



Broadview

Web标准组织创始人Zeldman力作全新登陆中国
国内四家知名开发网站和五大设计网站联合推荐

网站重构

——应用Web标准进行设计
Designing with Web Standards

[美] Jeffrey Zeldman 著
傅捷 王宗义 祝军 译



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

国外专家推荐：

“Zeldman为网站设计师和开发者们提供了一个幽默的、聪明的、重要的伙伴——《网站重构》。此书是对网站原有的艺术结构的一种深入的检查和挑战，因此将会使本书成为Web设计史上的罗塞塔石碑（译者注：Rosetta stone，破解古埃及文字的关键文物，奠定了埃及学基础）。”

Don Buckley
华纳兄弟公司交互市场副总裁

“我的生活现在已经有了一个新目标了：从我的代码中抛弃表现层的标志，专注为我自己和我的客户建立一个向后兼容的站点。我毫不怀疑，本书的知识将有助于我提高工作的质量，在竞争激烈的市场中保持优势。本书将成为理解和执行Web标准的权威指南。”

Chad Vrandt
Best软件公司CRM分部Web Services经理

国内四家知名开发网站——

CSDN(<http://www.csdn.net/>)
CoDelphi(<http://blog.codelphi.com/>)
ChinaUnix(<http://www.chinainix.net/>)
博客园(<http://www.cnblogs.com/>)

向广大网站开发人员联合推荐

为了网站能“活”得更长久，为了提高网站的可访问性，更为了降低成本，我们必须采用Web标准！

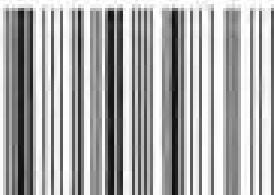
国内五大设计网站——

设计联盟网站 (www.chinadu.org)
闪客帝国 (www.flashempire.com)
5D多媒体 (www.5d.cn)
视觉中国网站 (www.chinavisual.com)
蓝色理想网站 (www.blueidea.com)

向广大网页设计师联合推荐

我们要在飞速发展的网络中用超前眼光来思维，推断将要来到的技术标准，之后取其精华，去其糟粕，纳为己用！

ISBN 7-5053-9836-9



9 787505 398368 >



责任编辑：孙学瑛

封面设计：张子建

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。

ISBN 7-5053-9836-9 定价：38.00元

网 站 重 构

——应用 Web 标准进行设计

Designing with Web Standards

[美] Jeffrey Zeldman 著

傅 捷 王宗义 祝 军 译

电子工业出版社

Publisbing House of Electronics Industry

北京 · BEIJING

内 容 简 介

本书是为了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。

本书着重分析了目前网站建设中存在的一些问题，以及“Web 标准”思想的产生、发展和推广，并从技术细节上讲解了网站实际制作和开发的过程中如何向 Web 标准过渡，如何采用和符合 Web 标准。本书的出版目的就是帮助读者理解 Web 标准，创建出用最低的费用达到最多的用户的，并维持最长时间的网站，并且提供一些相关的技术和技巧。

不要停滞不前，不要拒绝接受全部观点！

Authorized Translation from the English language edition, entitled Designing with Web Standards 1st Edition, 0735712018 by Jeffrey Zeldman, published by Pearson Education, Inc. publishing as New Riders, Copyright © 2002 New Riders.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PUBLISHING HOUSE OF ELECTRONICS INDUSTRY, Copyright © 2004

本书中文简体版专有版权由 Pearson Education 授予电子工业出版社，未经许可，不得以任何方式复制或抄袭本书的任何部分。

版权贸易合同登记号 图字：01-2003-6398

图书在版编目（CIP）数据

网站重构：应用 Web 标准进行设计 / (美) 塞尔达曼 (Zeldman, J.) 著；傅捷，王宗义，祝军译。

- 北京：电子工业出版社，2004.5

书名原文：Designing with Web Standards

ISBN 7-5053-9836-9

I. 网… II. ①塞… ②傅… ③王… ④祝… III. 网站—设计 IV. TP393.092

中国版本图书馆 CIP 数据核字 (2004) 第 031952 号

责任编辑：孙学瑛

印 刷：北京智力达印刷有限公司

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

经 销：各地新华书店

开 本：787×980 1/16 印张：24.25 字数：501 千字

印 次：2004 年 5 月第 1 次印刷

印 数：6000 册 定价：38.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：(010) 68279077。质量投诉请发邮件至 zhts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

看看他们是怎么说的

“自从 Jeffrey Zeldman 发起成立 Web 标准组织以来，我就已经了解和钦佩他的工作。Zeldman 的才能是将激情和实用主义结合在一起，支持基于标准的 Web 设计并产生漂亮的网站。设计新手和专家拥有此书将做得更好，不管你是刚刚开始规划网站，还是进行升级。”

Karl Dubost
W3C 的质量保障经理

“我的许多同行都在关注这个主题，但总是缺乏容易理解的文章。要写好这个主题，不让它太枯燥是很重要的。本书用清晰的、风趣的、富有见解的风格做到了这一点。”

Jen deHaan
交互式网站架构师、作家

“《网站重构》是一颗炸弹。对，正如你期望 Zeldman 做的。它分析了不同时期困扰我们的出版问题，然后给我们提供多个方法来对付这些问题。书中包含了这些方法的具体细节，以及如何创建最节省时间的代码的指导。”

Kathy Keller
得克萨斯公司和野外生存网站交流集团

新媒体从来是技术和艺术的完美结合体。作为一个新媒体设计师，如果既没有创作的意识和创意思维，又缺少对技术的理解和掌握，将是很可怕的事情！这

本书是为一个富有创意的新媒体设计师而准备的，它会帮助你认识新媒体设计必需的另一部分——技术标准！

雷海波（Byteart）

视觉中国网站主编

当我看到这本书的时候，心里很是欣慰和高兴，终于有了一个标准可以给同行们参考和学习了。这对业界的发展是非常有好处的。此书不仅是一本规范标准的技术书，更可以说是一本描述思想的书。它描述的不是要你去做这个标准，而是教你思考为什么要用这个标准。我觉得这个意义远大于书本身。

贺敏华（风子）

设计联盟网站主编

我想《网站重构》这本书，并不是一本指导读者进行网站基础建设的入门类书籍。然而关心网站标准，正准备升级网站或者想有更好网站访问的朋友可以从本书中获益良多。

曾沐阳（蓝色）

蓝色理想网站站长

原则只有一个，就是通过唯一的途径——Web 标准。目的只有一个，就是最佳兼容性、最佳架构与用户体验。

ALLAN

闪客帝国技术总监

对于那些想把一个网站发展壮大的人来说，这是一本难得一见的书，原作者及译者都在从事着大型网站的规划建设，有丰富的行业经验，而他们愿意把这些经验分享给大家，是值得庆幸的事情。从这本书中，我发现了我们自己网站的很多不足，在网站建设方面，本书是对我产生很大影响的几本书之一。

胡海（Lakesea）

5D 多媒体站长

译者序

“99.9%的网站是过时的”——猛然看到这个标题，你也许同样会吃一惊，但在仔细地阅读完本书后，你会觉得这句话一点不夸张。

这是一本什么样的书

这是一本具有“里程碑”意义的书，它即将在 Web 发展过程中开创一个崭新的时代！

这不是一本纯粹的技术教材，它更注重介绍的是一种思想，一种观念。这种观念对我们的网站，对我们的设计人员和开发人员，对网站的所有者都至关重要。那就是：为了网站能“活”得更长久，为了提高网站的可访性，更为了降低成本，我们必须采用 Web 标准！

书的第 1 部分着重分析了我们前些年网站建设中遇到的问题，以及“Web 标准”思想的产生、发展和推广，让你明白为什么要采用和推广 Web 标准。第 2 部分从技术细节上讲解了网站实际制作开发过程中如何向 Web 标准过渡，如何采用和符合 Web 标准。通过阅读本书，你可以清楚一些基本问题：

- Web 标准是什么？
- 网站一定要用标准吗？
- 用标准和不用标准的区别？
- 如何采用标准？
- 如何向标准过渡？

适合什么样的人看

这本书适合的读者是：所有的网站设计者、开发者和所有者。

这不是广告词。不论对这一行业的新手还是资深人士，本书都将对你产生深刻影响。它告诉我们怎样做是正确的，以及应该如何做。也许本书介绍的思想和技术你可能不会马上接受或贯彻到工作中，但它是 Web 发展的趋势，是阻挡不了的发展趋势。它的重要性已经并将继续在 Web 领域体现。

对我们有什么好处

如果你是 Web 设计师、开发者或所有者，如果你正在困惑自己应该学习什么，怎么做，如果你正在犹豫自己网站的发展方向，如果你正在想如何把自己提高一个层次，如果你正在头疼不断的网站升级问题，本书正是你需要的！

它告诉你 Web 开发的技术将如何发展，未来的 Web 将是怎样的。看清楚了这些，如何发展网站，自己如何定位就容易得多。

任何书都有它的价值所在。如果需要一个理由购买这本书，那么我会这样说：你可以不买这本书，但是本书的思想和技术你一定要了解，总有一天你会需要它，这一点是百分之百肯定的。与其被动地、等到迫不得已的时候再来了解和学习，为什么不现在就行动？

译者的感受

对于本书的作者 Jeffrey Zeldman，国内设计师和开发者可能并不熟悉，但在国外，同行无人不知。他领导的 Web 标准组织（www.Webstandards.org）帮助终止了 Microsoft 与 Netscape 之间的浏览器之战，使得我们不再需要考虑浏览器不同版本问题。他也是资深的设计师，对网站技术的历史和发展了如指掌。通过翻译本书，我们对 Web 标准的发展历史、Web 标准技术的应用进行了系统而全面的了解。特别是许多技术细节的来龙去脉，常常让我们顿悟：原来如此！

本书不像 Flash 或者 Photoshop 教程那么有趣，也不像程序语言教材那样立竿见影，但我们还是决定翻译这本书，不仅仅因为这是一本好的技术书籍，更是因为这是一本有意义的书。或许一开始它可能会被淹没在成堆的计算机图书中（尽管我们希望它一开始就受到重视），但我们相信随着时间的推移和观念的转

变，它的重要性会逐渐显露出来，它的价值会被大家所认同。

记得一开始学网页设计的时候，发现国外的大站点都很简单，就自大地认为中国的设计师肯定可以赶超国外。但随着学习的不断深入，发现一个大型网站的简单背后藏着丰富的技术内涵，例如，样式表、动态信息发布系统，要达到这样的“简单”和高效，我们仍然需要花大力气。

当国内的设计师现在都以为 HTML 已经没有什么可学，只要会使用 Dreamweaver 就行的时候，当我们沉醉在网页特效、Flash 动画的时候，Web 标准的推广已经成为国外一种普遍的现象。目前国内大多数客户都不清楚对网站的具体要求，他们注重的往往只是外观是否漂亮，基本功能是否实现，而不会去查看页面的原代码是否符合标准，也不会去问数据格式是否易于扩展交互。网站是否符合标准，很多时候都取决于设计者、开发者的知识面和认识。但是不能说因为客户不清楚、不要求采用 Web 标准，我们就可以偷懒和省略。作为网站设计师，有责任和义务去推广和采用 Web 标准。

都说 IT 界浮躁，总是“拿来主义”，缺少原创。那么即便是拿，我们也不能只拿表面的东西，更需要拿那些思想的内涵和精髓，否则我们永远都是落后的。国外一向注重规范和标准，我们也迫切需要这样的共识。本书所介绍的关于 Web 标准的思想，能让越多的设计师了解越好。虽然国内现在好像还没有完全采用标准的有影响力的网站，只有一些研究性质的个人网站，但越来越多的有识之士已经认识到并开始推动 Web 标准的应用。同时，由于经济利益的驱动，企业和客户也将逐步接受和认识到遵循 Web 标准所带来的好处。

相信有一天，Web 标准将成为进入这一行业的基本知识。

关于书名

本书英文原名为“Designing with Web Standards”，如果直译应该为《应用 Web 标准进行设计》，感觉有点枯燥，并且不能够使得本书和市面上大多数单纯讲述 HTML 及代码堆砌起来的书籍区分开来。因此，我们借用软件开发行业中的“代码重构”一词（重构的意义在于不改变程序的功能而改变程序的架构，从而更利于修改、维护和扩展等）。同样地，在国内 Web 领域，翻开漂亮的网站外衣，看到的是臃肿的、肮脏的、不可维护的代码（却从没人重视和理睬这些，更不用说去改变它），我们希望使用 Web 标准在不改变外观的情况下能够改变 Web 的技术本质。因此我们采用了“网站重构”作为书名。

感谢

非常感谢电子工业出版社给我们这次机会，让我们能为推动 Web 标准出一份力，也使我们自己获益匪浅。感谢郭立、朱沫红编辑的信任和鼓励，使得我们信心百倍地来翻译这本书，也感谢孙学瑛编辑，她细心的工作帮我们弥补了很多细节上的漏洞。

关于译者

傅捷

网名：阿捷，早期知名网页设计师之一，致力于网站设计和规范的推广，代表文章有“网站设计的思考”系列。原个人网站《网页设计师》、《网站项目管理协会》在网页行业有一定影响力。主要翻译本书思想理论部分。

王宗义

网名：redeye，Opensource 热衷者，技术爱好者，是 linuxforum 早期版主，领导开发过网站内容管理软件和多个项目的实施，长期进行软件开发研究，对软件开发、架构设计有比较丰富的经验。主要翻译本书 XML/DOM 部分。

祝军

网名：dodo，国内较早接触互联网的一批人，曾经是深圳热线非常男女的站长，即时通讯软件欧姆 omme 的创始人之一。对人机交互有较深的研究。主要翻译本书 CSS 部分。

关于作者



Jeffrey Zeldman 是世界上最知名的网站设计师之一。他的个人站点 (www.zeldman.com) 受到 1600 万访问者的欢迎，每天都有来自 Web 设计和开发行业的数千人访问。

他是 A List Apart (www.alistapart.com) 的出版者和创作主管，在线杂志《献给网站创建者》和 Happy Cog (www.happycog.com) 的创始人，还是一个设计师和咨询顾问，其客户包括 Clear 娱乐频道、华纳兄弟娱乐公司、

Fox Searchlight 公司 (www.foxsearchlight.com)，以及纽约公共图书馆。1998 年，他创建了 Web 标准组织 (www.webstandards.org)。这是一个网页设计师和开发人员的联盟，用来帮助终止 Microsoft 与 Netscape 之间的浏览器之争，并且劝说他们在新版本浏览器中支持相同的技术。

Jeffrey 是 “Talking Your Talent to the Web” (New Riders: 2001) 一书的作者，同时也在 A List Apart, Adobe, Creativity, Digital Web, Macworld, PDN-Pix 和其他站点上发表过许多文章。他曾经担任过 Communication Arts Interactive Festival、纽约艺术指导俱乐部、5K、Addy Awards 及 Radio Mercury Awards 的审查委员，并且是一个咨询联盟会议创建者和 i3Forum 的董事会成员。

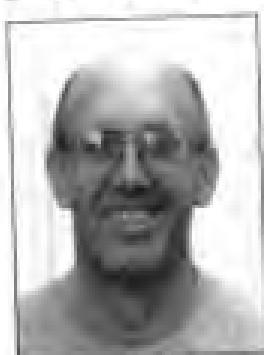
他曾在美国图形艺术学会 (AIGA)、哥伦比亚大学图书馆、洛杉矶国家实验室、纽约公共图书馆团体和纽约州信息资源管理部门论坛，包括 Builder, CMP, Seybold, SXSW Interactive, Web Design World 和 Webvisions 等会议演讲，但是他真正想做的事情是自己亲自动手去做网站。

关于技术评论家

那些评论家为 Web 标准设计思想发展的整个过程提供了相当多的专家意见。当本书开始撰写时，这些潜心研究的专业人员对书中的技术、组织和流程等内容及素材进行了重新校对和审核，他们的意见和建议保证了本书为读者提供高品质的 Web 标准设计技术信息。



Eric A. Meyer 从 1993 年开始就从事网络方面的工作。他住在一个非常美丽的城市——克利夫兰，受雇于 Netscape 公司，作标准设计培训师。Eric 在这个行业中相当有名，他经常就 Web 标准，跨浏览器兼容性、CSS 和 Web 设计等方面在各种会议上发言。作为一个凯斯西保留地大学（Case Western Reserve University）的网络管理员和毕业生，Eric 整理了 W3C 设计的 CSS1 测试套件，并于最近推出了尽其所能发现的 CSS 设计的局限。Eric 也是“Eric Meyer on CSS: Mastering the Language of Web Design”(New Riders)、《样式表：权威指南》(O'Reilly & Associates)、《CSS2.0 程序员参考》(Osborne' McGraw-Hill)，以及众所周知的 CSS 浏览器兼容性表的作者。



J. David Eisenberg 与他的宠物小猫 Marco 和 Zoë 住在加利福尼亚州的圣何塞市。他教授 HTML、XML、Perl 和 JavaScript，喜欢编写教育软件和在线教程。

David 在伊利诺斯大学从事计算机自动化教育软件项目，同时在 Burroughs 和苹果电脑公司工作。

感谢和赞扬

在写这本书期间，我得到了很多人的指导和帮助。

首先我非常感激 New Riders 出版社的 Jennifer Eberhardt，没有她耐心的指导和鼓励，这本书会耽搁很长时间。我要感谢的另外一位是 Michael Nolan，是他在 2000 年带我到 New Riders，并且甘愿当本书的一个配角。还有 Chris Nelson 也经常鼓励我，当有人提议我将论文更名为“*The Little Orange Book of Web Design*”时，David Dwyer 建议我写这本书。

在校对本书时，我从两位天才技术编辑的洞察力受益匪浅。J. David Eisenberg 是一个教师，也是 Web 标准组织中的成员和“*SVG Essentials*”(<http://www.oreilly.com/catalog/svgess/>) 的作者。Eric Meyer 是 Netscape 标准的强有力的推动者，并且是“*Eric Meyer on CSS*”(<http://www.ericmeyeroncss.com/>) 的作者，不过这标题（Eric Meyer on CSS）一直令他很尴尬。很幸运，他们都是我的朋友。

我要感谢我的搭档和同事，他们帮助我认识到在这本书中提到的很多技术，他们是：Brian Alvey, Leigh Baker-Foley, Hillman Curtis, Nick Finck, Dennis James, Jamal Kales, Erin Kissane, Bruce Livingstone, Tanya Rabourn, Brad Ralph, Ian Russell 和 Waferbaby。我也要感谢在我写这本书那年 Happy Cog 的客户，尤其是 Steve Broback, Don Buckley, Eric Etheridge, Andrew Lin, Alec Pollak 和 Randy Walker，非常感谢他们为我们的工作提供各种机会，并容忍我偶尔的失败的实验，还报销了不少的费用。

我之所以可以做好每件事情，都应该归功于 Tantek Celik, Joe Clark, Todd Fahrner 和 Eric Meyer，他们掌握的知识比我的丰富，思考得也并不比我少。

没有 George Olsen 和 Glenn Davis，就不会有 Web 标准组织，没有这个组织，现在我们的每个站点的编码方式起码要有 15 种。感谢各位改变了世界！

我还要感谢组织指导委员会的每一个成员，尤其是特别感谢 Tim Bray 的才智，还有 Steven Champeon，他使组织保持在一个较高的水平，以及 Dori Smith，他使标准可以真正实现。另外，我还要感谢 Rachel 和 Andrew，他们使 Macromedia 及它的用户群可以很好地工作。

Jeffrey Veen 也许没有意识到，是他帮助我像公众演讲人一样学会了放松，从而让我可以更有效地传播 Web 标准的信息。

我从每一位 Web 标准组织的支持者那里学到更多知识，甚至从批评者那里也获得很多，1998 年至 2002 年之间在他们的要求和异议声中，形成了组织的策略。

感谢那些在过去三年努力工作的浏览器工程师们，他们克服了技术跨度大、预算少的困难，最终真正实现了遵循 Web 标准的诺言。

还要感谢 Janet Daly, Karl Dubost, Håkon Lie, Molly Holzschlag, Meryl Evans 和 Michael Schmidt，还有 CSS 设计的倡导者 Douglas Bowman, Owen Briggs, Chris Casciano, Eric Costello, Todd Dominey, Craig Saila, Christopher Schmitt, Mark Newhouse 和 Waferbaby。还要感谢数以百计的鼓励我们的 Web 设计师。假如我列出所有要感谢的人（你知道我说的是你），那这本书起码要再加厚一倍。

特别要感谢那些工作在不同行业的设计师们，像在本书里主要描述的 Warren Corbitt, Joshua Davis, Matt Owens 和 Lee Misenheimer。你们的工作促使我对每件事都站在设计师的角度思考多遍。

我要感谢我的合著者——我的父亲 Maurice，并且祝福他和他的新婚太太 Katherine 健康、恩爱和幸福。

这本书也同时献给 Carrie，我爱你，宝贝。

告诉我们您的想法

作为本书的读者，您是最重要的批评家和评论家。我们非常重视您的意见，并且想知道我们应该怎样做才是对的，我们怎样才能做得更好，您通常在什么地区看到我们发行的这本书，以及其他任何明智的建议。

对于这本书，我们组织了一个项目团队，他们是一—

组稿编辑：朱沐红

责任编辑：孙学瑛

市场推广：张 推 王 斌 孙学瑛 张子建 朱沐红

出版统筹：郭 立

在这本书的制作和推广的过程中，还得到了许多其他同事、译者及各大网站站长和主编们的大力支持，在此表示感谢。

电子工业出版社博文视点的同仁们欢迎读者提出自己的意见。您可以发传真，写电子邮件，或者直接写信，或者填写本书最后的读者调查表寄给我们，让我们知道这本书您喜欢或不喜欢的地方，也可以告诉我们如何把书做得更好。我们会仔细阅读您提出的意见。

电子邮件：editor@broadview.com.cn

邮寄地址：北京市复兴路 47 号天行建商务大厦 604

邮政编码：100036

导言

早前一段时期，许多汽车司机认为，向车窗外扔空瓶子并没什么。数年以后，人们开始意识到乱丢垃圾并不是处理垃圾的合理方式。目前网站设计团队在看法上正经历着一个类似的转变，而 Web 标准正是这次转变的关键。

历史上，我们的网站总是用明天的开支解决今天的问题。本书展示了“先建站后付费”的方法将不再有效或必需的；展示了解决今天问题的方法，而且它不会使网站在以后产生进退两难的局面；同时消除“采用 Web 标准会损失部分用户”的陈旧观念，事实上，恰恰相反，采用标准将使你获得更多用户。

一种尺码不能适合所有人

在本书中，我们将讨论一些可以解决设计和开发的普通问题的标准方法。没有一本书能覆盖到每个问题及其解决方法，其他作者可能采取一个完全不同的方式来阐述，而本书则偏向于用实用的、直接的方式来满足未来的需求。本书论述的先进技术和概念，都已经被实践检验是有效的，并且在我自己参与的设计和咨询的项目中被证明都是非常有用的。这些思想已经被用在数以千计的、有远见的站点中。

不要求每个读者都立即使用本书论述的方法。如果你的设计风格倾向于采用紧凑的表格，就不必遵守像第 16 章“CSS 重新设计”中描述的设计规则。如果你的站点必须用 4.0 版本以上的浏览器观看，你可能对第 8 章“XHTML 举例：混合布局（第一部分）”到第 10 章“CSS 应用：混合布局（第二部分）”中描述的混合技术感兴趣，而不需要采用在第 16 章中描述的纯 CSS 布局技术。

任何有思想的设计师、开发人员，或者站点所有者都会支持本书中先进的观点。标准对于任何网站都是至关重要的，因为支持 Web 的软件最终都是要遵循

Web 标准的，因此学习和正确使用它非常有意义。这样做可以节省时间和金钱，减小企业的管理费用，延长站点的可用寿命，并且提供更多的途径来访问我们的内容。

最后一点，对于那些希望获得更广泛的用户（比如采用非传统网络访问方法的用户）的网站特别重要。它也更符合法律，越来越多的国家和美国的州政府开始制订可访问性的法律条款。Web 标准及其可访问性可以帮助站点保持其合法性。

理论与实践

本书中的一些特殊观念和先进技术还有争议。如果你是一个绝对标准支持者（我的意思是用最挑剔的方法），可能你不愿意使用 XHTML，除非所有的浏览器都完全支持用 application/xhtml+xml 代替 text/html 发送 XHTML 文件。具体细节请看 Ian Hickson 的《用 Text/HTML 发送 XHTML》(<http://www.hixie.ch/advocacy/xhtml>)。

如果你同意 Ian 的观点，你可以：现在就使用 HTML 4.01，或者在 Web 服务器端进行配置，向支持 XHTML 的浏览器发送 application/xhtml+xml 文档，向不支持的 XHTML 的浏览器发送 text/html (<http://lists.w3.org/Archives/Public/www-archive/2002Dec/0005.html>)。在本书中，我已经尽量避免这种问题，因为我倾向于在已有的环境下完成工作，希望本书的大多数读者可以分享其中的内容。

混合布局：这是出路吗

同样地，一些纯 CSS 的支持者可能不重视 CSS/表格的混合布局方法。混合布局的方法（第 8~10 章）提供给那些需要的人，特别是那些需要在老的、不符合标准的浏览器上获得与新的、兼容标准的浏览器几乎一样好的效果的设计师。

我们可能不会长时间地使用混合布局的方法。在我写这一页的时候，ESPN.com 又使用 CSS 布局（如图 1 所示）重新设计，而每大概有一千万次访问这个体育站点。当大型的商业站点也开始采用 CSS 布局技术时，Web 标准真正成为标准的日子也即将到来。虽然站点的艺术指导 Mike Davidson 既不是标准纯化论者也不是完美主义者，但是 Mike 和他的团队还是做出了正确的选择：

- 全部采用 CSS 布局。除了超出设计师控制的广告横幅外，布局中不包含表格。
- 禁止使用``标签。

- 节约带宽，节约带宽，节约带宽。虽然显示更丰富的页面，但标记和代码只有重新制作之前的一半（用第 5 章“现代置标语言”中所描述的结构标识及第 9 章“CSS 入门”中的方法可以达到节约大量带宽的目的）。
- 对于所有的浏览器，只用一个样式表——不需要浏览器版本检测或根本没有必要。用所有支持 `getElementById` 的浏览器来浏览这个站点，效果都差不多一样，包括 2002 年的新产品——目前还在测试中的苹果的 Safari 浏览器。



图 1

采用 Web 标准对商业站点是否太冒险？ESPN.com (www.espn.com) 不这么认为，这个流行的体育站点在 2003 年 2 月使用 CSS 技术重构了网站

ESPN.com 使用了本书中提到过的许多技术而获得巨大利益，并且为访问者节约了大量带宽。但该站点的第一次应用 CSS 时并没有采用本书提到的方法（毕竟，当最初使用 CSS 重新设计 ESPN.com 的时候，本书还没有出版）。该站点使用了大量的浏览器检测技术，而且由于检测代码中含有错误代码，站点的可访问性并没有变得更好。事实上，用 CSS 完全能够使站点结构更加合理和紧凑。

当你阅读本书时，ESPN.com 可能改善了一部分甚至全部内容。但即使它还没有这么改善，也至少已经开始使用 Web 标准。在 ESPN.com 的带领下，少数大胆的、流行的商业站点肯定会跟着使用标准。当每天有一千万人次访问使用了 CSS 布局的站点时，它标志着基于标准的设计和开发方法的胜利。

在 ESPN.com 冒险尝试的前一个星期，Netscape 的 DevEdge 网站（如图 2 所示）就按照 Web 标准重新开发并推出。这个站点一直提供关于网络开发的权威信息——包括正确应用 Web 标准的教程——但它原来的代码结构与 Web 标准并不相符。按照标准重新设计后，此站点开始遵循自己的原则并且作为一个标准示例站点服务于大众。所有的特点如下列出。

- CSS 布局中无表格
- 用户可自行选择样式
- 基于标准的下拉菜单
- 通过 CSS 内容属性为浏览者提供一个便于打印网页的链接

图 2

2003 年 2 月，Netscape 的 DevEdge 遵循自己提供的原则，使用基于标准的布局和标识重新构建了站点 (<http://devedge.netscape.com/>)



一个连续统一体，而非一套不可改变的规则

正如本书中所强调的一样，Web 标准是一个不断发展的连续统一体，而并非一套不可改变的规则。在向 Web 标准转换的过程中，在你的第一个站点甚至到第五个，你也许都不能完美地将结构从表现中分离（第 2 章，“根据标准设计和制作”）。你在可访问性方面（第 14 章，“可访问性基础”）的早期努力可能仅仅达到 WAI Priority 1 的最低要求，甚至无法完全正确做到。

让我们开始吧！总要有开始的一步。举个例子，当我们开始发胖，如果因为羞愧而不去健身房，直到身体检查出现问题才不得不去。那么同样地，如果我们

不在某个地方开始使用 Web 标准，我们的站点就一直无法达到向后兼容。删除 `` 标签也许就是你的开始，或者用有意义的 `<h>` 和 `<p>` 标签替换非结构性的标识，也通常是一个很好的开始。你有必要花费一些时间来考虑现代标识和 XHTML 技术。

展现，并非销售

有时，设计师会在标准的“销售”方面陷入困境。这些年以来，我收到了数以百计的邮件，均来自想使用标准的设计师，“怎么办？我的客户不让我这样做。”如果标准具有连续性，客户又如何会反对，甚至不愿做一点点的尝试呢？举个例子，即使大多数表格驱动的站点坚决反对采用 HTML 4.01 或 XHTML 1.0 过渡技术，但它也能达到 U.S. 508 条款或 WAI Priority 1 的可访问性要求。客户只要有一个可访问性好且没错误的站点就可以了。

