normal 100%/1.3 Helvetica, Arial, sans-serif;
}
a {
    color: #36C;
}
a img {
  border: none;
  margin-left: 0.1em;
  vertical-align: baseline;
}
```

也没什么特别的，毕竟这只是一个简单的测试页面，应用一点基本的 CSS 就可以了（如图 15.5 所示）。

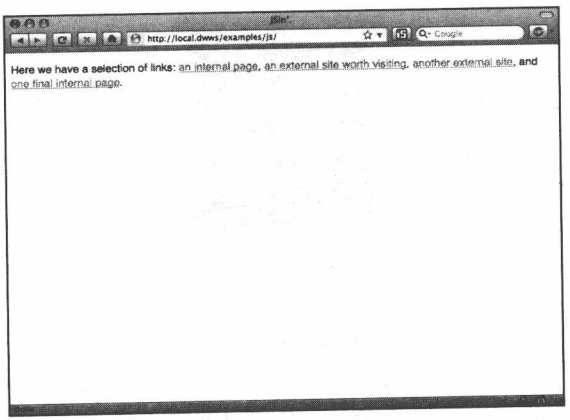


图 15.5　我们的标记和 CSS 等待着 JavaScript 层

现在，手头上的工作就是：书写一点 JavaScript。在参阅了上面所列出的那些好书之后，让我们在文档的 head 元素中放进一点 JavaScript 吧。

```
<script type="text/javascript">
function externalLinks() {
  var allLinks = document.getElementsByTagName("a");
  var icon = document.createElement("img");
  icon.setAttribute("src", "external.gif");
  icon.setAttribute("alt", "[external link]");
  for (var i = 0; i < allLinks.length; i++) {
    var oneHref = allLinks[i].getAttribute("href");
    if (oneHref && oneHref.indexOf("http") == 0) {
      var newImg = icon.cloneNode(true);
      allLinks[i].appendChild(newImg);
    }
  }
}
window.onload = externalLinks;
</script>
```

如果你从未写过 JavaScript 代码，那么上面的这些代码可能会让你感觉有点不知所措。让我们简要地看一下重要的部分。

```
function externalLinks() { ... }
```

script 代码块的第一行定义了一个 JavaScript 的函数（function），我们把它命名为 externalLinks。一个函数是一段 JavaScript 的代码，当另一段 JavaScript 代码显式地用此函数名调用时，里面的代码才会执行。事实上，这就是我们最后一行 JavaScript 所做的事情：

```
window.onload = externalLinks;
```

当我们的 HTML 页面下载完成后，这一行代码就会简单地去调用 externalLinks 函数。方便，省事。

```
var allLinks = document.getElementsByTagName("a");
```

我们已经声明了函数，现在把目光移到函数里的 JS 段落。首先，我们先来看一下等号（=）右边的部分，我们使用了第一个 DOM 方法，document.getElementsByTagName()。它的名称非常具有描述性：如果我们在圆括号内插入一个 HTML 元素的

名称，那么 getElementsByTagName()将会从我们的文档中找出每一个同类元素，并把它们存储到一个列表（也叫集合 collection）中。document.getElementsByTagName("a")告诉我们的 JavaScript 要找到 HTML 中的每一个链接元素（锚），以便将来我们能对它们进行操作。

等号左边的部分（var allLinks）很简短，但是同样重要。我们声明了一个叫 allLinks 的变量（variable）。就像在数学中一样（一个我不会假装很了解的领域），变量是一个实际值的替身；声明了 allLinks 变量之后，我们就可以把 document.getElements ByTagName("a")的调用结果存储到内存里。从这一点出发，我们就能在函数内部的其他地方引用 allLinks，而不需要一再地调用 document.getElementsByTagName("a")。

```
var icon = document.createElement("img");
icon.setAttribute("src", "external.gif");
icon.setAttribute("alt", "[external link]");
```

接下来，我们建立了另一个叫 icon 的变量。这次我们用 document.createElement()来创建一个 HTML 元素——具体来说就是一个 img 元素，因为我们在圆括号里写了"img"。接着的两行为这个元素赋予了属性——一个 src 属性指向 external.gif 文件，一个 alt 属性指定为"[external link]"文本——但是这个元素是存在于浏览器的内存里的（而不是页面上），并且被 icon 变量所引用。

```
for (var i = 0; i < allLinks.length; i++) { ... }
```

这是一个 for 循环，一个非常普通的 JavaScript 工具。在它的圆括号里，我们声明了另一个变量（一般命名为 i），并且赋予一个初始值 0（var i = 0）。然后，只要 i 的值小于 allLinks 内存储的链接元素的数量，那么就会重复执行花括号里的代码。另外，每一次循环都会把 i 的值加 1。一旦 i 的值等于或大于 allLinks 的长度，我们就停止循环，继续后面的代码。

如果上面的段落把你绕晕了，也不用担心——其实我也有点晕了，不过我只是描述了事实而已。让我们分步骤，用通俗的语言再梳理一下。

1. 通过遍历上面的样例 HTML 文档，我们的 allLinks 列表或数组得到了四个项，对应着页面上每一个链接元素。因此，allLinks.length 的值是 4。

2．当我们的 JavaScript 函数开始运行时，碰到了 for 循环，并把 i 的值赋为 0。

3．然后询问 i（这时它的值是 0）是否小于 allLinks.length（4）。

4．因为 0 小于 4，所以就接着执行循环体的花括号里的代码。

5．当花括号里的代码执行完毕之后，循环体做的最后一点工作就是把 i 的值增加 1（i++），并且把这个新的值（0+1）重新赋予 i。

6．这个过程将一直持续，直到循环运行四次，每次处理文档中的一个链接。在第 5 次循环开始后，i 的值就是 4 了。当 JavaScript 检查 i 的值（这时是 4）是否小于 allLinks.length（4）时，循环就会刹车了：因为 4 不小于 4（它们是相等的），它将退出循环，不再去执行里面的代码。

总之：我们的循环每碰到一个 HTML 页面中的链接元素就运行一次，并且每次把 i 增加 1。一旦把链接数循环完了，它就会简单地停止运行。

```
var oneHref = allLinks[i].getAttribute("href");
```

在循环内部，我们声明了一个叫 oneHref 的新变量。当我们每次迭代循环时，allLinks[i]就将引用列表里的一个不同元素——allLinks[0]是第一个链接，allLinks[1]是第二个链接，依此类推。我们现在所做的事情，就是获取当前正在查看的链接的 href 属性，并把它保存下来，以给下一行所用：

```
if (oneHref && oneHref.indexOf("http") == 0) { ... }
```

这里我们用了一个条件语句，就是用 if 来开头的语句行。在圆括号里，JavaScript 询问是否有一个 oneHref 变量——如果我们所检查的链接元素有一个 href 属性，就会返回 true——并且那个 href 属性是否是由 http 开头（oneHref.indexOf("http") == 0）。换句话说，这是不是一个外部链接？如果是，那么花括号里的代码会被执行。

最终，具有魔力的事情就会发生在这里：

```
var newImg = icon.cloneNode(true);
allLinks[i].appendChild(newImg);
```

在我们的 if 条件语句里，做了两件事：首先，创建一个叫 newImg 的变量，并使用 cloneNode()方法，从前面的 icon 变量中复制一个新的图片元素。然后，使用 appendChild()把这个新图片作为子元素来附加到当前的链接里。记住：因为判断过链接是否指向网站外，所以只有外部链接才会加上我们的小图标。从 HTML 的测试页中可以看到，我们的 JavaScript 工作得非常好（如图 15.6 所示），简单得很。

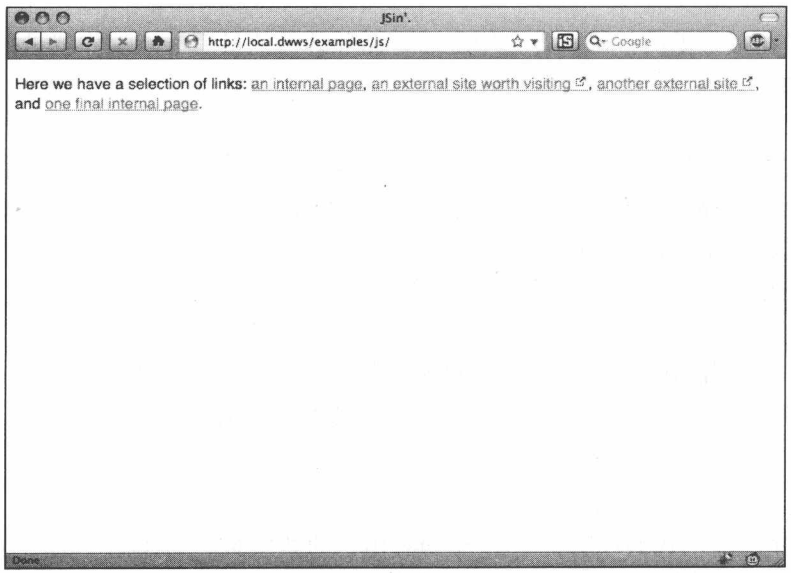


图 15.6　我们的 JavaScript 函数完成了，使用 DOM，仅在外部链接里适当地插入了图标。太棒了

15.3.2　检查是否支持

然而，我们的函数还有一个问题。如果页面在一个支持 JavaScript 但并不支持 DOM 的浏览器里查看时，就会产生一个错误。让一个老旧的不支持 DOM 的浏览器执行 document.getElementsByTagName()或 document.createElement()，就好像是在用世界语向某个老朋友询问时刻一样：你将可能会得到一堆白眼，还可能会被认为精神有问题。除非，你的朋友也是说世界语的。

好吧，这不是一个最好的比喻。但是，当碰到不理解我们所使用的 DOM 方法的浏览器时，还是需要保护好我们的 JavaScript。我们应该简单地询问浏览器是否支持我们所使用的 DOM 方法——在这里是 document.getElementsByTagName 和 document.createElement，而不是去分析用户代理的字符标识（一种在第 1 章里讨论过

的不可靠的方法）。我们可以用很少的代码来询问这个问题，只需要两行额外的代码就可以了：

```
<script type="text/javascript">
function externalLinks() {
  if (document.getElementsByTagName && document.createElement) {
    var allLinks = document.getElementsByTagName("a");
    var icon = document.createElement("img");
    icon.setAttribute("src", "external.gif");
    icon.setAttribute("alt", "(This is an external link)");
    for (var i = 0; i < allLinks.length; i++) {
      var oneHref = allLinks[i].getAttribute("href");
      if (oneHref && oneHref.indexOf("http") == 0) {
        var newImg = icon.cloneNode(true);
        allLinks[i].appendChild(newImg);
      }
    }
  }
}
window.onload = externalLinks;
</script>
```

我们所做的修改仅仅是把函数的内容包裹进一个条件语句中。修改之后，我们的函数将只在浏览器理解 document.getElementsByTagName 和 document.createElement 时运行；如果浏览器不理解，那么这个函数将默默地退出，我们的 HTML 页面上的链接将保持原样。瞧，这就是渐进增强（progressive enhancement）的工作方式。

15.3.3　不同的代码书写方式

在一个全局的.js 文件中，代码看起来就跟上面一样。使用一个全局的.js 文件是减轻你工作量的首选方法，它可以被缓存从而节省带宽，并能够帮我们把行为从结构中分离出来，让标记就是标记，脚本就是脚本。使用一个全局的.js 文件，你只要简单地把脚本代码粘贴到一个空的文件中（把它命名为 scripts.js），并且把下面这一行加到文档的 head 元素中：

```
<script type="text/javascript" src="scripts.js"></script>
```

链接到标记中的 scripts.js 所起到的效果，就跟用 link 把样式表应用到我们的(X)HTML 中一样。一旦我们把 JS 移到了一个外部文件中，那么其中的那些功能就可以被应用到数十个或数万个页面上，只要用上面的 script 标记把它包含进页面就行了。

如果你只是把 JavaScript 应用在单独的页面中，那么可以把它们写在<head>和</head>之间。在 HTML 4.01 的文档或 XHTML1.0 Transitional 文档中（或者在作为 text/html 的 XHTML 1.0 Strict 文档中），脚本可能是像下面这样的：

```
<script type="text/javascript">
<!-- //
function externalLinks() {
    ...
}
window.onload = externalLinks;
// -->
</script>
```

当 XHTML Strict 文档作为 application/xhtml+xml 时，你也可以使用全局的.js 文件，或者像这样直接插入代码：

```
<script type="text/javascript ">
// <![CDATA[
function externalLinks() {
    ...
}
window.onload = externalLinks;
// ]]>
</script>
```

完整地讨论 JavaScript 能做的事情将会超出本书的范畴。但是我在本章结束之前，至少还要介绍一个漂亮（且有用）的使用基于 DOM 的方法来实现的交互功能。

15.3.4 样式切换器：有助于增强可访问性，提供更多的选择

在第 13 章，我们讨论到，要想在不疏远至少是部分潜在访问者的情况下提供合适的 Web 字体几乎是不可能的，而且悲哀的是，在 Web 逐渐演变为一个媒体的十六年中，像素值仍然是设置字体大小的最可靠的方法——不过也是 IE 7.0 之前的用户最恼火的方法。怎样才能提供给访问者一个字体自选方法呢？甚至让他们能改变当前页面的布局？

通过 CSS，你就可以做到这一切。CSS 不仅可以让一个默认（持久）的样式表应用到页面上，而且可以应用一些可替换的 CSS 文件。为了增强可访问性的益处，这些可选的样式表可以提供更大的字型（如图 15.7、图 15.8 所示）或者对比更强烈

的配色。甚至可以完全改变站点的外观，从而实现“用户定制”的目的。

图 15.7　将在第 17 章详细讨论的 New York magazine 网站（www.nymag.com）中，在某些页面上使用了一个基于 DOM 的样式切换器，来让用户可以改变字体的大小

图 15.8　通过结合应用 JavaScript 和 CSS，同一个页面可获得更大的字体

W3C 推荐浏览器提供一个能够允许用户选择这些替代样式表的方法，基于 Gecko 的浏览器比如 Mozilla Firefox 这样做了。但是大多数的浏览器并没有做到。因此我们还以为本来从可替换性样式表上，能够获得的创造性和可访问性方面的好处，可能永远都无法得到了。但是在 2001 年，一个十几岁的 Web 开发者 Paul Sowden 利用可替换样式表跟其他页面组件一样可以被 DOM 访问到的这个事实，解决了这个问题。

在 A List Apart 网站上的一篇文章中（www.alistapart.com/articles/alternate），Sowden 写了一个 JavaScript 文件（www.alistapart.com/d/alternate/styleswitcher.js），可以允许站点访问者通过点击一个链接、表单元素或者另外一些交互组件来加载可替换的样式表。在解释了如何使用他的脚本之后，他把代码开源了。

数以万计的设计师开始用这个代码来解决可访问性的问题、增强创造性的效果。和其他一些功能一起，Paul Sowden 的 ALA 样式切换器帮助了 Dave Shea 心爱的网站 CSS Zen Garden 的诞生。而且，除了能处理字体大小和切换网站皮肤之外，样式切换器还可以驱动“缩放布局”（www.alistapart.com/articles/lowvision）来帮助那些视力不佳的用户们。

15.4 学会使用你喜爱的（JavaScript）开发库

正如前面我们提到的，在过去几年里 JavaScript 确实有了进一步地流行。不久之前，JavaScript 还是被看作是丑陋的技术，仅仅应用于表单验证的领域和制作令人厌恶的弹出窗口。但是今天，客户和公司们都开始要求用 JavaScript 驱动的富应用交互界面来构建在线网站。因此，设计团队们就需要有一些方法来减少单调琐碎的 DOM 脚本开发工作，以便他们能够快速简便地构建出那些健壮的交互界面。

JavaScript 开发库登场了。一个开发库就是提供一个框架，把那些单调琐碎的 DOM 脚本抽象出来，同时解决一些遗留的跨浏览器的差异，因为这些差异依旧存在于不同浏览器的 DOM 实现中。好的 JavaScript 开发库有很多，比如 Prototype（www.prototypejs.org）、Dojo（www.dojotoolkit.org）、MooTools（www.mootools.net）和 Scriptaculous（script.aculo.us）。

现在，先来看一下 jQuery（如图 15.9 所示），这是我们将在第 17 章使用的 JavaScript 库。它使用了类似 CSS 的选择器作为接口，因此我发现这对于善长设计的开发者来

说非常容易使用。举个例子，如果我想在文档中选择出所有的链接元素，只要简单地写成：

```
$("a");
```

如果要选择 class 名为“alert”的链接元素，我可以这样写：

```
$("a.alert");
```

或者，如果我想实现我们前面所写的那个函数的功能，只要超级简单地写成下面这样就行了：

```
$("a[href]^='http'");
```

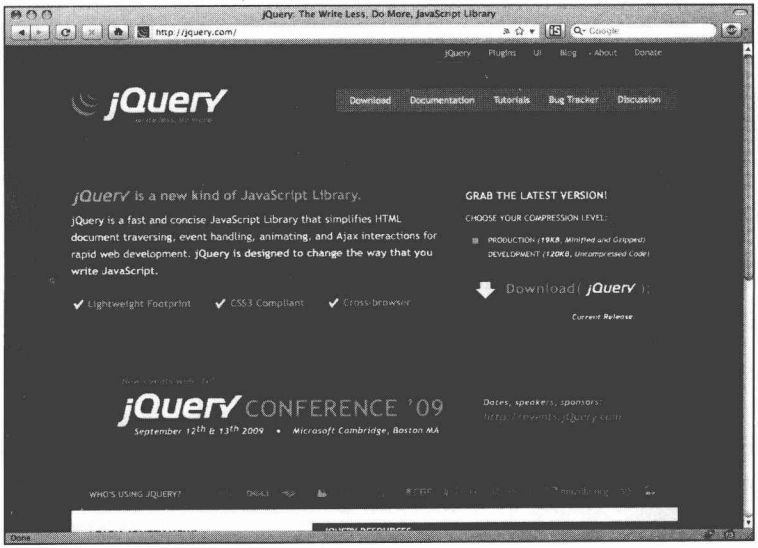


图 15.9　jQuery 的首页（www.jquery.com）

看上去好像有点复杂，但是 jQuery 的开发文档（docs.jquery.com）可以非常好地帮助你理解这类语法（docs.jquery.com/Selectors/attributeStartsWith#attributevalue）。这里我们很容易地挑选出了所有 href 属性是由 http 开头的链接元素。

一旦获得了元素的集合，你就可以用 jQuery 方法来处理它们。我可以给那些外部链接增加一个新的 class：

```
$("a[href]^='http'").addClass("external");
```

或者，给每个外部链接插入一个图片：

```
$("a[href]^='http'").append('<img src="external.gif" src="(This is an external
link)" />');
```

或者（你可以看到我有多么强力）我可以把这两件事都做了，只要简单地把两个方法“链”在一起：

```
$("a[href]^='http'").addClass("external").append('<img    src="external.gif"
src="(This is an external link)" />');
```

事实上，只需要用很少几行的 jQuery 就可以实现原先那个函数的功能，我们用的是 jQuery 简便的类 CSS 语法。

```
<script type="text/javascript>
$(document).ready(function() {
$("a[href^='http']").append('<img src="external.gif" src="(This is an external
link)" />');
});
</script>
```

这就是全部代码。我们的 JavaScript 代码减少到三行，而原先写了十几行的代码。并且实现的功能是完全相同的：只要页面完成下载和渲染，我们就选择出所有的外部链接元素，然后给它们每个内容里加上一个一个图片元素。

然而，还有一点平衡需要考虑：为了有效地使用一个 JavaScript 库的接口，我们必须把这个库包括进来：

```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript>
$(document).ready(function() {
   $("a[href^='http']").append("<img src=\"external.gif\" src=\"(This is an
external link)\" />");
});
</script>
```

这将给页面增加一些份量。比如，jQuery 的基本代码有 20K 字节左右。这并不是很大的数量——比大多数的图片文件还要小——但是你应该把这个额外的开销考

虑到你的 JavaScript 策略计划中。如果你所需要的仅仅是一个或两个简单的函数，那么直接书写一些 DOM 脚本应该更有意义；如果你构建的是一个高交互性的富应用界面，那么可以挑选一个自己喜爱的 JavaScript 开发库为基础来进行开发。

15.5　你应该如何使用 DOM 呢

本章仅仅是略微介绍了一下 DOM 能做的事情，并且展示了如何简单或精练地把它应用到商业、个人和公共站点上。你可以使用 DOM 创建 Web 富应用、创造有创造力的特效、增强可访问性等。在有经验的程序员手中，DOM 能够提供有力的帮助来构建那些能运行于现代浏览器上的基于标准的软件。但是即使你是一个讨厌写代码的设计师，也能用它来为你的站点增进可访问性。因此，不要问 DOM 能为你做什么——而要问你能用 DOM 来做什么。

第 16 章

一个网站重构的实例

让我们运用所有基于标准的设计知识来试着重构一个网站。这个有问题的网站是 zeldman.com，自 1995 年发布以来，网站的主要内容是“每日报告”，一个最古老的、不断发布的博客。

在重新设计之前，明智的做法是先问问你的同事。当征求客户或者公司内部用户意见时，我们称之为“竞争性分析”；当自己审核自己的时候，我们称之为“寻找灵感”。通过分析我了解到：一年多来，我同事们的博客早已突破了 800x600 的布局限制，建立了更宽的、多列的布局（如图 16.1～图 16.4 所示）。灵感来自 Josef Muller-Brockmann，他也是网格平面设计的作者以及现代布局之父。

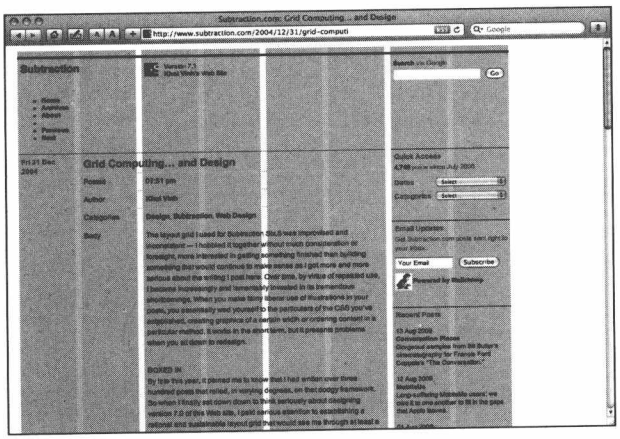


图 16.1　Khoi Vinh 介绍了如何使用背景图片来帮助校正一个基于网格的 1024 布局（www.subtraction.com/2004/12/31/grid-computi）

这种流行的网页布局方法是为 1024 像素（以及更宽）的显示器设计的，通常包含 8 个或者更多平分的网格列（如图 16.1 所示），通过不同列的结合，形成各种宽度的列，可以给用户带来一致性和多样性的浏览体验。为了预留一定的边框和浏览器滚动条，一个 1024px 的网格布局可能只有 960px 到 970px 宽。khoi Vinh，纽约时报（nytimes.com）网站的设计总监，在他的个人网站（subtraction.com）上采用了这种更宽的布局设计。A list Apart 也采用了这种布局，因此它也很适合网络杂志。Cameron Moll 建立了一张背景图片，用来帮助设计师建立 960px 宽屏的模块化布局（www.cameronmoll.com/archives/2006/12/gridding_the_960）。全世界的 Web 设计师都开始转向采用宽屏和留白设计。

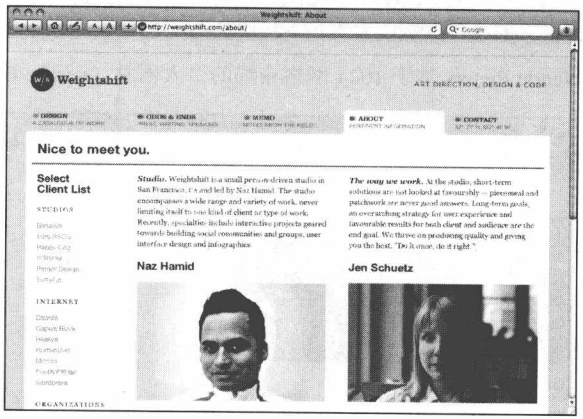


图 16.2　在 Naz Hamid 的网站 Weightshift 上（www.weightshift.com/about）基于网格的 1024 布局

图 16.3　Jason Santa Maria 网站基于网格的 1024 布局（www.jasonsantamaria.com/articles/whats-golden）

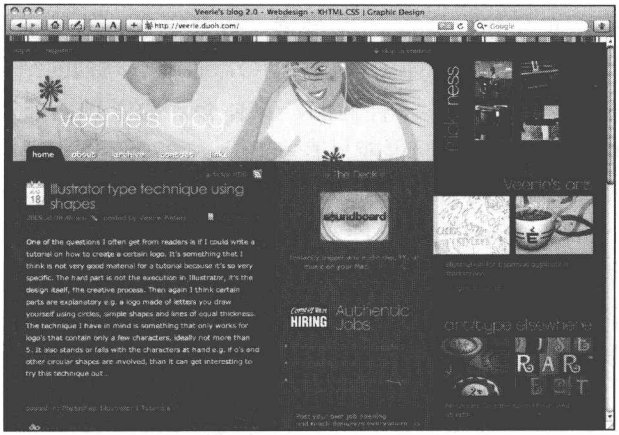


图 16.4　Veerle Pieters 基于 1024 网格布局的个人网站（veerle.duoh.com）

图 16.5　2004 年，一个下拉阴影的风格的 Zeldman.com

我的网站也将采用一个基于 1024 网格的布局？网站重新设计之前的样子（如图 16.5 所示），是本书第 2 版时候建立的，拥有那一年的所有特征：精细的下拉阴影灵感来自 MAC 新操作系统 OS X；柔和的淡色调标志着成熟、不再狂野的 Web；一个较窄的版面为了适应 800x600 分辨率的读者，也能在 1024x768 或者更高分辨率下看起来自然。如果想让 zeldman.com 跟上 2009 年的风格，那么我需要做的是：取消背景颜色，增加 Twitter、Flickr 以及其他社区网站的消息源（feed），再加入那些社区网站的精髓功能，然后采用宽屏、模块化的网格布局。

16.1　来自过去的灵感

这是一个很好的计划，对很多网站来说是一个合适的策略。例如，一个模板网站、文字比较少的平面形式的专业设计师网站，以及想要在首页展示 Web 应用程序的设计师——写作者的网站等，对于这些网站，上述策略可以很完美地实现目的。但是，人们来我的网站不是来看我的照片或者寻找 Flickr 信息源的，他们是来阅读的。这是一个简单的事实，引导我寻找合适的设计方案。

鉴于我的网站宗旨，正确的布局应该是一个方便阅读并减少任何让阅读分心的元素的设计。在理想状态下，那里应该只有文字，就像一本书，没有导航，没有侧栏，没有链接，没有按钮。当然，这是不可能的。但是新的设计应该把重点放在让访问者的注意力集中在阅读上。或许有潜在的网格，但它不会是 1024x768，超过两列（一列内容，一列侧栏）对阅读来说是没有必要的，而且一个适合阅读的版面不应该太宽。作为额外的好处，我的新设计应该最适合在 iPhone 手机里阅读，该设备带来了美观时尚和可用性的移动体验，开创了智能手机新时代。

此外，如果阅读和老式的 800 像素宽度的布局是必要的话，我的灵感不应该来自现代网站，而应该来自网站过去的版本。我的网站在过去的 14 年里历经过许多版设计（如图 16.6、图 16.7 所示），但是外观和感觉最有特色的、最“著名”的，一提到我的网站大多数人就想到的版本，是 20 世纪末的一个让人眼前一亮、高对比色的版本：白底、黑字、橙色背景和辅助色（如图 16.8、图 16.9 所示）。在很多人看来这个外观就是 zeldman.com 的同义词。如果我复活这个老旧设计，但是用 CSS 添加一些浏览器支持的字体效果是不是可以？页面其他部分可以用模块化的网格设计，但页面的核心看起来更像是 1999 年版的改良版。

听上去这是个可行的计划。

图 16.6　zeldman.com 网站已经发布将近十五年，这是其中的一个版本

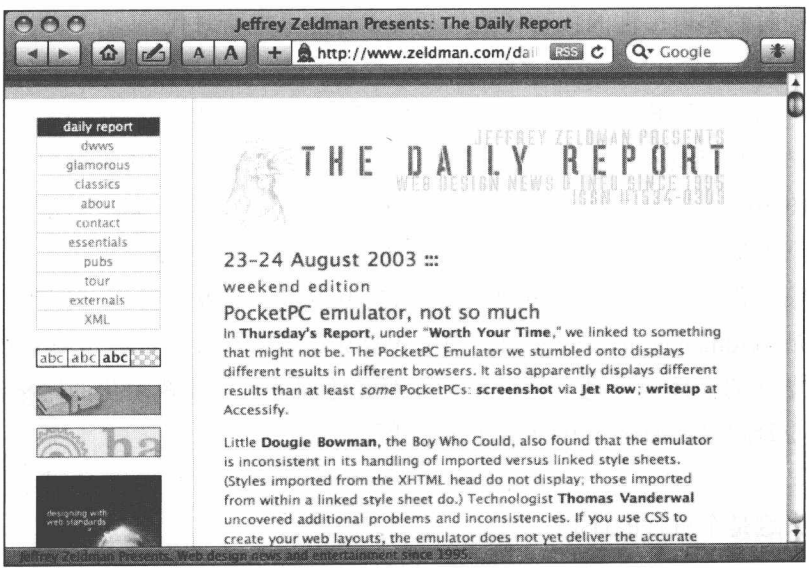


图 16.7　另一个版本

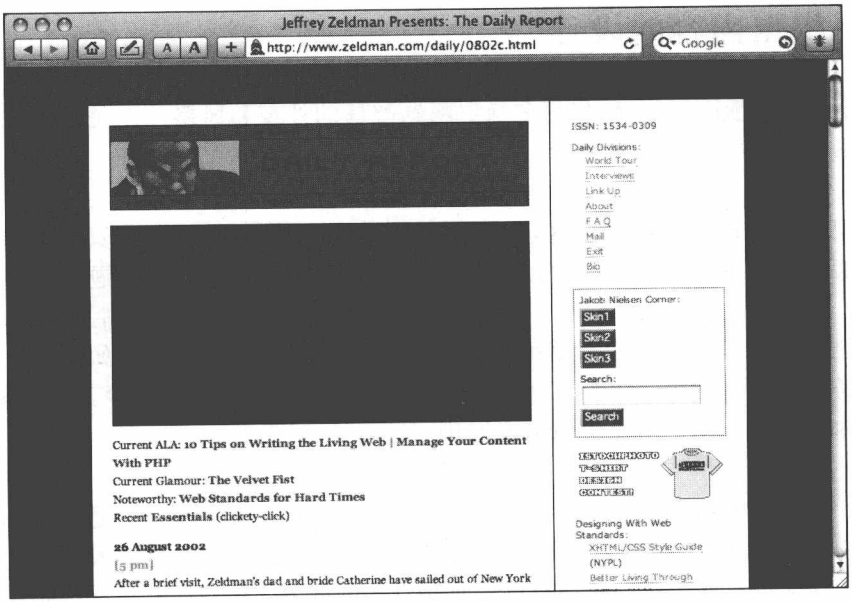


图 16.8　这个外观…

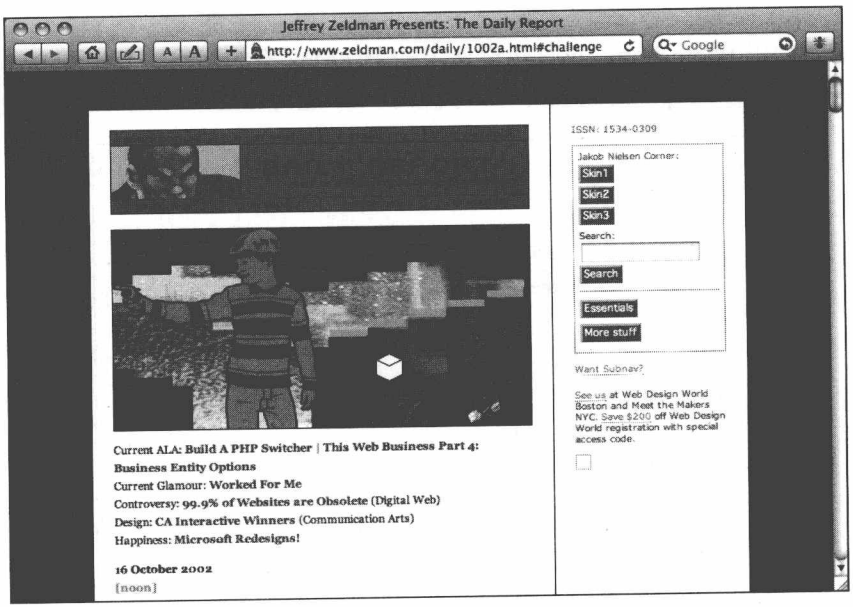


图 16.9　…是 zeldman.com 最有名的一个，也是复古灵感的来源，我们将对它重新设计，使之更现代、更具可读性

16.2 根据内容的设计

我并没有先画一个完整草图来确定网站整体的外观和风格，而是决定先设计第一页的核心。

如果网站的设计将由内容驱动（这始终是个好主意），我决定用真实的内容来制作，就像我们在 Happy Cog 做客户的项目一样。为特定的文字、主题和信息进行网站设计，这是最好的。设计了一个漂亮的模板，当你把内容填入之后，它却影响了对网站的理解，没有比这更糟糕的了。好吧，有些事情更糟糕，例如，踩上一只水母。但是在我们的领域，很少有比“一个漂亮但不恰当的设计”更令人悲哀的事情，用真实内容设计有助于防止类似情况发生。

由于将用真实内容进行设计，我决定直接写代码而不是用纸、铅笔和 Photoshop。我只关注页面的中间，那里将放置文章内容。我不担心页头会是什么样子，或者页脚有什么功能，只是简单地从现有网站复制真实的内容，用我所知道的网站结构和语义写真正的标记，把它们粘贴到一个文本编辑器里，并保存成为一个新的 HTML 文件。

不需要使用 Photoshop，我直接设计了一个 770px 宽的网格，其中 550px 用来放网站的主要内容，220px 是侧栏内容。我做了一个速写计算（如图 16.10 所示），在

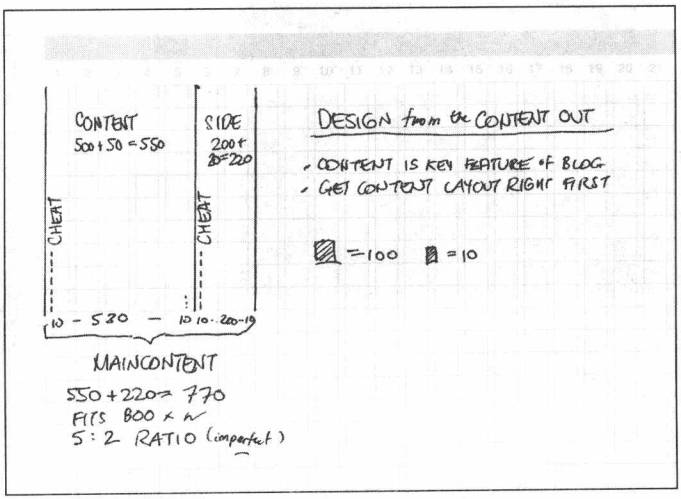


图 16.10　通过速写来演算我们的网格布局。我们需要这些数值来写 CSS

我的新 HTML 文件中，我写 CSS 来定位这两个部分“maincontent”和“sidebar”。页面初步布局的 CSS 代码看起来如下：