大多数关心标准推广的设计师在实际项目中总是不能顺利采用标准。例如，他们不能采用纯 CSS，只是因为他们的客户（或者老板）使用的还是不完全支持 CSS 的 Netscape 4。这种情况是存在的，但这并不是一个逃避标准的理由，不能因此而不采用有效的标识和正确的 CSS，或者因此不采用第 9 章描述的可以在多浏览器中显示满意的外观的双样式表的方法。

我的网站代理商是 Web 标准和可访问性的拥护者，但并不是我们的方法和纯标准的盲目追随者。我们使用最适合项目的方法将标准“销售”给他，为此做了两件事：

- 在我们的方案中，明确地陈述了我们使用的技术，并把它描述得简单易懂。举个例子，“XHTML1.0 过渡技术，即现在的标准，将应用于代码标识。”在客户已经同意此方案并已签好合同，“允许”使用指定的标准，进一步的技术控制就不需要客户参与了。我们的选择将会影响站点在老版本的浏览器上的视觉效果，在最初的方案中也要加以说明。
- 当工作开始后，在给客户的演示不同阶段中，即使面对技术型客户，我们也保持最低限度的技术讨论。当提交一份只有以前版本三分之一大小并保持着原有格式（甚至是更高级的格式）的重新设计时，无论改变或更新了多少次，我们也不要说“CSS 使它成为可能”。我们应该声明：“我们已经创建了一套保持原有格式并占用很小带宽的系统”，如果客户认为我们所做的非常棒，并继续给我们带来更多项目，标准就生存下来了。

让你的工作为你“推销”标准

当 Hillman Curtis 公司和 Happy Cog 合作重构 Fox Searchlight 图片站点时，我们接触的最主要的客户是一个富有经验的网页设计师兼开发人员。对他来说，我们使用的混合CSS/表格技术还是比较新的。“这个站点访问速度真快！”他不断地惊叹道。我们在网站交付之前提供了一个样式指南，这个指南清楚地覆盖了整个站点的各个技术层面。他非常高兴地学习，最终也掌握了这个新技术。从这个例子可以看出，我们已经不用再“推销”标准了，因为结果已经帮我们把标准“卖”出去了。

当你因做这样的工作而出名的时候，客户将会因此而主动来找你，你就不必自己到处去推销。虽然相对于纯 CSS 布局设计来说，Happy Cog 采用了一种混合技术，但是我所有的个人站点都使用 CSS 布局设计，并且越来越多地应用到我代理的小站点中。我们甚至发现用 CSS 做设计工作比 Photoshop 更简单，因为这样可以节省时间和操作步骤。

最近做的一个项目是给 Clear Channel Entertainment 做的，这个项目中需要几个设计上的变化（大多数项目开始都是这么做的）。“顺便说一下，这些是采用 CSS 的布局设计，”我们告诉顾客。“我明白，”他说。随着这些方法逐步标准化（越来越多的商业站点采用这种方法，比如第 4 章“XML 征服世界（和其他 Web 标准成功案例）”中讨论的 ESPN.com、DevEdge、Wired.com、政府、公共部门站点），渐渐地，我们使用 CSS 就像使用 GIF 或 JPEG 图片一样，不需要再跟客户特别说起。

内部推销

我已经讲述了一件网站空间代理商那里发生的事情，同样地，内部工作也可以这么做。要尽量避免陷入基线浏览器的讨论和其他保守观点的误区。选择合适的规范，在老板要求的文档中简要地描述这些规范，并开始工作。

在写这本书的两年前，我曾在一个大规模的美国政府组织中给一批 Web 开发人员做演讲，他们对 Web 标准很感兴趣。可惜的是代理商依旧相信老的、不符合标准的浏览器，他们中的一些人认为在内部不可能使用标准（这种迷信是没有事实根据的，记住，Web 标准是个不断发展的连续统一体）。

在此导言之前，我再次拜访了代理商并进行了一次简短的谈话，发现他们的思想已经改变了。大部分代理商现在使用的是 Netscape 7，只有少数仍旧使用 Netscape 4，因为一些内部应用程序仍然是利用 Netscape 的 `document.layers`

建立起来的，类似的 Netscape 4 私有的 DHTML 技术一直影响着一些 Web 设计和开发团队成员。在场的参与者都希望讨论怎么把他们的应用程序升级到 W3C 的 DOM 上，而不是举手投降，表示放弃。

变革的气息

变革到处都在发生，时快时慢。这些变化将促使人们不断地进行思考，应该如何创建、如何升级网站。在几乎被忽视的经济和政治上的困扰，以及最终期限的折磨中，我们关于网站如何工作，以及它应该是被如何创建的观念正经历着一场深刻而持续的变革。Web 标准将很快会像网络可用性和信息体系结构一样被广泛地讨论和学习研究，它们会像其他原则一样重要，因为它们的每一个细节完全是为了使我们辛苦建立的站点和媒介更加健康。

本书介绍的内容多、程度深，但还仅仅是涉及 Web 标准的表面，还有更多针对 CSS、可访问性、结构化标志，以及比本书或其他参考书中介绍的 DOM 更多的知识等待我们学习。我们已经提到，除了本书作者介绍的，还有更多的方法来解决本书提到的问题。

如果在一个房间里有两个设计师，那么会产生三种看法。两个设计师不可能对布局样式、标记、导航条或颜色的每个方面的看法都一致。对标准也是一样。在这个领域里，争论从未停止过。

没有一本书能够针对所有的人表达清楚所有的事情，本书仅仅是提供给你自己将来发展的一般指导。本书的出版目的就是帮助读者理解 Web 标准，创建向后兼容的站点，并且提供一些相关的技巧。

在使用传统的站点设计方法三年之后，我对网络标准感到困惑，于是在过去的五年时间内我逐渐明白了本书的基本理论。可能你并不同意我所说的观点，也可能六个月甚至两年后我会反对自己现在的意见，但关键是：不要停滞不前，不要拒绝接受全部观点。如果观点开始被采用，就会帮助你的项目用最低的费用来达到最多的用户，并维持最长时间。

采用 Web 标准吧！如果不是现在，那会是什么时候呢？如果不是你，那会是谁呢？

目 录

第1部分 休斯顿，我们出问题了

在开始之前	3
费用上升，效益下降	3
终止网站淘汰的怪圈	4
什么是向后兼容	6
没有规则，没有教条	7
实践，不是理论	9
这样的旅行真的必需吗	9
第1章 99.9%的网站都是过时的	11
1.1 现代浏览器和 Web 标准	11
1.2 “版本”问题	13
1.3 向前兼容的思想	15
1.4 当好事情发生在坏标记上	24
1.5 治疗	26
第2章 根据标准设计和制作	29
2.1 跳出那个禁锢	30
2.2 标准出现以前的设计成本	31
2.3 时髦站点，古老方法	32
2.4 Web 标准三剑客	35
2.5 实际应用	38

2.6 过渡方法的好处	39
2.7 Web 标准组织：跨平台性	41
2.8 A List Apart：一个页面，多种浏览方式	44
2.9 我们要去往哪里	48
第 3 章 推广标准的困难	53
3.1 看起来可爱，代码却丑陋	53
3.2 2000 年：浏览器之年	56
3.3 太少，太迟	60
3.4 糟糕的浏览器养成坏习惯	61
3.5 混乱的网站和令人困惑的标签	65
3.6 字母 F	69
3.7 兼容是一个禁忌词语	72
第 4 章 XML 征服世界（和其他 Web 标准成功案例）	75
4.1 通用的语言（XML）	75
4.2 XML 应用程序和你的站点	85
4.3 与生俱来的兼容性	86
4.4 协作的新时代	87
4.5 Web 标准和创作工具	89
4.6 CSS 布局的出现	92
4.7 Web 标准的主流	100
4.8 执行概要	106

第 2 部分 设计与构建

第 5 章 现代置标语言	111
5.1 垃圾代码的可耻秘密	114
5.2 重新阐述了什么	116
5.3 执行概要	117
5.4 什么样的 XHTML 适合您	118
第 6 章 XHTML：Web 重构	121
6.1 转换到 XHTML：规则简单，容易上手	121

6.2 语言编码：无趣，很无趣，真的无趣.....	131
6.3 结构康复——对我有益.....	133
6.4 视觉元素和结构.....	136
第 7 章 紧凑而坚固的页面保证：以严格和混合的标记组成的结构	139
7.1 所有的元素都必须是结构化的吗.....	139
7.2 混合布局和简洁的标记：要做什么和不要做什么.....	145
7.3 过时方法的展示.....	153
第 8 章 XHTML 的示例：混合布局（第一部分）	161
8.1 本章所使用的过渡方法的好处.....	161
8.2 基本方法（概述）	162
8.3 开始的标记和结束的标记是对应的.....	169
第 9 章 CSS 入门	173
9.1 CSS 概述	173
9.2 样式解析	175
9.3 外联、嵌入、内联样式	185
9.4 “最合适方案”设计方法	188
第 10 章 CSS 应用：混合布局（第二部分）	191
10.1 准备图片	192
10.2 设置基本参数	193
10.3 导航元素：第一个步骤	203
10.4 CSS 导航条：在第二个步骤的第一次尝试	206
10.5 CSS 导航条：最后一个步骤	207
10.6 最后一个步骤：外联样式和“ <i>You Are Here</i> ”效果	208
第 11 章 使用浏览器 第一部分：DOCTYPE 转换和标准模式	213
11.1 DOCTYPE 转换的传奇故事	214
11.2 控制浏览器性能：DOCTYPE 转换	216
11.3 感谢浏览器的多样性（或至少学会接受）	221

第 12 章 使用浏览器 第二部分：盒模型、bug 和工作区	225
12.1 盒模型和它的不足之处	225
12.2 IE/Windows 上的空格 bug	235
12.3 IE6/Windows 上的“漂浮”bug	238
12.4 Flash 和 Quick Time，期望的对象	240
12.5 一个平凡的工作区世界	244
第 13 章 使用浏览器 第三部分：排版	247
13.1 字号问题	247
13.2 用户控制	247
13.3 守旧的方法	248
13.4 目前使用的标准尺寸，但是它又能延续多久	250
13.5 em 理论的失败	258
13.6 像素的作用	260
13.7 字号关键字法	265
第 14 章 可访问性基础	271
14.1 有关访问性的书籍	272
14.2 普遍的错误观念	273
14.3 法律和布局	275
14.4 直逼可访问性的错误说法	277
14.5 关于可访问性的 一点技巧，一个元素接一个元素	281
14.6 所需工具	291
14.7 和 Bobby 起工作	292
14.8 为访问性做计划，你会受益良多	296
第 15 章 使用基于 DOM 的脚本语言	299
15.1 初步接触 DOM	299
15.2 DOM 请不要让我伤心	305
15.3 显示和隐藏	309
15.4 动态菜单（下拉和展开）	313
15.5 样式切换器：增加可访问性，提供更多选择	314

第 16 章 CSS 重新设计	319
16.1 设定目标	319
16.2 建立基本参数	323
16.3 基于规则的设计	330
16.4 具有 CSS 滚动效果的“返回首页”按钮	332
16.5 CSS/XHTML 导航条	337
16.6 最后的加工	342

第 3 部分 结尾

附录 A 现代浏览器：品质优良的、低劣的和非常低劣的	349
术语表	355

Part I

第1部分 休斯顿，我们出问题了

- 在开始之前
- 99.9%的网站都是过时的
- 根据标准设计和制作
- 推广标准的困难
- XML 征服世界（和其他 Web 标准成功案例）



在开始之前

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设备，那么本书正是你所需要的。

由于技术的快速发展，大部分网站已经不可避免地经历了几轮淘汰。每当浏览器版本升级或者新的网络设备和技术出现时，刚完成（或刚支付了建设费用）的网站看起来就已经过时了。

我们的建设仅仅就是为了重新建设，通常，改版并没有增加客户需要的功能或者提高网站的可用性，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的站点的情况，我们采用的“向前兼容”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 上做商务必须付出的代价”——我们已经习惯这样，甚至成了标准。但是这样的成本是大多数人不能长期承受的。

费用上升，效益下降

复杂的代码、层层嵌套的表格、``标签，以及其他冗余代码使简单的

译者注 第1部分标题“休斯顿，我们出问题了”（Houston, We Have a Problem）是美国家喻户晓的一句名言。1970年4月11日，阿波罗13号宇宙飞船升空3天后，在太空中发生爆炸，宇航员与休斯顿控制中心通力合作，终于使飞船返回地球。“休斯顿，我们出问题了”就出自这个事件。作者在这里引此为标题表示 Web 发展遇到问题了，但只要努力就可以解决。

网站需要两至三倍的带宽。网站访问者要花费长时间等待页面的打开，除了少数有毅力和耐心的客户能够等到所有页面打开，大多数人早已因厌倦而离开。这样的网站是难以让人去访问的。

我们支付服务器费用，用以支持一个每次页面浏览为 20KB 就可以满足需求，却要耗费 60KB 带宽的网站——我们也要为页面浪费的带宽向 ISP 提供商支付费用（或者增加我们的 IT 预算）。我们吸引的访问者越多，所需的成本就越高。为了应付特别的前台设计，我们的数据库不得不支持更多查询，费用也进一步增加。最后，我们被迫购买或者租借额外的服务器来满足需求——不是因为访问者增加，而是臃肿冗余的标记和代码。

创建网站时，我们聘用按小时计价的程序员为网站开发程序，6 种版本的方法（不同的版本服务于不同的浏览器和设备的访问者）使成本非常高，以至于用尽所有的钱。这时，一种新浏览器或无线设备又出现了，而我们已经没有钱修改代码，以适应新浏览器和新设备。淘汰的周期重新开始。

大部分人都曾经有这样的吃惊体验：用新浏览器访问一个网站时，被告知浏览该网站需要一个“时髦”浏览器，其实这个“时髦”浏览器已经比我们正在使用的版本旧很多。实际情况是，那个网站的站长或开发者并不愚蠢，也不是不顾及别人，只是他们已经用尽了他们的升级预算，没有更多的钱去维护这个网站。

在另外一些案例中，问题并不是缺少资金，而是缺乏知识，或是投资被误导。**Connected Earth**，这个口号为“*How communication shapes the world*”的公司，据报道最近花费了 100 万英镑（大约 160 万美元）重新设计了网站。不管在开发上浪费了多少资金，它居然不支持近来多数流行的浏览器，网站拒绝 Mozilla（如图 1 所示）、Netscape 6/7 及 Opera（如图 2 所示）浏览器访问，同时也不支持非 Windows 操作系统，Macintosh 的 IE 用户就更不幸了。

不管网站是因为不断的升级超出预算而荒废，还是因为一开始就采用了过时的标准被淘汰，结果都是一样的：他们损失了潜在的不断增长的用户。总之，不论你花费了 100 万英镑还是 1000 英镑，目的都是欢迎用户来访问，而不是赶走他们。

下面介绍解决方案。

终止网站淘汰的怪圈

现在，由于 W3C 组织（参见后面的阴影部分）和其他标准组织已经建立起来的技术，以及大部分主流浏览器和设备对标准的支持，使网站的设计及其长久

运行变为可能，即使标准和浏览器在不断演化。已经在此行业拼搏奋斗多年的老手可能怀疑这种说法，但本书将展示它是如何实现的。



图 1

花费了巨资资金，Connected Earth 公司网站几乎不支持所有的流行浏览器，它拒绝任何平台上的 Mozilla（图中显示的）、Netscape 6/7、Opera 浏览器用户，也不支持 Mac 操作系统的 IE 用户 (www.connected-earth.com)



图 2

同样的站点，在 Windows XP 系统中用 Opera 7 浏览效果，Connected Earth 公司的技术问题可能已经通过本书介绍的方法消除了

本书将教你如何摆脱网站的“建立、废弃、再建立”的怪圈，设计现在和将来的超越桌面系统的网站，不再拒绝潜在用户，不再因为目光短浅的解决方案而浪费日益稀少的时间和金钱。

这不是一本给空谈者或纯理论家的书，而是为那些需要继续工作、不断实践的人们写的。它是专门为创新者和那些寻找明智建议的专业人员而提供的，帮助

他们调整自己的技术和思路，继续工作并创造能够吸引越来越多访问者和用户的成功站点。

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。

阅读本书的网站所有者和管理者将会立刻停止在怪圈上浪费金钱，他们将懂得如何为向后兼容的网站撰写需求文档。

什么是 W3C

W3C (World Wide Web Consortium, <http://www.w3.org/>) 创建于 1994 年，研究 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 500 名会员组织加入这个团体，它的主任 Tim Berners-Lee (<http://www.w3.org/People/Berners-Lee/>) 在 1989 年发明了 Web。W3C 推行的主要规范有 HTML, CSS, XML, XHTML 和 DOM (Document Object Model)。

多年以来，W3C 把那些没有被部分会员公司（如 Netscape 和 Microsoft）严格执行的规范定义为“推荐”(Recommendations)。自 1998 年开始，“Web 标准组织”(www.webstandards.org) 将 W3C 的“推荐”重新定义为“Web 标准”，这是一种商业手法，目的是让制造商重视并重新定位规范，在新的浏览器和网络设备中完全地支持那些规范（看另外一个阴影部分：什么是“Web 标准组织”）。

其他的标准组织包括 European Computer Manufacturers Association (ECMA) 将 ECMAScript 定义为“标准 JavaScript”。请看第 3 章“推广标准的困难”了解更详细的资料。

什么是向后兼容

什么是向后兼容？我们认为是：采用正确的方法设计和建设，发布的任何文档可以正确显示在多种浏览器、平台、设备上，并且能继续在未来发明的新浏览器和设备中工作。开放的标准使之成为可能。

最具吸引力的是，用标准来设计和建设网站，可以用更少的生产和维护成本，并使网站的可访问性更完美（说明：更多的客户，更少的成本，提高大众亲和力，减少网站不易用的投诉）。

我们说的 Web 标准是什么呢？是指结构化的语言（如 XHTML 和 XML），解释性语言（如 CSS），对象模型（如 W3C DOM）和脚本语言（如 ECMAScript），所有这些（甚至更多）都会在本书做解释和说明。

经过委员会专家的推敲，这些技术被精心设计，将最大益处提供给数量众多的 Web 用户。这些技术的集成应用，已经成为合理的、易用的、稳定的和高价值的网站标志（网站的所有者和管理者：不用担心不懂技术，你可以跳过技术章节，只要确认你的员工或卖主理解这些就可以）。

以上所说的标准兼容的浏览器是哪些呢？我们认为那些能理解和支持 XHTML，CSS，ECMAScript，DOM 的产品都是，例如：Mozilla，Netscape 6+，MSIE5+/Mac，MSIE6+/Win 和 Opera 7+。这些浏览器都能完全支持每个标准吗？当然不是，没有任何软件完全没有 bug，而且标准本身也复杂，它们互相作用的方式就更复杂了。

不过，主流浏览器足够可靠，所以我们可以抛弃旧方法，并更加灵活地工作，满足更多用户的需求，因为标准已经很自然地被包含在浏览器中，我们能适应那些使用老版本浏览器和设备的人们，当然必须在向后兼容的前提下。

没有规则，没有教条

这不是一本宗教式或教条的书。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标准融进到你的工作流程中。本书不提倡死板地遵循标准，也许一些特别的网站和任务有变通的方法。本书更不会假惺惺地吹嘘 W3C 推荐的标准没有任何缺点。

本书将告诉你在 IE、Netscape 和其他主流浏览器中工作需要知道的特殊细节，并提供应付旧版本浏览器用户的策略，因为也许你的网站浏览者中，就有一位躲在办公室角落使用旧版本浏览器的顽固家伙。

本书不会说谎。有一些私有的方法和捷径比 W3C 的规范更容易使用。举例来说，IE 独有的使用 Microsoft 捷径的脚本（比如 innerhtml）比 W3C 标准 DOM 的脚本更快更容易。而从商业角度看，适合所有浏览器的代码比只适应一个浏览器的代码更理性，DOM 就是这样的解决方法。

同样，探究结构化标记和 XHTML 的好处时，我们不会假装说每一个网站、每一个页面、每一个标记都需要结构化。我们也不会告诉你每个网站必须立刻从 HTML 转换到 XHTML——尽管我们希望 XHTML 的优势迫使你考虑尽快进行转换。

本书的作者因为提倡尽可能用 CSS 替代传统的 HTML 表格而闻名。CSS 标准为开发者和读者解决了大量问题，CSS 设计是未来趋势，并已经被许多网站使用。从大型商业站点，比如 [wired \(www.wired.com\)](http://www.wired.com)（如图 3 所示），到搜索引擎 (www.alltheweb.com)，到公共部门和个人网站。

图 3

这是大企业网站，它是用 Web 标准和 XHTML 及 CSS 建立的 (www.wired.com)



虽然 CSS 具有不可否认的优势，但是彻底抛弃表格设计还需要很长时间，表格依然是很多设计工作的方便工具，它能够完成工作，且能够符合标准或向后兼容。这是你的网站，你有权利选择。

在一些纯标准论者的头脑中，流行的专利技术，如 Flash 和 QuickTime 将在 Web 上没有任何地位。这种观点理论上也许正确，但是我们还有工作要做，这些技术如 Flash 和 QuickTime 对解决某些问题是最好的工具。除了一些相关的标记和可访问性，本书不会对这类技术讨论更多，并不是因为讨厌 Flash 和 QuickTime，而是因为这些技术超出了本书范围。

你也许会遇到一些 Web 标准鼓吹者反对你使用 GIF 格式，他们也可能会指责你用 GIF 制作标题而不用 CSS。通常推荐用 XHTML 制作标题，这有时还是最好选择，但在实际应用中，一种用 nicely kerned 字体制作的 GIF 格式图片也许比用 CSS 定义的 Arial、Verdana 或者 Georgia 字体更适合你的网站访问者。这是你的网站，你有权利选择。

实践，不是理论

我们并不反对那些用激情驱使 Web 标准创建的纯理论者，相反，我们由衷地赞美和非常庆幸从他们那里获得的帮助和知识。

但这本书是写给正在工作的设计师、开发者、客户和为技术支付费用的雇工的。我们关于 Web 标准的探讨将立足于设计、内容和出版上，我们的目标是指导你了解整个 Web 标准体系，并帮助你做决定——做出那些让设计师或者客户能够永远真正受益的决定。

如果说本书有一个教条的、坚决的、僵硬的观点的话，那就是：商业成本普遍偏高，没有一位阅读本书的人能够负担得起用过时的方法设计的网站的高昂费用。

当一些标准刚起草时，守旧的技术有绝对的地位，当时主流浏览器对标准的支持少得可怜，但那些日子一去不复返了。时间推移，在网络繁荣和预算过度膨胀时期达到顶峰的“6 种版本代码”的方法曾经看起来是合理的实践，但在今天，一样地逝去了。

现在是 XHTML、XML、CSS、ECMAScript 和 DOM 的时代，虽然它们不是最终的技术，但它们组合起来成为一个解决方案，能够解决自从<blink>标记时代以来的困扰所有者和开发者站点的问题。

什么是“Web 标准组织”(Web Standards Project)

创立于 1998 年，Web 标准组织帮助终止浏览器战争，说服了 Netscape、Microsoft 和其他浏览器开发商正确地、完全地支持标准，减少因为开发的复杂性而增加的成本，保证对所有浏览器简单有效的访问。除了浏览器制造商，该团体还与开发工具制造商（如 Macromedia）、网站所有者和开发者合作。Web 标准组织是一个联合组织，它的活动是完全自愿和非营利的。

这样的旅行真的必需吗

为了使网站避免被淘汰的困境并达到新的水平，设计师和开发者必须学习和使用 Web 标准。网站所有者和管理者必须清楚标准为什么对他们的业务有帮助。公布的 Web 标准并不能说明白它自己，商业用户不太可能去浏览 W3C 的站

点，查看其“字典型”文档，以及直观地领会那些含义模糊的只取首字母的缩写词（如 XHTML 或 CSS）的关系他们利益的技术。同样，忙碌工作的设计师和开发者，总在为进度期限而努力奋斗着，只有很少的时间通过邮件列表和搜索到的易于理解在线教程来学习那些网络新技术。

为了提供一些标准的实际用例，我们不得不写这样一本书。作为 Web 标准组织的创办人之一，这个工作落到我身上。我的名字是 Jeffrey。虽然是我主笔，除了绝对必要，我都以“我们”作为第一人称。这是我从 1995 年成为一个网站编辑开始就一直采用的，相信读者不会因为我这个习惯而困扰。

虽然你可能宁愿阅读有关图形和动画设计、新的网站架构思想及其可访问性，也不想了解 Web 基础技术的变化，但是我也宁愿写那些东西。但是如果我们的网站不能在浏览器 X 或者设备 Y 中工作，我们最好的设计、结构和可访问性都将被浪费。如果电影规格、镜头、录音技术没有广泛的行业协议，就不会有电影。同样地，Web 设计健康的发展依靠 Web 标准的采用，没有那些标准就没有真正的可访问性和连贯的设计。

在我们接受标准的期间，Web 仍然以比任何传统媒体快的速度发展着，但是商业的发展远远快于行业标准，这就把我们建设的网站扔在了一个非常危险的位置，当新的浏览器或者设备出现后网站就不断地荒废。我们仍将忙于建设而没有时间思考我们的工作方法是否正确。今天，若想继续我们的工作和建设，就必须检查和调整我们的方法。

我们所有人都可以采用 Web 标准建立长久的、漂亮的网站，并且可以在将来仍像今天一样很好地提供服务。在这本书中，我将解释每个标准，以及它们如何一起工作建立一个向后兼容的网站，但具体的工作还是由你自己去做。

Jeffrey Zeldman
于纽约市
2003 年

99.9% 的网站都是过时的

几乎 Web 上的所有网站都有同样的机会得一种病，从简陋的个人主页到花费上百万的商业站点。这种病很狡猾而且隐藏很深，可以让你的站点不知不觉中受到伤害。这种病目前尚未被很多人认识到，因为它是基于行业惯例的。尽管网站的所有者和管理者可能不清楚，但事实上 99.9% 的网站是过时的。

那些站点在主流的版本为 4 或 5 的浏览器看起来工作得很好，但是，一旦离开了可包容这些错误的环境，弊病的症状和危害就开始显现。

在 Microsoft 的 Internet Explorer、Opera 软件公司的 Opera 浏览器、Netscape Navigator，以及 Mozilla（一种开放源代码，基于 Gecko 的浏览器，Navigator、CompuServe、AOL for OS X、AOL China 及其他浏览环境都是基于它的代码开发的）的当前版本中，谨慎构建的布局已经开始部分失效，一些代价昂贵的 Web 交互行为也停止工作。在主流的浏览器发展的同时，网站性能继续在恶化。

在不知名的浏览器中，在残障人士使用的屏幕阅读器中，在日益流行的从 Palm Pilots 到上网手机等非传统设备中，大量的网站将无法工作。根据需求和预算，网站所有者和开发者既不会支持那些不知名的浏览器和设备，更不会为支持它们而专门定制标记和代码，因为网站只为“正规”浏览器服务。

探究过时的行业惯例中的有害部分，看它如何导致成本的不断增加，如何使得 Web 网站开发越来越复杂而从未真正达到过预定目标，我们来比较一下现在的浏览器和过去非标准兼容的浏览器之间有多少区别。

1.1 现代浏览器和 Web 标准

贯穿整本书，当我们提及“现代”或者“符合标准”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

及 W3C Document Object Model (DOM) 标准。所有这些形成的标准，允许我们超越表现层的标记、不兼容的脚本语言，以及它们所造成的永久的淘汰怪圈。

在写这本书的时候，类似的浏览器包括：Mozilla 1.0 和更高版本、Netscape Navigator 6 及其更高版本、Windows 系统下的 Microsoft Internet Explorer 6 及其更高版本、Macintosh 系统下的 Microsoft Internet Explorer 5 及其更高版本、Opera 7。若需要浏览器的比较列表，可以看本书第 3 部分附录 A “现代浏览器：品质优良的、低劣的和非常低劣的”。注意那不是一张详尽的列表，我们试图在本书出版前列出每一个符合标准的浏览器。虽然我们称这些浏览器“符合标准”，但请记住本部分的“在开始之前”中说过：没有一个浏览器能够完美地符合标准。

缺少完美浏览器不是逃避标准的理由。目前有上百万人使用 Windows 系统下的 Internet Explorer 5 或者 5.5。从标准的角度出发，这些浏览器都是劣质的，包括 IE6/Windows、Netscape 6+，等等。如果你的访问者使用的是 IE5/Windows，你就可以抛弃 Web 标准？你是不是应该告诉使用 IES/Windows 的人去升级浏览器，否则就拒绝他们？我们认为不应该这样。符合标准的设计和开发并不是说：“只为最新的浏览器版本设计”。

同样地，使用 XHTML 和 CSS 不必告诉 Netscape 4 用户去做大的改变。一个设计合理并采用标准的站点，在 Netscape 4 和其他符合标准的浏览器中看，不一定每个像素都要一样，事实上，通过你的设计技巧，它们看起来可以完全不同，这样做更被认同。我们将在本书第 2 部分“设计和构建”中解释其中原因。

1.1.1 为新工作新代码

现代浏览器不仅仅只是老浏览器的一个新版本，它们的基础也发生了变化。在很多案例中，新版本完全被重新设计，Mozilla、Netscape 6/7 浏览器不是 Netscape Navigator 4 的新版本；IE5+/Mac 也不是 IE4 /Mac 的简单升级；Opera 7 不是基于老版本的代码开发的。这些浏览器已经为新工作写了新代码，也就是说，能够尽可能地、完美地、更贴切地符合本书所说的标准。

与之对比，20 世纪 90 年代的专有浏览器（Netscape、Microsoft）技术很少考虑标准，老的浏览器完全忽略了一些标准，它们也造成许多开发的麻烦。例如，如果浏览器不支持 Portable Network Graphic (PNG) 标准，那么开发者也不会使用 PNG 图片，这都没问题。可是麻烦在于，那些老浏览器嘴上说得好听，要支持一些标准，但实际上只是支持了部分，甚至是错误的支持。比如漫不经心地支持基础的 HTML 标准，造成混乱的 Web 发布环境。

当病人发生阑尾炎，有资格的外科医生将做完整的阑尾切除手术。现在想像

一下，一个新手只切除了一半阑尾，还随意刺伤一些其他器官，最后又忘记缝合伤口，结果会怎样。我们为这个恶心的假设向您道歉，但这就和老浏览器对待 Web 的健康一样：它们对标准的支持是极其不完整的、不合格的、冒险的。

如果 Netscape 4 忽略作用在<body>元素上的 CSS 规则，在页面的每一个结构元素中随意增加一些空格，如果 IE4 能正确读取<body>标签的 CSS，但是对齐很糟糕，那么哪一种 CSS 写法是安全的呢？一些开发者选择根本不写 CSS；另外一些开发者则针对 IE 写一个样式表，针对 Netscape 4 再写一个样式表。为弥补跨平台字体和 UI 的细节差别问题，开发者或许还要定义不同的样式。这样做还没有考虑浏览器用的是 Windows 还是 Mac 系统（对于那些 UNIX 或者 Linux 用户就更糟糕了）。

这些 CSS 问题只是九牛一毛。浏览器在 HTML、标记表现和脚本语言上可能都不支持 CSS。没有一个正确的方法组织页面内容（是的，还有一个正确的方法，但是没有浏览器支持）；没有一个正确的方法设计页面（是的，有一个，但是没有浏览器支持），也不能在页面上增加复杂的交互行为（最后有办法了，但是老浏览器不支持）。

如果希望站点能够在多个浏览器或操作系统中正常工作，设计师和开发者在艰难应付不断增多的兼容问题的同时，还要为每一个不同的浏览器定制设计（非标准的）标记和代码。在今天来说这是错误的，因为现在的浏览器已经支持同样的开发标准。然而事实上这样的情况仍然在继续，不必要的浪费资源，切分 Web，导致网站的可访问性和可用性更差。

1.2 “版本”问题

用非标准的标记和代码创造多个版本，每一个都迎合不同浏览器的“癖好”，这是困扰大多数网站“永久淘汰”的根源，目标总是在后退，游戏规则永远在改变。

尽管这种做法是昂贵的、无效的、不可忍受的，但实际工作中仍然有很多人这样做。面对一个支持 Web 标准的浏览器，许多开发者仍然把它当成不支持，因此，尽管 IE6 能够处理标准的 ECMAScript 和 DOM，而人们却继续写探测脚本来判断浏览器是否是 IE6，并调用针对 IE 的脚本来适应它。尽管 Netscape 能够处理标准的 ECMAScript 和 DOM，他们仍然习惯于为最新的 Netscape 浏览器写独立的脚本和代码。

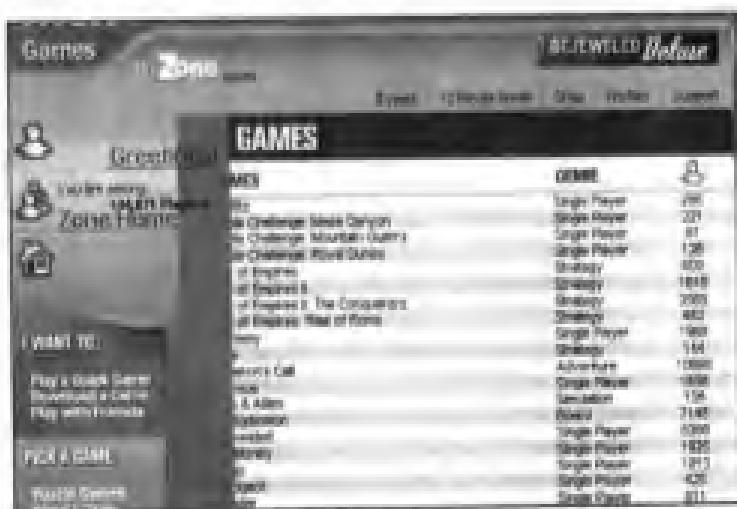
用一个例子来说，在今天有利于标准的环境下，浏览器探测技术和针对性的版本创建是不必要的。事实上，这样做更加糟糕，在不断升级变化的情况下，不是所有的站点管理者都能负担得起，因此探测脚本常常失败。

例如，在 Windows 系统下，Opera 浏览器将自己定义为 IE，这样做主要为了避免被一些只允许 IE 访问的网站（比如许多银行站点）屏蔽。但是为 IE 专门写的脚本很可能在 Opera 浏览器中失败。当 Opera 浏览器定义自己为 IE（这是安装时的默认值），并且开发者写了 IE 专有脚本时，访问网站就会失败，用户会受到极大挫折。用户有权选择他们的用户代理（User Agent）和强迫 Opera 以自己的身份而不以 IE 来替代，但是只有很少用户知道这样的操作，他们也不需要知道。

除了私有脚本外，开发者还写那些需要占用两倍带宽的表现层标记，或者提供一个难以被搜索引擎、非主流浏览器和设备访问的服务页面。这些策略会导致很多问题，他们被迫进行解决：在不同浏览器下看到的界面不一样，如图 1.1 所示。

图 1.1

MSN 游戏乐园(cone.msn.com/blog.asp) 提供了 7 种外部样式表，仍然不能在大多数现代浏览器上完全地表现出来。网站同时欢迎您有 14 个脚本（大部分是内联（inline）的），包括复杂的浏览器探测，但是仍然有代码在多个版本下有问题，不能正常运行。



多版本带来不断增加的成本和难题。“DHTML”网站产生于 Netscape 4 和 IE4 的不同的不兼容的脚本，但是它们现在又不能工作在新版浏览器上了，网站所有者应该将更多的钱花在这个问题上吗？请开发者建立第 5 甚至第 6 个版本？如果他们没有为这些版本做预算怎么办？那么许多用户将被网站拒绝。

同样，开发者可能花费巨大的时间和资源来建造一个“无线”的版本，却发现他们用的无线标记语言已经过时，或者他们的无线版本在新的流行设备中无法访问。一些网站重新建立一个新版本，另外一些只能发布令人为难的消息，许诺在“不远的将来”支持新设备。

那些对旧式方法忍痛割爱的设计师和开发者在运用像 XHTML 和 CSS 这样的 Web 标准技术时，往往不得要领。而更多墨守成规的设计师和开发者由于坚决不采用标准来避免多版本问题，创建的多个特定浏览器和平台的 CSS 文件均自相矛盾，自取灭亡，如图 1.1 和图 1.2 所示。



图 1.2

Adobe 网站 (www.adobe.com)
较深页面的内容显示：糟糕的文字
交叠。不是采用建立一两个替换样
式表来适应所有浏览器，开发者过
度地估计具有极强交互性的特定
浏览器和平台的样式。设计师不知
道采用 CSS 和 XHTML 就可以不需
要多版本这重要的一点

这些做法浪费时间和金钱，自从西方经济进入百年低迷后，日用品不再丰富，也没有特别的补给。更糟的是，用昂贵的做法去解决问题依然失败。网站依旧破产，用户依旧被拒绝。

1.3 向前兼容的思想

查看任何一个 2003 年的站点的页面源代码，从 Amazon 到 Microsoft.com，从 Sony 到 ZDNet，检查他们复杂的非标准的标记，他们私有的 ActiveX 和 JavaScript（常常包含断链检查），以及他们拙劣使用的 CSS（当他们完全使用 CSS 后）。这样的站点能工作在任何浏览器上简直就是一个奇迹！

因为早先的 Netscape Navigator 和 Microsoft Internet Explorer 的第 4 代、第 5 代浏览器产品支持非标准的标记和特定浏览器代码，这些站点能工作在昨天主流的浏览器中。为了自己浏览器的市场份额，他们居然在拙劣的战争中鼓励私有的标记和脚本。

通常，非标准站点能工作在以前的浏览器上，是因为它们的所有者已经投资购买了昂贵的能够适应多种浏览器的发布工具，可以建立多样的、非标准的版本适应特定浏览器和平台的特殊要求，就像上一节“‘版本’问题”中所述。实际上，他们为了不同版本代码分支，大量嵌套的表格，空像素和其他图片处理，过时的或者不完善的标记，以及属性浪费带宽，使拨号用户负担加重。

什么是代码分支

代码是程序员写的用于在我们的数字世界里建立软件产品、操作系统或者其他大量事物的语句。当多组程序员工作在一个项目上时，每个开发组为了解决不同的问题或者遵循不同的进程，代码就可能被“分支”到多个不相容的版本中。

本书提及的代码分支是指创建不兼容的多版本，以应付不支持 ECMAScript 和 DOM 标准的浏览器的需求。

在同一时间，这些多版本浪费了网站的带宽，越大的站点浪费越严重，越多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。

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，一个组织一年花费 2500 美元的话就可以节省 875 美元，如果花费 160000 美元就可以节约 56000 美元。

Yahoo 的首页（如图 1.3 所示）每天服务数百万次，每在过时的 HTML 标记上浪费一个字节，都将成倍增加大文数字的页面负载，导致 10 亿字节通信浪费，Yahoo 服务器的负担将成倍增加。如果 Yahoo 简化它的代码，用节约带宽的 CSS 代替耗费带宽的标签，如图 1.4 所示，每页的服务成本会减少，公司收益得到增加，那么 Yahoo 为什么不做这样的改变呢？

我们只能推断 Yahoo 公司希望他们的站点在现代浏览器中看起来和 1995 年的不支持 CSS 的浏览器中一样。具有讽刺意味的是，没有一个 Yahoo 管理者关心 Yahoo 页面看起来像什么。站点巨大的成功是因为他们提供的服务，而不是漂亮的视觉设计（就像不存在的一样）。

还有那些网站外表华丽的公司，浪费不计其数的带宽只是为了一个看上去好看却没有人会赞扬的页面。究其原因只是开发者们根深蒂固的“向前兼容”思想，并把这看得比网站可访问性、合理性或者公司利益更重要。



13

Yahoo 站点 (www.yahoo.com)
看起来就像一个朴素的白板



69

查看 Yahoo 首页的源代码，你会发现如此简单页面的代码竟会如此地繁杂冗长。

1.3.1 过时的标记：网站所有者的成本

假设一个旧式的网页的代码和标记有 60KB，那么，替换过时的``标

签和清除其他私有的、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（在我的实践中，我们常常可以用 22KB 甚至更少来替换 60KB，保守地说，至少可以节约 50%），看下面两个典型的案例。

1. T1 终结者

情节：一个经常有几百人在线的自有主机的小型业务或者公共网站服务组织，在通过清理表现层标记，用 XHTML 结构化页面，切掉一半页面垃圾代码后，节约了 1500 美元/月。

如何做到的：在代码精简前，为这些用户提供服务需要租用 2 条 T1 线，每条线月租是 1500 美元（一条 1.544MB/s 的 T1 线的正常成本）。当裁剪文件尺寸 50% 后，该组织发现只要 1 条 T1 线就可以有效提供服务，因此从这一处理上节约了 1500 美元/月。带宽节约后，同样将节约一些硬件费用。简化你的代码标记，使页面显示更快：页面显示越快，对服务器的压力就越小，你就可以购买、维护、升级更少的服务器。这个做法对于现在应付动态的，基于数据库的内容网站是切实有效的。

2. 节约每一兆字节的流量

情景：当商业主机托管变得流行，网站拥有者发现他们每月都支付了一种意想不到的文件流量罚款，从几百甚至到几千美元。裁剪文件一半尺寸后，每月的账单恢复正常，费用趋于合理。

如何做到的：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，你每月只要付一般费用，超过了，则必须付更多。

在一个声名狼藉的案例中，因为超出限定流量，主机提供商 Global Internet Solutions 要求独立设计师 Al 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，Sacui 在核对后避免了支付，但经历了漫长的司法战争，因为那家主机提供商改变了服务条款而没有通知他 (<http://thewebfairy.com/gis01/>)。谁愿意冒高额账单风险或者和主机提供商拖到司法程序中去呢？

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。例如，在 Pair.com，超出 1MB 的费用一般是 1.5 美分。一个采用 Pair 主机的小通信量的站点通过裁剪代码尺寸可以每年节约 200 美元，更大更高通信量的站点可以节省更多。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，占用带宽越小，和你的主机提供商因为流量超额而产生冲突的可

能性就越小。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。

精简与压缩代码

在对 Web 标准发表了一次演讲时，一位听众中的开发者对作者宣称，由干净的和结构良好的代码标记所节约的带宽对于采用压缩 HTML 技术的公司来说微不足道。

除了通过精简代码压缩你的页面之外（或者仍旧采用过时的“HTML 设计”方法），你还可以在一些服务器环境下压缩代码标记。例如：Apache 服务器加载的 mod_zip 模块可以在服务器端压缩你的 HTML，在用户浏览器重新把它解开。

那个开发者举了个例子：如果 Amazon.com 在过时的代码和垃圾代码上浪费 40KB，但是用 mod_zip 压缩掉 20KB，Amazon 的肿胀的标记的费用将比演讲中建议的要少。

事实上，Amazon 没有使用 mod_zip，很少有商业网站使用这个工具，可能是因为发通页面前需要额外的加载原因。但是，从另一个角度说，越小的文件压缩后更小。如果你从 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，可以大量地减少操作费用，也可以预防额外的费用。（就像前面的例子：你可以不必要再租借另外一条 T1 线来应付超额带宽。）

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 技术的网站也是有现实意义的。

1.3.2 向前兼容

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，不管怎么说，“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 或者 Netscape7。在专业开发领域中，“向前兼容”理论的声音洪亮，就像举着圣杯。但是这样做成本太高，实践也总是基于一个谎言。

没有真正的向前兼容，总是有一个切断点。例如，不论 Mosaic（第一个可

视化浏览器)还是 Netscape 1.0，都不支持基于表格设计的 HTML。需要解释一下？好，那些用远古浏览器看网站的人们不可能和用稍新一点的浏览器浏览的用户获得同样的视觉体验，比如 Netscape 1.1 或者 MSIE 2.0。

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 Netscape 3，网站将不支持更古老的浏览器 (Netscape 2 的用户就不幸运了)。为了履行他们支持基线浏览器的承诺，开发者针对不同版本的浏览器细节，用非标准的处理方式反复设计他们的标记，导致每一个页面都在加大加重。

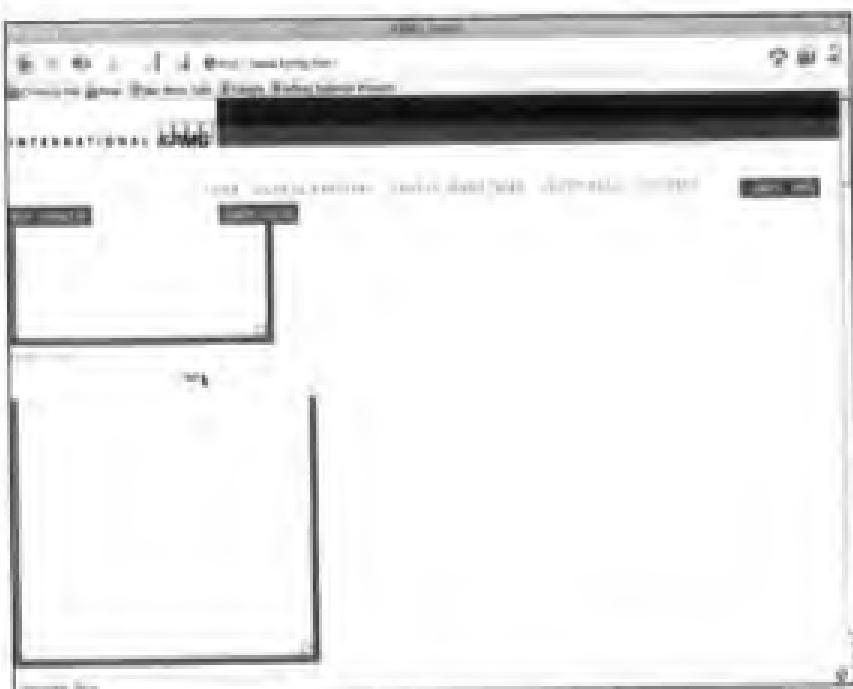
在同一时间，开发者也需要写多个脚本来适应不同浏览器，检测用户的浏览器并运行不同代码以使页面看起来最好。正是因为这样做，页面尺寸不断加大，服务器压力不断增加，想挣脱“永久淘汰”的怪圈，直到他们财力枯竭或者被淘汰。

1.3.3 屏蔽用户对业务不利

一些公司愿意多花点钱试图保证在旧版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，在减少开支的目标压力下，越来越多的网站只针对 IE 浏览器设计，甚至某些情况下只针对 Windows 平台，因此他们失去了 15%~25% 潜在的访问者和客户。参见图 1.5、图 1.6、图 1.7、图 1.8、图 1.9。

图 1.5

KPMG (www.Rgnyg.com) 网站在 Navigator 中的画面效果，我们宁可不叫 Navigator 看，这样的页面得感谢愚蠢的只针对它的代码



因为这种目光短浅的做法而会失去大量的客户。如果有公司说不要 1/4 的潜

在客户，那么谁都会理解这种商业模式。

根据 NUA Internet 调查统计报告 (www.nua.ie/surveys/)，截止 2002 年 9 月有超过 650 600 000 的因特网用户，你可以自己计算损失。

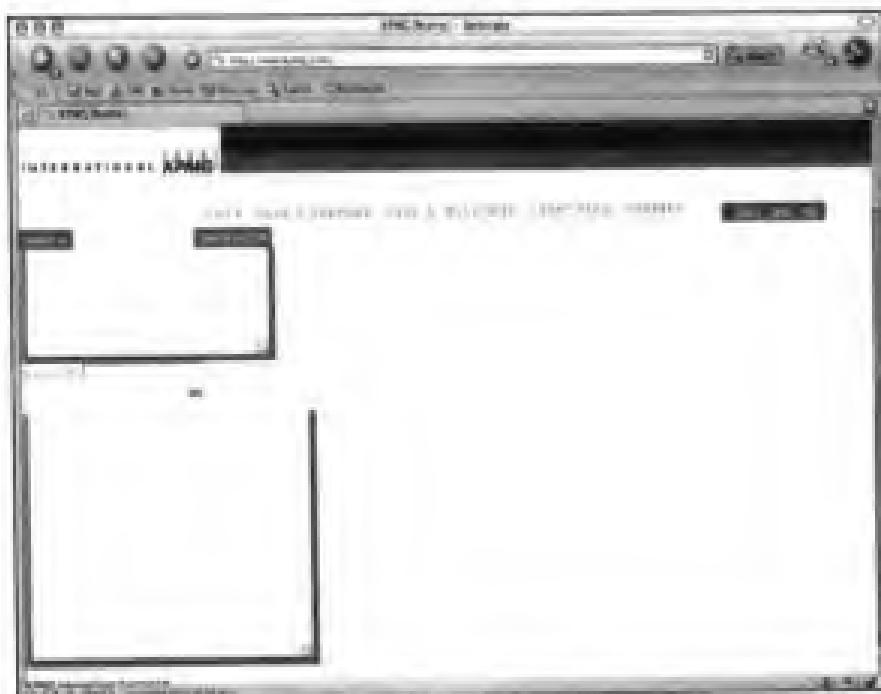


图 1.6

KPMG 网页在 Netscape 7 中同样无效，我们猜想公司根本不关心他们的客户



图 1.7

好，如果这个站点只针对 IE 浏览器，那么让我们试一下 Macintosh IE5 版本。噢！它也不能工作。（这个站点在 IE5/Macintosh 下一切正常，一半失败，没有任何理由）

说你不介意你的站点损失 25% 的访问用户，而继续“IE-only”方法也是不理性的，因为你不能保证 IE（或者桌面浏览器）在 Web 领域会持续保持主导优势。

图 1.8

同样的站点在 IE6/Windows 下，这才是最初的设计



图 1.9

当 Opera 7/Windows 定义自己是阳时，网页非常漂亮。（如果定义自己是 Opera，站点无法打开）

在写这本书的前几年，Netscape 的 Navigator 浏览器占领的市场份额远远大于 Microsoft 的 Internet Explorer，在那时，惯例是只支持 Netscape。在 Microsoft 投入数不清的金钱过后，市场改变了，那些只支持 Netscape 的站点变成了信息高速公路上的数字垃圾。

同样的命运会不会落到只支持 IE 的站点上？看上去难以想像，但是在 Web

上没有什么是不变的，惟有变化是永恒的。由于日益广泛使用的非标准网络设备和个性化设计的桌面浏览器等因素，排斥所有其他浏览器和设备的业务决定绝对是不明智的。

此外，正如本书所展示的一样，使用标准就可能设计出适用所有浏览器和设备的站点，简单而快捷，在花费大量成本建立一个向前兼容的网站和压缩成本建造只服务单一浏览器的网站两个极端之间，Web 标准提供了唯一的出路。

不论浪费金钱的多版本方法，还是深思熟虑决定只支持一种浏览器或平台的方法，都不能帮助今天的网站继续工作在未来的浏览器上，或者是超越桌面的、日益繁荣的、每天都在变化的世界中。而如果继续这样做，成本和复杂性就会不断升高，直到大部分的公司都无法负担和升级网站的费用。

当我们努力在不同浏览环境下提供相同的体验时——使网站看起来像印刷物，操作起来像桌面软件——我们损失了它的真正的本意，即网站对所有人都应该是丰富的、多层次的、易接近的媒体。

在短暂的网络繁荣期，当设计师和开发者不规范地维护网站，无标准地学习，为某种浏览器细节而设计页面的时候，我们逐渐失去了它的本意，最终导致我们现在面临网站淘汰的关口。

许多网站已经发展到荒废期，只是还在垂死挣扎，当你读到这些文字，无数的网站都已经倒闭。如果你是网站拥有者、管理者或者创建者，警钟已经敲响！

1.3.4 愚蠢之路

早在 1997 年，有一个通用惯例：为 Netscape 浏览器写 JavaScript，为 Microsoft 浏览器写 JScript（一种类似 JavaScript 的语言）。使用 JavaScript（只工作在 Netscape）和 ActiveX（只工作在 IE/Windows）为不同的浏览器写代码也有类似的通用惯例，那就是我们为 3.0 浏览器做的。

这种方法对其他小的没有品牌的浏览器（比如 Opera）没有一点好处，并且不能正确作用在 Macintosh 平台的 IE 上。但是它可以为大部分的 Web 用户工作，并很快成为当时的行业惯例，如果我们想建立比静态页面看起来更可爱动态网站，我们除了使用这些程序别无选择。

1997 年下半年，Netscape 和 Microsoft 都推出了 4.0 浏览器，每个都吹嘘有强大的“动态 HTML”（DHTML）能力。你可能已经猜到，它们完全互相不兼容，而且和它们自己早前的版本也互相不兼容（在 Netscape 4 中工作的不能在 Netscape 3 中工作），更不用说那些不知名的浏览器了。它们从顺从地支持 HTML 等基础规范，转变为创造它们自己的语言和属性。

规则可以自己随意制定吗？Netscape 和 Microsoft 认为可以，许多设计师和开发者也同意。那些持反对意见的人除了咬咬牙用功做出需要的版本，发布一个可以接受的“专业”站点之外，别无选择。

我该如何编码呢？让我想想

既有针对 Netscape 4 的 DHTML，又有不兼容的、更适合在 Windows 系统下工作的、针对 Internet Explorer 4 的 DHTML，但是没有针对 Netscape 3 和 IE 3 的非 DHTML 的脚本语言，针对其他不知名的浏览器和更早期的浏览器或许有附加的代码，或许根本没有。简而言之，甚至最简单的网页也需要比意大利面更缠绕的代码。

一些开发者限制他们自己的网站只能适合两个浏览器版本（一个针对 IE4，另一个针对 Netscape 4），并要求访问者安装一个 4.0 浏览器，否则将无法访问。另外一些预算更少的，就将所有事情限制在一个浏览器上。

在 4.0 浏览器推出市场不久后成立的 Web 标准组织，评估当时 Web 开发方法，每个功能都需要写四个或者更多版本，因此设计和开发任何网站上将会至少增加 25% 的成本。

一些开发者对这一评估的反应是非常不屑的。因为网络越来越热，客户心甘情愿地付费，这样大网站运营商何必担心成倍代码和标记给网站带来的高成本？然而因特网泡沫的破灭，网站预算随之缩小或者冻结，运营商开始缩小规模或者转移到其他行业，一夜之间，几乎没有人能够负担得起多版本网站的费用。

当此行业从裁员和倒闭中蹒跚走出来时，新一代的支持公共 DOM (W3C 制订的) 的浏览器出现了，这个发展意味着什么？它意味着版本问题找到出路，一个新的以标准为基础的设计和开发的时代的到来。我们低迷的行业经济如何对这渴望已久的消息做出反应？这个行业仍旧处于继续写分支代码、开发只针对 IE/Windows 的版本或者转到 Macromedia Flash 的状态中。因为过度的商业幻想，所以 Web 行业目光短浅。

1.4 当好事情发生在坏标记上

早在接受计算机语言教育时，他（她）就学会一个短语“Garbage In, Garbage Out”（输入的是垃圾，产出的是垃圾）。不仅仅因为语言（类似 C 和 Java）鼓励严格的代码习惯，他们也需要它。

同样，图形设计学习的第一件事情就是：素材质量决定最终产品的效果。开

始于一个高起点、高质量的照片，打印和制作 Web 图片都会看起来很好。而试图用一个低质量的快照或者低分辨率的 Web 图像设计，最终结果往往不值得一看。你可以将一个高质量的 EPS 转换为合适的优化的网页标志，但是你不能将一个低分辨率的 GIF 转成一个高质量的 Web、打印或者电视标志。这就是“输入的是垃圾，产出的是垃圾”。

但传统的主流浏览器却不是这样工作。不严格的、甚至荒谬的是，它们吞噬所有断掉的标记和 JavaScript 源文件中的坏链接，一点都不停顿。在大多情况下，站点显示似乎都正确。因此设计师逐渐在工作中养成不严格的习惯，开发者也养成了大量他们未察觉的坏习惯。在同一时候，这种工作方法已经使中间件和服务器端开发者相信像 XHTML、CSS 和 JavaScript 这些技术也是粗糙的、卑劣的。

不尊重工具的人就不太可能正确使用它。参考下面的 HTML 代码片段，这是从一个正在市场为生存挣扎的公司的电子商务站点上截取的：

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

无意义的``标签是为了抗议``标签而故意的排版——这个``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！错误的标记看起来对你很熟悉，类似的标记可能也出现在你的站点，在上面的网页中，其实只需要下面一句代码：

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

结合一个适当规则的样式表，可以用上述更简单更结构化的标记替代臃肿的、非标准的、残废的标记。同时还能节约服务器和用户带宽，并易于将来转换到一个以 XML 为基础的标记的灵活的站点上。上述例子中的电子商务站点还包含下列断链接的 JavaScript：

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

这段代码的问题是，语言属性“language”和 source 标签连在了一起，没有分开。另外，浏览器会告诉用户一个不存在的脚本语言“JavaScript1.1src”。

据任何理性的评估，这个站点最终都将失败，应该警告开发者所犯的错误，督促他们尽快修改。然而，直到最近，这个站点的 JavaScript 仍旧能工作在主流的浏览器上，因此形成代码糟糕的站点和纵容它们的浏览器都存在的怪圈。我们已经不奇怪那些熟练的代码者为什么总是看不起前台的设计者。

垃圾代码威胁你的站点长期健康

当最新的浏览器遵循 Web 标准时，它们将对设计师和开发者的要求日益严格，因此逐渐地不会再宽容残缺代码和标记。“Garbage in, garbage out”的准则开始被全世界的浏览器所了解，使得任何一个设计师和开发者都必须了解 Web 标准知识。

过去所受的损失已经无法挽回，我们只能用更好的方法设计和建造新网站，使它们可以适应众多的浏览器、平台和设备，解决网站淘汰和屏蔽用户的问题，Web 朝向更远、更强、更易用、更理性的方向发展。

治疗网站“重复建设、不断淘汰”的弊病，就会发现其核心是建立一个普遍支持的技术，并以此作为“Web 标准”。学习使用 Web 标准设计和建造网站，我们可以保证我们创建的每个站点都向后兼容。

“写一次，即可发表在任何地方”，Web 标准的许诺，是非常令人向往的概念。今天它已经开始实现，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工作中的设计师和开发者那里，大量的新站点依旧被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。

1.5 治疗

设计师和开发者历经在长期地依赖主流浏览器制造商的深渊中的挣扎之后，终于可以使用能够保证网站交互行为和外观的技术，不再是针对某个单一的制造商的浏览器，而是所有的。

经过 W3C 会员和其他标准组织的锤炼，像 CSS、XHTML、ECMAScript（JavaScript 的标准版本）和 W3C DOM 等技术已经被当前浏览器支持（包括 Netscape、Microsoft、Opera 和其他厂家），并可以设计和做到如下内容。