```
html {
min-width: 770px;
}


div#wrapper {
width: 770px;
margin: 0 auto;
padding: 0;
text-align: left;
}


div#maincontent {
float: left;
margin: 0;
padding: 0;
width: 550px;
}


div#sidebar {
float: left;
margin: 0;
padding: 0;
width: 220px;
}
```

html 声明规定，这个页面的宽度是 770px，如果需要的话，可以在这里添加一个背景风格。“wrapper”层用来包装两个浮动的列（“maincontent”和“sidebar”），这个层将居中布局，层里的文本将左对齐。正如你清楚看到的，主要的内容区域有 550px 宽，侧栏有 220px 宽，就像我们计划的一样。

16.2.1　留白

列与列之间需要留一定的空隙，“以方便文本呼吸”。中间部分的布局代码类似这样：

```
html {
min-width: 770px;
}

div#wrapper {
width: 770px;
margin: 0 auto;
padding: 0;
text-align: left;
overflow: visible;
min-height: 1000px;
 }

div#maincontent {
float: left;
margin: 0;
padding: 0 30px;
width: 490px;
 }

div#sidebar {
float: left;
margin: 0;
width: 180px;
padding: 0 20px 36px;
 }
```

结果页面（如图 16.11 所示）还很简陋，但我们可以验证布局是居中的，两列的宽度比例接近 5:2，正如我们所希望的，中间有一个合适的间隔。

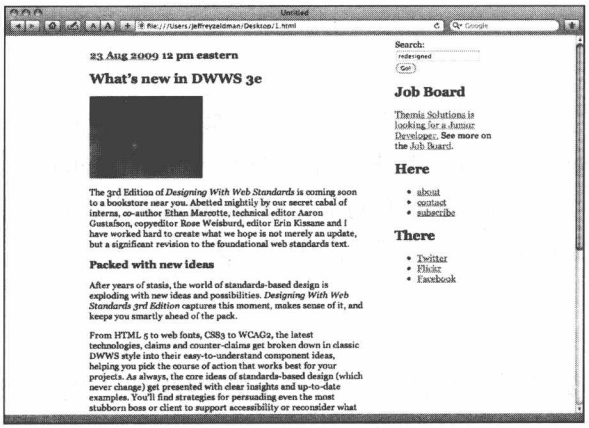


图 16.11　包含真实内容的基本布局（www.zeldman.com/dwws/1.html）

为了帮助布局能在 iPhone 中看起来很好，我们增加一个“viewport”的控制 meta 元素在页头：

```
<meta name="viewport" content="width=770" />
```

设置 viewport 为 770px，将确保 iPhone 按我们的期望大小显示页面。想了解更多关于 iPhone 的技巧，阅读 Craig Hockenberry 的“把你的内容放进我的口袋”（www.alistapart.com/articles/putyourcontentinmypocket）。

16.2.2　字体、介绍和首字母下沉

接下来，我们尝试在布局中增加不同的内容（但仍然是真实的），并开始用 CSS 添加颜色和字体细节（如图 16.12、图 16.13 所示）。

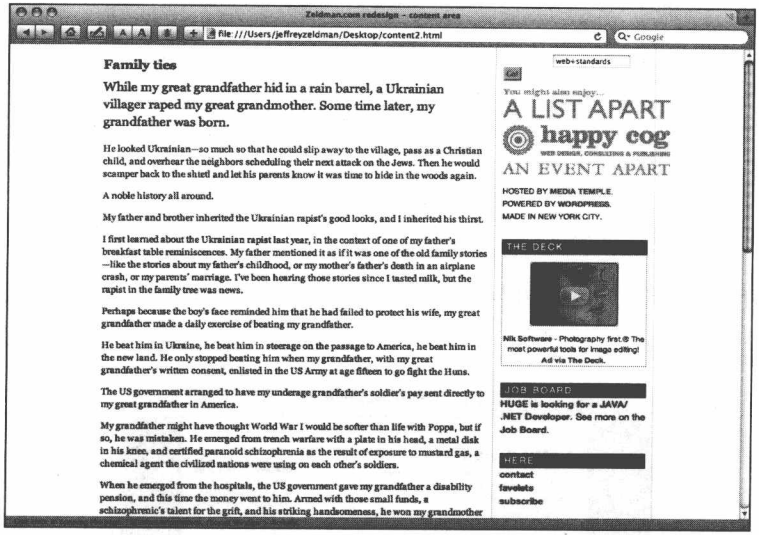


图 16.12　把真实内容放入布局，在设定字体之前，我们先设定边距和间隙

```
/* Set out the main layout divisions */

html {
min-width: 770px;

background: #f9f8f3;

}
```

```
div#wrapper {

color: #222;
background: #f9f8f3 url(/i/z3bg.gif) top left repeat-y;
 width: 770px;
margin: 0 auto;
padding: 0;

text-align: left;
overflow: visible;
min-height: 1000px;
}

div#maincontent {
float: left;
margin: 0;
padding: 0 30px;
width: 490px;
}

div#sidebar {
float: left;
margin: 0;
width: 180px;
padding: 0 20px 36px;

font-family: "Helvetica Neue",  Arial, Helvetica, sans-serif;

 }
```

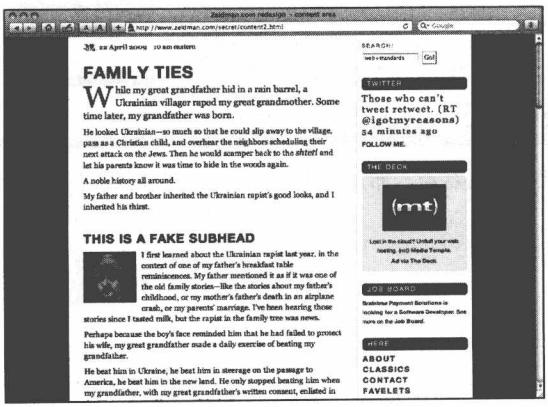


图 16.13　增加了颜色、间距，定义了字体（以及层次结构）以及一个简单的图形元素增加了品牌特征

我们给网站加了一个橙色的背景，并确定所有侧栏的内容将用 Helvetica Neue 字体显示，除非我们另外定义。这里没有太多的代码，每行都在“努力工作”。

说到 Helvetica Neue 字体，让我们确保整个站点都是这个风格，除非有另外的定义。我们将使用一个细线状图片（如图 16.14 所示）通过垂直重复它来填充背景，以避免当页面太长而链接速度又慢的时候背景闪烁。

```
body {
text-align: center;
margin: 0;
padding: 0;
border: 0;
background: #c30 url(/i/ff-flash-fix.gif) top center


repeat-y; /* this will fix the flashing */
color: #333;
font: 11px/18px "Helvetica Neue", Arial, Helvetica,


sans-serif;
}
```

Helvetica 字体用在标题、子标题和小的侧栏文本正好，但是我们想用一些更“书本化”的字体来显示大段文本，Georgia 字体是一个好的选择：

```
p {
margin: 0 0 9px 0;
line-height: 1.4;
font-family: Georgia, "Times New Roman", Times, serif;
}
```

图 16.14　用一个细长的背景图片，通过纵向重复规则给 wrapper 层添加背景颜色

我们通过标记定义元素并把 Grotgia 字体指定给这个元素。我们使用的这个模式很简单：先定义，然后设置风格。在绝对必要时，使用特定的选择器针对特定的部分。例如：一些列表项需要 Geotgia 字体，而其他部分需要 Helvetica 字体，我们使用派生选择器，以确保正确的元素获得正确的风格。

```
div#maincontent div.endpost ul li {
```

```
font-family: Georgia, serif;
list-style: square outside;
line-height: 18px;
padding: 0 0 0 6px;
margin: 0 6px 6px 15px;
}


div#maincontent div.endpost ol li {
font-family: Georgia, serif;
font-size: 14px;
line-height: 1.4;
}


div#sidebar ul li {
font: 13px/21px "Helvetica Neue", Arial, Helvetica,


sans-serif;
letter-spacing: 2px;
}
```

我们不仔细讨论 CSS 里的每条规则（你可以检查它在 www.zeldman.com/wp-content/themes/zeldman-v2/style.css），但是你应该了解这种思路。使用真实的内容和真实的标记，可以为我们明确段落、有序和无序列表、定义列表，以及其他一些需要风格化的结构内容，还可能需要定义他们的大小、高度、边距和其他要调整的细节。把这些细节用 CSS 表现得和用 In Design 或 Photoshop 画出来的草图一样就完成了。当然，与 Photoshop 不同的是，在 CSS 里有风格继承和级联。

首字母下沉

在书籍和杂志里，艺术总监往往设置一个章节或者一个文章的介绍段落字体比后面的段落更大一些。大字体更醒目，更容易将人们引入到故事中去。首字母放大并下沉一行，同样增加了醒目效果，有助于引起读者的的关注与参与，引诱她进入作者的文字中去。让我们来创建一个“intro”的段落样式，以及首字母下沉效果：

```
body div#maincontent p.intro {
font-size: 18px;
line-height: 1.4;
font-style: normal;
```

```
font-weight: normal;
}

/* IE7 and IE6 false value hacks on span.drop courtesy Paul of
hell.com */

span.drop {
display: inline;
float: left;
margin: 0;
padding: .25em .08em 0 0;
#padding: 0.25em 0.08em 0.2em 0.00em; /* override for

Microsoft Internet Explorer browsers*/
_padding: 0.25em 0.08em 0.4em 0.00em; /* override for IE

browsers 6.0 and older */
font-size: 3.2em;
line-height: .4;
text-transform: capitalize;
color: #c30;
}
```

下面是样式作用在标记上的代码：

```
<p class="intro"><span class="drop">T</span>his paragraph will
be bigger than the others, and the letter "T" will be styled as
a drop cap.</p>
```

我们真的需要一个样式为“intro”的段落吗？同样的问题，我们真的需要一个样式为“drop”的吗？Firefox、Safari 和 Opera 直接支持 CSS2.1 中 first-child、first-letter 这些伪类，那样我们可以直接可以将样式定义到元素上，而不需要增加额外的 class。哎，IE 8.0 之前的版本不支持，为了让 IE 用户也能体验到这盛行的风格，设计师必须为每个设计增加额外的 class。（当然，在某些项目中，这由客户说了算）。虽然我的读者大多数使用 Firefox 和 Safari，我还是决定用安全的方法增加额外的 class。这样做不仅仅是为了让 IE 7 的用户体会到我设计中的小小“乐趣”，也给了我机会来控制什么时候需要，什么时候不需要使用段落介绍和首字母下沉。

关于标准技巧（Hacks）

说到 IE，我们都不想说为什么它不能正确显示“首字母下沉”，需要特定技巧（hacks）才能让 IE 实现其他浏览器都能实现的效果。最可悲的是，我们宣称基于

标准的设计，却依然需要给不同的浏览器（主要是 IE）提供特定的代码。这样做的情况并不多，不再需要在前台或者后端脚本中增加标记。幸运的是，采用标准技巧（standards hacks）只需要调整样式表中的一两行，或者通过条件注释链接针对 IE 的单独样式表。

请注意，技巧（hacks）始终都应该提供注释，这样你或者你的继任者可以知道你为什么把它们放在那里，当坏版本的 IE 浏览器最终死亡的时候，可以将这些技巧移去。注释也应该提供是谁发现和解决了问题，并留下链接，以便其他开发人员在他们的网站使用你的方法遇到问题时候进行沟通。共享就是关怀。

16.2.3 基本模式

这个网站上有大量的字体设计细节，但它们都遵循同样的基本设计模式（或工作模式）。让我们来回顾一下：

1. 指定一个元素并设置风格；

2. 尽可能指定一个通用元素（例如：h2 要比 h2.subhead 或者 h2.bigredletterz 好）；

3. 如果你的风格出现在不应该出现的区域，例如：如果你的 h2 内容样式同时作用到了侧栏 h2 的子标题上，可以使用后代选择器来针对特定的元素，避免过多地使用 class：

```
maincontent h2 {font-size: 24px; }
```

16.2.4 页脚的创新

经过对网站主要内容布局和风格定义后，可以开始考虑页脚了。Derek Powazek（www.powazek.com）可能是第一个打破“不重要的东西都堆到页脚去”规则的设计师，他在自己的博客页脚大胆地放上丰富的内容。这种做法迅速得到了推广。Dan Cederholm（如图 16.15 所示）、Jason Santa Maria（如图 16.16 所示），以及 Veerle Pieters（如图 16.17 所示）这些现代设计师都在他们的博客采用了丰富页脚的做法。受他们的创意启发，我决定在我的网站也采用这种做法。很明显的好处是，我可以把原本放在侧栏的宣传内容和促销活动信息放到页脚，使用户获得更好的阅读体验。页

脚真是一个好地方，它允许我打破网格，用不同于中间博客布局的方法来设计，如图 16.18 所示。

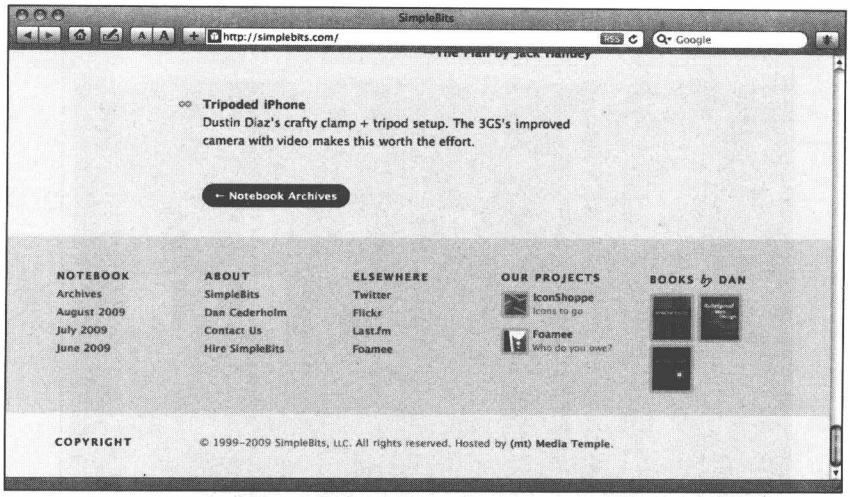


图 16.15　Dan Cederholm 网站的页脚（www.simplebits.com）

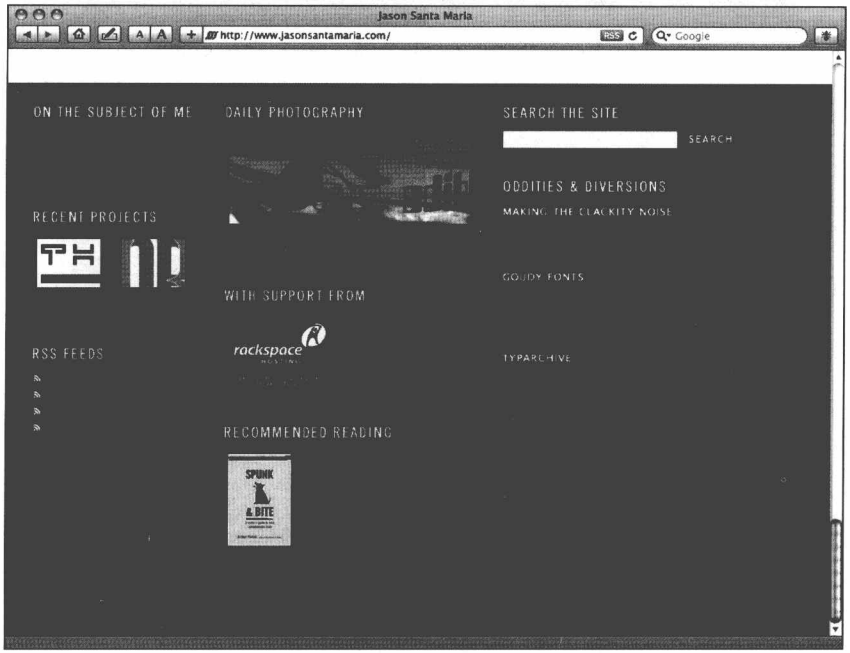


图 16.16　Jason Santa Maria 网站的页脚（www.jasonsantamaria.com）

我们的页脚分为两个部分：上面是两个带文字介绍的宣传横幅（一个美丽的页脚），下面是版权信息等（标准的页脚方式，包括地址元素，并可再用 HTML5 的方式进行处理）。

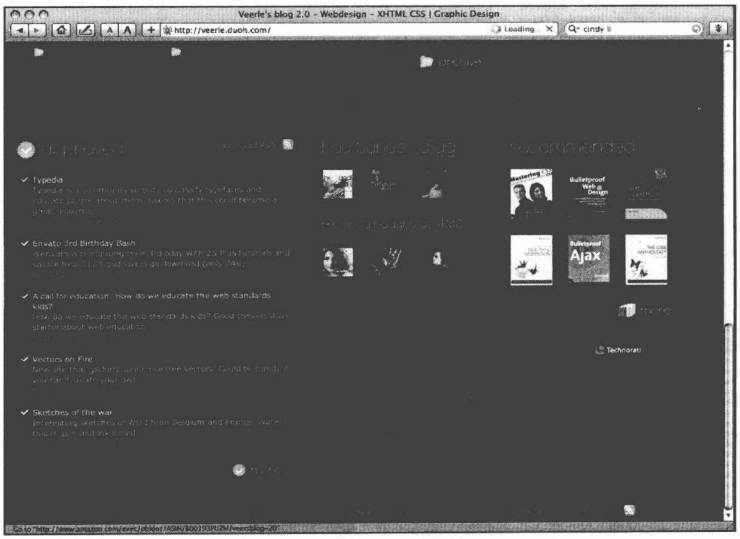


图 16.17　Veerle Pieter 网站的页脚（veerle.duoh.com）

图 16.18　zeldman.com 网站的页脚。

让我们看一下页脚的标记，从漂亮的部分开始（下面的例子已经简化，移除了 PHP、JavaScript 以及其他许多与讨论关系不大的内容。）

```
<div id="footer">


<div class="first">
<a href="http://www.aneventapart.com/" title="Daniel Mall makes
Flash and web standards play nice at An Event Apart, for people
who make websites.">
<img src="/secret/danmall.jpg" alt="Daniel Mall makes Flash and
web standards play nice at An Event Apart, for people who make
websites." /></a>
<p>Daniel Mall weds Flash to web standards at
<a href="http://www.aneventapart.com/">An Event Apart</a>,
for people who make websites. (Photo &copy;
<a href="http://www.flickr.com/photos/pete-karl/">Pete Karl
II</a>.)</p>
</div>


<div>
<a href="http://www.alistapart.com/" title="Now in A List
Apart, for people who make websites.">
<img src="http://www.zeldman.com/wp-content/themes/zeldmanv2/
i/ala.jpg" alt="Now in A List Apart, for people who make
websites." /></a>
<p>In <a href="http://www.alistapart.com/issues/290">ALA No.
290</a>, build respect for content strategy, Motown style;
produce modular, reusable JavaScript the MVC way.</p>
</div>
<div>
<a href="http://www.happycog.com/design/krylon/" title="Happy
Cog redesigns Krylon.">
<img src="/secret/krylon.jpg" alt="Krylon" /></a>
<p><a href="http://www.happycog.com/design/krylon/">
Happy Cog&#8217;s redesign</a> of Krylon, the aerosol paint
leader, puts creative projects and products front and center.</p>
</div>
<hr />
<div>
<a href="http://www.webassist.com/professional/etc/"
title="Improve your web markup with Jeffrey Zeldman's Web
Standards Advisor."><img src="http://www.zeldman.com/i/
webassist.jpg" alt="Jeffrey Zeldman's Web Standards Advisor"
```

```
/></a>
<p><a href="http://www.webassist.com/professional/products/
productdetails.asp?PID=255&amp;WAAID=889">Improve your markup</
a>. Validate HTML, CSS, microformats for single pages or entire
sites. Get advice on page structure, headings, ID, class, and
more. </p>
</div>
<div>
<a href="/talent/" title="Taking Your Talent to the Web - a free
downloadable book."><img src="http://www.zeldman.com/wp-content/
themes/zeldman-v2/i/talent.gif" alt="Taking Your Talent to the
Web" /></a>
<p>Rated Five Stars at Amazon.com since the day it was
published, this classic text on transitioning from print art
direction to web design is now a <a href="http://www.zeldman.
com/talent/">free downloadable book</a>.</p>
</div>
<div>
<a href="/dwws/" title="Designing With Web Standards - 3rd
edition coming soon."><img src="http://www.zeldman.com/Web Standards" /></a>
<p>Create search-friendly sites that load
faster, reach more users, and cost less to design, update, and
maintain. <a href="/dwws/">Read about the third edition</a>,
coming this year.</p>
</div>
</div>
```

哇，看起来真不少，但所有这些足有 6 页的代码都放在一个叫“footer”的容器里。下面是对应的 CSS，先扫一眼，后面会详细解释。

```
div#footer {
clear: both;
border: 0;
border-top: 1px solid #666;
border-bottom: 12px solid #09f;
background: #000;
color: #dddcd8;
width: 760px;
padding: 36px 0 12px 10px;
margin: 0;
}
hr {
clear: both;
visibility: hidden;
height: 1px;
```

```
}
div#footer p {
font: 11px/18px "Helvetica Neue", Arial, Helvetica, sans-
serif;
margin: 6px 0 21px 0;
}
div#footer p#credits {
clear: both;
background: transparent url(/i04/author.gif) center left
no-repeat;
line-height: 18px;
margin: 0 0 0 20px;
padding: 0 0 0 48px;
}
div#footer div {
float: left;
width: 210px;
padding: 0;
margin: 0 0 0 30px;
min-height: 190px;
}
div#footer div.pleasure {
width: 100px;
margin: 0;
padding: 0;
}
```

正如你看到的那样，footer 样式中的 clear 用来清除它之前的浮动元素（“maincontent”和“sidebar”）所继承下来的浮动属性，原因我们已经在第 10 章解释过。每个 footer 里的 div 都在 footer 里浮动，每个 div 宽是 210px，两个 div 之间的间距是 30px。让我们看其中一个 div：

```
<div>
<a href="http://www.alistapart.com/" title="Now in A List
Apart, for people who make websites."><img src="http://www.
zeldman.com/wp-content/themes/zeldman-v2/i/ala.jpg" alt="Now
in A List Apart, for people who make websites." /></a><p>In
<a href="http://www.alistapart.com/issues/290">ALA No. 290</
a>, build respect for content strategy, Motown style; produce
modular, reusable JavaScript the MVC way.</p>
</div>
```

当你只看其中一个 div 的时候，很明显，这是很简单的一段 HTML。我们放了一

个加了链接的图片，紧接着是一段解释的文字。这段文字之所以会在图片下面是因为包含它的 div 宽度限制。下一个 div 浮动在这个 div 的边上，第三个 div 又浮动在第二个边上；三个 div 填充了第一行。hr 元素清除了浮动，好让第二行开始。

我解释一下 hr 的语义，它用来创建一个水平线以分割上下两组相关的项目。水平线可以显示在非图形环境中，将两行（每行三个事物）略微分隔开。我想这是一个打印记号，传达了页面结构语义。

你可能不同意这个观点，为什么不用两个独立的 div 来分别包裹两行。这是大多数“标准人”会采取的方案，而我认为这属于滥用 div，使用 hr 更加简洁。当然，旧的设计方法还可以用表格的行来实现同样效果，但是我们都明白为什么自己不要那样做。我们可以很轻松地用 CSS 来实现任何表格布局，并对干净的、语义的、搜索引擎友好的标记进行风格定义。

接下来的两行是常见的页脚标记，不值得我们浪费时间来讨论。它通常包含了版权和作者信息，还有 HTML 和 CSS 的校验链接，那样当我在帖子里忘记关闭 li 的时候读者就会提醒我。我是从上世纪 90 年代开始在页脚加 HTML 和 CSS 校验链接的，虽然一些人认为这种做法古怪，我还是会坚持，直到大多数的网站站长意识到有效标记的重要性，直到大多数的网站校验有效为止。

16.2.5 刊头设计

刊头（如图 16.19 所示）主要关心的是它不是什么。本书的第 1 版介绍了 Fahrner 的图片替换法，一种显示是图片，但选择和访问是文本的技术。Mike Rundle 的 Phark 图片替换法（最出名的地方是-9999px 技巧）替代了 Fahrner 方法，它解决了可访问性问题（phark.typepad.com/phark/2003/08/accessible_imag.html）。过去，zeldman.com 刊头使用了这些图片替换技术，以使网站的名字使用 h1 标签，又可以在页面上展示 logo，在本书的前面版本也解释了这些技术的应用细节。Fahrner 方法导致了 phark 方法，Phark 方法又产生了 sIFR 方法，sIFR 又导致了 Cufon 方法，现在我们已经进入了真实字体时代。但是，这次改版，我简单地使用一张 GIF 图片作为我的刊头：

```
    <div id="header">
    <h1>
    <a href="/" title="Jeffrey Zeldman Presents: Web design news &amp;amp; information
since 1995">
```

```
    <img src="http://www.zeldman.com/wp-content/themes/zeldman-v2/i/masthead.gif"
alt="Jeffrey Zeldman Presents: Web design news &amp;amp; information since 1995." /></a>
    </h1>
    </div>
```

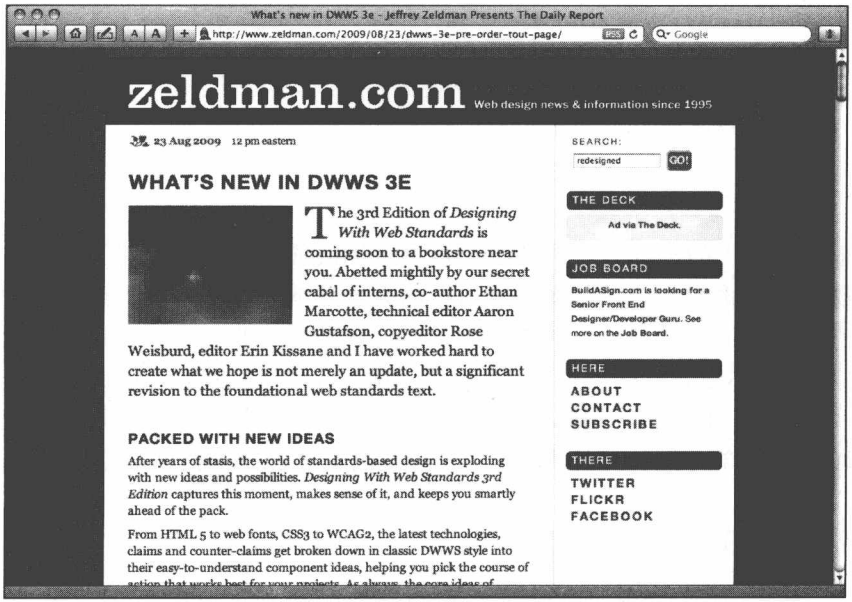


图 16.19　zeldman.com 的刊头（www.zeldman.com/x/57）

除了其他一些原因之外，使用图片替换方法还将使我的网站有能力为支持 CSS 的小屏幕的移动设备服务，但是，非智能手机往往不支持 CSS，又考虑到价格便宜的智能手机充斥市场，故选择老式的 GIF 图片，然后再用 alt 和 title 属性支持是明智的。

同样，先前的设计使用了 CSS 子画面技术（www.alistapart.com/articles/sprites）创建精美排版，只要几个简单的图片和一些 CSS 就可以实现多个动态的滚动导航。但是为了这次以阅读为核心的改版，我决定放弃那些漂亮的导航。

细节，细节

对于这个超简单、新复古的设计来说，它的一切都在于细节。我设置了行高和边距以加强垂直的基线网格，使用来自 Wilson Miner 的方法（www.alistapart.com/articles/settingtypeontheweb），通过简单标记和 CSS 设置作者评论和读者评论样式（如

图 16.20 所示）：

```
<li class="comment zeldman_speaks">

...

div#maincontent ol.commentlist li.zeldman_speaks {
background-color: #f9eae1;