- 在图形桌面浏览器上达到更精确的控制、定位和排版，允许用户使用适合他们的表达方式进行编辑。
- 可以开发工作在多浏览器和平台的复杂交互行为。
- 遵守可访问性原则和指南，而不需要牺牲美观、性能或者精巧性。
- 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工作烦恼。
- 支持多种浏览器，从而不需要争论和考虑多版本的成本，很少或根本就不

需要代码分支。

- 支持非传统的设备，从无线设备到孩子们想像到的、可以上网的智能手机，以及盲人阅读器、屏幕阅读器等残疾人使用的设备，都不需要再争论开发特殊版本的费用。
- 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”或者依赖昂贵的私人出版系统来建立类似的版本。
- 通过把样式从结构、行为中分离，以及严格的文档结构，易于在高级发布流程中重新设计 Web 文档。
- 从老的 Web 语言 HTML 转换到更强大的以 XML 为基础的置标语言。
- 可以在当前符合标准的浏览器和平台上确保正确地工作，也可以在老浏览器中工作。
- 保证这样设计的站点将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……

在我们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，找出老技术是如何产生网站不断淘汰的怪圈的，在第 2 章“根据标准设计和制作”将展示所有这些内容。

根据标准设计和制作

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制作一个网站的呢？Suck.com（一个早期的、幽默的、独立的 Web 期刊，如图 2.1 所示）在每天的头版，读者不可能错过的地方，疯狂地用一种犀利的写作风格，巧妙地、公正地展现自己的内容。现在，这是显而易见的，但在 20 世纪 90 年代中期 Suck 刚成立的时候，大多数网站把自己的内容埋藏在外观好的网页、欢迎页、任务条，以及内容列表页之后。



图 2.1

Suck，一个简单明了的商业网站先驱者（www.suck.com）

在大多数商业站点过度包装页面（例如“提高我们的计数器，获得 Web 女神的垂青”）的时代，Suck 的正直超前的文字、不夸张的比喻上让人耳目一新。同样，Suck 简洁的网站外观，在那个多数站点过度设计（金属感的斜角和高技术，发光字）及被自学 HTML 的系统管理员毫无设计地胡乱堆在一起的时代显

得尤为突出。在主要使用 Netscape 1.1 的时代中，许多站点造就了所谓的版式设计家，从重复背景图片到私有的<center>标签，还有一些站点依旧沉迷于技巧展示（如图 2.2 所示）。在越来越多的 Web 上，Suck 大胆的做法显得格外突出。

图 2.2

“Moon’s Design”网站的 Ulead 发光 Frame 的教程(moondesigns.com)是 20 世纪 90 年代中期的典型 Web 设计。内容居中放在中间的表格中，外面的表格使用一个重复的背景，前面的表格覆盖在上面



为达到 Suck 网站与众不同的以内容为主、外观简洁的目标，创建者 Carl Steadman 和 Joey Anuff 放弃了传统方法。那时，HTML 缺乏设计工具，有一个很好的原因：当发明 Web 的物理学家 Tim Berners-Lee 构思时，HTML 是一个结构化的置标语言(<http://www.w3.org/MarkUp/html-spec/>)，源自 SGML，而不是一个类似 Adobe 的 PostScript 或者标准样式表的设计语言（CSS 至今仍然是 W3C 推荐认可的，并且一经认可，它存在了至少 4 年，直到浏览器支持和标准本身规格充分完善）。

所以，Steadman 和 Anuff 是如何控制站点的表现呢？他们利用创造力和想像力完成这些。

2.1 跳出那个禁锢

为创建 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后回车，并在后面插入一个<p>段落标签。

```
<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.
```

整个作品被`<tt>`(“typewriter”)标签包装，以强迫早期的浏览器(主要是Netscape 1.1)把文本的字体改成一种叫monospace(类似Courier或者Monaco)的字体。

运用基本的排版控制和强迫的标题模拟，这样的HTML处理手段是1995年用于实现设计效果的惟一方法。(图2.1中的视觉效果来自1996年，在设计技术稍微加强一点后，Suck重新设计——依然保持简洁的设计。原始的设计已经不能再访问。)

相同的强迫HTML符合排版设计效果的创作方法被网页设计师广泛采用，并在早期的像Lynda Weinman和David Siegel这样的作者写的Web设计“圣经”教程中被传授。HTML创建者喋喋不休地讨论他们的HTML处理方法。当客户强烈要求好看的Web界面时，设计师没有其他选择。

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且达不到目的的设计方法。另一方面，2002年优秀的Web设计书籍依然直接面对面地建议它的读者使用font标签和“HTML scripts”来控制字体样式。事实上，Font标签早就被反对使用了(W3C的说法是：请不要使用这种过时垃圾)，HTML也不能脚本化。但是像这样错误的、毫无意义的建议广泛地分布在Web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 标准出现以前的设计成本

依靠HTML创造性的操作，Suck达到了与众不同的外观，但是花费了两倍的成本：网站排斥了一些读者并且要求读者升级浏览器版本才能浏览。

如果使用早期的Mom-and-Pop屏幕阅读机(一种为视觉障碍的人们所使用的声音浏览器)大声朗读Suck的文本，因为不断的段落标签，会在几个单词就停顿一会儿，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，理解这些话都比较困难，Suck 的令人费解的句子结构被没有语义的段落标签分割后变得更加难以理解。间断地提交这些声音是屏幕阅读机无法克服的问题，对他们来说这个站点是不可用的。

在图形浏览器中做设计时采用的 HTML 技巧使得部分用户无法访问，Suck 的创建者在升级的时候也遇到同样的问题。

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，最后成了一个社团的领导站点，网站创建者被迫需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的基本发布需求。

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？但是不管标准的出现，目前大多数的商业站点仍旧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。这种状况如此广泛，以至于许多设计师和开发者从未想过要停止它。

2.3 时髦的站点，古老的方法

从 1995 年跨越到 2001 年，我们考察一个当代站点 “The Gilmore Keyboard Festival (www.thegilmore.com)”。如图 2.3 所示，它是一个漂亮的站点，是用早已成为行业惯例的劳动密集型的表格排版技术制作的。

事实上，站点已经考虑了访问者的显示器规格和浏览窗口大小，那么这些图片有什么问题呢？从拥有者的角度看，这个站点有两个麻烦的地方。

- **改版的巨大成本**——如果有任何关于艺术节的信息变化，例如，如果一个额外的连续独唱会加入到本年计划中，站点就无法通过简单的文本链接进行更新。设计师也不能很简单地增加一个热点链接到原来的 GIF 图片上，不能映射到服务器，增加到导航菜单中。包含多个图片的 HTML 表格外观上是一个整体，如图 2.4 所示，任何部分图片的尺寸改变都将使表格变形。



图 2.3

Gilmore Keyboard Festival 网站 (www.thegilmore.com) 看起来很可爱，但是难于维护和更新



图 2.4

同样的站点，用 CSS 扩大了边框和空白以展现页面构造方法（以及潜在的令人头疼的维护问题）

因此，即使最小的改变也将需要很大的成本。图片将不得不被重新设计、分割和优化。表格标记将不得不重新写，连同图片热点链接和 JavaScript 代码。当一个如此简单的任务（如：增加一个链接）需要用几个小时来完成时，你就会怀疑行业惯用的生产方式是否已经证实是有效的。

- 把众多潜在用户排除在外——不明显并不代表不重要，这个站点对那些使用屏幕阅读器、文本浏览器、掌上电脑、可上网手机及那些使用常规浏览器但不显示图片的人们是难以使用的。这些访问者将什么都看不到，因此损失的商业机会将更多。下面是网站在一个无图像的环境（如图 2.5 所示）下看到的所有内容：

[INLINE]

[INLINE] [INLINE]

[INLINE] [INLINE] [INLINE]

[INLINE] [USEMAP] [INLINE]

[INLINE]
Contact Us
[INLINE] Privacy Statement
Copyright 2001

图 2.5

在一个不支持图像的浏览器中，网站没有提供任何信息：没有文本，没有链接，没有任何东西。使用一个免费的在线 Lynx emulator 浏览器 (www.delorie.com/) 就能看到这样的屏幕。去试试你的网页在无图像的环境下是什么样子。用 Lynx 或者一个屏幕阅读器，比如 JAWS（或者其他）测试一下。



“INLINE INLINE INLINE”的信息是 Gilmore 网站创建者希望得到的吗？我们表示怀疑。“INLINE INLINE INLINE”对潜在的访问者有帮助吗？显然不是。

不是每一个有类似遭遇的访问者都不做反抗。理论上，一个耳聋者有权利要求站点区别对待他们。但我们不是律师，不能故意挑剔这样有很好目标、清晰创建的站点。

我们也不能说那些漂亮图片是不好的。相反，图片是必需的，漂亮也很重要，Gilmore 站点的创建者已经在艺术上做得很好，给用户美观的在线体验。那些体验是很容易接近的。排版布局看起来严谨，对用户也容易理解，但这并不是全部。

尽管非同一般的漂亮，但 Gilmore 在设计和构建方法是非常典型的，进而由这些方法所引起的要求网站所面对的问题也是典型的。我们发现，当客户希望改变时，我们精心设计的表格布局总是被破坏，当所有的客户经常这样要求，我们要么让客户交费要么自己承担成本。

当有限的预算被严格控制时，例如，当经济低迷时，客户可能需要快速而不美观的解决方案，就会破坏已存在的典雅和清晰的界面外观。

例如，在 Gilmore 站点的顶部或者底部增加三行文本链接，以便让访问者能够浏览最新增加的内容，但这样做会引起误解。（为什么呢？访问者会感到奇怪，为什么这些链接与其他的不一样？它们是否比其他链接更重要或者更不重要？）

增加这样的链接也会破坏站点精心设计的、严谨的艺术效果，减少艺术节网站品牌的严肃性，也因此影响站点的市场价值。

像 Gilmore 站点的创建者，我们发现守旧的设计排版方法已经不能满足需求。它们在主流浏览器的“通常”环境和“一般”个性设置情况下看起来很好，但超过标准环境，我们的站点也许就停止传达正确信息，至少，如此的“不可访问性”肯定阻止了部分的潜在用户。

Gilmore 的拥有者和设计师不是唯一的例子，他们遵循已经长期形成的行业惯例。问题是那些所谓行业惯例在今天大部分的网站继续产生同样的问题，包括不采用 Web 标准而是使用某些流行的浏览器的私有技巧来设计的网站，以及强制用 HTML 把表现和结构混杂在一起的站点。

幸运的是，现在有了另外一个设计和建造网站的方法——一种解决所有守旧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 2.6 示意的是 Web 标准是如何通过把网页分成三个独立组成部分来解决我们已经讨论过的问题的：结构（Structure）、表现（Presentation）和行为（Behavior）。

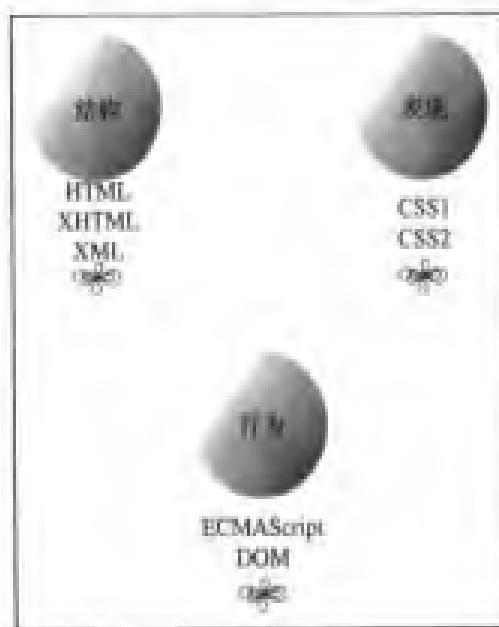


图 2.6
结构、表现和行为：在 Web 标准中组成任何网页的三个部分

2.4.1 结构

一个置标语言（XHTML；<http://www.w3.org/TR/xhtml1>）包含的格

式化的文本数据，它的结构包括：标题、副标题、段落、数字列表、定义列表，等等。

在 Web 上，这样的文本可能是定义列表`<dl>`中的一部分。标题“Structure”将被`<dt>`标记定义为标题，内容片段将被`<dd>`定义为内容数据。

```

<dl>
  <dt>
    Structure
  </dt>
  <dd>
    A markup language (http://www.w3.org/TR/xhtml1) contains text data formatted according to its structural 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. The subhead, "Structure," would be marked up as a definition title. The paragraph you're now reading would be wrapped in definition data tags.
  </dd>
</dl>

```

第 2 个段落还可以再使用一个`<dd>`子元素。

```

<dl>Structure</dl>
<dd>
  <p>A markup language (http://www.w3.org/TR/xhtml1) 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. The subhead, "Structure," would be marked up as a definition title. The paragraph you're now reading would be wrapped in definition data tags.</p>
</dd>

```

“’的含义

如果你花时间阅读 XHTML 例子，你可能想知道“’是什么意思。非常简单，这是标准的 Unicode 的 Escaped 字符。更多细节，参看第 5 章“现代置标语言”。

标语言，能够提供更多的选项。但现在我们强迫自己先去使用 XHTML，它是一种当前 W3C 推荐的，可以在最近所有浏览器和网络设备上工作的，就像 HTML 一样的，过渡期的置标语言。

正确撰写（指没有错误、残缺标签和属性）XHTML 标记是十分方便的，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。

这种标记也能包含设计师认为需要额外的结构，例如，内容和导航可能需要用适当的自定义标签预定义：

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

这种置标语言也能包含嵌入对象，就像显示文本一样，在浏览器中显示图片、Flash 或者 QuickTime 电影。

2.4.2 表现

表现语言（CSS1——<http://www.w3.org/TR/REC-CSS1>；CSS2——<http://www.w3.org/TR/REC-CSS2>）用来格式化网页、控制字体、布局、颜色等。

在许多案例中，CSS 能够代替旧式的 HTML 表格。在所有的案例中，它都可以替换非标准的 font 标签和浪费带宽的、过时的垃圾代码，就像这样：

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

这样的垃圾代码可以用一个不修饰的表格加一个样式表代替，或者根本不需要加样式。

因为表现与结构相分离，所以改变一部分而不影响其他部分是可能的，例如，你可以在众多页面使用同样的布局，或者在不改变布局的基础上改变文本和链接。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，它不像设计语言负担双重责任。

同样地，你可以改变布局而不触及标记。读者抱怨你的站点字体太小吗？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，不论他们在 PC Web 浏览器屏幕上看起来是什么样，你的页面就可以打印得非常漂亮（我们将在第 2 部分“设计和构建”中介绍如何做这些）。

2.4.3 行为

一个标准的对象模型（W3C DOM 参见 <http://www.w3.org/DOM/>

DOMTR#dom1), 工作在 CSS、XHTML 和 ECMAScript 262(<http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>)。JavaScript 的标准版本上, 使你可以创建出运行在多平台和浏览器上的交互行为和效果。不再有只为 Netscape 服务的 JavaScript, 不再有只为 IE/Windows 服务的 ActiveX 和 JScript (就像我们在第 1 章 “99.9% 的网站都是过时的” 中所描述的。)

根据站点目标和服务对象, 设计师和开发者能够运用 Web 标准将表现、行为和结构充分相分离, 或者他们可以选择建立一个混合旧的和新的技术过渡期网站。例如: 混合简单的 XHTML 表格布局, 用 CSS 来控制字体、空白、标题和颜色。

2.5 实际应用

如果 Suck 网站能够活到今天, 那么像 XHTML 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 XHTML 模板将表现文档结构, 用 CSS 将控制外观, 不需要额外的设计工作, 而只要准备配文图片即可。段落标签将明确表示出段落的起点和结束, 不需要在每行文字间强制行间距 (CSS 完全可以做到)。

在一个类似 IE、Mozilla/Netscape 和 Opera 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。通过结构化的 XHTML, Suck 的内容不仅仅可以表现在浏览器中, 还可以表现在个人数字助理 (PDA)、屏幕阅读器和文本浏览器中, 不再需要设置假的段落的间隔和类似标记。

作为一个以内容为本的站点, Suck.com 是严格地将其内容和样式转换 XHTML/CSS 的最好候选站点: 它用 CSS 控制布局, 用 XHTML 控制内容结构。但是 Suck 也从过渡方法中受益: 简单 XHTML 表格负责主要内容区域的定位, CSS 控制其他。

Gilmore 站点创建者就不能从使用模板中受益, 至少, 像它现在设计的站点首页就不能。但是他们可以用 CSS 来控制已经存在的布局, 当设计师改变一部分页面而不需要重新设计整个布局, 这样就节约了带宽。

Gilmore 站点能够用 CSS 背景属性来放置主要图片, 即使在对 CSS 支持不完善的 4.0 浏览器中, 也可以用一个单独的 JPEG 文件代替一打的小图片 (如图 2.4 所示), 使用成熟的 CSS 方法能够更容易地覆盖一个或多个菜单图片。

Gilmore 的创建者也可以使用有效的 XHTML 及其可访问性的属性, 包括 alt, title 和 longdesc, 保证站点内容容易理解, 而不是在大多数人看来

是晦涩难懂的，如图 2.5 所示。图片较少的网页可以采用过渡方法（表格加上 CSS）或者严格的方法（纯粹的 CSS）转换。

2.6 过渡方法的好处

采用 XHTML 和 CSS 的过渡方法是我们今天的重要进步，它将解决我们讨论的众多问题。虽然与一个无表格的方法相比，用过渡期的 CSS 方法需要花费更多的成本和费用（在大多数情况下，还需要更多带宽），但使用过渡技术——CSS 控制字体、颜色、空白等，XHTML 表格控制基本布局，这样可以增加网站的可用性、易访问性、协同性、长期生存能力，虽然比用无表格、纯 CSS 方法要花费更多的工作和代价（需要更多的带宽）。

Happy Cog (www.happycog.com) 是我的网站代理商的一个商业站点，是采用过渡方法的网站例子之一，如图 2.7、图 2.8、图 2.9 所示。它结合了 XHTML 表格布局技术和 CSS1、CSS2 以及简单的基于 DOM 的脚本，它符合 XHTML1.0 和 CSS 标准，也遵守 508 条款的可访问性要求和 WAI Priority 1 指导方针（关于指导的更多细节在第 2 部分介绍）。



图 2.7

Happy Cog，这个商业站点 (www.happycog.com)，是一个过渡期的例子。它用 CSS 和 DOM 结合最新的 XHTML 表格布局做到向后兼容，用任何现代浏览器访问，都表现得十分正确。

因为符合 XHTML、CSS 和 508 条款，即使浏览器和标准发展，Happy Cog 站点也能长期生存。使用一些守旧的方法（主要是表格布局），它在新浏览器中保持良好的外观，也能在浏览器战争时代的不完全支持 CSS 的浏览器（像 Netscape 4）中保持适当的外观。在同一时间，Happy Cog 使用简单的结构标记和易访问性技术允许它的内容可以在非图形浏览器、屏幕阅读器和新的无线设备中正常显示。

Happy Cog 站点的过渡期策略提供了向前和向后兼容，这是一个适合今天大多数网站使用的策略。在使用 Web 标准设计和建造网站获得巨大收益时，你的思维方式和方法也会发生根本的转变。

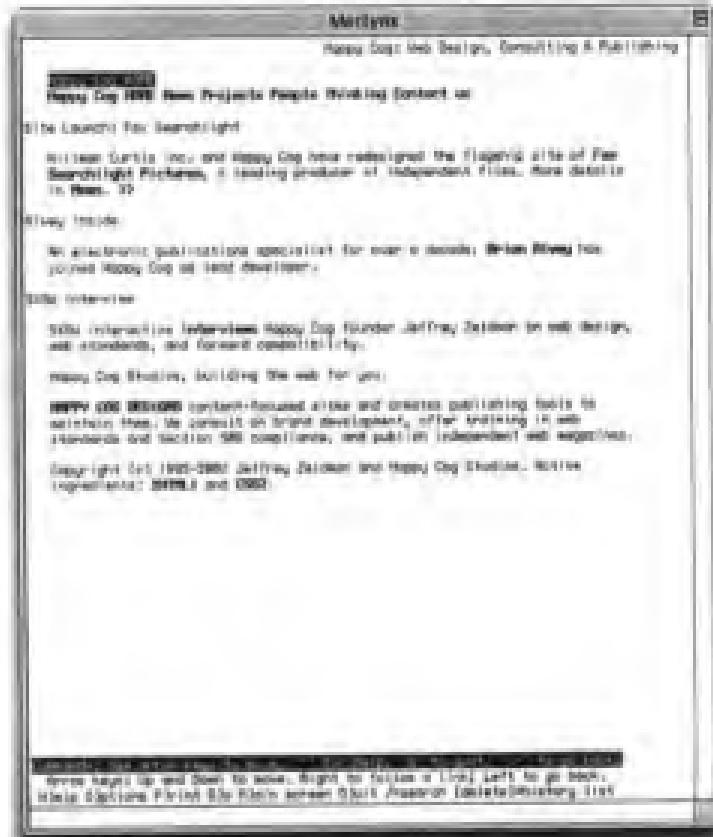
图 2.8

用一个 CSS 支持性不太好的老浏览器（Netscape 4）打开站点，除了少数细节设计部分无法显示外，大部分 Happy Cog 的网页都表现得较完美。但我们不会介意：大部分 Netscape 4 浏览器的用户对此都已习惯，也不会介意。



图 2.9

再说说 Happy Cog 网站。这次用文本浏览器 Lynx 浏览，就像图片中显示的一样，当分离视觉表现，文本内容能够完全显示出来。与图 2.5 进行对比，Gilmore 站点根本不能在非图形浏览器中工作。Happy Cog 在设计上并不比 Gilmore 聪明，它只是使用了标准保证站点的易访问性。Gilmore 却不能。



当你的网站希望达到流水线生产，保持内容的完整性和可移植性，并服务于不同用户的访问设备的目标，采用 Web 标准不仅仅是应该的方法，也是它们今天工作惟一方法。

结构与表现及行为相分离是这种设计方法的基础，这也是未来所有站点的设计方法（除了使用全 Flash 的站点）。它已经被无条件使用在向后兼容站点上，就像我们展示的那样。

2.7 Web 标准组织：跨平台性

Web 标准组织(WaSP)（如图 2.10 所示）发起于 1998 年，目的是说服 Netscape、Microsoft 和其他浏览器生产商彻底支持本书介绍的标准。他采用从容进行、坚持和策略性的方法（又称“叫，大叫和辩论”方法），最终使得浏览器生产商同意引进 WaSP。通过共同的标准协同工作，是 Web 向前发展必需的观点。

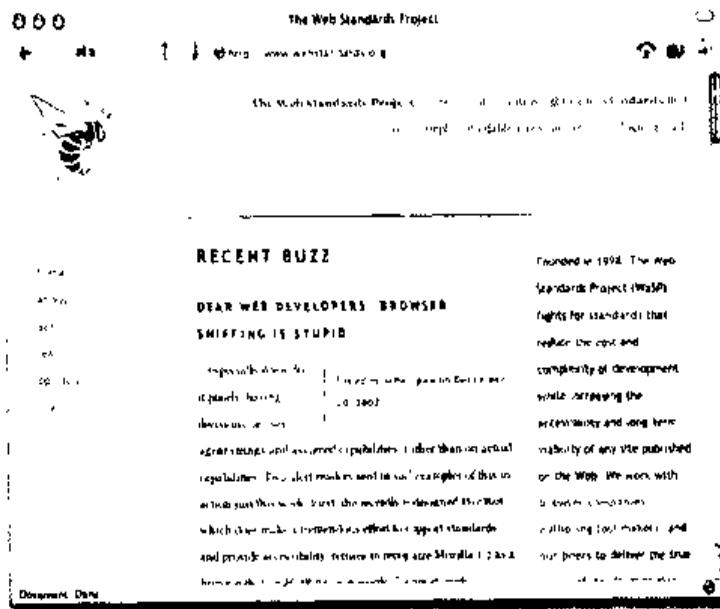


图 2.10

Web 标准组织的主页在 Mac OS X 上的 Chimera（一种基于 Gecko 的浏览器，www.webstandards.org）中浏览的画面。它的 CSS 布局在所有现代的、符合标准的浏览器中看起来都一样。但是等等，还有更多

小贴士

在努力使浏览器符合标准的过程中值得讽刺的一件事就是：Netscape 和 Microsoft 都是 W3C 的会员，而且都已经为 Web 标准做出重大贡献，但却不得不逼迫他们完全支持这些标准。

浏览器终于开始在完整意义上支持标准（参见第 3 章“推广标准的困难”），Web 标准组织在 2002 年重新启动，以鼓励设计师和开发者学习和使用这些强大

的、好不容易形成的标准技术。为了表示 Web 标准组织新的使命，从权威讲坛到教育资源，站点都进行了重新改写和设计。

就像预期的那样，站点在符合标准的浏览器中看上去很好，在老的、支持标准不完善的浏览器中也看起来可以接受（如图 2.11 所示），而且站点还超越了 PC 的浏览器的范畴，不需要额外附加的标记、代码或者设备版本检测，就能工作在其他的设备上（看吧，没有多版本代码）。

图 2.11

同样的站点在 Netscape 4 中看起来大方且可以接受，而不再需要一个特别的“Netscape 4 版本”



2.7.1 一个文档服务于全部

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，也不再需要多版本，当你使用标准去设计和建造，一个文档可以服务全部。

图 2.12 是 Webstandards.org 网站在一个 Palm Pilot 中显示的效果。图 2.13 是在 Microsoft 的 PocketPC 中显示的效果。更让人惊奇的是，图 2.14 显示站点很好地工作在你喜欢的 Newton 手持设备中（Apple 的一种已经停产的 Palm Pilot 前身产品）。提供截图的 Grant Hutchinson 告诉我们：“用一个过时的操作系统能看现代的网站，这样的事情从未有过。”

这对于那些希望用最少的努力获得最多访问者的设计师或者网站所有者来说，无疑是最美妙的事情。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。

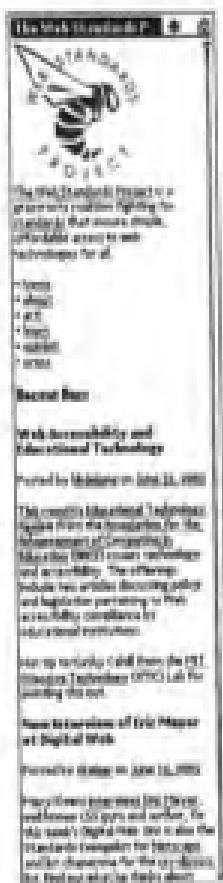


图 2.12

同样的站点在 Palm Pilot 中的截图。看，不需要 WAP 版本。感谢 Porter Glendinning (www.servyc.com/adg/) 提供屏幕截图

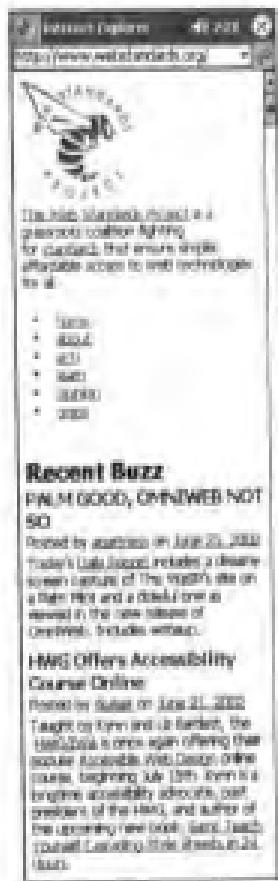


图 2.13

同样的站点显示在 Microsoft's PocketPC。感谢 Ant Dush (www.dashes.com/en1/) 提供屏幕截图

注意：在图 2.12 到图 2.14 中，在普通桌面浏览器中的屏幕左边显示的 DHTML 菜单到 Palm、PocketPC 或者 Newton 中转变成了普通的列表，是魔法完成的吗？其实很简单，DHTML 菜单实际上是一个无序列表，用 CSS 在符合标准的桌面浏览器中改变它的外观，菜单通过普通的服务器端包含(Server Side Includes,SSI) 实现一页到另一页的转换。

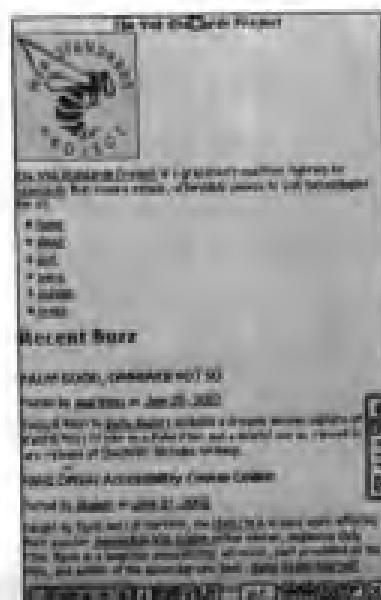


图 2.14

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.spliro.com) 提供屏幕截图

2.8 A List Apart：一个页面，多种浏览方式

A List Apart (www.alistapart.com) 是作者专门为“制作网站的人们”开办的在线杂志，于 2001 年 2 月转为只用 CSS 排版的网站。外观的设计采用的是站点标榜的“最少限度的 HTML 代码”的观点，如图 2.15 所示。站点以前曾用 HTML 表格完成过。

A List Apart 是作者为网站设计师开办的在线杂志，图 2.15 是在支持 CSS 的浏览器中的效果 (www.alistapart.com)。ALA 在 2001 年 2 月转为只用 CSS 排版的站点，数百家其他站点也很快跟随改变。