}
```

有很多设计上的小细节。我通过公众测试期间的读者反馈完善了一些，并一直微调到现在。我很少会花这么多的努力在这么普通的事情上。话说回来，设计一个自己拥有和维护的网站是最快乐的，你可以持续查找和修正自己在审美和可用性上的失误。（如果你像我一样，关注设计的时间越长，就越能体会到）。这也是为什么 CSS 那么重要，当我选者了错误的背景颜色的时候，不用再悲哀自己缺乏美感，而是直接通过输入样式表来改变颜色，一个样式表就可以控制整个站点。

至少，直到我下一次的重新设计为止。

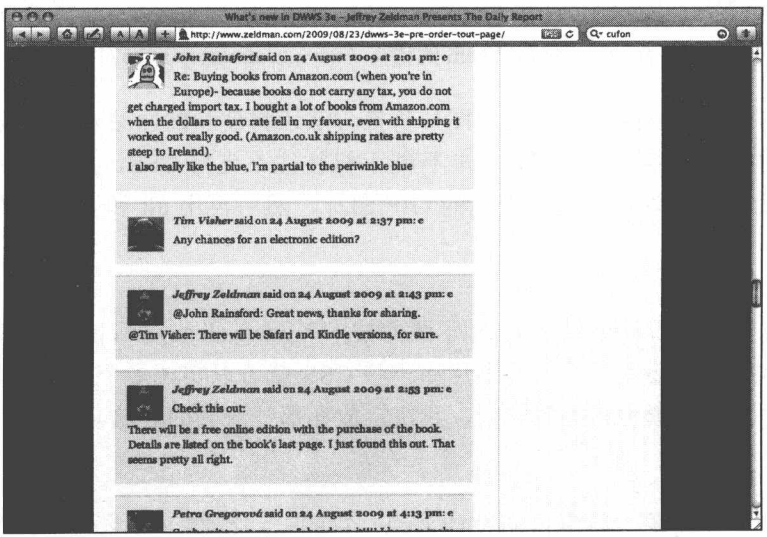


图 16.20　使用一个 class 和一个规则，就将“作者”评论和读者评论区分开来

第 17 章

NYMag.com：简单的标准，迷人的界面

1968 年由 Clay Felker 和 Milton Glaser 创办的“*New York*”是美国历史最悠久的生活时尚杂志之一。它的网站 www.nymag.com（如图 17.1 所示）编辑设计做得非常好：生动的配色和大胆的排版完美地配合着精致的杂志品牌，充满着个性的页面令读者赏心悦目。此外，网站界面的复杂程度在通常的网站编辑中也是很少见的；几分钟的察看就可以令好奇的读者领略到一些界面设计的思想，错综复杂却又很直观，内容很多却非常引人注目。

图 17.1　“*New York*”网站的首页（www.nymag.com）：丰富多彩，但清晰整洁

“*New York*”的“日程表”（www.nymag.com/agenda）证明了这个杂志通用的界面策略。这是由设计总监 Ian Adelman 和 nymag.com 的设计与用户体验团队创作的，“日程表”是一个看上去相当简单的应用：它是一个日历，由“*New York*”的文化编辑们经常更新，精选出未来一周的一些突出事件。不过，就像你点击“日程表”之后所看到的（如图 17.2 所示），设计师们并没有满足于创作一个简单的静态的日程表。取而代之的是，不同的表格列默认是收缩着的，隐藏着信息；直到用户点击了特定的单元格，相应的表格列才会展开，显示出用户需要的信息。这是一个相当聪明而直接的设计，在本章中我们将会看到，这么强大的功能仅仅是通过 Web 标准来实现的。

图 17.2 “日程表”，这是我们将要在本章中构建的

在本书的最后章节，我们不会像第 16 章那样深入地探讨另一个 CSS 布局，而是缩小一点我们的关注范围：我们并不构建整个页面栏目或建立一个版式，而是仅仅聚焦在重新创建自己的日程表。但是，这并没有降低难度。这个看起来简单的日程表隐含了一个驱动这个界面的复杂且标准化的框架，我们通过重新构建它可以学到很多东西。

17.1 检视内容

看一下杂志的设计团队提供给我们的样稿，可以看到出现在界面上的是一种复合文档：默认状态下，页面表格的第一列是展开的（如图 17.3 所示），不过当鼠标停留在

某个列上时，它的外观会发生改变（如图 17.4 所示）。最后，当点击某一个单元格的时候，那么其他的列都会收缩起来，而点击的那个列将会适当地展开（如图 17.5 所示）。

图 17.3　“日程表”初始的样子，第一列默认是展开的

图 17.4　当鼠标停留在列上时，设计上会有细微的变化

图 17.5 “点击一个列”会呈现出更多的信息。

显然，这要比我们之前创建的界面更复杂一点。不过，我们通常采用的策略正好可以帮助我们：在我们开始担心“日程表”界面的细节之前，需要先检视一下内容，以及建立我们的基础标记。让我们开始吧：使用什么样的 HTML 代码来构建一个日程表才是最合适的呢？

在了解到语义标记的价值前，我首先计划使用一堆嵌套的 div，每个单元格一个，然后用一堆的 CSS 来模拟日程表的行和列。或者，也许我会考虑用一个 ol 元素，它的 li 元素感觉上比 div 元素更具有语义性。然而，我最终放弃了这些想法，因为 HTML 规范已经有一个更健壮的选择提供给我们使用：那就是 table 元素。

我意识到，在第 16 章中自己反对过在现代页面布局中使用表格，因此上面的最后一句话看上去似乎就有点怪异了。也许你已经准备好火把和棍棒，冲进我的城堡，希望我能收回我的话。但是，如果可以的话，请你先息怒片刻：table 元素对于页面布局来说是一个糟糕的工具，在语义上是毫无意义的。而事实上，table 元素是设计用来标记表格数据的，诸如电子表单、会议日程表，是的，还有日程表。

也许你仍然有点怀疑我是心血来潮地吹嘘 table 元素。如果是那样的话，那么让

我们粗略地看一下样稿的内容类型，看看一个 table 元素能够多么好地实现我们的目的。首先，我们从一个空的 table 元素开始，并赋予一个描述性的 id。

```
<table id="agenda-week">
</table>
```

在样稿的顶端恰好有一行头部（如图 17.6 所示），当展开时，每个单元格呈现出相应列的日期。我们可以使用 thead，这是一个“行组”元素（http://www.w3.org/TR/html401/struct/tables.html#h-11.2.3），包含着一个由表头单元格（th）组成的行（tr）。说起来很拗口，但是标记却非常的直接：

```
<table id="agenda-week">
  <thead>
    <tr>
      <th scope="col">Today: 6/10/09</th>
      <th scope="col">Wednesday: 6/11/09</th>
      <th scope="col">Thursday: 6/12/09</th>
      <th scope="col">Friday: 6/13/09</th>
      <th scope="col">Saturday: 6/14/09</th>
      <th scope="col">Sunday: 6/15/09</th>
      <th scope="col">Monday: 6/16/09</th>
    </tr>
  </thead>
```

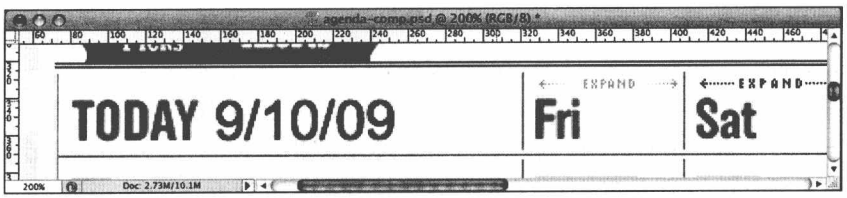


图 17.6　日程表的头部将很好地转换为一系列的 th 元素，并包裹在一个 thead 内

提示：th 元素的 scope 属性可以接受两个值，col 和 row，用来标示 th 是应用到数据列还是数据行上。关于表格的更多的语义秘诀，强烈推荐查阅 Zoe Gillenwater 所写的关于这个主题的文章（www.communitymx.com/content/article.cfm?cid=0BEA6）。

在日程表的底部，有一行单元格是链接到所有事件的（如图 17.7 所示）。我们可以使用另一个行组元素，tfoot，它简单地包含了一个排满了 td 元素的 tr 元素。不过，有一个警告：与它的名字（和排版位置）所暗示的有点差别，tfoot 元素在一个合法

的 table 中是应该紧跟在 thead 后面的：

```
</thead>
<tfoot>
  <tr>
    <td>All of Today's Picks</td>
    <td>All Wednesday Picks</td>
    <td>All Thursday Picks</td>
    <td>All Friday Picks</td>
    <td>All Saturday Picks</td>
    <td>All Sunday Picks</td>
    <td>All Monday Picks</td>
  </tr>
</tfoot>
```

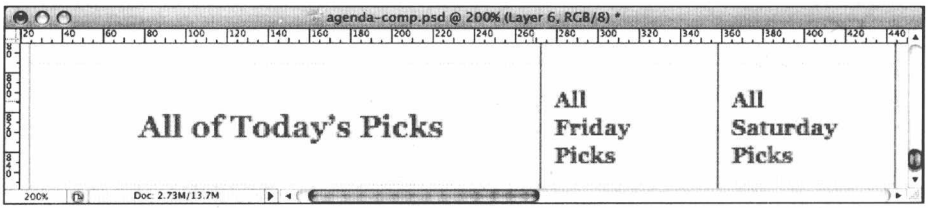


图 17.7　一个底部行变成一个 tfoot。这些名字很好记忆，不是吗

不用担心：桌面浏览器会正确地把 tfoot 的内容定位在包含它的表格的底部，不须要增加额外的 CSS 技巧。

日程表的头部和底部都已经转换到 table 里了，只剩下那些事件内容——“日程表”的主体内容——需要转换成 HTML（如图 17.8 所示）。我们将使用第三个，也是最后一个行组元素，tbody，它包含着每一个显示着相应行事件内容的 tr 元素。而这些行将包含对应着每天内容的 td 元素，就像这样：

```
</tfoot>
<tbody>
    <tr>
      <td>Event</td>
      <td>Event</td>
      <td>Event</td>
      <td>Event</td>
      <td>Event</td>
      <td>Event</td>
      <td>Event</td>
```

```
    </tr>
    <tr>
      ...
    </tr>
  </tbody>
</table>
```

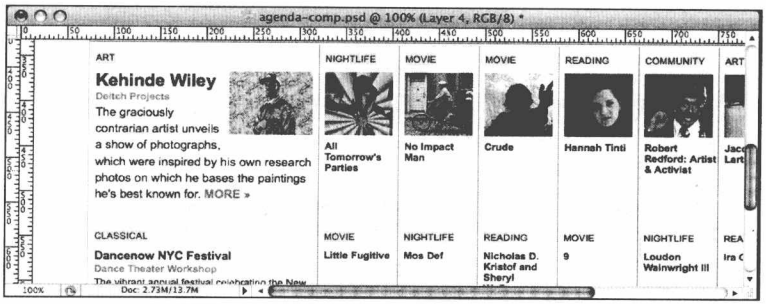


图 17.8　对于表格的主体，我们将使用——就像你猜测的那样——tbody 元素

如果曾经使用过 table，那么你可能还记得我们用在这个日程表里的核心元素（tr、th 和 td）。而那三个行组元素——thead、tfoot 和 tbody——你可能不怎么熟悉，不过它们对于结构化我们的表格是非常需要的，这使得我们的 CSS 可以很容易地区分从头部到底部的各个单元格，而无需用额外的 class 或 id 来标识我们的标记了。但是，更重要的是，它们在语义上是最适合描述我们的内容的，而且证明了 table 确实是最合适在这里使用的元素。

在粗略地构建好我们的表格的基础结构之后，我们的工作就差不多准备开始从标记转到 CSS 上了。但是，到底我们应该如何进行样式化才能实现像这个日程表一样的动态效果呢？

从内容检视到实现策略

为了回答这个问题，我们需要重新仔细地看一下设计稿。不过，这次我们检视的方面会有一点不同；并不是考察不同的内容类型，而是把不同的视觉状态分一下类，以便于我们能更好地理解用来样式化它们的 CSS。在这里我们不关心“如何”去改变这些状态——当一个用户点击了一个单元格之后如何去展开一个列，或者如何改变背景颜色——而只关注每个部分的设计细节。

让我们来仔细看一下。

1. 从顶部开始，我们可以看到头部的单元格有三种基本状态（如图 17.9 所示）。“默认”状态只显示星期几，一种非常淡的蓝色背景衬托着一种可爱的字体。在单元格的顶部有一个非常小的“EXPAND”图形，为用户提供了一个非常好的视觉提示。但是在第二种状态——“悬停”状态——中，它的背景图片明显地变暗了，当中的字距变得更宽了。而单元格的背景也大大地提亮了，从粉蓝色变成了完全的白色。最后，当用户点击了一个单元格之后，就进入了“展开”状态：仍然是白色的背景，但是“EXPAND”的提示消失了，而令人愉悦的大字体日期出现了。

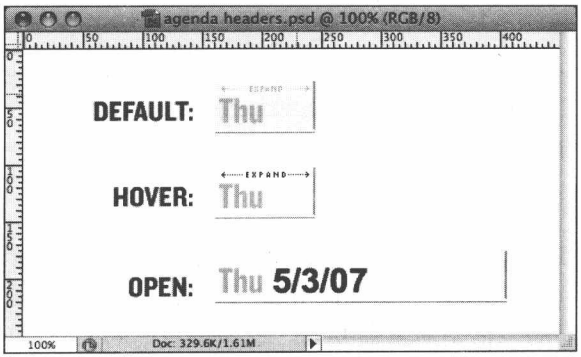


图 17.9　日程表头部的不同视觉状态

2. 再看底部，这三种基本状态仍然适用（如图 17.10 所示）。默认显示的是一种亮粉蓝色背景，带有左对齐的链接文字。跟之前的一样，当用户的鼠标悬停在列上时，背景会变亮。最后，当单元格处于展开状态时，链接文字明显地变大了，并且水平居中。

3. 日程表的事件部分包含了很多信息（如图 17.11 所示），但是我们仍然可以处理成三种基本模型。具有跟之前相同的背景颜色切换，不过在默认和悬停状态下，只显示了事件的种类（“Reading”、“Play”、“Movie”等）和标题。而在单元格的展开状态下，所有事件的信息都显示出来了：地点和简介，还有一个“MORE”的链接，并且标题也明显变大了。

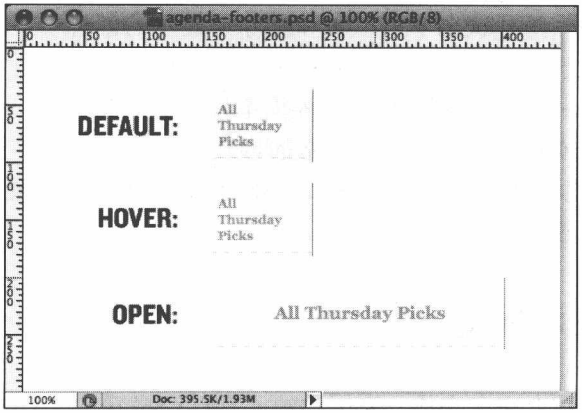


图 17.10　底部单元格的不同视觉状态

4. 在第一行，事件的风格有略微变化（如图 17.11 所示）。默认和悬停状态下，图片是在事件的种类和标题之间的。然而，当单元格处于展开状态时，图片浮动到了右边。而且，展开版本的事件标题的字体尺寸明显地要比其他行的事件标题大得多。

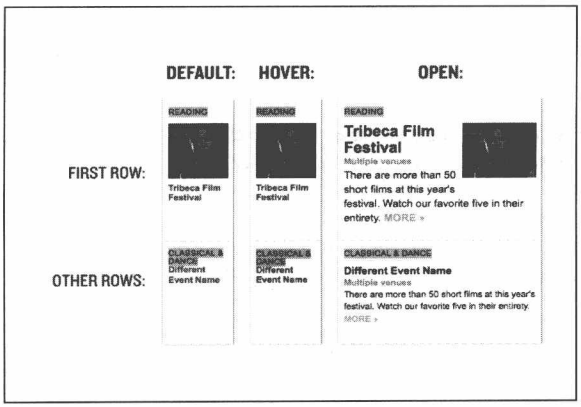


图 17.11　事件部分的不同样式。注意第一行的事件与其他行之间有稍微的不同

因此，无论从哪个部分查看日程表，我们都能处理成三种基本的视觉状态：默认、悬停和展开。当然，在每种状态下，日程表的头部、底部和主体之间都有很大的变化，但是这三种状态模型是完全相同的。

除了底部之外，两种状态下（默认和悬停）显示的信息明显要比相应的展开状态下少。对此我们已经找到了实现策略：我们的表格标记将包含“所有可能会在某个时刻被用户显示出来的内容”。然后，根据单元格所处的状态，我们使用 CSS 选择性的隐藏、显示，或者仅仅对内容设定新的样式。因此，每个单元格里的(X)HTML 都已经为展开状态准备好了，我们将使用 CSS 来改变 HTML 的显示，以匹配我们的设计，对应默认和悬停状态。

17.2 再次深入标记

我们的表格还没有完全完成。高层次的元素已经写好，我们的 th 和 td 中的内容还很简陋。因此，现在让我们进行三个不同类型的单元格的内容检视，并让这些标记更智能一些。根据我们新的策略，我们将仅考察展开状态下的 XHTML 代码；在我们完成标记代码之后，CSS 将负责实现另外两个状态模式。

更仔细地看一下我们的设计（如图 17.12 所示），我们可以看到头部的展开状态有两个元素需要我们包含到标记中：蓝色的星期图标和相应的日期，这是 CSS 样式的 HTML 文本。按照这个清单，让我们书写相应的标记：