灵感的源泉

所有在 A List Apart (以及 zeldman.com) 中使用的样式表都是开放源码的，你可以在创建站点时随意使用和修改。若要查找站点的样式表，可在你的浏览器中选择“查看源代码”，注意 CSS 文件的引用位置在文档的 `<head>` 区，然后将那个文件位置剪切并粘贴到你的浏览器地址栏，例如，如果 XHTML 文档的 `<head>` 中类似这样：

```
<style type="text/css" media="all">
@import "/styles/basic.css";
</style>
```

或者

```
<link rel="StyleSheet"
      href="/styles/basic.css"
      type="text/css" media="screen" />
```

你可以在浏览器地址栏输入 <http://www.domain.com/styles/basic.css>。大部分浏览器将以普通文本方式显示样式表，然后你可以剪切并粘贴到 HTML 编辑器或者下载到本地硬盘中。

为了节约时间，你可能宁愿访问 www.favelets.com 并将查看 CSS 样式表的 bookmarklet 安装到浏览器工具条。单击一次，可将任何站点的样式表立即导入到浏览器的新窗口。

我创建了 ALA CSS 与 Todd Fahrner 和 Tantek Çelik 一起重新设计，他们两个都参与了 W3C 的 CSS 工作组，Fahrner 是 CSS 专家及 Web 标准组织的核心成员。Çelik 是 Microsoft 的浏览器工程师，也是 Favelets 的创建者。

不喜欢默认的文本设置的 ALA 读者可以通过 Paul Sowden 开发的样式表转换器转换成更大更清晰的字体。ALA 出版的第 126 期杂志 (www.alistapart.com/issues/126/) 解释了这个转换器如何工作并提供开放源代码，你可以在网站中使用和修改这个样式表转换器。（参考第 15 章“使用基于 DOM 的脚本语言”和第 16 章“CSS 重新设计”获得更多信息）

图 2.15 显示了 A List Apart 在符合 CSS 标准的浏览器（比如 IE5+、Mozilla、Netscape 6+ 及 Opera 5+）中的外观，在其他环境下，网站外观变得完全不同。

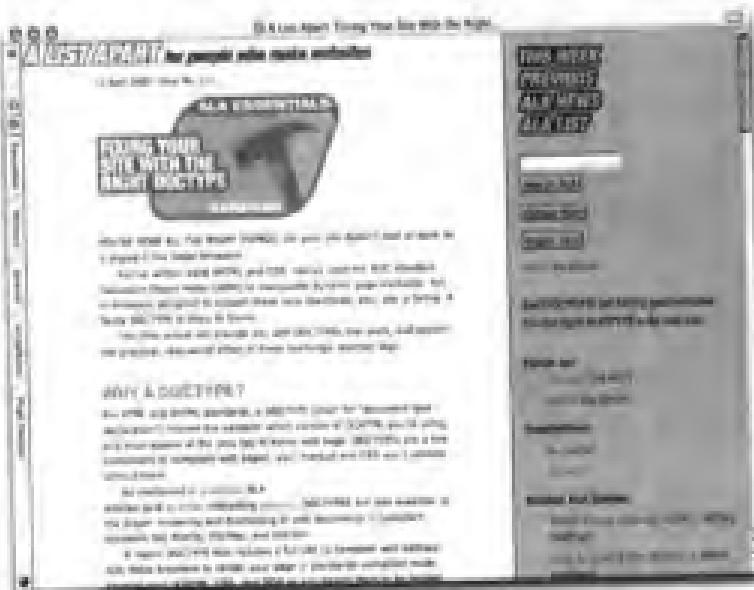


图 2.15

A list Apart 是作者为网页设计师开办的在线杂志，如在 CSS 兼容浏览器所看见的。ALA 在 2001 年 2 月转换到仅 CSS 布局，其他成百的网站纷纷跟随。

图 2.16 显示了同样站点在一个不支持 CSS 的浏览器的浏览结果，这里用的是 Netscape Navigator 4。站点依然完美和清晰可用。

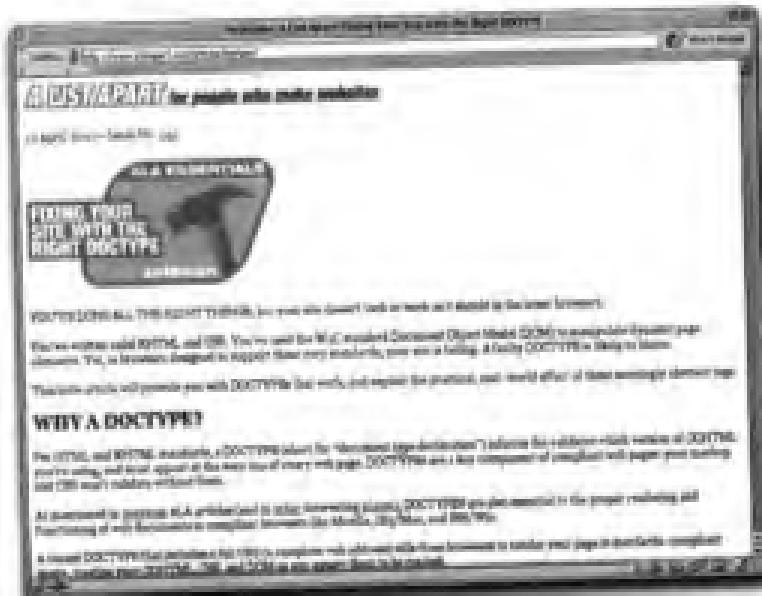


图 2.16

同样的站点在一个 4.0 浏览器中的效果，没有 CSS 排版也没有问题，站点依然能阅读和使用。

事实上，一些读者可能更喜欢这个方法：那些使用 Netscape 4 的新的 ALA 的访问者，因为我们的 CSS 设计方法，而可以看到一个简单清晰页面，不再是早前惨不忍睹的页面。那些持有“标准会伤害老浏览器用户”观点的人们从这个故事可以得到恰恰相反的启示：只要合理使用，标准可以帮助每一个人。

在图 2.16 中，你会注意到导航条好像不见了，事实上，它只是简单地移动到页面底部。文本的布局和尺寸完全根据用户的参数设定，并没有使用检测浏览器的方法来判断是否使用简单格式，代替的方法是使用样式表 CSS，在屏幕上呈现的就是一个用这种方法的链接，不符合标准的浏览器会忽略它（我们将在第 2 部分“设计和构建”详细说明）。

用这种方法，将不会有表格或者其他非结构化的垃圾代码。惟一还使用的旧“设计”标记适用于`<body>`标签里的背景颜色属性，所以在不支持 CSS 的浏览器中，头条图片也能和那些页面的背景相匹配。

2.8.1 超越屏幕的设计

最后，但并非最不重要，图 2.17 展示的是一篇 A List Apart 文章打印出来的效果。如你所见的，工具条已经被移走，字体和颜色已经针对打印而最优化。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。

图 2.17

从网页上打印：ALA 上的文章便能实时友好打印，这归功于打印样式表



这是通过一个单独的打印样式表完成的，是由工作在更早期打印样式表的 Eric Meyer 设计，并被 Todd Fahrner 和本书作者继续开发的。Meyer 是一位 CSS 专家，是“Eric Meyer on CSS”(New Riders, 2002)一书的作者。他的 ALA 文章“Going to Print”(www.alistapart.com/stories/goingtoprint/)，说明了创建打印样式表技术的基本原理，我们将在本书的第 2 部分介绍类似技术。

现在需要领会的重要观点是：用一个单一的轻量级的文本（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”版本，同样你的站点也不需要。除了一些例外：那些使用多篇文章显示格式的站点，像 www.wired.com 或 O'Reilly Network (www.oreilly.com) 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单一文档。但是，他们也可以通过使用打印样式表获得便利，显然，输出那个全文链接页面可以使用打印样式表。

让我们来回顾一下上述两个站点的获益。

2.8.2 节约时间和成本，增加扩展性

如果采用标准设计和构建，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：

- 不需要仅仅针对 Netscape 的版本
- 不需要仅仅针对 IE 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 WAP 和 WML 版本
- 在大多数情况下，不需要打印友好版本
- 不需要浏览器和平台检测，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

许多人或许想给他们的用户提供无线或者适合其他非传统设备的服务，但是许多组织可能负担不起独立的无线版本或者简单的纯文本。感谢 XHTML 和 CSS 标准的出台，他们不再担心，甚至连手指头都不用抬，就可以获得大批的新读者和用户。

严格兼容标准也提供了一个解决可访问性问题的高起点。如果你的站点能够工作在一个 Palm Pilot 上，它也就可能像工作在 Jaws 的屏幕阅读器上。当然，你需要测试确认。你可能需要做一部分工作保证确实可用。我们将在第 14 章“可访问性基础”，以及在第 2 部分的一些章节讨论可访问性和 508 条款。

2.9 我们要去往哪里

基于 XML 的置标语言（请看第 4 章“XML 征服世界”）将使今天的 Web 看起来很初级，但是用昨天的设计和开发惯例就无法进入未来的 Web 时代。

有两个前进的方法：过渡型的向后兼容（一种极好的混合传统和标准技术的方法），以及严格基于完全分离（或几乎分离）结构、表现和行为的向后兼容。

过渡型的向后兼容在今天混乱的浏览器环境中更容易接受。它很好地迎合了外观至上的需要，以及适合大多数仍旧使用不完全符合标准浏览器的用户。严格的向后兼容，就像它的名字含义那样，坚持最新的标准精神，是最好的向后兼容的方法，应用在适当的环境下并且能够获得最大的益处。让我们来看看那些最新的方法。

2.9.1 过渡型的向后兼容

组成因素

- 正确的 XHTML 标记（HTML 4.01 也能使用）。
- 正确的 CSS 控制字体、颜色、空白等。
- 尽量少使用 XHTML 表格设计布局，通过 CSS 来避免表格的多层嵌套。
- 可选的：在重要的表格单元中应用结构化标签（便于 CSS 和脚本技术的采用，帮助以后向较少表格的布局过渡）。
- 基于 DOM 的 JavaScript/ECMAScript 可能有代码分支，用以适应 IE 和 Navigator 4.0 浏览器。
- 可访问性和测试。

推荐给

过渡型的向后兼容方法被推荐给那些访问者中高比例使用 4.0 或者更老版本浏览器的网站，可以不用一直等到所有工作都充分地支持 CSS；也推荐给那些表格做得比 CSS 控制的布局更好的场合使用。从可访问性和长期生存的角度出发，过渡期方法已经被用在纽约公共图书馆的各分支图书馆中（如图 2.18 所示），以适合数量巨大的 Netscape 4 用户，直到完全符合 XHTML 和 CSS 标准。

优点

- 合理的向前兼容：站点可以在老浏览器中显示相当好的效果。它们将在更新更符合标准的浏览器中表现得更好。



图 2.18

纽约公共图书馆 (www.nypl.org/branch/) 使用过渡方法转向 Web 标准：用浅显的表格和正确的 XHTML 布局。过渡型的向后兼容是一个简单的标准转换，适合大多数的组织。

- 向后兼容：站点将在今后的浏览器和设备中继续工作。
- 开始铺平通向基于 XML 的标记和纯 CSS 的布局的过渡之路。
- 当垃圾标记和私有代码被清除后，在当前的浏览器中只有很少的维护问题。
- 可访问性加强：减少可访问性问题带来的客户损失和起诉风险。
- 部分地修复了文档结构（我们说“部分”，是因为代码标记依旧包含一些表现设计代码）。
- 开始恢复典雅、清晰和简单的标记，因此缩小了文件尺寸，从而减少带宽浪费，降低发布、生产和维护成本。

不足

- 结构和表现仍将混合在一起，使维护和更新困难，成本很高。
- 同样的原因，网站将来转向基于 XML 的内容管理系统会很困难，并且成本很高。这对一个小小的站点可能不是问题，但是对于包含成千上万动态生成页的大型内容和商业站点来说，将是一个很大的问题。

2.9.2 严格的向后兼容

组成因素

- 从表现和行为中完全分离的结构。

- 正确的 CSS 排版布局，表格仅仅用做它原始的目的：表现一个表格数据，类似电子表格、地址簿、股价、记事本等。
- 正确的严格的 XHTML 1.0 或者使用过渡的标记。
- 强调结构，在标记中没有表现的处理或者尽量少用表现处理标记。
- 结构化标签和提取设计元素。
- 基于 DOM 的脚本处理交互行为，仅在需要的时候限制代码分支。
- 可访问性和测试。

推荐给

严格的向后兼容推荐给任何网站，只要它的用户不是一个高比例老浏览器的使用群体（典型的 4.0 或者更早的浏览器）。严格的方法可以允许非标准浏览器的访问内容，但在提供交互行为给那些老浏览器的时候，经常失败。

优点

- 向后兼容：在现有的和未来的浏览器和设备中（包含无线设备）增强的交互能力。
- 从基于 XML 的标记向更高级转变。
- 用极少的工作量获得更多用户。
- 不需要多版本。
- 较少的可访问性问题，这样设计的内容一般可用于所有人。
- 恢复典雅、简易和逻辑性的标记。
- 恢复文档结构。
- 更快、更容易、更便宜的发布和维护，在低预算时避免紧张；在高预算时，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。

不足

- 站点极有可能在老浏览器中看上去比较朴素。
- 支持 CSS 浏览器是不完美的。
- 一些用 HTML 表格轻松实现的设计用 CSS 布局变得困难，一些设计需要重新考虑。
- 一些符合标准的浏览器（如 Opera 预览版本 7）可能阻止基于 DOM 的交

互行为。

- 基于 DOM 的交互行为将不能在 4.0 和更老的主流浏览器、屏幕阅读器、文本浏览器和多数的无线设备中工作。你需要使用<noscript>标签和 CGI 来让那些浏览器和设备进行访问。

第 2 部分解释了标准是如何工作的，提供了技巧和策略来解决关于各种各样 Web 开发的设计和商业问题。但在我们开始钻研前，先讨论一些可能已经长期困扰你的问题。

如果标准增加交互能力，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、老板和卖主来阅读呢？为什么 Web 标准没有被更广泛地理解和使用呢？

这一堆奇怪的现象，我们在下一章一一解说。

推广标准的困难

Web 标准是通向可访问性、高效的 Web 设计和开发大门的钥匙，但是在重要的商业网站和过去几年创建的网站中你很难找到它。在这一章，我们将探究为什么 Web 标准还没有成为所有的设计工作室和公司设计团队的工作习惯，为什么 Web 标准还不是每个站点计划或者需求中必需部分的原因。

如果你更喜欢阅读 Web 标准成功的案例，请直接跳到第 4 章“XML 征服世界”，如果你已经接受标准并卷起袖子准备大干一场，请直接跳到第 5 章“现代置标语言”。但是如果你需要使同事也相信 Web 标准的价值，或者你想简单了解一下为什么行业能够不使用它们就达到标准，那么这一章就是为您准备的。

3.1 看起来可爱，代码却丑陋

在 2002 年中期，我和 6 个来自新媒体团体的人担当第 8 届每年一度的交流艺术的交互奖（www.comarts.com）的评委，这是一个行业中有名的设计竞赛。竞赛中提交的作品是近几年来设计和开发技巧最好的。

我们最初花了 10 周的时间浏览数千的网站和提交的光盘，缩小范围到数百个，最终选出获奖者不到 50 个。最后的评奖地点选在一个海湾，我们 7 个人在那里被“关押”了一星期，直到获奖者被选出，我们才能离开。到周末，我们已经选出了 47 件作品，终于可以从束缚中解放出来了。

为了庆祝最后的评定（以及重新获得的自由），我与一位旧金山的朋友共进晚餐。竞赛引起了我朋友的兴趣，他也了解一些关于 Web 开发的事。

我朋友问：“你是否关注站点是不是符合标准？”

我回答：“没有一个站点是符合标准的。”

这是事实。数千个已发布的站点中，没有一个是用正确的、结构化的 HTML

写的。许多站点有夺目的视觉效果（如图 3.1 所示）和技巧很高的程序（如图 3.2 所示），个别站点还提供了引人注目的写得很好的内容和一些原创。但是没有一个站点是有一点正确结构标记、简洁的 CSS 或者以标准为基础的脚本的痕迹。

图 3.1

Team Rahal 公司站点 (www.rahal.com) 是一个每年一度的第 8 届交流艺术节比赛中获奖的作品，它非常漂亮，给人的印象非常深刻，但是不符合标准。



图 3.2

World Resources 协会网站 (<http://earthtrends.wri.org/>) 是另一个依靠视觉冲击获得胜利的作品，有灵活的程序，但一样，仅仅用了一点点的 Web 标准。



在提交的作品中，有超过一半的采用 Flash 开发。大多数作品在 4.0 浏览器中不能工作，或者只能在 IE4 或者 Netscape 4 下工作。还有一些只能工作在 Windows 平台上。数百的参赛者，大部分都有华丽（而昂贵）的作品，每个人都有自己的方法表现行业最专业的效果，但是没有一个人会稍微使用 Web 标准。

3.1.1 共同的目标，共同的手段

网站用缤纷的艺术创造来传达不同的信息，主要是为了一个市场目标——不论是你自己的还是我的站点都一样。我们都希望我们的站点能够引起站点访问者的注意力，并鼓励他们积极参与。网站要易于理解和使用。页面内容要展示我们团体、产品或者服务中所有的好处，不仅仅用单词，还要包括界面和操作。

大多数人都希望在预算内获得最好的价值，我们希望网站尽可能在更多环境下为更多的人服务。我们希望网站能够一直活跃在不断变化的技术前沿，避免陷入浏览器和平台的兼容问题中。

我们大多数人都希望网站在将来也能很好地工作，不需要连续的技术升级，就像第2章“根据标准设计和制作”中描述的一样。我们宁可在内容更新和服务的增加上花费有限的时间，也不愿意每次都为了一个新浏览器或者新设备重新写代码。

标准是达到目标的钥匙，但为什么他们还不用标准来开发呢？

3.1.2 理解 vs 真实

出于某种原因，对于可访问性，许多设计师持有一个错误的观点，他们认为Web标准与好的图形界面设计需求是相互矛盾的。另一方面，那些建立标准的人却不在商业应用中推广它们。视觉和结构呆板的W3C或者ECMA站点（如图3.3所示），没有一点点的设计灵感和客户向导的设计。非专业外观的W3C或者ECMA只做了一点事，来与“标准和视觉设计是对立的”这一荒谬观点做斗争。只有使用标准设计出漂亮站点（如图3.4所示），才能转变那个错误理解。

那么，那些花时间学习私有脚本编制和书写的多样性的设计师、开发者及作者也许有理由去学习新东西——也许他们忙于学习JSP、ASP或者.NET，正在考虑改变他们的前台技术。那些依赖WYSIWYG（所见即所得）编辑器的人们有另外一个不使用标准的理由，就是他们还没有察觉到现在的主流WYSIWYG（所见即所得）编辑器已经支持标准。许多高技巧的开发者使用类似Dreamweaver和GoLive的WYSIWYG工具。当然，如果没有工具，许多半专业的设计师将无法建立一个基本的网站。

最近有一个相关的消息：主流的浏览器已经同意符合标准。许多Web专家还没有用起来，因为他们还没有注意到浏览器已经改变了。让我们先从这个理由开始调查。

图 3.3

ECMA（欧洲计算机制造商协会）是一个好的标准团体，也是许多伟大的想法之归宿。不幸的是，这些伟大的想法却没有使用客户友好的字体、图形设计或者站点结构。因此，ECMA 站点 (www.ecma.ch) 不太可能激发设计师去学习 ECMAScript 或者其他标准（与图 3.1 和 3.2 形成鲜明对比。你将在本章后面读到更多关于 ECMA 的内容）。

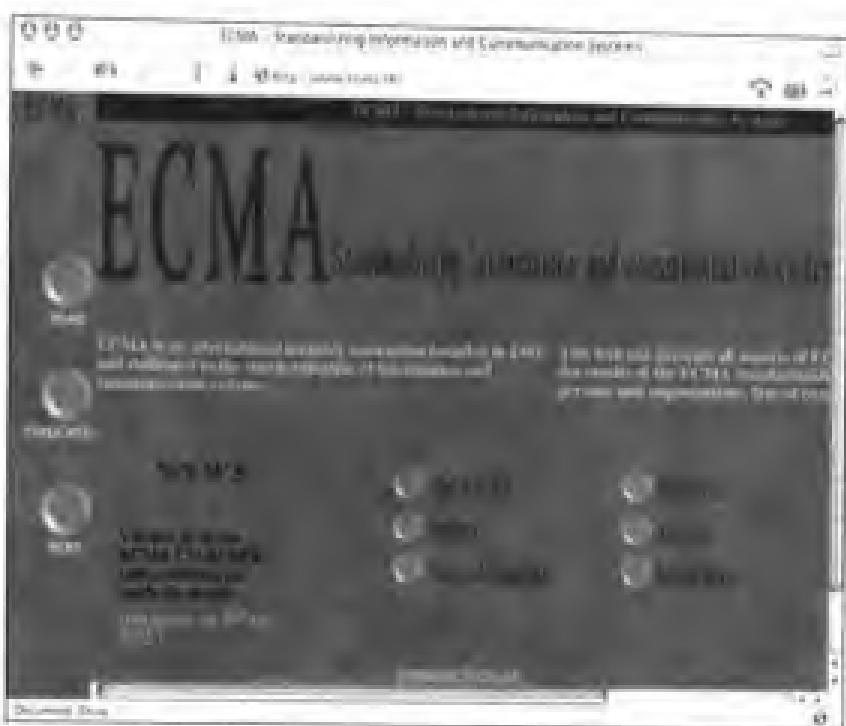


图 3.4

Kaliber 10000 (K10k) 是一个学习 Web 设计的热门门户网站 (www.k10k.net/)，而且是构建在 XHTML、CSS 和 W3C 文档对象模型 (DOM) 基础上的。虽然不是第一个包含这些标准的站点，但是 K10k 站点的标准使用是非常重要的，因为它展示了设计团体：标准也能支持好的图形设计。



3.2 2000 年：浏览器之年

随着 2000 年 3 月 IE5 Macintosh 版本的发布，世界（至少使用 Mac 的用户）终于得到更多的 Web 标准体验。IE5/Mac 支持 XHTML、ECMAScript、几乎所有的 CSS1 规范，大部分的 CSS2 规范，以及大部分的 DOM。IE5/Mac 同时能显示未处理的 XML，尽管它不清楚谁要那么做（若要了解更多请查看第 5 章，或访问 http://bugzilla.mozilla.org/show_bug.cgi?id=64945）。

3.2.1 IE5/Mac：转换和放大

IE5/Mac 与标准相结合得如此紧密，以至于在页面顶部写一个<!DOCTYPE>标签来控制显示和行为。这一技术称为 DOCTYPE 转换将在第 11 章“DOCTYPE 转换和标准模式”中论述。若简单地放置正确的 DOCTYPE，一个页面将可以用 Web 标准来显示和执行。用一个老的或者局部的 DOCTYPE（或者根本不放），页面将按原有的模式显示和执行，以便能够访问 Web 上 99.9% 的非标准兼容的商业站点，至少目前是这样（如图 3.5 所示）。



图 3.5

Hello, world. 这是 IE5 Macintosh 版本，第一个支持大部分 Web 标准的浏览器。一个开创自己道路与其他产品竞争的产品，一些创新最后被用于 IE Windows 版本，不幸的是，没有全部采用。

IE5/Mac 同样包含一个名为“文本放大镜”（如图 3.6 所示）的功能，能让用户自己放大或缩小页面文本，包括用 CSS 设置大小的文本，用以解决长期存在的可访问性问题。Prior 到 IE5/Mac，只有 Opera 浏览器允许用户缩小或放大所有的页面文本，包括用像素来定义大小的文本。Opera 用“zooming”命令操作整个页面、图片和其他各项。在设计师设计和用户使用需要之间偶尔有冲突产生，这是针对此冲突的一个创新解决方法。如图 3.7、图 3.8、图 3.9 所示。

3.2.2 Netscape 的大胆转移

IE5/Mac 的发布引发了大量浏览器支持标准，于是 Netscape 6 和它的开放源码版本 Mozilla 也都开始在所有平台上支持 XML、XHTML、CSS、ECMAScript 和 DOM，并同样提供 DOCTYPE 开关和文本放大镜功能。它已从一个基本浏览器成长为一个完全适应标准的浏览器。

图 3.6

使用命令或者单击下拉菜单执行 IE5/Mac 的文本放大镜功能。不论文本是否被设置了大小，用户都能够放大或者缩小页面上的文本，而图片不受影响。只有文字尺寸被改变。很快文本放大镜被应用在 Netscape、Mozilla、Chimera 和其他领导标准的浏览器中。哎——使人恼火的是，在 IE/Mac 文本放大镜功能推出 3 年后，IE 的 Windows 版本依然不支持这个实用性很强的功能。



图 3.7

PixelSurgeon (www.pixelsurgeon.com/news/) 是一个领导设计先锋的门户，是 KIOX 的姐妹站点。这是在 Opera 7 浏览器中的实际尺寸的衡量效果。



图 3.8

同样的站点用 Opera 的页面放大功能放大到 200%。该功能提供了阅读小图片按钮和字体缩小的文本图片的方法，解决了可访问性问题。





图 3.9

还是同样的站点，这次用 Opera 的页面放大功能放大到 500%，对于设计师，这个功能提供了一种更方便的学习热点设计元素的方法，不再需要将屏幕截图到 Photoshop 中。

为了完全符合标准，在 WaSP 的促进下，Netscape 大胆地抛弃已经存在的 Navigator/ Communicator 4.0 浏览器和每一个遗留的过时代码，从零开始，放弃升级，重新开发一个新的浏览器。这些年 Mozilla/Netscape 6 的开发使 Netscape 损失了相当大的市场——开始于 1998 年，直到 2000 年后期一直没有推出商业化产品（直到 2002 年还没有推出一个可行的商业版本）。

那些 Netscape 的管理者和工程师并没有疯狂，他们显然像 WaSP 一样相信标准。新浏览器原计划大约在一年完成，可是一年过去了，两年甚至三年也过去了，管理者和工程师仍然英勇地坚持着，决心要坚持到底。

许多公司并不在乎，干脆放弃了这个项目。他们宁愿在遗留的代码上发布一个非标准的 5.0 浏览器，也不愿意牺牲额外的时间和与一个厉害的竞争者分享市场，就像 Microsoft。虽然股东可能不同意，但 Netscape 的管理者和工程师依然值得感谢和信任，为了协同工作和 Web 未来的健康发展，他们放弃了短期的利益和他们自己个人的利益。

3.2.3 放开的水闸

接着 Opera 6 发布了，没有 DOCTYPE 转换，也没有 DOM，但是 Opera 能很好地支持大部分标准。Opera 避开 DOCTYPE 转换是因为不同于众多商业浏览器，Opera 总是按照 W3C 规范寻找和显示页面。因此，Opera 的制造者发现没有理由提供一个向前兼容的“怪癖（quirks）”模式（Opera 5 和 6 不支持标准的 W3C DOM，但是 2002 年发布的 Opera 7 支持）。

最终 Microsoft 发布的 IE6 Windows 版本，主要为了能够追上它的 Macintosh 产品正确的 CSS 表现的。IE6 对 XML、ECMAScript 和 DOM 提供了强大的支持，并且加入 IE5/Mac、Mozilla 和 Netscape 6+ 提供 DOCTYPE 转换的行列（在浏览

器的发布中，Windows 的商业联盟最终被迫在 IE6/Windows 使用 DOCTYPE 转换)。

IE6/Windows 依然不支持文本放大功能。不过，虽然文本放大功能从可访问性和友好性的角度看是非常有价值的，但这只是一个创新，而不是标准。IE6/Windows 中依然存在一个使用浮动属性破坏 CSS 布局的 bug(看第 16 章“CSS 重新设计”)，尽管如此，IE6/Windows 还是一个高度符合标准，制作精良的浏览器，比前一版本 IE5.x/Windows 的工作性能更强。

没有浏览器是完美的（也没有软件是完美的），但是每种浏览器在使用标准和可访问性上都各有所长。没有人，至少 Web 标准组织的所有人都相信，这些公司将走得更远，最终促使浏览器完美地符合标准。设计师和开发者将免费使用 CSS 布局和其他标准为基础的技术。

2000–2001：标准先锋

一些设计师和开发者的先行者，在 IE6/Windows 发布之前就开始使用 CSS 布局和其他标准（包括基于 DOM 的脚本）为基础的技术。有些人通过创建不同环境实现标准，他们强迫 IE5/Windows 精确解释 CSS，并在 4.0 浏览器中关闭 CSS，是因为老浏览器不能正确地解释 CSS。另外一些人使用分层的方法，给所有浏览器一些可看的内容，并为新的更符合标准的浏览器预备更“完整”的设计。这些经验丰富的设计先锋将在第 4 章中讨论，整个第 2 部分“设计和构建”将会描述它的工作区（workaround）和策略。

3.3 太少，太迟

主流的、稳定的、支持标准的浏览器的发布，对 Web 用户和开发者都是一个巨大的好消息。但是在知道令人兴奋的消息以前，许多设计师和开发者认为 Web 标准只是一个梦想，许多人即使曾经尝试正确地实现它们也已经停止。这并不难解释，理解需要多年才能形成。

3.1.1 CSS：第一次尝试

CSS1 规范大约在 1996 年圣诞节就已经发布了，几个月以后，IE3 发布，在其功能中包含基本的 CSS 支持。在当时是 Netscape Navigator 统治 Web 的时期，支持 CSS(Netscape 3 完全不支持)给 Microsoft 的浏览器开始带来一点信任。IE3 支持的 CSS 足够让你扔掉非标准的标签，并开始试验使用空白、标题和

其他初级的 CSS 布局。在 Microsoft 演示页面（如图 3.10 所示）看到的新浏览器性能令许多设计师兴奋不已，他们进行了第一次 CSS 设计尝试，然后立刻到处炫耀。



图 3.10

这是 1998 年 Microsoft 的 CSS 展示集中的一页。[\(http://www.microsoft.com/Ltypography/css/gallery/\)](http://www.microsoft.com/Ltypography/css/gallery/)。重叠的字体和所有其他设计效果都是完全用 CSS 实现，没有 GIF 图片，也没有 JPEGs 图片。IE3 能够显示这些效果。Netscape 3（当时浏览器市场领导者）不能显示。虽然这些 CSS 使用了不正确的属性，需要依靠 IE3 有缺陷的 CSS 引擎，它一点都不符合标准。但是魔鬼从瓶子中被释放出来，只看 CSS 可以做什么，许多人根本就不管将来会怎样。

IE3 支持 CSS 是大胆的第一步，但是像所有的第一步，它是不完善的，带有许多 bug。那些第一次使用 CSS 的作者，在欢呼它提供的创造性自由后，又很快地陷入 IE 早期页面不能用的 bug 中。例如：在某些环境下，CSS 驱动的页面中的图片会跑到文本顶部，而不是文本旁边。这就像你把手放在本段落上，试着阅读手覆盖着的内容，是不可能的。

早期的 CSS 在 IE3 中表现的 bug 是：每一个图片和段落都要放到它自己的表格单元中，这样会加倍页面容量，也不符合使用 CSS 的目的（不再用表格控制布局，不再超出带宽）。设计师们很快断定 CSS 并不适合 Web 的黄金时代——在不支持 CSS 的 Netscape 3 占据市场领导地位的时代，这是一个看起来很合理的断定。

3.4 糟糕的浏览器养成坏习惯

当 4.0 浏览器出现后，虽然依旧 bug 多多且不完善，但 IE4 比 IE3 大大改良了对 CSS 的支持。Netscape 4 在最后一刻也决定首次支持 CSS。

坦率地说，Netscape 4 对 CSS 的支持远远好于 IE3 (<http://www.webreview.com/style/css1/leaderboard.shtml>)，但是当今几乎没有

人使用 IE3，却有好几千万人仍然使用 Netscape 4。这样，许多站点拥有者开始被迫支持 Netscape 4——这是一个错误的支持。总的来说，这是一件好事情，设计和交互保持了一致性，这也是一件坏事，因为束缚了开发者的手脚，强迫他们写糟糕的代码和无用的标记。

3.4.1 默认样式的缺点

Netscape 4 在支持 CSS 上有很多失败，主要的失败在于默认样式和缺乏继承。

CSS 的设计目的是从结构中抽象出表现，并没有假定它要支持和显示多少元素，也没有假定哪种置标语言。而浏览器却经常做这些假定（一些现代浏览器使用 CSS 强制假定一些默认样式并允许设计师的样式表覆盖它们）。在大多数浏览器的默认状态下，在没有 CSS 的时候，`<h1>` 标题将是一个大而粗的字体，上下都保留一定空白。

CSS 能让你改变哪些呢？使用 CSS，`<h1>` 可以是小的，斜体的，空白可自由留出，就像设计的那样。哎，在 Netscape 4 中就不行。默认情况下，它在设计师指定的 CSS 规则中加入默认规定的表现。如果你在 CSS 中设定大标题下面没有空白，Netscape 4 还是会抢先在标题下方插入一个空白。

当设计师在标准的 HTML 标记上应用 CSS 时，他们很快发现 IE4 大体能表现出他们想要的，而 Netscape 4 则总是弄乱他们的布局。

一些设计师放弃了 CSS，另外一些设计师从一开始就工作在不断的问题中（很悲哀，包括我），避开结构标记，采用像`<div class="headline1">`语句代替`<h1>`。这样做解决了显示问题却损失了文档的结构和语义。因此，将短期利益放在长期生存之前带来了无数的问题，现在我们终于尝到了自己种下的苦果。

自从放弃破坏文档结构的习惯，到写这本书已经很长时间了，但是大多数的设计师和开发者依然用向前兼容 Netscape 4 的借口写垃圾标记，这是标准化实践过程中的遗憾。在努力使数据驱动的工作流规范化与合理化的过程中，的确会产生可访问性的问题。

那些在 4.0 浏览器时代开发的系统，如内容管理系统、发布工具和可视化 Web 编辑器（又称 WYSIWYG 编辑器）等，产生大量的无意义的垃圾标记，这给引导网站采用标准开发或者将原有内容转换到 XML 驱动的数据库，带来巨大的困难和高额的费用。一个大型站点由多个设计师和开发者共同创建维护，每个设计师可能使用不同的非标准的标签，这使得集中所有数据，按照更有用的方案重新格式化变得不可能（想像一下，一个已经建立索引的公共图书馆，不是用杜

威图书分类法，而是按照奇怪的 Joe、Mary 和各种各样其他的分类，每个分类用他们自己制定的规则编辑整理，会乱成什么样子）。

在图形浏览器以外的领域，使用无意义的非结构标记也会使页面变得难以访问。对于一个 Palm Pilot、Web 智能电话或者屏幕阅读器的用户来说，`<div class="headline1">`只是一个文本，而不是一个标题。因此，我们必须购买或开发内容管理系统，将这批非标准标签转换为标准标签。否则我们只能强迫 Palm Pilot、Web 电话和屏幕阅读器用户浏览在结构上无意义的代码标记，并猜测我们的含义。

我们感谢 Netscape 4（以及我们适应它的缺点的积极主动性）让我们陷入这场混乱。所以我们也不会奇怪，那些 Netscape 和 Mozilla 工程师需要在 Mozilla 项目中花费长达 4 年的时间，他们根本无法借鉴原有产品的代码。

3.4.2 缺乏继承

Netscape 4 在理解和支持继承上也很失败。这个潜在的闪耀的继承概念给了 CSS 强大的力量，CSS 通过授予的通用规则而格式化所有子标签，除非设计师特别指定某个子标签的样式规则，这样做的好处是可以减少带宽。

例如，用 CSS 你可以在 `body` 标签中定义一个字体、尺寸和颜色，同样的字体、尺寸和颜色将作用在 `body` 标签的所有子标签中，从`<h1>`到`<p>`以及更多。但是在 Netscape 4 中却不是这样，在 Netscape 4 中是 $2+2$ 还是等于 $2+2$ ，而不是等于 4 。

有一定经验的开发者在浏览器缺乏支持继承下徘徊时，就会写出多余的规则：

```
body, td, h1, p {font-family: verdana, arial, helvetica,  
sans-serif;}
```

在上面的例子中，`td`、`h1` 和 `p` 选择器是多余的，因为在任何符合标准的浏览器中，子标签的样式将自动继承父标签的样式。

经验不足的开发者会把规则写全，这样的方法是冗余和浪费带宽的：

```
body {font-family: verdana, arial, helvetica, sans-serif;}  
td {font-family: verdana, arial, helvetica sans-serif;}  
h1 {font-family: verdana, arial, helvetica sans-serif;}  
p {font-family: verdana, arial, helvetica sans-serif;}
```

这样的方法，它虽然浪费了用户和服务器的带宽，但也能使工作完成。另外的一些开发者则断定 CSS 不能工作在 Netscape 4 中，或者 CSS 有缺陷（他们的

想法是错误的，但是有这样理解的人却很多）。

Netscape 4 还有其他 CSS 的缺点——足够填满一个大城市的黄页了。但是，它已经足以扰乱局面，也是以妨碍 CSS 标准的广泛采用。

3.4.3 混乱的交互行为

伴随着 CSS 技术实现的混乱，早期的浏览器也没有通过一个通用的脚本方法来促进交互行为的成熟。每一个支持脚本的浏览器都有一个对象模型，它可以规定哪种浏览器行为能够应用在页面上。Netscape 4 有一个私有的 `document.layers` 模型，IE4 也使用自己私有的 `document.all` 模型。这两个浏览器都不支持 W3C DOM。那些想在站点中显示交互行为的开发者不得不写两种代码适应不同的浏览器，支持早期的浏览器（向前兼容）需要更多的代码和条件跳转，就像第 2 章描述的那样。

早期的浏览器甚至不能运行一个普通的脚本语言。在早期，Netscape 发明了 JavaScript，承诺将其开放为标准，这样其他浏览器开发商就能支持它。但是几年后，尽管他们答应开放，Netscape 依然掌握着 JavaScript 的秘密，并把它看成是一个竞争优势（如果 Navigator 是支持 JavaScript 的唯一浏览器，为什么别人要在支持 JavaScript 稍差的浏览器上开发呢。所以 Netscape 有充分的理由这样做。如果处于相同情况，Microsoft 也可能这样做。事实上，Microsoft 已经采用私有的 ActiveX 技术这么做了）。

为了竞争，Microsoft 对 JavaScript 进行了反向工程，按照他们的方法做一些改变，这在反向工程中是常见的。结果就是这种语言工作像 JavaScript，但又不完全像 JavaScript。这两种语言的差别足够把你搞糊涂。Microsoft 称他们的脚本语言为 JScript。其间，Microsoft 提供了一个独立的 ActiveX 技术，设想是能在所有版本的 IE 浏览器中提供无缝的交互功能，但实际上只能正确工作在 Windows 平台上。

JScript，JavaScript，ActiveX：在跨浏览器和向前兼容的技术气氛下，开发者发现他们自己必须与多个技术“跳舞”，但是他们的节奏都不一样——客户不得不承担不断提高开发和测试费用。

3.4.4 最终的标准化脚本语言

最后，ECMA 批准了一个 JavaScript 的标准版本，他们命名为 ECMAScript (www.ecma.ch/ecma1/STAND/ECMA-262.HTM)。W3C 也及时出版了标准的 DOM 规范。终于，Netscape 和 Microsoft 都支持这两个规范。但是几年前两

者仇敌般的对立已经使得许多开发者在私有的、不兼容的脚本技术和对象模型方面成为专家，也使许多站点拥有者以为 Web 发展总是诸侯纷争的，因此，还会有只支持 IE 的站点、残缺的检测脚本，另外一些案例中则放弃 Web 标准而采用像 Flash 的专利技术方案。

顺便说一句，如果你想了解 ECMA 代表什么，不要试图去访问这个令人讨厌、混乱的官方网站。ECMA 是 European Computer Manufacturers Association（欧洲电脑制造商联合会）的缩写，它是个好的标准的制定组织，而不像 W3C 只把标准称为“推荐”。糊涂的网站和使人困惑的标签是另一个标准在现代 Web 广泛普及困难的原因之一，我们将在下一节详细论述。

3.5 混乱的网站和令人困惑的标签

CSS2 规范是 W3C 献给全世界的礼物，如图 3.11 所示，是一个强大的标准表现语言，可以让设计师更容易达到创作目标。但是你光看 W3C 的相关页面，根本就看不出这些。那些页面就像我叔叔用 Microsoft FrontPage 和价值 50 美元的图片编辑器在一个下午就做好的个人网站一样。

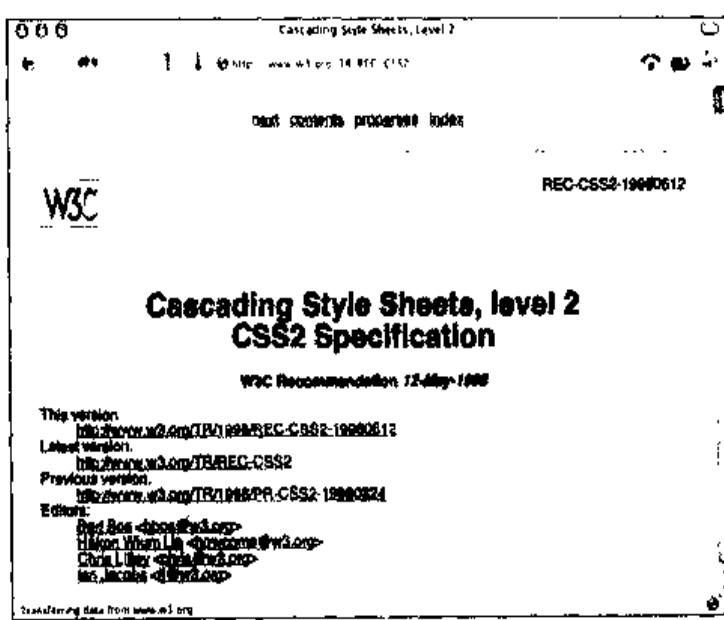


图 3.11
由 W3C 制定的 CSS2 规范 (www.w3.org/TR/REC-CSS2/)：一个令人鼓舞的表现语言

克服恐惧和痛苦感觉和乏味的外观，你努力阅读和领会 W3C 的说明，毕竟，W3C 是由科学家组成的，而不是图形设计师，所有的关键实质在于那些文字，对吧？阅读 20 分钟以后，你就觉得看得太吃力而眼睛流泪。失望之余，你转而去了在线电脑商店购买 Macromedia Flash。

公平地说，W3C 不是一个商业化的图形设计、可用性或者信息结构顾问，也不是商业化的、专门针对设计师撰写教材的公司。W3C 站点是由思想领先者和技术专家用连续精确的技术文档建立起来的网站。它假定的服务对象也是这类人员。

在“如何阅读 W3C 规范”(www.alistapart.com/stories/readspec) 中，O'Reilly 的作者和 WaSP 领导委员会成员（其中一位是本书的技术编辑）J. David Eisenberg 说明他的方法：“当你寻找答案时，你会查看用户手册或者用户参考指南。你想使用技术，那不是 W3C 规范的目的，spec 的目的是告诉实施技术的程序员，它必须有的特点和如何实现。这是与汽车使用和修理手册不同的地方。”

根据组织使命，W3C 交流的对象是工程师而不是大众。它并不想解释或者出售建立的标准。要注意的是，在早期他们甚至不将其称之为“标准”，尽管它们的確是（事实上，在近来的处理原料和网站的部分内容中，W3C 已经开始使用标准的“s”字母来代替被动的“推荐”。这是一件好事）。

与商业公司不同的是，当你使用 Web 标准，W3C 不要求你支付标准、专利和版税的费用。这使得它的成员公司有点气馁。

3.5.1 理论 vs 经济

从“狗咬狗”和“狗买其他狗的公司，目的只为了将它们赶出商业竞争”的世界分开，W3C 居住在一个冥想的空间，他关注 Web 潜力，而不是竞争压力。W3C 的行为有点发烧，因为发烧，为了发烧，其站点反映了团体在科学上的目标，超越了样式和对消费者友好的意向。

但矛盾是：设计师、开发者和站点所有者很关心样式，关心更多的是在用户友好的可访问性。在他们自己的站点上，从不故意发布使读者迷惑的、理解困难的、神秘的文本，从不任性地将他们最重要的内容放在偏僻的角落，从不故意把内容发布在一个缺乏美感的足以让访问者立刻单击离开的页面中。

从心理学角度讲，那些关心品牌、审美、清晰、易用的人，以及花费时间努力把属性用在他们自己网站项目的人不太可能相信：一个充满神秘技术文档却只有业余外观的网站，掌握着通往未来的钥匙。那么怎样做才能让那些人们相信？他们只相信商业巨头漂亮的陈述。

3.5.2 软件会帮你做

在西方，当我们面对一个商业或创新问题时，只要打开一个应用软件来处理

就可以了，或者期望处于市场领导地位的公司发布我们需要的产品。网站所有者用 Microsoft Excel 格式的电子表格检查他们商业的健康状态，设计师用 Adobe Illustrator 设计 logo，用 Macromedia Flash 设计动画，他们在 Adobe Photoshop 中准备图片和 Web 布局。对应每一个问题，都有一种软件，对应每一种软件，都有一个领导者。

尽管提倡 Web 设计师和开发者学习用 Notepad 和简单文本创建页面，但后来更多人依赖可视化编辑器（WYSIWYG）进行创作。当私有脚本语言和不兼容的对象模型使得开发越来越复杂的时候，就出现类似 Macromedia Dreamweaver 和 Adobe GoLive 的产品帮助人们建立网站，工具隐藏了潜在的复杂性。怎样才能支持多浏览器？单击按钮，程序会自动帮你做。

这些行业主流的视觉编辑器是复杂和强大的，但是直到最近，它们创建的代码和标记只应用了少许的 Web 标准。像开发者一样，Dreamweaver 和 GoLive 产生了针对特定浏览器的脚本和无语义的标记结构。

Dreamweaver MX 和 GoLive 6 已经较以前版本大大符合标准（看 4.5 节“Web 标准和创作工具”），尽管它们仍旧需要开发者学习，但它们的用户都在学习什么知识呢？他们转到吸引人的、良好写作的、提供在线用户手册服务的、以及培养 Dreamweaver 和 GoLive 团体的发展的产品站点去了。我们稍后将讨论这样的站点。

3.5.3 了解产品 vs 了解标准

当众多公司努力交付“一键到位”解决方案用来解决前端设计问题的期间，后端也发生同样的问题。当类似 XML 和 DOM 的标准仍然在标准委员会锤炼的过程中，私有的发布工具和来自大品牌公司（类似 IBM, SUN, Lotus 和 Microsoft）的数据库产品和中坚公司（类似 Allaire，现在是 Macromedia 的一部分）提供了必需的功能。

你不需要手工制造一辆汽车，那为什么你要手工写代码建造网站呢？立刻注册，立刻购买，如果有什么错误，我们会在下一个升级版本中修改它。对进度急迫的开发者来说，这个定位是有道理的，同样的理由，曾经将 HTML 视为一个设计工具也是有道理的：也许正在会见的客户是正好需要它的。

这样一来，“宁肯知道产品也比知道标准好”的思想控制了许多 Web 开发者，特别是许多 Web 设计师。所以如果说有 1 个设计师或开发者通过 W3C 了解规范，就有 10 个设计师和开发者是通过 Netscape、Microsoft、Macromedia（如图 3.12 所示）、Adobe（如图 3.13 所示）和其他主流公司的网站获得这些信息。

图 3.12

像 W3C 一样, Macromedia 为设计师和开发者提供了无价的工具 (www.macromedia.com)。与 W3C 不同的是, Macromedia 出售他们的工具, 并努力工作以培养与设计团体的密切合作, 并在自己的网站上用设计师团体的语言发表



图 3.13

与竞争对手 Macromedia 一样, Adobe (www.adobe.com) 不断地与它的用户保持沟通, 从而发现关于他们最想要和最希望产品改变的地方。与竞争对手一样, Adobe 也努力在他们的网站使用“设计师的语言”, 形成鲜明对比的是图 3.3 和 3.11。



与 w3.org 不同的是, 为了加深公司和用户之间的交流, 增强公司的品牌形象, 这些站点是为服务用户(专业的设计师和开发者)而建立的, 因此, 这些站点往往很好地(至少是充分的)设计自己的品牌规范, 他们编写的教程对于专业观众是非常容易理解的。

不用说, 这些教程强调的是产品的功效。当这些公司提及 Web 标准, 他们很可能一带而过, 或者就是标榜他们的产品比其他公司的更符合标准。毕竟, 这类站点的目的是帮你确信要买产品的价值, 并且明年继续购买和升级产品。

简而言之, 许多 Web 专业人士——设计师和开发者, 还有他们的客户和老板——对私有解决方案了解相当多, 但几乎不了解 Web 标准。他们也无法了解

更多，因为他们总是跟着单一的公司或产品线，将自己锁定在必要的升级、个性化定制、培训、顾问等费用永远增长的怪圈中。

有个产品需要特别提起，下面我们就介绍它。

3.6 字母 F

在市场上所有成熟的、有竞争力的专利技术解决方案中，没有一个像 Macromedia Flash 那样成功和辉煌（如图 3.14 所示）。这个产品起初只是一个微不足道的名为 FutureSplash 的插件，设计师用于在他们的页面里嵌入矢量图形和动画。



图 3.14

欢迎使用 Macromedia Flash! Flash 的创作环境可能是丰富的、深入的和复杂的。但是 Macromedia 尽一切可能指导设计师和开发者用这个助手来开始攀登产品的峭壁和学习的曲线。

FutureSplash 刚出现的时候，设计师只是稍稍注意一下，但聪明的 Macromedia 公司立刻发现了它的潜在价值。Macromedia 买下了插件和它相关的创作工具，重新命名为 Flash，并且开发成为一个丰富灵活的创作工具，用一个强大的类似 JavaScript 的程序语言 ActionScript 来驱动。

Macromedia 同样也维持和鼓励一批 Flash 的爱好者。

3.6.1 Flash 的价值

当 4.0 浏览器不兼容的脚本语言与对象模型严重不兼容及抬高成本时，Flash 4 和它强大的脚本语言可以平等地很好地运行在 Navigator, IE 和 Opera 中，并且可以像在 Windows 中工作一样很好地在 Mac OS, Linux 和 UNIX 上显示。因此，许多设计师向 HTML、修补 4.0 浏览器的 CSS，以及老鼠窝一样混乱的不兼容

容的代码说再见，开始学习 Flash。

旋转的标志，冗长乏味的“loading”屏幕，和无止境的、不必要的“介绍”，最初 Flash 给用户留下的印象并不好。少数有眼光和创新意识的代理商急切地希望 Flash 流行，但是用 Flash 制作的有魅力的站点却很少，就好比你不能因为木工不好而责备锤子和钉子。但在设计高手设计出类似 OneNine（如图 3.15 所示），Juxt Interactive（如图 3.16 所示）和其他高级购物的网站后，Flash 的威力开始逐渐体现出来。Flash 吞食多媒体应用空间，就像 Microsoft 的浏览器在吃掉 Netscape 的午餐一样。

图 3.15

将一个强大的工具放在一个极赋天才的视觉设计师手里，你会得到什么？当然是丰富的用户体验。OneNine（www.one9ine.com）是纽约的一个商店，它的高端设计几乎完全采用 Flash 实现。



图 3.16

位于加利福尼亚州的 Juxt Interactive（www.juxtinteractive.com）是另外一个重要站点，它极富巧妙的用户体验设计，推翻了 Flash 只是一个暗藏机关的、创建令人讨厌的“介绍”工具的观点。



尽管 Flash 适合许多项目，如果任何地方都使用 Flash，也会发生很多错误。Flash 4 糟糕的可访问性和可访问性问题被开发者和客户注意到。对此产品批评声音最响亮的是批评家 Jakob Nielsen (<http://www.useit.com>)，他来自尼尔森·诺曼底咨询组织。

在 2002 年，Macromedia 忙于在它的升级产品 Flash MX 中大幅度提高可访问性和可用性问题，并聘请 Nielsen 为顾问。Nielsen 因此改变了他的论调（如果 Microsoft 和 Netscape 也这样聪明地聘请苛刻的批评家，本书的作者将大声笑着在私人海滩度假，而不用辛苦地在这里写书——我跑题了）。

依靠其才能，Flash 促进了多媒体交互体验。而使用标准标记，CSS、SVG (Scalable Vector Graphics) 和 DOM 很难效仿它。

SVG 对于你和我是……

SVG 是一个 XML 应用，标准的矢量语言，具有动画和脚本能力，完全符合 Web 标准。但是在写这本书的时候，还没有流行的浏览器“天生”支持 SVG，使用它需要安装一个像 Flash 一样的插件（W3C Amaya 浏览器支持有限宽度的 SVG，一些 Mozilla 能够局部地支持 SVG，但是这些环境还远未到主流）。

通常，如果你想建立复杂的类似应用程序的界面，用 Flash 很容易实现，因为它有巨大的安装基础和单一的开发环境。有一天，它也许会提供更多功能，建立一个使用 XML、XHTML、CSS、ECMAScript、SVG 和 DOM 的标准联合体的应用。

3.6.2 使用 Flash 的问题

使用 Flash 的主要问题是它不适用于大量内容站点和商业站点。然而开发者还是在不适当的情况下使用它，因为 Flash 的界面表现好像比较时髦，并且使客户觉得超值，因为，不论成功与否，Flash 站点看起来是很漂亮的。

新的站点、门户、商店网站、制度站点、团体站点、杂志、目录和其他强调文字或者包含网络实践行为的站点仍旧采用 XHTML、CSS 和其他标准是最佳的。但是许多开发者却用 Flash 来替代，不是因为它能服务于项目目标，而是因为一开始使用 Flash，能吸引新客户。

3.6.3 另外一些使用 Flash 的问题

使用 Flash 的另外一些问题是：一些设计师如此倾心于 Flash，以至于他们

忘记了如何使用 Web 标准——如果他们曾经学习过，结果是，他们发现内嵌 flash 的页面只能在一种或者两种浏览器上显示。而实际上 Flash 文件可以在任何包含 flash 插件的浏览器上运行，但这些站点可能由于糟糕的 HTML 代码而导致大多数的用户都不能访问这些 Flash 内容。有些 Flash 站点要求使用 IE 浏览器，只不过是因为要加载一个 Flash 文件。这就好比苛求用你的手表收看有线电视。

当要开发一个新的“传统”的站点时（指不使用 Flash，而采用我们前面介绍过的会产生大量问题的老方法），这些工作常常会交给新手的团队，而资深的设计师和开发者则继续将焦点放在 Flash 项目中。

XML、XHTML、CSS 和 DOM 对新手团队和初学者不是一种没有意义的技术，而是成熟的、有强大标准能力、提供丰富的用户体验的技术。要学习 Flash，通过购买专门的、极其漂亮的、实用性强的 Flash 书籍，已经没有什么难题，但我更喜欢看到 90% 的设计师和开发者花同样的关心和注意力在 Web 标准上。我不能强迫你，你有权利选择是否购买这本书。

3.7 兼容是一个禁忌词语

其他阻碍 Web 标准被广泛接受的原因，就是错误地认为：标准不知何故会减少创造力，束缚开发者手脚；或者与守旧的、私有的方法相比，采用标准会导致用户减少体验。这种错误观念是从哪里来的？

这或许来自那些尝试在 4.0 和版本更老的浏览器中使用 Web 标准并被混乱结构惹怒的开发者，但是标准支持不完善的时代已经一去不复返。

3.7.1 语言对于形成理解的威力

短语“Web 标准”可能并不准确，Web 标准组织创造了这个短语只是为了宣传。我们曾经寻找到一组词汇（基于道德原则的）来传达给浏览器开发商面临的危险是什么，提醒浏览器开发商做到他们的承诺：对他们参与创建的技术，要保证支持。我们现在也需要一个短语，可以传达给开发者、客户和科技新闻记者这样的信息：我们需要一个共同遵守、执行，具有广泛行业基础的、可靠的技术，这是急迫的，重要的。我们认为“推荐”不足以推动 Web 标准快速发展，所以采用“标准”。

我们没有预算，也没抱太大希望，然而不知道为什么，我们成功了。今天，像 Netscape、Microsoft、Adobe 和 Macromedia 等公司正努力适应标准，并将之作为一个期待的、渴望的功能来大力宣传——就像当年的四轮驱动技术。尽管这

些公司“了解”，但许多设计团体仍不知道。一些设计规则中强加和武断的采用错误的“Web 标准”（比如一些可用性的方法），应该向设计师们解释：Web 标准至今没有对外观审美做任何指导方针和戒律。

另外，单词“兼容”可能也不够准确。设计师希望按他们自己的创作想像设计，而不是约束在一个复杂的技术标准中。其实他们应该知道：Web 标准对站点的外观和感觉没有做任何限制，它们只不过使浏览器能够正确表现设计师创建的站点，同样，帮助客户根据公司的市场需求和用户分析来制定目标。Web 标准只是保证站点为更多的人和更多的平台工作。

3.7.2 灵感问题

当遇到讨论 Web 标准或者吹嘘它们适应一种或全部 W3C 标准的站点的时候，因为视觉效果糟糕，设计师和客户可能会立刻离开。当我们在讨论可访问性时，会遇到同样的问题（一些可访问性好的站点显然很难看，但问题出在那些站点的设计师的设计能力，而不是高可访问性。对于 Web 标准也一样，看看 W3C 和 ECMA 的站点，无设计的页面绝不会激发设计师开始学习 XML 和 CSS2 的热情）。

Wthremix 竞赛协会（如图 3.17 所示）成立于 2002 年 12 月。他们的目标是寻找丢失的审美情趣，发起人对竞赛说明如下：

“W3C 创造了强大的标准和指导方针，促进 Web 发展更加合理，提高用户体验。类似 XML、CSS、XHTML、DOM 的技术和类似 Web 的指导方针能帮助我们建立更强大的、适应所有环境的站点。但是 W3C 只是执着的超级标准发烧者，他们的站点缺乏客户向导的设计和开发，缺乏撰写技术信息的专家。”

结果，W3C 强大的技术和指南堆砌在一个不引人注目的、使用不方便的、无规划的站点上。我们希望 W3C 的站点能够转变成更加好看，更加有组织，更加体现其品牌，更加容易使用和理解，因此举办这个竞赛。”

竞赛内容

Wthremix 是一个向编码者的设计挑战，也是一个向设计师的编码挑战。这是一个重新创作设计 W3C 主页的想法。设计一个直观的布局和导航，根据用户的想法组织内容，建立一个反映协会重要性和影响性的美观站点。告诉我们你认为 W3C 的主页应该是什么样子，它应该如何与用户沟通，用你的观点，用正确且少量表格的 XHTML 和 CSS，要求符合 WAI 的可访问性等级 1。

图 3.17

Wikimix 竞赛协会成立于 2002 年 12 月，挑战设计师和程序员重新设计 W3C 的网站 (<http://w3mix.webgraphics.com/>)