```
<thead>
  <tr>
    <th scope="col">
      <h3><img src="day-today.png" alt="Today:" />
      <b title="May 1, 2007">5/1/07</b></h3>
    </th>
    <th scope="col">
      <h3><img src="day-wed.png" alt="Wednesday:" />
      <b title="May 2, 2007">5/2/07</b></h3>
    </th>
    <th scope="col">
      <h3><img src="day-thu.png" alt="Thursday:" />
      <b title="May 3, 2007">5/3/07</b></h3>
    </th>
    ...
    <th scope="col">
      <h3><img src="day-mon.png" alt="Monday:" /> <b title="May
      7, 2007">5/7/07</b></h3>
    </th>
  </tr>
</thead>
```

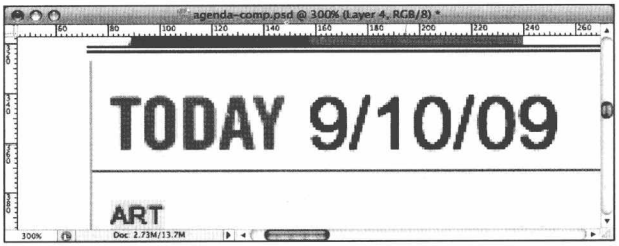


图 17.12　进一步仔细查看展开状态的头部，可以看到它包含了一张图片和一个日期，我们最好更新一下标记来匹配它们

看一下 th 元素本身，我们知道星期的标记需要一张图片，因此我们照做：把每一个星期标记转换成一个 img 元素，并且带有包含着图片文字的 alt 属性。然后，对于日期本身，我们把它包裹进一个无语义的 b 元素中；下面的章节中我们会看到，在我们的标记中需要一个特别的钩子以便让 CSS 能够区分出文字和旁边的图片。最后我们把内容都包裹进 h3 元素。毕竟，它是一个标题，在页面源码中前面还排有 h1 和 h2。

提示：你可能会问，为什么不使用一个 span？不错，对于我们的日期，span 是非常合适的选择，因为我们寻找的正是一个没有语义性的标记。不过我通常先使用 b 或 i 标签，仅仅是因为它们的字母更少。转换为更少的代码，可以为你的客户和用户节省更多的带宽。

通常，我们不会担心我们的内容需要看起来怎样，而是怎样用最好的方式描述它们；我们为设计选择最具有适当语义性的标记。对于底部，语义性就更直接了。在每一个 tfoot 的单元格里，我们简单地把文字包裹在一个链接中：

```
<tfoot>
  <tr>
    <td><a href="#">All of <i>Today&#8217;s</i> Picks</a></td>
    <td><a href="#">All <i>Wednesday</i> Picks</a></td>
    <td><a href="#">All <i>Thursday</i> Picks</a></td>
    <td><a href="#">All <i>Friday</i> Picks</a></td>
    <td><a href="#">All <i>Saturday</i> Picks</a></td>
    <td><a href="#">All <i>Sunday</i> Picks</a></td>
    <td><a href="#">All <i>Monday</i> Picks</a></td>
  </tr>
</tfoot>
```

tfoot 完成后，让我们处理事件内容，这是日程表的主体。

```
<td>
  <h4><b>Reading</b></h4>
  <dl>
    <dt><a href="#"><i><img src="img/pics/girl.jpg" alt="" />
    </i>Tribeca Film Festival</a></dt>
    <dd class="venue">Multiple venues</dd>
    <dd class="summary">
    <p>There are more than 50 short films... <a class="more"
    href="#">More &raquo;</a></p>
    </dd>
  </dl>
</td>
```

看一下设计稿（如图 17.11 所示），我们看到事件内容——不管它们是在第一行中突出地显示，还是处于随后的行中——都有着相同的类型。对于事件内容的处理策略是，我们把它放在一个 h4 元素中（和另一个令人不快的无语义的 b 元素一起），因为它在层次上低于 thead 中的 h3 元素。而对于事件内容本身我们选择使用了 dl 元素：dt 列出了事件的标题，包含着地点和事件简介信息的 dd 元素使用了不同的描述性的 class。

到此，我们的表格终于完成了。我们不仅通过 thead、tfoot 和 tbody 元素形成了主要的结构，而且同样地对它们内部的内容使用了适当的标记。当然，我们的表格还不够漂亮（如图 17.13 所示）。不过，随着这些简单的标记代码的最终完成，我们就可以开始应用适当的 CSS 了。

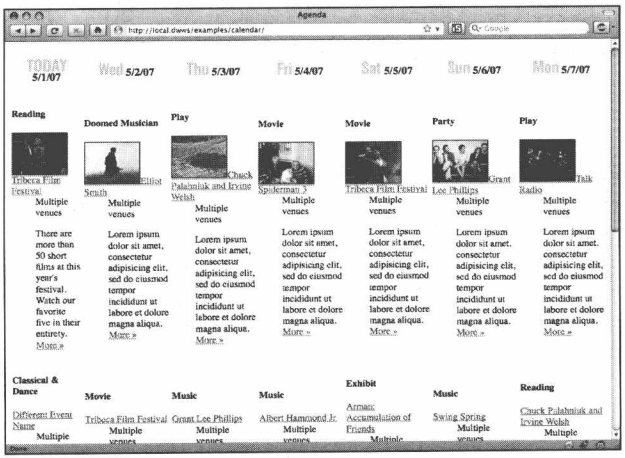


图 17.13　完全语义化的日程表，但是还没有应用样式。不漂亮吗（是的，不够漂亮）

17.3　开始应用样式表

首先，让我们先设置一些基础的设计样式：

```
body {
  background: #FFF;
  color: #000;
  font: normal 100% Arial, Helvetica, Geneva, Verdana,
  sans-serif;
}
a {
  color: #05A7D4;
  text-decoration: none;
}
a:hover {
    text-decoration: underline;
}
a.more {
  font-weight: bold;
  text-transform: uppercase;
  white-space: nowrap;
}
#agenda-week {
  border-collapse: collapse;
  clear: both;
  font-size: 0.625em; /* 10px / 16px = 0.625em */
  margin-left: 2px;
  width: 764px;
}
#agenda-week p {
    margin-top: 0;
}
```

我们通过这几条短短的规则已经大大地改变了页面的外观（如图 17.14 所示）。我们为 body 元素设置了默认的色彩和字体特性，这将被文档中所有的其他元素所继承（除非重载它们）。我们对文档中的链接也设置了 color 和 text-decoration 属性，这也将应用到整个设计中。

在上面的代码块最后，我们把视线缩小到单个的#agenda-week 上。我们为 table 元素自身设定了一个相对的字体尺寸，这是相对于 body 的 font-size 为 100%来计算的，粗略估计是 10px。不过更重要的是，我们已经对表格设置了基本的约束：限制

宽度为 764px，具有一定的外边距，并且清除之前所有的浮动状态。

提示：我们声明了 0.625em 的规则，并不是因为我们想设定字型为小字体（如图 17.14 所示），而是因为我们不擅长复杂的数学计算：以这个设定为基准，我们可以很容易地计算出相关的字体大小。（想了解更多这方面的知识，请查阅第 13 章的关于 62.5%技术的讨论。）

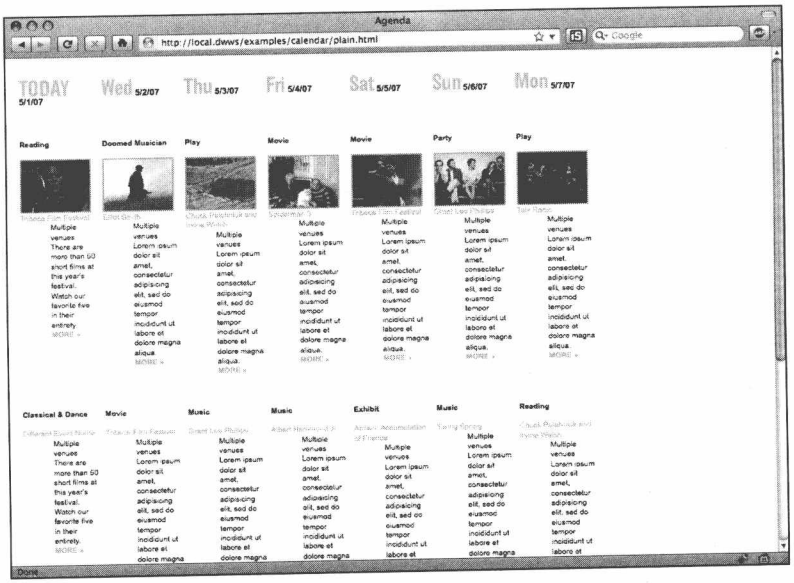


图 17.14　少量的 CSS 就使我们的表格看上去漂亮多了

现在，很明显这个初步样式化的表格还要做更多的工作才能匹配我们的设计。在这些基本设定的基础上，我们再次看一下之前所分类的三种不同状态：默认、悬停和展开状态。让我们先实验一下，在表格的每一行的第一个单元格上加上名为“open”的 class 属性，不管它是否在 thead、tfoot 或 tbody 中。

```
<table id="agenda-week">
  <thead>
    <tr>
      <th class="open" scope="col">...</th>
      <th scope="col">...</th>
     <th scope="col">...</th>
      ...
```

```
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td class="open">...</td>
      <td>...</td>
      <td>...</td>
      ...
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td class="open">...</td>
      <td>...</td>
      <td>...</td>
      ...
    </tr>
    ...
  </tbody>
</table>
```

现在把下面的规则加到样式表中：

```
#agenda-week th,
#agenda-week td {
    padding: 1em 1px 1em 7px;
    text-align: left;
    vertical-align: top;
    width: 72px;
}
#agenda-week th.open,
#agenda-week td.open {
    width: 242px;
}
```

第一条规则设定了我们日程表中 th 和 td 元素的一些基本样式——默认状态下。然后我们建立了一个特定规则，在所有 class 为“open”的单元格上应用一个更宽的宽度。

虽然只是一个微小的改变（如图 17.15 所示），但是最后我们得到了完成其余 CSS 的实现模式。我们可以把那些不同的状态模式转换为 class 名称，由此可以控制 th 和 td 来有条件的改变那些单元格的显示。为此，我们加上下面的两条规则：

```
#agenda-week th h3 b {
  left: -1000em;
  position: absolute;
}
#agenda-week th.open h3 b {
   position: static;
}
```

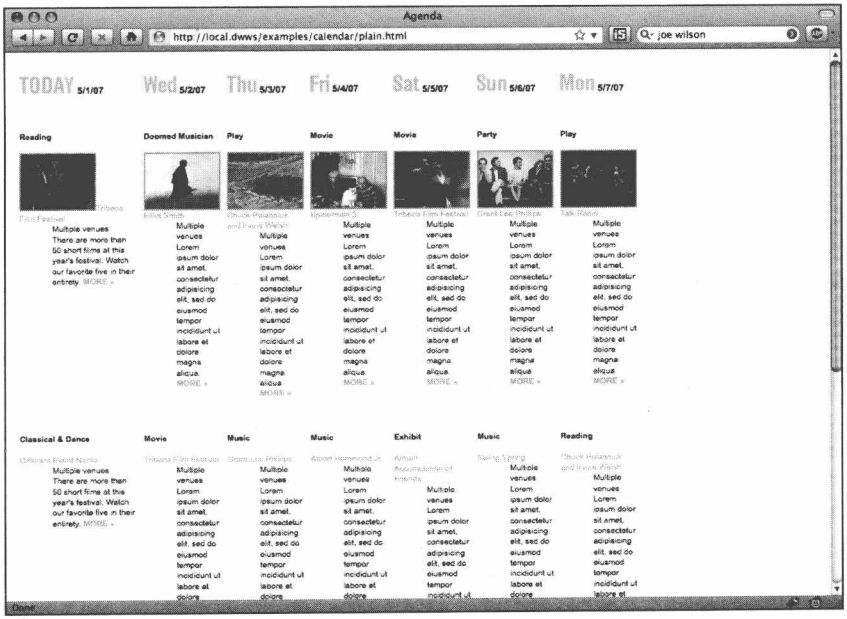


图 17.15　转换三个视觉状态为 class 名称，我们就可以改变表格中某个部分的表现

一点细小的改变，就能实现这个效果（如图 17.16 所示）。我们为包裹在 thead h3 中的无语义 b 元素设置了一个默认规则：把它们向左移出到浏览器窗口的外面，有效地隐藏了起来。不过，当“open”class 被应用到一个单元格时，就像我们在表格第一列中的一样，元素的定位信息被去掉，日期被带回到视图中。

我们完全可以把这个可视开关应用到日程表的事件部分。记得每一个事件都是一个 dl 元素，并且 dd 元素中是地点和简介信息。当我们想控制默认状态下的那些 dd 的显示的时候，可以简单地增加最后几行 CSS：