3.7.3 其他问题

一些对 Web 标准的不信任是因为不好的体验，在这些浏览器发布一年多后，Netscape 6 第一版中有大量的 bug，而 IE6 中仍旧有没修正的 bug，其他的已经按照 Web 标准，从 HTML 转换为 XHTML 的站点，忽然发现它们的布局在 Netscape 6+、Mozilla 和 IE6 中看起来不同。在第 11 章我们将解释为什么会这样，以及如何简单、快速地修正使站点重新正确。但是如果你不知道那些简单快速的修正方法，你也许会不再信任 Web 标准。也许你会把头埋在沙里，对好的建议充耳不闻，坚持陈旧且非标准的方法。

不要只看到黑暗面，尽管无知和偏见在 Web 设计行业中继续蔓延，但是还是会有一些人在努力，Web 标准在继续坚持——本书会在这里给你帮助。

XML 征服世界（和其他 Web 标准成功案例）

我们在前一章讨论了标准推广的困难，以及是如何在因特网中立足并且超越网络的，在进一步深入之前，这是有必要的。尽管对标准的误解妨碍了一些项目采用标准，但 Web 标准在许多方面已经获得成功，并且迅速改变着我们的科技、商业、以及网络和非网络的出版物。

在本章，我们首先了解一下自 HTML 以来最成功的 Web 标准：可扩展置标语言（XML）。XML 是一种可以包含一切的数据格式，已经被广泛地应用并且适用于复杂需求。我们会发现 XML 如何帮助软件产品屹立于快速变化的市场，如何解决当前数据驱动型的商务需求，以及如何引发新一代的可共享的 Web 应用和服务。

我们同样可以看到，Web 标准已经直接使两个主要的浏览器竞争者开始促进和鼓励，并进行协作。我们将会知道为什么一些以前曾经忽略标准的 Web 专业开发工具又重新拥护标准。还有一些有远见的设计师和开发者的个人网站会广泛接受 CSS 布局、XHTML 标记，遵循 WAI 及 US 508 条款关于可访问性的指导。

所有这些成功案例都表明一个共同的事实：因为他们的工作，标准正在获得更大的支持。这些标准越被广泛支持，他们工作得就会越来越好，并且使我们所有人走上采用标准的道路更平坦。

4.1 通用的语言（XML）

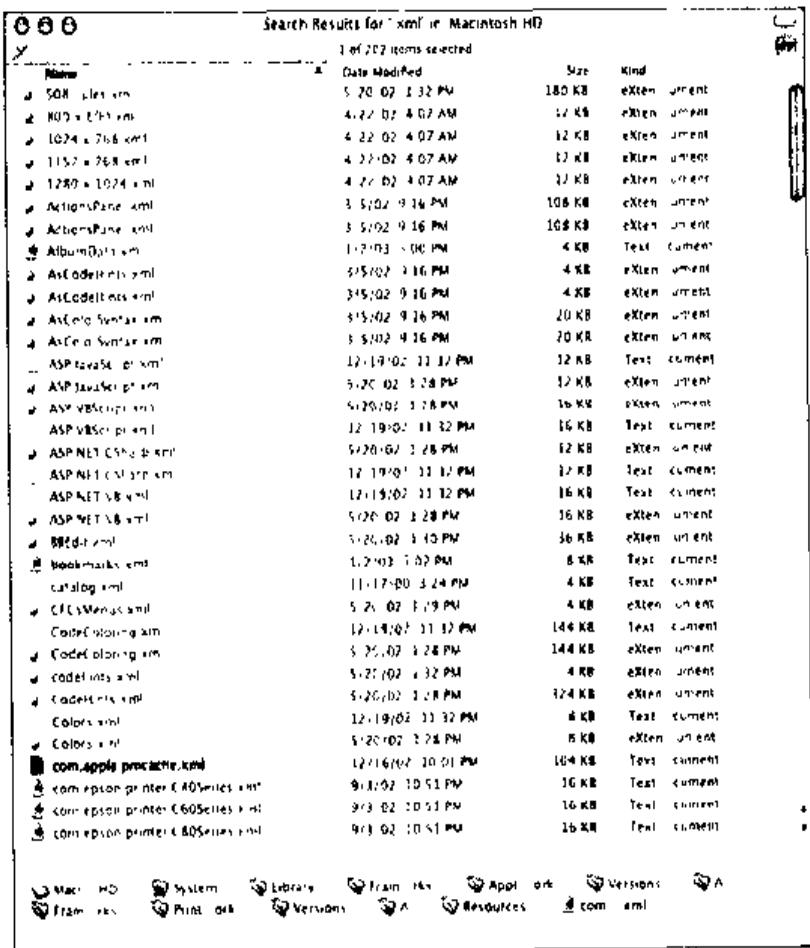
在第 3 章“推广标准的困难”中，描述了在早期浏览器环境下，说服大多数设计师和开发者遵循标准简直是白日做梦。部分人固守着达不到目的且陈旧的

方法，另一部分则放弃使用 HTML、CSS 和 JavaScript 等技术，转向使用 Flash。在那个章节中，我们知道 Web 标准化正面临着一个在接受标准和正确使用标准的日益激烈的战争。那现在我们如何对待 XML 呢？

XML (extensible Markup Language) 标准 (www.w3.org/TR/2000/REC-XML-20001006)，在 1998 年 2 月被提出，并在软件行业引起了风暴（如图 4.1 所示）。这是第一次，结构化文档和数据有了一个通用的、可适应的格式，不仅仅应用在 Web 上，也可以应用在任何地方。全世界都看着它，就像泥坑中来了一个身穿节日盛装的少年。

图 4.1

“咦？在我的电脑上有好多 XML 文件！”，在任何一台 Macintosh 计算机上进行检索，你都会发现很多 XML 文件。这些文件保存着计算机的个性配置，还有一些保存设备信息。另外一些则可能包括 Acrobat、iPhoto、iTunes、Eudora、Internet Explorer、Mozilla、Chimera 和 Flash MX 等应用程序组件的配置文件。XML 是一个 Web 标准但却远远超出了 Web 应用的范畴。



4.1.1 XML 和 HTML 的比较

虽然和优秀的 HTML 来源于同样的技术 SGML（类似 HTML，也使用标记、属性和值来格式化文档结构），XML 却完全不同于早期可代替的置标语言。

HTML 是构架 Web 页面的一种基本语言。它由一些固定数量的标记和看起来好像有些矛盾的规则组成。在 HTML 中，你可以随意使用或者不使用某些标记，组合或者不组合其他标记。对于任何人来说，利用这种松散的随意性，使创

建一个Web页面变得很容易，甚至他们不需要完全知道他们在做什么，这便是核心所在。

在早期，那时候Web只需要基本的内容并不需要更多其他信息，HTML是个很好的解决方法。但在今天，那些经历过长期成长的网站，经常需要频繁通过发布工具重新组装页面，经常需要将内容从数据库发布到Web页面或者无线设备。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文件格式。

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內容，因此限制了我们重用这些內容的能力。当然，HTML确实是适用于Web页面的。

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，你不仅要考慮把它们在Web页面中显示出来。你应当对标记进行编码，以便使得任何支持XML的环境都能理解你的XML文档。

4.1.2 一个父母，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分录的定制标记自由地创建XML标记。音乐公司也可以制作他们自己的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。

这些定制的XML语言我们称为应用，并且它们都是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。

4.1.3 专业和定制软件的基本要素

XML的格式化、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不仅可以保存在线或者公共数据库的数据內容，而且也成为了像FileMaker Pro的面向数据库处理软件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。

苹果公司的基于 UNIX 的操作系统 Macintosh OS X 采用 XML 来保存所有的配置信息。打印设计软件 Quark XPress 5.0 及 Adobe InDesign 2.0 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”（如 Macromedia 的 Dreamweaver MX 和 Adobe 的 GoLive 6），从而使得在打印的 Web 页面、Web 布局以及运行你的在线商店和全局目录数据库之间的数据转化变得更容易（至少也会变得有可能）。

有一些软件除了处理 XML 就没有其他内容，软件竟然就是这样制作出来的。Macromedia 公司的 Dreamweaver MX 就是由一些最终用户也能看得见的 XML 文件组成的（如图 4.2、图 4.3、图 4.4 所示），由此看来，通过编辑这些文件（www.alistapart.com/stories/dreamweaver/）来修改应用程序变成了可能。用这种方式定制 Dreamweaver 和出售这些定制化版本已经成为了一种职业。

图 4.2

Dreamweaver MX，一个流行的 Web 开发工具，它由一些 Web 开发者熟悉的格式的文件组成。瞪大眼睛仔细看，Dreamweaver MX 由此 XML 文件组成……

Search Results for ".xml"			
	Date Modified	Size	Kind
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
DataSource.xml	5/20/02, 1:29 PM	4 KB	exten ument
Default.xml	5/20/02, 1:29 PM	4 KB	exten ument
documentTypes.xml	5/20/02, 1:32 PM	8 KB	exten ument
Dreamweaver 3.xml	5/20/02, 1:30 PM	36 KB	exten ument
dynamicData_reJSP.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_checkbox.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_fileField.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_form.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_hiddenField.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_option.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_passwordField.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_radioButtonsTableRow.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_radioGroupTableRow.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_select.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_submitButton.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_table.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_tableRow.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_textArea.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_textAreaTableRow.xml	5/20/02, 1:29 PM	4 KB	exten ument
editOps_textField.xml	5/20/02, 1:29 PM	4 KB	exten ument
EncodingMenu.xml	5/20/02, 1:29 PM	4 KB	exten ument

Macintosh HD
Applications
Macromedia Dreamweaver MX
Configuration
Encodings
EncodingMenu.xml

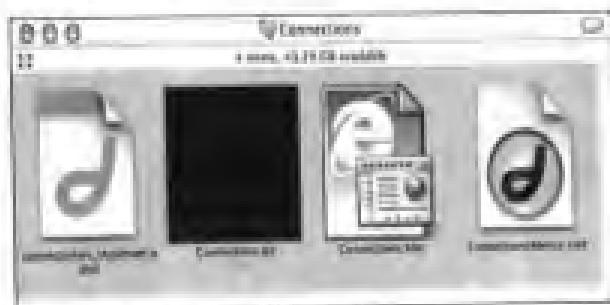


图 4.3

还有 GIF、HTML 及 JavaScript。就像一个网站，Dreamweaver 的组件被组织在多个子目录。



图 4.4

理解能力高的 Dreamweaver 用户可以对这些文件进行编辑，从而定制 Dreamweaver。下面是一个用户通过编辑一个名叫 menu.xml 的 XML 文件来改变 Dreamweaver 默认的快捷键设置。

一般通用软件也是喜欢采用 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可以读写 XML 信息，也通过第三方的产品比如 Palm Pilot 上的 *Alfred XML 解析器* (www.xml.com/pub/t/216) 来处理。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 的度假照片集，日落时刻的美丽照片很可能采用的就是 XML 格式。

甚至在一些惹人喜爱的图像处理软件也能处理 XML 格式（比如 Apple iPhoto，如图 4.5 所示）。当你打印一张小奥斯卡第一次遇见小狗的时候的照片，小奥斯卡涨红的脸颊，以及小狗明亮的眼睛表现得非常真实，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。

图 4.5

用户界面友好的 iPhoto 软件 (www.apple.com/iphoto/)，是 Apple 的 OS X 操作系统中的一个免费组件。它通过 XML 来组织图像库数据，记住打印设置及其他信息。



4.1.4 比 MTV 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。

作为一种不受到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。如果你在你的软件中集成了 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的费用。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，就越容易能在一家厂商和另外一个厂商的产品之间进行数据传输。

另外，可以想像 XML 的工作方式。过去如果你能从一个程序中导出用 tab 字符分隔的文本，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，但能电视中使用

然而 XML 并不是解决任何软件问题的万能药。比如一个 JPEG 图片的数据在二进制格式下能比 XML 文本格式得到更好的压缩比例。虽然大多数专业软件和大众软件采用 XML，并且数量还不断增长，但我们并不是说市场上任何的软件都是采用 XML 的。不管实现上是否有缺陷，XML 还是改变了软件工业。

即使是那些不支持 XML 的软件作者们也相信他们的软件最终将会支持 XML。在 2002 年 4 月，惨淡的销售和破碎的中间件市场令人悲哀，在 iTV 标准协会 (iTVA Production Standards Initiative) 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：展示和支持一个基于 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” (www.allnetdevices.com/developer/news/2002/04/09/iTV_Firms.html)。

听起来很熟悉？这个正是 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。

4.1.5 选择 XML 的五个因素

在 Web 上，工作在大型的协作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：

- 类似 ASCII 文本文件，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。
- 不像 ASCII 文本文件（或者 HTML），XML 是一种智能的、同一的格式。XML 不仅保存数据，它还可以保存数据的数据（元数据），易于检索和用于其他功能。
- XML 是一种扩展性语言，容易定制以应用于任何协会、公司或者商业及其他领域，通过派生的其他基于 XML 的语言，可以应用并处理特定任务，比如数据同步或者 Web 服务的传递。
- XML 基于如下规则：它要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，从要出版的 Web 页面到年度报告，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。没有人能怀

疑 XML 可以轻松地提高效率。

4.1.6 创造力的宝藏

下面的四个例子和课程，它们会揭示 XML 在 Web 上受欢迎的深度，以及用图例展现 XML 派生的语言和协议是如何解决那些曾使最聪明的开发人员都感到沮丧的问题的。

扩展样式语言转化语言 XSLT (www.w3.org/TR/xslt)

这种基于 XML 的标记语言可以提取和排序 XML 数据，格式化为 HTML 或者 XHTML，且可以立即给用户提供在线浏览。如果你愿意，XSLT 还可以把你的数据转化成 PDF 文件或者文本文件，或者用来产生一个用可扩展向量图像语言格式（SVG）渲染的可持续更新的图表或者类似的商业图像。XSLT 甚至可以同时处理这些事情。如果想看看一个简单的教程，可以参阅 J. David Eisenberg 的“使用 XML”教程 (www.alistapart.com/stories/usingxml/)。

资源描述框架 RDF (www.w3.org/RDF/)

这种基于 XML 的语言为应用程序在 Web 上交换数据元提供了一致性的架构。从实用角度来说，RDF 集成了库分类和目录，将新闻、软件和其他各种内容收集和集成，并方便不同的集合之间通信和共享（例如个人照片和音乐收藏）。RDF 的功能还在于它能驱动软件。如果你刚好在你的计算机上安装有 Mozilla 浏览器，打开它的文件夹进行查看。你会发现 RDF 文件（还有 CSS 文件），它们用来帮助浏览器实现它的功能。特别的是，在文件夹进行查找，每个配置文件都有自己基于 XML 的文件的设置。

丰富站点摘要 RSS (<http://backend.userland.com/rss092>)

丰富站点摘要（Rich Site Summary）是一个轻量级的 XML 框架，它用来描述一个 Web 站点，最早是由 Dan Libby 开发，用来组装 AOL/Netscape 的“我的 Netscape”门户网站的。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。现在，RSS 已经被成千上万的网站所使用，使它成为了在 Web 上最被广泛接受的 XML 格式（如图 4.6 所示）。

XML-RPC (www.xmlrpc.com)

另外一个 UserLand 软件公司的创新就是 XML-RPC，它是一个规范，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 Internet 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。



图 4.6

在 Splorp.com 上的网志日志和个人期刊，提供了一个 XML RSS，从而能够使这个站点的内容非常容易地被其他站点同步（www.splorp.com）。

4.1.7 Web 发布工具

正如这个简短的调查所显示的一样，本章先前讨论的一些基于 XML 的软件产品价格昂贵，而基于 XML 的语言对聪明的开发者却是免费的。因此，这些开发者经常会为设计师、开发者和作者们提供符合他们需求的新产品。

个人发布工具，比如 Movable Type (www.movabletype.org) 可以让那些不太懂技术的发布人员和 HTML 专家一样方便地管理网站杂志、新闻站点和 Web 网志。Movable Type 使用了 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，如图 4.7 所示。

Movable Type 给它的用户提供了极其强大的内容发布功能，而正是 XML 赋予了它生命力。



图 4.7

Movable Type 是一个方便的 Web 发布工具，使用了 XML 来衍生出站点管理，以及使内容能让那些同样采用 XML 技术的站点所用 (www.movabletype.org)。

Movable Type 只不过是众多采用 XML 来管理和同步内容的网站发布产品中的一个。其他产品还包括 Radio 的 UserLand，如图 4.8 所示。UserLand 的 Frontier

站点 (<http://frontier.userland.com/>) 和 Pyra Software 的 Blogger 站点 (www.blogger.com)。这些产品持续的发展和普及，使那些个人（包括错过了第一次 Web “革命”的人们）能方便地在线上分享他们想法和乐趣，从而掀起了第二次 Web 浪潮。

图 4.8

Radio UserLand (<http://radio.userland.com/>) 使用类似 SOAP 的 XML 技术来“抛开所有 Web 发布问题的争论，让您专注于写作、发布图片和链接到您喜欢的站点”



因此，随着个人发布工具的传播，XML 就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。

Flash MX 使用 XML 来导入、导出数据并且基于 XML 工作，因此能够和 Macromedia 的私有的、有创造性能力的、被广泛使用的设计师工具交换标准数据而获益。开发人员可以使用同样的 XML 数据来驱动一个站点的 Flash 和非 Flash 版本，从而能够更好地使用资源，节省时间和费用。

4.1.8 听候您的吩咐

XML 的逻辑性也促进了整个 Web 服务市场的形成。基于 XML 的简单对象访问协议（SOAP）(www.develop.com/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，以及消息的处理变得非常便利。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 服务 (www.w3.org/2002/ws/) 领域中的

一个协议，一些大公司急切希望能够拥有它，如 IBM (www-106.ibm.com/developerworks/webservices/) 和 Microsoft (www.microsoft.com/.net/)。规模小的、独立的和开放源码的开发者们提供了一些与之竞争的分布式 Web 服务软件，但没有哪个公司能够占明显优势。David Rosam 定义 Web 服务如下：

Web 服务由基于 XML 的可重用的软件组成，并且相关的协议可以使商业系统以接近零成本进行交互。它们可以用来在 Internet 上为你的客户、提供商及合作伙伴提供快速构建低成本的应用集成。

[原文：www.dangerous-thinking.com/stories/2002/02/16/Web-ServicesDefined.html]

XML 驱动着大部分的 Web 服务的协议，它内置的强大交互能力是实现这样服务的主要原因。只要 XML 对所有人都是免费的，那就没有任何理由让任何公司（不论它有多人和多强大）垄断这个领域。

4.2 XML 应用程序和你的站点

可伸缩性矢量图形 SVG (www.w3.org/TR/SVG/) 和扩展超文本置标语言 XHTML (www.w3.org/TR/2002/REC-xhtml1-20020801/) 都是基于 XML 的语言。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 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 SVG 的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 (www.adobe.com/products/illustrator/main.html)。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 ECMAScript 和 DOM 进行操作和处理。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。

4.2.1 仍处于发展初期

目前，使用 SVG 必须在浏览器上安装一个类似 Flash 的插件（www.adobe.com/svg/），因此它的威力很大程度上受到限制。而且这个插件在跨平台和浏览器上工作得还不是很好。如果浏览器能够内置支持 SVG，那么对所有站点来说，可视化交互能力就会大大增强。

同样，在当前，即便是有了不少 XML 驱动的软件、数据库和 Web 服务，一些浏览器也已经可以有效地显示原始 XML 文件，并且创建的 XML 应用程序早就远远超出了大多数设计师和站点拥有者的需求，但是浏览器对 XML 的支持也仅仅处于发展初期。

开发者们已经通过创建基于 XML 的语言、协议和产品来解决前面的问题，使得我们可以“坐享其成”。W3C 组织也已经通过把 XML 的扩展功能与传统的简单的 HTML 相结合（我们将在第 5 章“现代置标语言”中详细解释）解决了浏览器对 XML 的支持问题。

4.3 与生俱来的兼容性

因为都是源自同一个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 普遍存在于专业和大众软件，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 解决了我们在第 1 章“99.9% 的网站都是过时的”中描述的网站不断被淘汰的问题。XML 已经非常成功地超越了所有人的想像力，它解除了“非兼容技术死亡论”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 会使得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，也都不愿让私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，私有格式的数据将被淘汰，即便它们曾经是潮流。XML 拉平了技术的门槛，并且任何人都可以参与。XML 是一个 Web 标准，而且它已经在工作着。

好的标准有一个的特点：用这种标准能完成一项工作，并且能和其他标准配合很好。这称为互操作性（W3C 这样称呼），或者可以称之为简单协作。不论你怎么称呼它，XML 比过去任何私有的 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。

4.4 协作的新时代

正如我们前面多次提到的，Microsoft、Netscape 和 Opera 最终在他们的浏览器中支持了相同的标准。由于技术的合作而带来一个意想不到的结果，这些曾经苦苦竞争的公司也学会了在一起很好协作，而且经常是在令人惊讶的方面。

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 的拥有者共同贡献的，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。

4.4.1 测试套件及其规范

W3C 的测试套件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）的测试套件。当缺少测试套件的时候，那些浏览器开发商只能通过他们自己的方法而希望很好地遵从标准。

此外，当缺少测试套件的时候，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，你怎么能确认你发明的技术可以充分地解决那些问题？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。

为了标准的制定者的利益，也为了浏览器开发商的利益，测试套件姗姗来迟。

4.4.2 套件是怎么工作的

当 Microsoft 着手开始纠正由于缺乏测试套件而引起的问题时，他们不是把

竞争对手搁置一边，而是邀请他们及外部组织（WaSP）来参与这个标准兼容的工作。就像人家所期待的，那些竞争对手和外部组织十分乐意地接受了这个机会。虽然这些工作中专利技术或者版权都是免费的，并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均没有意见。

通常情况下，Microsoft 不会关心什么对 AOL/Netscape 是最好的，也不会想办法帮助后来的公司，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意义的冒险上。但是，他们都聚到一起来了，不是为了争夺一个时髦的专利技术，而是为了“卑微”的 HTML 4。

忽略那些商业新闻的炒作，这个事件意味着巨大的改变。我们都经历过那个漫长时代，当时浏览器厂商们在对于 Web 标准和用户上采用了“只用我们的浏览器”的策略，并且希望在市场上打击竞争对手。现在浏览器的开发商还在不断改革创新，而且他们也仍然希望你能够选择他们的产品而不是竞争对手的。但他们在一起合作的事实还是说明，他们尊重 Web 标准的系统工作，并且也显示了在这个行业领域已经大大改变了，完全不同于浏览器大战时代（1996 年-1999 年）。

不要相信谣传

有时候，某些报纸或者商业杂志会宣称浏览器市场大战又开始了之类的观点。这发生在 2002 年 6 月份，当时 AOL 把它的 CompuServe 用户从基于 IE 的浏览器转到基于 Mozilla/ Netscape 的一个浏览器上。商业杂志说：“Web 市场的变化使得开发者心烦！”“第二次浏览器大战！”某些消费者报纸叫嚷着。在几个月之后，当 AOL 把它的 Mac OS X 用户切换到基于 Gecko 的浏览器上时，类似的消息又发生了。所以不要相信谣传。

在当前的新闻业，编辑们如果想保住他们的工作就必须将他们的报纸能够卖出去。危机和冲突比那些详尽的报告更容易使报纸卖出去，而且小报本身就很少关心迟到的真实内容。不管在市场份额上是否有一些变化，浏览器的战争其实早已经过去，也没有一个编辑真的希望浏览器升级战重新开始。

Web 将会被建立在本书所讨论的技术基础上，并且会被所有主要的浏览器所支持。AOL/Netscape 和 Microsoft 可能有时还会有冲突发生，但是它们之间的竞争已经大部分从浏览器领域转到与我们关系很少的方面去了（除了 FontPage 外，这个我们另外讨论）。

Web 新时代

虽然不像联合国的和平计划那样不可思议（和重要），Microsoft 非常平静地

向 W3C 贡献了“HTML 测试套件”，给其商业对手们一个信号：Web 还将按照这种方式继续前进。当竞争对手也通过这种方式进行协作，那就是说明这种方法已经成熟了。

任何成熟的行业发展过程中都会有类似的事情发生。例如在唱片行业，在 P2P 音乐下载流行起来的时候，当威胁到生存的时候，那些互相竞争的唱片公司就会联合起来提交一个新的行业标准。当 Microsoft、AOL/Netscape 以及 Opera 能够一起合作，这就告诉我们 Web 已经走向成熟了。他们联合起来贡献的东西（一个 W3C 测试套件）告诉我们为什么这个成熟期已经到来。我们的媒体也成熟了，因为本书正在讨论这些标准。

可以预想，未来几年内，会有更多的合作和更多对标准的支持。我们也可以轻松一下，因为我们明白那些用标准创建的站点能够继续工作在未来的浏览器上，甚至可以到 10 年以后。联合行动表明了他们支持向后兼容，正如设计师和站点拥有者一直希望的。

另外一个浏览器开发商承诺支持标准的例子，Netscape 资助了一个小型标准推广小组，他们的使命是：在浏览器和网站以及基于开放标准的跨浏览器方案中，改善标准支持。不再是“Netscape 最佳显示”或者其他。

4.5 Web 标准和创作工具

浏览器大战正酣之时，一些市场领导者及专业视觉设计商，如 Macromedia 公司的 Dreamweaver 和 Adobe 公司的 GoLive 通过为 3.0 和 4.0 浏览器优化，生成相应的标记和代码，来克服浏览器不兼容产生的问题。

Dreamweaver 和 GoLive 会为不同浏览器版本的准备好针对它们的非标准的、私有的 HTML 代码。如果每个浏览器都有它自己不兼容的文档对象模型(DOM)，并由各自独有的脚本语言驱动的时候，Dreamweaver 和 GoLive 也会自动生成相应的代码。

这种情况下，由 Dreamweaver 和 GoLive 生成代码会比开发者自己手工书写的代码出现的错误少。正像我们在第 2 章“根据标准设计和制作”解释的一样，在当标准仍在发展之中而浏览器又没有支持它们的情况下，站点创建者该怎么做。“要什么给什么”那个时候是有一定意义的，但今天它不再是个合适的策略。因为当浏览器支持标准的时候，类似 Dreamweaver 和 GoLive 的工具也相应会进行支持。今天，它们已经这么做了。

4.5.1 Dreamweaver 特别小组

Web 标准组织中的“Dreamweaver 特别小组”创建于 2001 年，它一直和 Macromedia 的工程师致力于提高用领先的专业化 Web 编辑工具 Dreamweaver 产生的站点与标准的兼容性和可访问性。这个特别小组的历史可以通过 www.webstandards.org/act/campaign/dwtf/ 获取。

这个组织的主要目标是由 Rachel Andrew 和 Drew McLellan 制定的，主要如下：

- Dreamweaver 应该产生有效的标记 out of box[有效的标记只是用标准的标签和属性，并且不会产生任何错误。我们将在后面的第 5 章讨论]。
- Dreamweaver 可以在 XHTML 和 HTML 版本中进行选择，而且为其中任何一个选择一个有效的 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置标语言，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 CSS2 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。
- 在没有用户的同意下，Dreamweaver 不应该通过插入默认的内置样式来纠正破坏合法的 CSS 布局。
- Dreamweaver 能让他们的用户充满信心，并认为他们用 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 中已经达到上述目标及其他一些目标。通过评估这个他们极力打造的产品，这个小组发现 Dreamweaver MX 达到如下特点：

- 产生有效的标记
- 帮助用户创建可访问性高的站点
- 处理 CSS2 到一个恰当精确的程度（虽然 CSS 的定位有些问题）
- 避免了破坏 CSS 布局
- 增强的 XHTML 和 CSS 校验（自动测试标准兼容性）
- 尊重和促进 Web 标准

你自己看看

你可以阅读 WaSP 的 Dreamweaver 特别小组的产品评估全文，地址为 www.Webstandards.org/act/campaign/dwtf/mxassessed.html。

4.5.2 WYSIWYG 工具的成熟

Dreamweaver MX 对 Web 标准的支持和强大的可访问性，已经远远超出了他们自己宣传的。Dreamweaver 的功能和对标准的支持都是和当前的浏览器相称的。

虽然 WaSP 未能够与 Adobe 进行合作，但是这个公司也在他们的专业可视化 Web 编辑工具 GoLive 中对标准支持进行了独立的、充分的改进(www.adobe.com/products/golive/main.html)。

除了 CSS 和 XHTML，GoLive 还支持同步多媒体集成语言，即 SMIL (www.w3.org/AudioVideo/)，一个 W3C 推荐的标准，用来创建可访问性高的多媒体表现。使用这些市场领先的可视化 Web 编辑工具的人们可以很容易地制作出标准兼容的可访问性站点。

4.5.3 FrontPage：非标准兼容的设计

还有第一个产品，即 Microsoft 的 FrontPage，在专业人员中应用非常少而在非专业人员中却广泛应用，可能是因为它捆绑在其他 Microsoft 的产品中。这个客户群体中的专业人士发现他们坚持使用 FrontPage，是由于预算经费不允许他们选择“其他”的 Web 编辑工具。

“我们已经拥有自己的 Web 编辑软件了”，他们可能会这么告诉你。但是他们错了，Frontpage 并不是一个可视化的 Web 编辑软件，它只不过是个 IE 页面编辑器。

虽然 Microsoft 开发了标准兼容的浏览器，但是 Frontpage 工具却产生不兼容的标记和代码，生成的站点只能在 Internet Explorer 浏览器中才能正常浏览和工作。Frontpage 的非标准的页面并不是由于不称职的原因。这个 bug 本身就是个“特色功能”，尽管这个特色功能并不是出于对用户的利益考虑进行设计的。