```
#agenda-week th h3 b,
  #agenda-week td dd {
  left: -1000em;
  position: absolute;
}
/* "Active" elements */
#agenda-week th.open h3 b,
#agenda-week td.open dd {
    position: static;
}
```

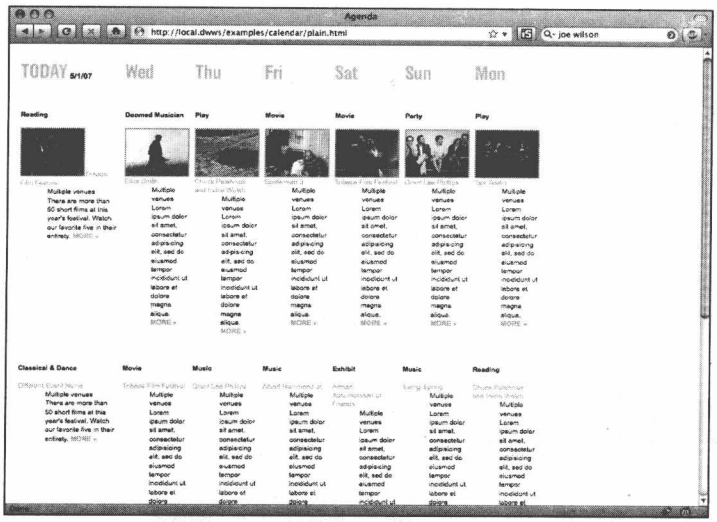


图 17.16　在默认状态下我们隐藏了日期，而在标示为“open”的单元格里则会显示出来

看，我们的日程表又有了一点进展（如图 17.17 所示）。用一个 class 作为开关，现在我们通过简单的样式规则就可以控制每个事件部分的地点和简介信息的显示了。

有了方法，其他就简单了

还需要为日程表设定更多详细的字体规则，但是我们没有那么多时间全部列在这里，因此，让我们快速地过一遍这上百行的 CSS。我们为单元格的边框加上了背景图片，为 thead 加上了“EXPAND”图标，还进一步调整了页面的排版。当我们把“open”class 应用到第一列的单元格上后，我们的模板最终匹配了期望的设计：展开的列中，事件信息显示出来了，顶部的日期也是一样，列底部也适当地居中了。而另一些没有 class 的列没有影响（如图 17.18 所示）。当然，如果我们去掉 class，那么

表格就根据我们所定义的样式规则而变回默认模式（如图 17.19 所示）。

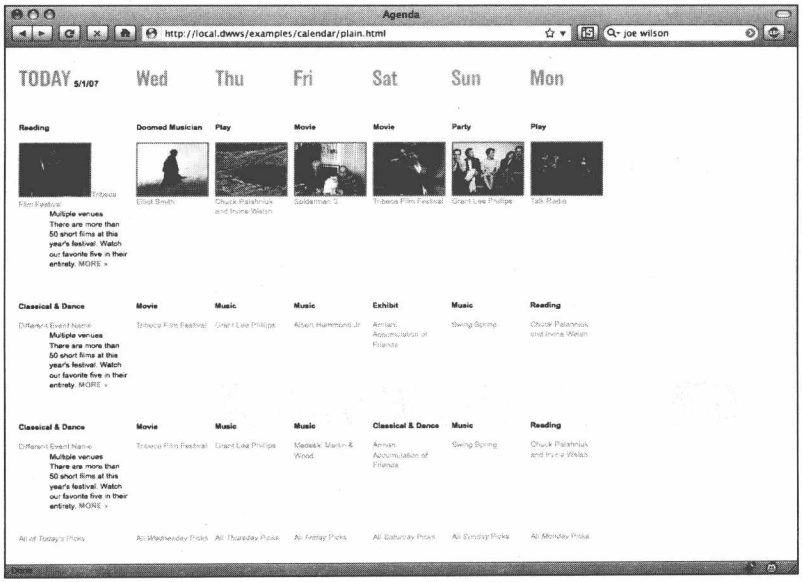


图 17.17　我们的事件部分最终开始符合我们的设计了

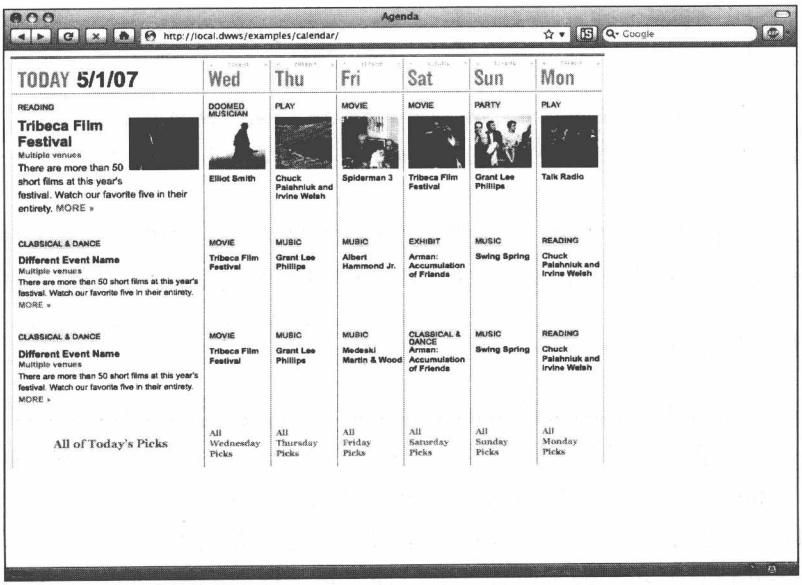


图 17.18　在 CSS 中加上不同的字体规则后，我们看到基本的 class 开关仍然保持得很好

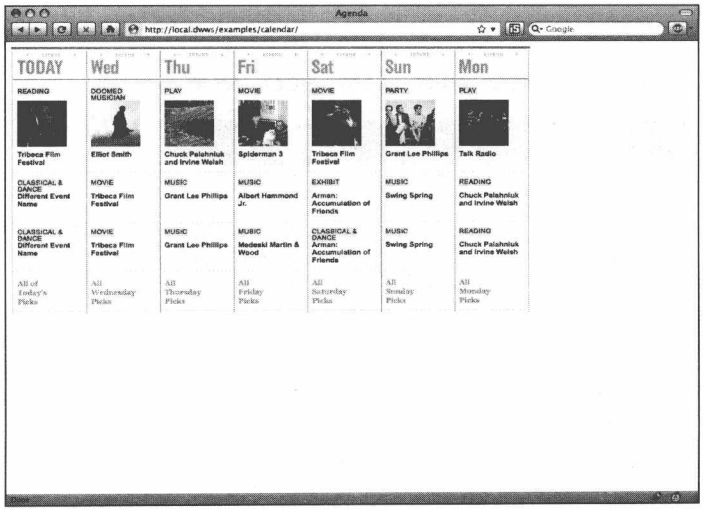


图 17.19　然而，如果我们去掉 class 的值，看上去肯定就没那么有趣了

进一步，我们把这个开关包含进一个“over”class，它对应设计中的悬停状态；把它加到每一个单元格，就可以很大地提亮我们的表格（如图 17.20 所示）。我们还可以把“open”class 应用到表格的每一个单元格上，结果肯定是不怎么好看的（如图 17.21 所示）。

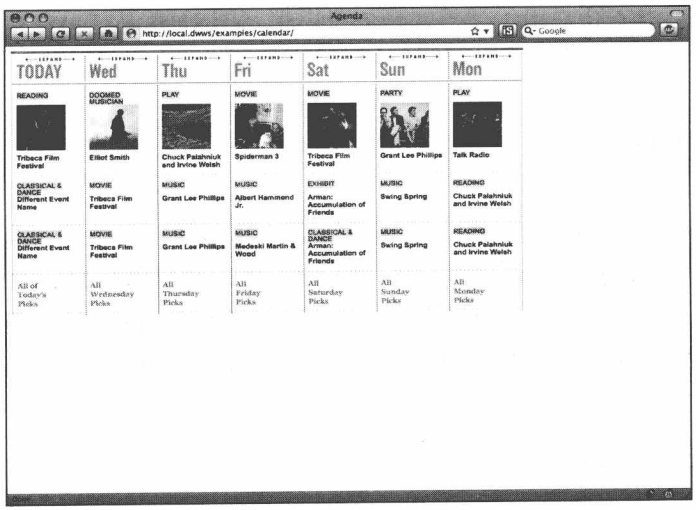


图 17.20　给每一个单元格加上“over”class，应用相应状态的 CSS 规则

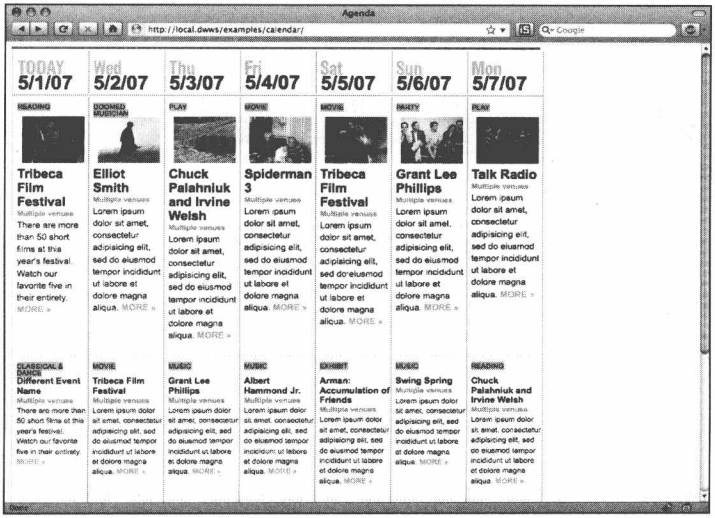


图 17.21　如果我们给每个单元格加上 class="open"，那么它们全部都展开了

那么，如果我们的样式检视都做完了，为什么不庆祝一下呢？（或者，至少结束本章？）虽然我们的三路 class 开关工作得很好，而且表格看上去也相当漂亮，但是我们是手动把 class 加到日程表的单元格上的。这可不是我们的理想状况，交互上还远没有做到：我们需要一些什么样的机制才能在用户把鼠标悬停在列上时把“over” class 加上去呢？或者在点击一个单元格后可以把它的 class 从“over”变为“open”呢？我们如何动态地改变这些标记呢？

17.4　使用 DOM

如果 DOM 是一个超级英雄，那么我们一定会为他的及时营救而高唱赞歌。虽然在进行描述和样式化我们的内容时（X）HTML 和 CSS 是我们必选的武器，但是它们面对交互却捉襟见肘。因此，我们需要依靠通过 JavaScript 来使用 DOM 脚本，它将使我们不仅可以侦测到用户悬停或点击某个单元格，而且还可以修改相应单元格的 class 名称。

17.4.1　结识“colgroup”

然而，我们并不想只修改个别的单元格。鼠标悬停在一个 td 或 th 上将提亮整个

列上的其他单元格，点击一个单元格也将展开这个列。因此，虽然我们表格的 XHTML 已经足够完美了，但是还需要编辑少量的标记，以使我们的 JavaScript 能够辨识出某个单元格与某个列之间的关系。

```
<table id="agenda-week">
  <colgroup>
    <col id="day-today" title="Today's events" />
    <col id="day-wed" title="Wednesday's events" />
    <col id="day-thu" title="Thursday's events" />
    <col id="day-fri" title="Friday's events" />
    <col id="day-sat" title="Saturday's events" />
    <col id="day-sun" title="Sunday's events" />
    <col id="day-mon" title="Monday's events" />
  </colgroup>
<thead>
```

在 thead 元素之前，我们加上了一个 colgroup 元素（www.w3.org/TR/html401/struct/tables.html#h-11.2.4），它包含了七个 col 元素——依次对应我们表格的每一列。表格自然地排列为水平的行（tr）和单元格（tr 或 td），而 colgroup 和它的子元素 col 让我们为那些单元格组建为一个个垂直的列。我们为每个 col 增加了一个 title，这样做是为了获得一个可访问性的特性：如果一个用户使用一个阅读器来访问我们的日程表，那么他们就可以运行一个命令来朗读出列的 title，而无论当前阅读的单元格嵌套得有多深。

然而，我们仍然需要把单元格与它们对应的 col 元素关联起来。为了实现这个目的，我们为表格中的每个 th 和 td 增加了一个 headers 属性，并依次设置了该单元格所属的 col 的 id。

```
<thead>
  <tr>
    <th class="first" scope="col" headers="day-today">
<h3><img src="img/day-today.png" alt="Today:" />
<b title="May 1, 2007">5/1/07</b></h3>
</th>
<th scope="col" headers="day-wed">
<h3><img src="img/day-wed.png" alt="Wednesday:" />
<b title="May 2, 2007">5/2/07</b></h3>
</th>
<th scope="col" headers="day-thu">
...
```

这是我们对已有的标记所做的仅有的修改，而且丝毫不会影响显示。headers 为给定的一个列里的所有单元格之间建立了一个垂直的关系，从而进一步加强了表格的语义性。至此，我们的脚本编辑工作终于可以开始了。这些 headers 属性将允许我们的 JavaScript 在用户与一个单元格发生交互时快速遍历表格，然后更新该列的所有单元格的 class，比如“Wednesday”列。那么让我们去除标记中所有的 class，把我们的表格设置回默认的状态（如图 17.19 所示），然后开始 JavaScript 的工作。

17.4.2 使用 jQuery

当“日程表”起初创建时，使用了大量自定义的 JavaScript 来实现交互功能。而为了这个章节的论述（和引起你的关注），我们将使用 jQuery 库（www.jquery.com）来重新实现相同的功能，还记得吧，先前在第 15 章中讨论过 jQuery 库。让我们把下面的代码加入到 HTML 文档的 head 元素中：

```
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/scripts.js"></script>
```

在第一个 script 元素中，我们把下载下来并存放在 js/目录下的 jQuery 库文件链接进来。第二个文件，scripts.js 当前还是空的，不过我们将在那里书写所有的日程表脚本。

提示：Google 实际上提供了几个流行的 JavaScript 库的免费存放服务，包括 jQuery。所以，如果你觉得自己在非常频繁地使用 jQuery 工作，那么也许可以考虑直接链接到这个文件，就像这样：

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/
jquery/1.3.2/jquery.min.js"></script>
```

链接到 Google 的服务可以节省你的带宽消耗，而且也帮助了你的用户：如果他们访问的另一个网站也使用了 Google 存放的 jQuery 版本，那么它可能已经缓存在他们的浏览器中了，因此也就使他们避免额外地从你的站点上下载了。想了解更多的关于 Google 的 JS 存放服务的信息，请查阅 code.google.com/apis/ajaxlibs/documentation。

我们的 jQuery 库和 scripts.js 准备好之后，让我们打开后者，加入下面几行代码：

```
$(document).ready(function() {
```

```
    var aCells = $("#agenda-week th, #agenda-week td");
});
```

代码的第一行（$（document）.ready（...））简单地指示 jQuery 在页面的 DOM 完成浏览器的渲染之后，去执行圆括号里的所有代码。到那时，JavaScript 将要做的事情就是声明一个 aCells 变量，用 jQuery 的类 CSS 选择器语法（$("#agenda-week th, #agenda-week td")）来建立一个包含了表格中所有 th 和 td 元素的列表。把它们存储到一个变量中，以备将来可以简便地引用。不过，我们并没有对这个变量做什么实际的操作，我们仅仅为了以后的使用而声明了它。刷新页面也不会产生什么显著的变化，我们的页面仍处于默认状态，它的列仍然是收缩着的（如图 17.19 所示）。

一旦这个单元格的集合存储在内存里，我们就可以用 JavaScript 来操作它们。让我们开始简单地编辑 scripts. js 文件：

```
$(document).ready(function() {
  var aCells = $("#agenda-week th, #agenda-week td");
  aCells.filter("[headers=day-today]").addClass("open");
});
```

在这新加的一行中，我们让 jQuery 去选择出单元格集合（存储在 aCells 变量中）的一个子集，它们都有一个设置为“day-today”的 headers 属性（aCells.filter（"[headers=day-today]"）]。一旦那些单元格被挑选出来，我们就给其每个元素加上 class=“open”（ .addClass（"open"））。如果我们刷新页面，变化是显著的。jQuery 尽职地完成了我们的指令，给每个 headers 值为“day-today”的单元格增加了一个“open”的 class，我们表格中的第一列根据 CSS 中对于“open”的规则而展开着（如图 17.22 所示）。这也许看上去跟我们早前的 class 实验时的显示效果是一样的，但是请记住：class="open" 实际上在我们的标记里是不存在的。我们是使用 JavaScript 在 table 里遍历找到特定的 headers 值，然后把 class 应用到相应的标记上。

用一行代码，我们就拥有了一个适当的策略来完成我们的 JavaScript 了。我们的 aCells 集合包含了表格中所有的单元格，每个列的单元格都按语义适当地设置了与该列相符的 headers 属性。因此，当一个访问者使用鼠标与单元格发生交互时，我们可以获得它的 headers 属性，以此来搜索出该列的所有其它的 th 或 td 元素。一旦我们找出所有与 headers 属性值相匹配的单元格后，就可以使用 JavaScript 来改变它们相应的 class。让我们开始定义当用户把鼠标悬停在一个单元格上将发生的事情。

```
$(document).ready(function() {
```

```
  var aCells = $("#agenda-week th, #agenda-week td");
  aCells.filter("[headers=day-today]").addClass("open");
  aCells.mouseover(
    function() {
      aCells.removeClass("over");
      aCells.filter("[headers=" + $(this).attr("headers") +"]").addClass("over");
    }
  )
});
```

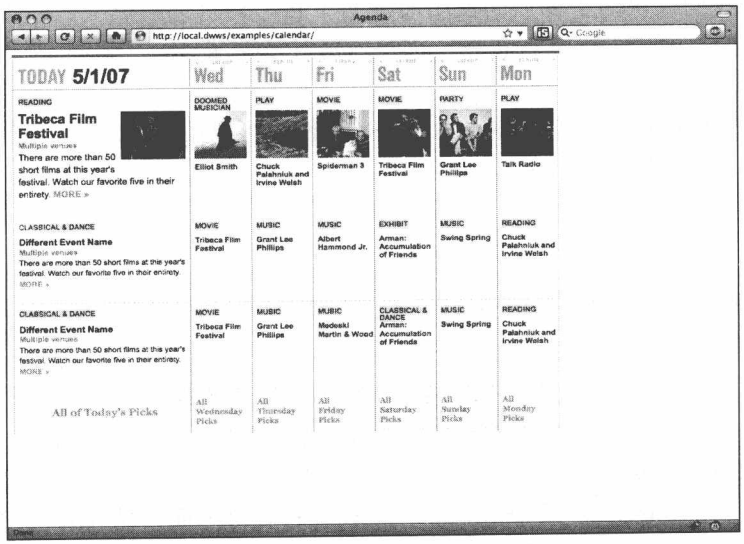


图 17.22　虽然这看上去像早前的那个展开的日程表，但这个却是在背后操作 DOM 所实现的。

在这里，我们为 aCells 数组附加上一个事件处理器，aCells.mouseover（...），不管用户什么时候把鼠标悬停在 aCells 中的单元格上，都会指示 jQuery 去执行圆括号中的代码。一旦用户的鼠标悬停在日程表中的一些 th 或 td 元素上，将会发生两件独立的事情：

1. 首先，我们去除表格中所有单元格的“over” class（aCells.removeClass（“over”））；

2. 接着，我们简单地修改一下用于展开第一列时所写的代码。我们并不是寻找一个特定的 headers 值（aCells.filter（"[headers=daytoday）"]），而是使用$（this）来引用用户当前所悬停的 th 或 td 元素。我们可以在所有的单元格中搜索匹配的 headers 属性（aCells.filter（"[headers=" + $(this).attr("headers") + "]"））），

然后把它们的 class 修改为“over”（.addClass（“over”））。

你可以看到，这工作得非常好：如果我们把鼠标指针悬停在一些未展开的单元格上时，“over”class 就会应用到该列，而 CSS 则会相应地修改它的显示样式（如图 17.23 所示）。

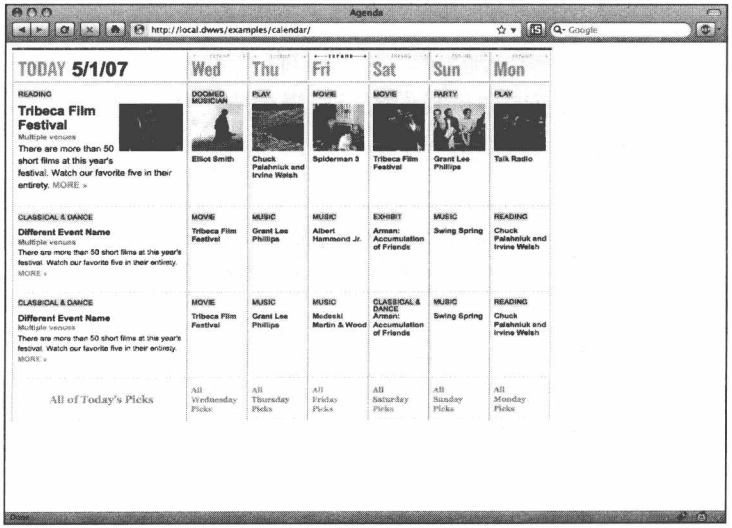


图 17.23　无论什么时候当用户悬停在一个单元格上时，我们的 JavaScript 会把 class=“over”应用到一整列上

如果用户的鼠标离开了该单元格，我们还需要把这个列收缩回去，实现这个功能只需要 mouseover()代码的简单变体：

```
$(document).ready(function() {
    var aCells = $("#agenda-week th, #agenda-week td");
    aCells.filter("[headers=day-today]").addClass("open");
    aCells.mouseover(
      function() {
        aCells.removeClass("over");
        aCells.filter("[headers=" + $(this).attr("headers") +
        "]").addClass("over");
      }
    ).mouseout(
      function() {
      aCells.removeClass("over");
    }
```

```
  );
});
```

这三行代码增加了之后，不管什么时候当用户的鼠标从一个单元格上移开时或完全移出整个表格时，每个单元格的“over” class 都将被去除，（可惜，这种情况很难在书中表现出来，不过请相信我，它就是这样工作的。）

现在，鼠标的移动已经很好地处理了：我们有了 mouseover 和 mouseout 事件处理代码，表现性的 class 相应地会从我们的日程表中应用或去除。剩下的事情就是定义用户点击一个单元格的行为。

```
$(document).ready(function() {
  var aCells = $("#agenda-week th, #agenda-week td");
  aCells.filter("[headers=day-today]").addClass("open");
  aCells.mouseover(
    function() {
      aCells.removeClass("over");
      aCells.filter("[headers=" + $(this).attr("headers") + "]").addClass("over");
    }
  ).mouseout(
    function() {
      aCells.removeClass("over");
    }
  ).click(
    function() {
      aCells.removeClass("open");
      aCells.filter("[headers=" + $(this).attr("headers") + "]").addClass("open");
    }
  );
});
```

跟之前一样，我们简单地重用使用在 mouseover()上的处理语法，只是做一点小小的（但是是关键的）修改。现在我们用 JavaScript 为 aCells 里和所点击的单元格 headers 属性相同的元素们附加上 class=“open”，而不是 class=“over”。这样，我们不仅能在默认状态下展开第一列（如图 17.22 所示），然后当悬停在另一列上时改变它的状态（如图 17.23 所示），而且也可以点击另一个单元格来展开那个列，至此，我们全部完成了“日程表”的创建（如图 17.24 所示）。

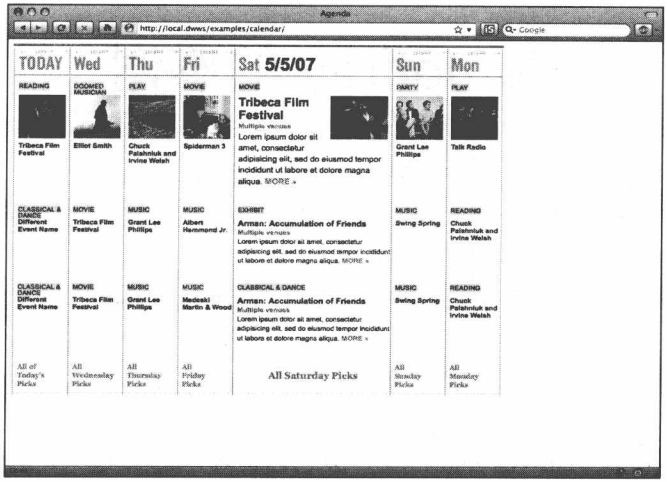


图 17.24　一旦我们写好了一个点击事件处理的代码，那么当用户点击一个单元格时相应的列就适当地展开了

17.5　标准贯穿始终

在本章中，我们再次强化了使用 Web 标准来进行设计所能带来的强大和简便。我们在一个合法且富有语义的 XHTML 的基础上建立了一个非常吸引人的界面，并且使用了很少的 CSS 和 JavaScript 代码，来为那些稳健的标记注入了活力。这个日程表下载起来非常快速，而且，如果万一用户不能应用样式表或者不支持 JavaScript，它也能优雅地适应。它实现了我们的设计目标，满足了客户的需求，因此，我们的章节到这里就可以结束了。

实际上，很明显“*New York*”网站上除了日程表还有很多内容是本章没有涉及的，比如：在切换视觉状态之间还有一些细微的图形差别；一个复杂的 CSS 框架控制着杂志上那些漂亮照片的校正；大量关联的规则控制着站点各个部分和字体的细节。我建议你访问 nymag.com，然后自己去发现这些。

当然，“*New York*”网站的设计和内容从日程表构建之后已经经过了不断地演化，而且，当然，它也还将在接下来的时间里继续演化。但那是另一个故事了，现在是停下来，开始思考你想讲的故事的时候了。

来自 Jeffrey 的感谢

这是一本凝结了众人心血的书，我要深深感谢他们。

感谢我们的实习生，Henry Li 和 Nicole Ramsey，他们熟练地使用微软 Word，令人难以置信地解决了书中的各种小问题。

感谢 Krista Stevens，她放下自己在“A List Apart”的编辑工作，来帮助我处理任务和计划。

感谢 Aaron Gustafson，我们的技术编辑高手，没有人比他知道的更多了，他真是个天才，谦虚又可爱。

感谢才华横溢的 Ethan Marcotte，这一版的合著者，他的另一个重要角色是：出版前的复审。我们俩的文字早已经分不出彼此了。非常感谢他，他的新娘和他之前所有的帮助。

虽然得到以上天才们的帮助，但是本书得以顺利出版还必须提到我们的明星编辑，有着坚强的毅力和战略智慧的 Erin Kissane 女士，对她的帮助，我只能说 6 个字：谢谢，谢谢，谢谢！

本书的这一版和 HTML5 规范都受益于一些极具天赋的个人。他们汇聚纽约，在 Happy Cog 工作室一起评论 HTML5 规范，肯定 HTML5 的优点，帮助 HTML5 更加完善。这里要极其感谢这些 HTML5 的特别朋友（www.zeldman.com/superfriends/）：Dan Cederholm, Tantek Celik, Wendy Chisholm, Aaron Gustafson, Jeremy Keith, Ethan Marcotte, Eric A. Meyer, 以及 Nicole Sullivan（他们的合影可以在这里看到：www.flickr.com / photos/zeldman/3813120876/）。

还有，非常感谢 Michael Nolan，他在 2000 年介绍我认识 New Riders 出版社，并在 2001 年对我说：“你应该考虑写一本关于 Web 标准的书”。

感谢 John Allsopp，“Developing with Web Standards”的作者，他的书里有大量的源码实例，是本书的“姐妹篇”。

我的感谢和赞扬还要给 Peachpit 的 Liz Merfeld、Kim Scott、Rose Weisburd 和 Tracey Croom，是他们完成了这本书的制作。

最后，想对新读者和老读者们说，感谢你们和我一起分享这段旅程。

-JZ

关于作者 Jeffrey Zeldman

被商业周刊誉为“Web 标准之王”的 Jeffrey Zeldman 是最早的网页设计师、blog 作者、网络独立作家和最早的网页设计教师之一。

在 1998 年，他和其他成员共同发起并创建了 Web 标准草根联盟网页标准计划小组“The Web Standards Project”（在 1999 年到 2002 年一直主持该组织的工作），致力于推动浏览器遵循 Web 标准，发起了 Web 标准运动。

自 1998 年起，Jeffrey 出版和主持了行业领先的杂志“A List Apart”（www.alistapart.com，专门面向网站设计师和站长）。他和 Eric Meyer 一起组建了网页设计会议“An Event Apart”（参见 www.aneventapart. com）。他也是 Happy Cog™（www.happycog.com）的创始人和执行创意总监。Happy Cog™ 是一个高端 Web 设计公司，在纽约、费城和旧金山都有分公司，主要的客户包括：AIGA（美国平面艺术协会）、美国大屠杀纪念馆、WordPress、Housing Works 组织、W3C（世界互联网标准组织）、Fetch 软件公司、圣丹斯国际电影节、W.W. Norton 出版公司、Mozilla 创作社区、MICA、Brighter Planet 以及阿曼达项目（The Amanda Project）。

Jeffrey 已经出版了两本书，除本书外，另一本是“Taking Your Talent to the Web”，一本为想转型成为网页设计师和美术指导（Art Director）的人而写的指南书（New Riders 出版社，2001 年，现在可以免费在这个网址下载：www.zeldman.com/talent）。他同时还担任 SXSW 互动艺术节和 Rosenfeld Media（出版公司）的咨询顾问，也是 Deck 公司（www.decknetwork.net）的创始人，一个提供创意、网页和专业设计的网络广告公司。

在成为网页设计师之前，他曾经当过作曲家、音乐演奏家；为华盛顿邮报和城市报当过记者；也做过广告文案创作和美术指导。Jeffrey 的 blog 在 www.zeldman.com（从 1995 年写起），在 twitter 的账号地址是 www.twitter.com/zeldman。他目前和女儿 Ava 以及一条名叫 Emile 的小狗居住在纽约市。

来自 Ethan 的感谢

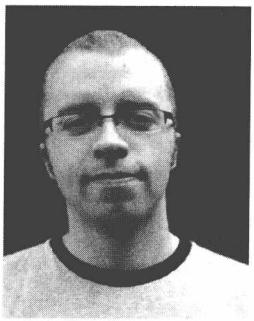
我非常感谢天才的 Ian Adelman 和他在纽约杂志的设计团队，作为最好的客户之一，我们依然保持着邮件联系，讨论他们站点上那些漂亮的网页。

Erin Kissane 是我共事过最细心、最周密、最一丝不苟的编辑。是的，英语这门语言需要更多像她这样的人，非常感谢你，Erin。

Aaron Gustafson 是一个神奇的技术编辑，他提供了详细的反馈和周到的批评，他就是一个真正的摇滚明星。

如果说，我今天有一个事业，那是因为几年前偶然拿起了一本橘黄色封皮的书（指本书的第 1 版），作者是我一直钦佩的朋友，和他一起写书是我的梦想。现在梦想成真，我都不知道用什么语言来感谢 Jeffrey Zeldman，也许永远都不知道。

关于作者 Ethan Marcotte



Ethan Marcotte 是一个多才多艺的用户体验设计师和开发者，他的工作展示了高质量代码和美观界面的完美结合。在加入 Happy Cog 之前，Ethan Marcotte 为这些客户工作，包括：纽约杂志（New York Magazine）、哈佛大学以及世界互联网组织 W3C 等。他也是前 Web 标准组织（The Web Standards Project）指导委员会成员，他在标准领域的工作已经包括在一些杂志和网站里了。

Ethan 是多个出版社的供稿作者，包括手写 CSS 专题（New Riders 2009），Web 标准创造专题（friends of ED, 2007），以及专业 CSS 专题（Wrox, 2005）。Ethan 也是一个有丰富经验的技术编辑，参与编辑的书包括“Bulletproof Web Design”（New Riders, 2005），以及“Designing with Web Standards”第 2 版（New Riders, 2006）。

Ethan 同时也是 A List Apart（为网页设计师服务）网站的供稿作者和技术编辑。在 at An Event Apart、SXSW 互动艺术节、哈佛大学以及 AIGA（美国平面艺术协会）等会议上，Ethan 是它们的特邀演讲者和受欢迎的教育人士。他几乎所有时间都在线上，并梦想以后成为一个忍者（他的个人网站：www.unstoppablerobotninja.com），哔（到此为止）！

Aaron Gustafson（第 3 版技术编辑）

1996 年迷上网络，并花费数年时间为 IBM 和柯尼卡·美能达推广像素（pixels）应用之后，Aaron Gustafson 创办了简单设计（www.easy-designs. net），一个精品网络顾问公司。Aaron 是 The Web Standards Project（WaSP）的会员、A List Apart 的技术编辑。并为 MSDN 提供著作和编辑书籍，包括“Accelerate DOM Scripting with Ajax，APIs，and Libraries”（Apress 出版社，2007 年 9 月），“AdvancED DOM Scripting”（Friends of Ed 出版社，2007 年）和“Web Design in a Nutshell”（第 3 版，O’Reilly 出版社）。他定期在网络上通过电话会议，为公众和个人提供 Web 标准和 JavaScript 培训。他的 blog 网址是 www.easy-reader.net。