在法庭证词上，Microsoft 的比尔·盖茨先生承认他直接对 Office 软件的开发小组，其中包括 FrontPage 发了一个备忘录，让他们停止对别的软件产品交互的工作（那就是说，让他们停止对标准的支持）。盖茨先生不希望竞争产品能够打开、保存和编辑 Microsoft 产生的文档。因此，如果你使用 FrontPage 来设计

或者开发一个站点，你很有可能会让你的站点只能在IE浏览器中正常显示和工作。

不过还是有希望能够解决的。在2002年7月，UsableNet宣称他们将LIFT可访问性工具集成到Microsoft的FrontPage中（www.usablenet.com/frontend/oneneWS.go?news_id=45），可以让FrontPage用户制作的站点具备兼容性（LIFT同样可以和Dreamweaver MX集成）。虽然不能保证FrontPage下一步的工作就是进行标准的兼容处理，但这个新东西还是能对FrontPage的兼容性做出一些小小的努力。

目前，如果你想创建标准兼容的可访问性高的站点，最好的选择就是通过手工在文本编辑器进行代码书写或者通过Dreamweaver或者GoLive工具来进行。除非你喜欢重新再做一遍书写代码的工作。

4.6 CSS 布局的出现

CSS1规范在1996年被建立，同时也结束了极富表现力的HTML，将它们与网站包含的数据分离开来。在2000年，所有的主流浏览器都完全具有合适显示纯CSS布局的能力。但是只要有一部分访客仍固执地使用4.0和更老的浏览器，设计师和开发者就会对采用CSS设计感到犹豫。因此必须做一些事情，让大部分的设计师与开发者感到Web标准是“安全”的。网页标准组织WaSP决定采用“游击策略”，这是必需的。

4.6.1 浏览器向标准升级之战

2001年2月，Web标准组织发起了“浏览器向标准兼容的升级战役”活动（www.Webstandards.org/act/campaign/buc/）。就像它的名字建议的一样，活动是想要浏览器使用新的更符合标准的浏览器，同时也鼓励了设计师使用Web标准代替HTML私有代码。

在一些情况下，活动鼓励开发者们行动起来支持标准，就当他们全部的用户已经使用且仅使用标准浏览器。在页面的<head>区中嵌入的JavaScript代码可以测试浏览器对DOM解析。如果浏览器能“解释”DOM，浏览器就能显示这个页面；否则访问者将会跳到一个他自己的浏览器可以解释的页面上。

网页会建议用户升级他们的浏览器，比如说升级到最新版本的IE、Netscape、Opera或其他合适的浏览器。同时，网页还会告诉用户升级浏览器能够改善他的网络体验。

与“本页面推荐使用XXX浏览器”不同的是，WaSP的浏览器向标准升级

战役是制造商没有想到的。只要这个浏览器遵循标准，我们才不关心你下载使用哪种浏览器。

这个强有力的技术一开始只推荐给那些能正确使用 CSS 布局和 DOM 的网站，并且仅仅推荐给非营利性和非正式网站，因为它们可以承担采用这种方法的风险。我们鼓励大多数设计师和开发者建立“浏览器升级”页面，并定制他们的访客的需求：这些技术仅仅适用于有效的、合乎标准的网站。采取这样的步骤，需要谨慎地衡量其有利和不利的方面。

浏览器升级的使用和滥用

哎！一些使用这个强有力的技术的偷懒的开发者却时常将访问者摒弃在不十分兼容标准的网站之外。更加伤害用户的是，他们始终将用户转送到 WaSP 那里，而非创建他们自己相应的升级页面。你可以预料到，那些想尝试一下标准兼容的浏览器的用户更加失望了。

举例，下面这个拉拉队队长的宣传站点 Raiderettes.com，如图 4.9 所示，它是参加 Web 标准组织的“浏览器升级战役”活动的一个站点，是一个漂亮的网站，但是如果尝试验证它的代码却产生如下报告 (<http://validator.w3.org/>)：

```
Fatal Error: no document type declaration; will parse without validation.  
I could not parse this document, because it uses a public identifier that is not in my catalog.
```



图 4.9

Raiderettes.com，Oakland Raider 拉拉队的官方网站 (www.raiderettes.com)，看看在结合 Web 标准支持方面做了些什么

（致命错误：没有文档类型声明，将无法确认解析。不能处理这个文档，因为它使用了一个无法识别的公共标记。）

与其他一些商业网站一样，Raiderettes.com 也使用了 WaSP 的“浏览器弹回”（browser-bouncing）代码。这并非是标准服务，仅仅是一种拒绝那些不能处理 DHTML 菜单的浏览器的简单办法。不用说，Web 标准组织收到许多来自激素分泌过剩的男人的邮件，他们想看美女图片而不是阅读 CSS 和 ECMAScript。

友好、温和的升级方式

除了上述的一些例子外，这项活动还是成功地鼓励了大多数设计师和开发者尝试标准化。同时因为销售压力而引起商业公司的注意力，提高了他们对标准化的认知。同时，它说服了用户下载那些兼容标准的浏览器。

虽然因为它那些奇怪的浏览器模块使这次浏览器升级活动声名狼藉，但这是从多层次上来努力提供一个用户友好界面的策略，做得越好压力就越小。为了支持重新启动该项活动，并证明它更具用户友好性的一面，A List Apart (www.alistapart.com) 同时使用 CSS 排版和有效的 HTML（很快地通向 XHTML）来重新设计，如在第 2 章中描述的所示。

为了示范活动的适应性，ALA 尽量避免排斥任何访问者，取而代之的是，网站对那些不兼容标准的浏览器只是隐藏了排版布局。使用 IE5+、Netscape6、Opera 5+和其他具有相同能力的浏览器的读者能够同时看到内容及其排版。而那些使用旧的浏览器的用户，只能看到内容而没有排版设计。使用不兼容标准浏览器的用户同样可以看到一个“浏览器升级”的提示，而这个提示对于版本较高的浏览器来说是隐藏的。标准仍然在被推广，但每个访问用户也都是受欢迎的。

为了不损害设计师的兴趣，ALA 宣称了一种用户更好的方法。如图 4.10 和图 4.11 所示。

图 4.10

“告别老式浏览器”，这是 ALA 宣言。设计师们将挑战使用 CSS 排版布局和使用 DOM 实现行为。并且使代码就是代码，而不是迫使 HTML 来做排版的工作 (www.alistapart.com/stories/tehelli/)



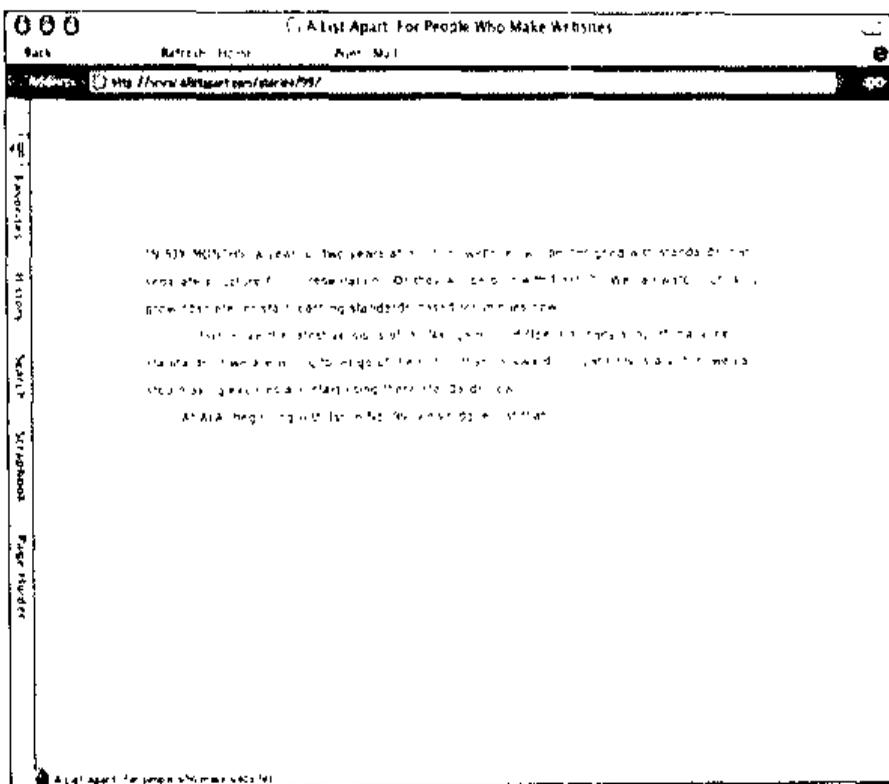


图 4.11

ALA 第 99 期文章中的浏览器升级活动同时出现在 2001 年 2 月 (www.alistapart.com/articles/upgrade/)

ALA 向它的每周 70 000 的浏览器宣传，让他们停止为不使用标准找借口，并开始在他们的网站上结合标准。在 Web 会议（由 CSS 排版）的草案上纂写了这样的序言：

“在 6 个月，一年或者最多两年，所有的网站都将采用表现层和结构层分离的 Web 标准设计（或者他们都由 Flash7.0 设计）。我们要么看着我们的技术变得落后，要么现在就开始学习基于标准的新技术。

实际上，因为最新版本的 IE、Navigator 和 Opera 已经开始支持许多 Web 标准，如果我们心甘情愿放弃‘向前兼容是一个优点’的观念，那我们就可以停止寻找借口并开始使用这些标准。

在 ALA，从第 99 期发布开始，我们已经这么做了，加入我们吧！”

4.6.2 风起云涌

社论引起了共鸣，在几天之内，一个接一个的独立网站开始转变到 CSS 排版设计和 XHTML 结构。Todd Dominey 的每日网站（如图 4.12 所示）是这些 CSS 转变者的代表。他们选择的新技术的质量都好于平均水平。在 Atlanta 的一个新媒体基地，Dominey 把 Flash 技术应用在它的多项获奖网站（如图 4.13 所示）上，并且在自己的每日网站日志上使用 CSS 和 XHTML 技术。

图 4.12

Todd Dominey 的两个方面：新媒体设计师。他优秀的每日网站在文件结构上使用 XHTML，并用 CSS 排版设计（www.whatdoiknow.org）



图 4.13

Dominey 的获奖网站更优秀地应用了 Flash 技术 (www.domineydesign.com)



在后来的网站上，Dominey 解释了为什么转变到纯 Web 标准：

“这个网站使用了成熟的 XHTML/CSS 排版技术，没有使用透明的 GIF 图来做间隔，也没有行、列，没有垃圾代码。原因很简单——我是一个商业新媒体

设计师，需要一个空间来体验先进的设计技术，但客户往往不允许我们使用的（在一些案例中，事实的确如此）。排版技术日新月异，而且浏览器也在持续更新。如果你正在使用 IE5.0 以上的版本，或者 Mozilla、Netscape 6 和 Opera，你就没有问题。其他更老的或者 Beta 版本的浏览器，比如 iCab 和 OmniWeb 就有问题了。如果你们看到的都是灰色的背景和蓝色的字体，你会很渴望升级你的浏览器。

www.whatdoiknow.org/about.shtml

4.6.3 无数正在转变的网站和他们依靠的帮助网站

在两年中，自从浏览器升级活动和 ALA 重新设计像一记组合拳冲击了设计团体，无数个人网站和网络日志都开始走上 CSS/XHTML 路线（当然，也许还不是很像组合拳，我们都是设计师，不是拳击爱好者，我们的运动比喻经不起仔细的推敲）。

为了使更多的人对 CSS 布局感兴趣，许多个人网站发布了公开的 CSS 布局源代码，可以让大家免费获得。如果你不明白 CSS 是如何工作的，或不确定如何用样式表排版页面，你随意访问任何一个他们的网站都可以学习并拷贝他们的代码。

- Blue Robot 的排版资源（如图 4.14 所示）。



图 4.14

Blue Robot 的布局资源提供整洁、有用的 CSS 布局技术，可以免费获得 (www.bluerobot.com/web/layouts/)

- Eric Costello 的 CSS 布局技术 (<http://glish.com/css/>)
- Owen Briggs 的“小盒子”视觉排版页面（如图 4.15 所示）

415

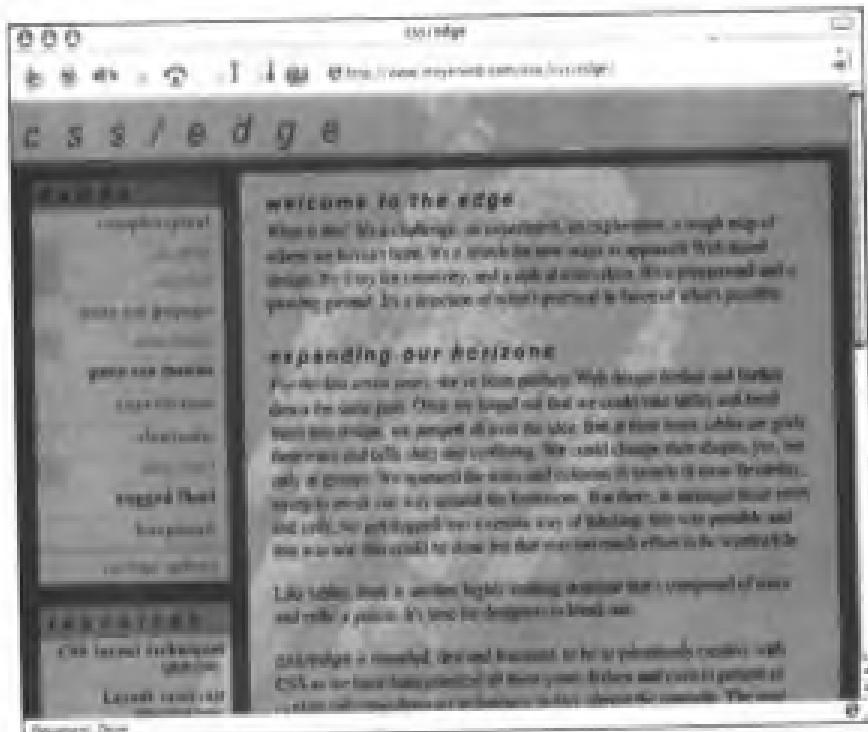
Owen Briggs 的小盒子視覺排版頁面 (http://www.thenoodle-incident.com/tutorials/box_lessons/boxes.html)，若想看到排版，單擊一下頁面即可取得他們創造的 CSS 代碼（你可以在自己的網站項目中使用）。



为了满足更大的求知欲望，Eric Meyer（“Eric Meyer on CSS”一书的作者）提供“css/edge”（如图 4.16 所示）的 CSS 排版技术，这项技术只在大部分标准兼容的浏览器上有效。这样的技术设计不能在客户端工作，但是他们为这一类工作提供培训基础，我们可以在此基础上继续来年的工作。

P4-16

Eric Meyer 的 `css/edge` 技术仅仅在最新的、最兼容的浏览器上工作。
(www.meyerweb.com/eric/css/edge/)



大多数 CSS 设计师都愿意设计可以在 IE5 或者更高版本的浏览器里正确浏览的网络，并且也能够接受低版本浏览器（至少能够显示内容），如 Netscape 4。Mark Newhouse's Real World Style（如图 4.17 所示）提供了在 Netscape 4 和其他更过时的浏览器中也能够接受的有效 CSS 布局。



图 4.17

相对于 casewide, Mark Newhouse 的 Real World Style 网站 (www.realworldstyle.com) 提供一个开放源代码的 CSS 布局技术, 即使在 Netscape 4 这样老旧的浏览器上也能够工作得相当好 (当然, 他们在更新的、更兼容的浏览器上也能够工作得很好)

4.6.4 流行时尚

当 CSS/XHTML 组合技术已经被个人网站和独立网站所掌握时, 符合 508 条款或 WAI 可访问性指南的可访问性规范加了进来。当写这本书的时候, 要符合可访问性要求可以采用以下两个方法之一 (就是前面展示的 Todd Dominey 的两种方法)。

- 向后兼容, 以标准为基础的网站使用 CSS 布局和 XHTML 来做文件结构, 并遵循 508 条款或者其他可访问性原则。
- 使用 ActionScript (Macromedia 公司的脚本语言基于标准的 ECMAScript) 建设的完全的 Flash 网站。

坦白地说, 如果你的个人网站还没有应用 Web 标准或者 Flash, 你的许多同行设计师将认为你是不懂时尚的落伍者。这样势利的观点看上去微不足道, 但事实的确如此。如果你不想被嘲笑, 就抓紧学习新技术吧。

来自个体的领导能力

个人和独立网站一直推动着技术发展，他们的改革创新也一直使主流网站的设计得到灵感。JavaScript、frames 和弹出窗口（pop-up windows）技术一直应用在越来越多的个人网站上，没有人因为浏览这些网站而遇到麻烦，这也说服了商业网站的设计师们结合使用 JavaScript、frames 和弹出窗口技术是很安全的。于是，我们也能在电子商务网站上看到 DHTML 的下拉菜单了（www.coach.com/index_noflash.asp），并在 www.nytimes.com 中每天都固定的弹出讨厌的广告窗口（很明显，这可不是一个好的事情）。

从前沿站点向主流商业站点的这种技术转换将越来越好。因为这些技术前沿站点现在更关心的是标准和可访问性，而不是暗藏的技巧。在个人站点的鼓舞和法律的条款的出台下，这种转变已经开始。

4.7 Web 标准的主流

在 2001 年，美国和加拿大公布了一个指导方案，要求那些政府相关的站点使用 Web 标准进行开发并且加强可访问性。英国和新西兰政府马上就对这个方案进行支持，众多的美国站点也遵循这个方案。

在 2002 年，主要的政府相关站点，包括 Texas Parks & Wildlife (www.tpwd.state.tx.us) 和 Juneau、Alaska (如图 4.18 所示) 的网站，都转向了包括 CSS 布局在内的 Web 标准。Texas Parks & Wildlife 解释了他们的转化方式 (www.tpwd.state.tx.us/standards/tpwbui.htm)：

“作为一个政府机构，Texas Parks & Wildlife 必须符合 Texas 行政代码 §201.12，并用 W3C 制定的 Web 标准来编写网页代码。这些需求背后的动力主要是可访问性。Web 页面必须能让所有用户可访问到，不论是否身体有残疾，或者是否懂技术。”

但是，如果一个页面对于所有的访问者看来都不是一样的，那他们怎么做到良好的可访问性呢？

可访问性不意味着每一个人都可以看到相同的页面，而是关于怎么让所有的内容（文本、图像和多媒体信息）能够让所有的用户都可以访问得到。为了达到这个目的而遵循标准，TPW 使用了 CSS 样式表来把内容从表现中分开。把表现和内容分开可以使得 TPW 提供更高质量的页面和动态及时的内容。

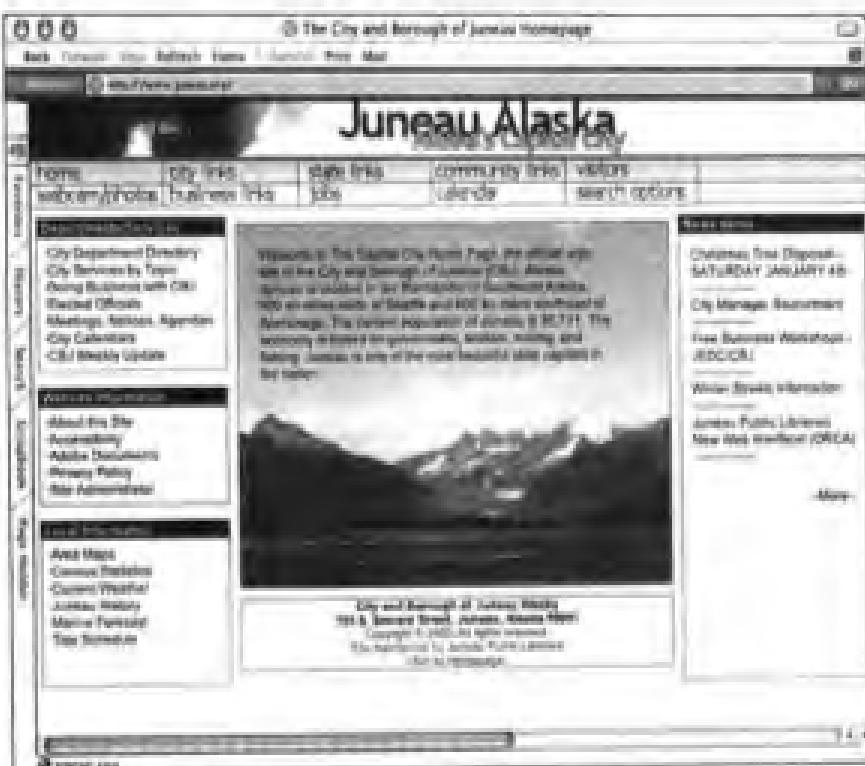


图 4.18

欢迎来到美丽的 Juneau, Alaska。旅游、采矿和钓鱼的圣地 (www.juneau.org)。Juneau.org 曾是第一批采用过时的框架技术的站点之一，而现在使用了 CSS 布局和结构化的 XHTML 赋标语言。

由 W3C 制定的 Web 代码标准能够创建可访问的页面。通过使用类似 CSS 样式表和 XHTML 的 W3C 标准，Texas Parks & Wildlife 已经着手创建兼容的 Web 页面。”

Alaska 首府网页的开发者提供了一个类似的例子 (www.juneau.org/about/stylesheets.php)：

“我们正在转化 CBJ 站点，从框架结构转变成样式表 (CSS) 标准。这个标准可以使我们进行更自由的设计，同时也能为我们的用户提供更强的可访问性。使用了样式表，就可以让我们的内容对几乎所有的浏览器和计算机平台兼容，包括手持设备和 ADA 设备。维护同一个页面而不用单独维护‘纯文本’和‘友好打印’的冗余版本。同样，它可以让我们避免很多 Web 设计陷阱，比如我们当前遇到的‘基于框架’的布局。

惟一不利的方面是很多老浏览器不兼容样式表。但不支持样式表的浏览器会将 Web 页面内容用文本显示出来。虽然这样的效果不是很漂亮，但是所有的信息（包括链接和图像）都会被显示出来。事实上，使用了样式表，Web 设计师可以在一个非兼容的浏览器中控制内容显示顺序，而不用关心这个站点在兼容浏览器中怎么显示。这个功能是样式表的主要优点之一。它允许 Web 页面可以在旧版本浏览器和新版本浏览器中显示。作为一个 Web 站点的设计师，我们宁愿让

用户说我们的 Web 页面‘看起来滑稽’也不让他们说‘看不了’。

大多数当前流行的浏览器的新版本都是兼容 CSS 的。如果你当前的浏览器不知道是否支持样式表，请单击下面的按钮下载一个兼容的浏览器。对于旧的系统（比较慢的处理器和很少的内存），请选择 Opera 浏览器。这个浏览器能比其他两个浏览器需要更少的系统资源。”

4.7.1 商业站点冒险尝试

在 2002 年，主流的商业站点都在开始转向 CSS 和 XHTML 技术，他们的开发者肯定曾经讨论过“向前兼容性”的问题。在那年的 7 月份，具有领先水平的搜索引擎 Lycos 的欧洲站点转换到了 XHTML 和 CSS 布局技术。差不多同一时间，超快的 AllTheWeb 搜索引擎（如图 4.19 所示）也同样转向了 CSS 和 XHTML 技术，并且为他们的客户提供了可定制的样式表。

图 4.19

一个和 Google 非常相似的搜索引擎 AllTheWeb，在 2002 年已经转成了 XHTML 结构和 CSS 布局（www.alltheweb.com）



在本书的写作过程中，Google 和 Yahoo 已经开始跟随这些比较小的竞争者，但正像我们所说的，这是个自然的变化：小家伙们会先涉足危险境地，而大家伙们如果确认这样做是安全的话，也会跟随着做。

4.7.2 Wired Digital 的转化

在 2002 年 9 月份，很受欢迎的 Wired Digital 站点以一个完全标准兼容的站

点而重生了：数据采用 XHTML 格式，CSS 用于表现层处理，如图 4.20、图 4.21 所示。团队的负责人 Douglas Bowman（如图 4.22 所示）负责管理整个重新设计，并且把兼容标准作为第一优先考虑的事情。



图 4.20

Wired Digital (www.wired.com) 在 2002 年的时候转向了标准支持，在架构上使用了 XHTML 和 CSS 布局

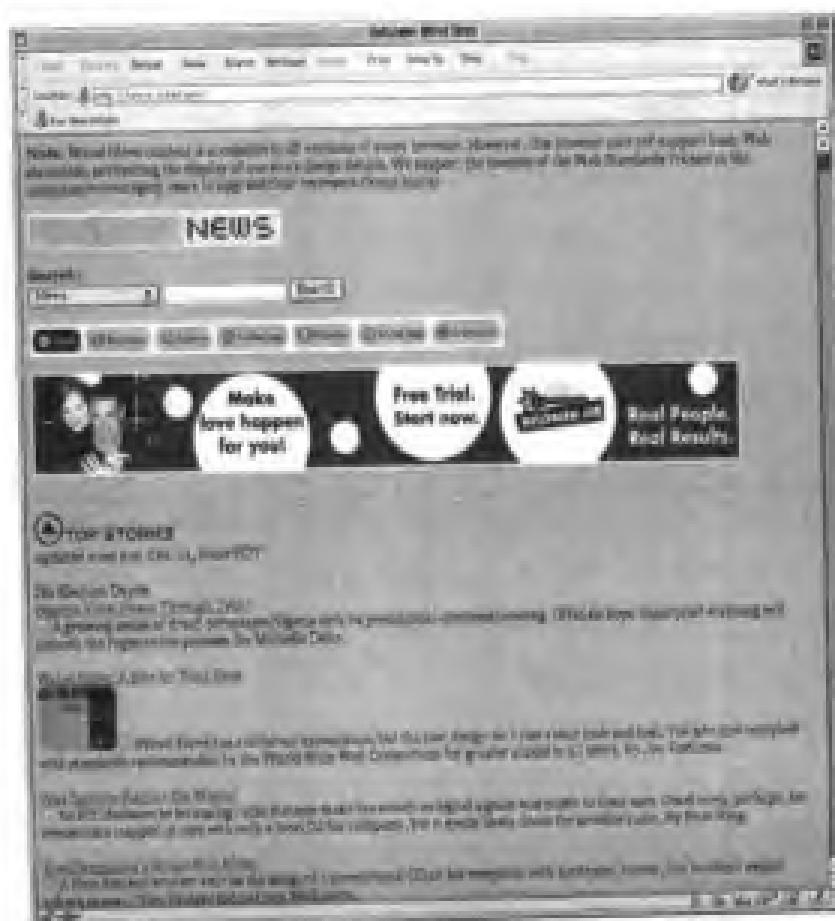


图 4.21

在一个非标准兼容的浏览器中浏览的模样（在这个例子中为 Netscape 4）。Wired Digital (www.wired.com) 能够提供所有的内容访问，不过布局就隐藏起来了，使用同样的技术部署。在 2001 年 ALA 进行了网站重构，并且链接到 Web 标准组织升级战役的页面，以激励用户升级到新浏览器

图 4.22

虽然很多设计师和开发者进行 Wired Digital 重新设计的工作，但是它的标准的跨越式很大程度上是通过 Douglas Bowman 一个人工作和努力获得的。随后他离开了 Wired 并且创办了自己的公司：Stop Design。Stop Design 的站点同样大量使用了 CSS 样式布局和良好架构的 XHTML。



作为一个大流量、高度直观且拥有理解和使用 Web 技术良好的声誉的站点，Wired 已经为开发社区进行了长时间的指导工作和服务。它对 Web 标准的转化和支持为那些产业发出了一个清晰的信号，向后兼容性的时代已经来到了。

一粒老鼠屎坏了一锅汤

起初校验错误影响了 Wired Digital 标准转化的启动工作。有些错误是由 Wired 内部使用的编辑器通过 Vignette 内容管理系统产生的。其他错误是由一些第三方用不标准的方法创建的广告内容和一些不恰当的 URL 处理。我的代理商 Happy Cog 也面临同样的障碍，你也可能会遭遇到同样的问题。当你开发完兼容的模板后，过时的内容管理系统（Content Management System, CMS）会加入垃圾标记和私有代码，因此影响到整个站点的兼容性。

在 Wired 这个站点中，这些问题很快就被解决了（主要是由于 Bowman 的坚持）。在开始后的一两天内，Wired Digital 就拥有了一个完美的标准兼容的大规模的商业站点。不是所有的公司通过 Vignette 介绍就能够解决问题的技术能力，也不是任何公司都有能力那样做。

为了 Web 的发展，发布工具必须能够和标准兼容（参见阴影部分，“发布系统和标准”）。站点的拥有者和管理者应该要求 CMS 厂商必须要进行兼容，就像设计师和开发者曾经要求浏览器支持标准一样。如果有足够的用户这么做，这

些厂商可能就会更乐意升级他们的工具，并且能够使得他们更容易地实现标准的兼容。

发布系统和标准

即便最初的模板是正确的，并且客户和设计师都完全支持 W3C 的规范，但是在大规模的商业站点上，也很少有完全能通过 XHTML/CSS 的校验的。不可否认，这一类的客户和设计师们还是比较少的，但是即便是这些人的努力也总是会被过时的中间件、混乱的数据库及其他障碍弄得很混乱。

就像我们在本章前面提到的，独立的设计师和开发者在根据 W3C 的 CSS 和 XHTML 上进行工作是没有任何问题的。有些人甚至喜欢修改那些跟他们毫无关系的商业站点，把这些站点带入到标准行列中只需要进行少数的工作（比如，在第 1 章中 Eric Meyer 就是这样对待 KPMG 站点的）。

完全正确地达到有效的 XHTML 和 CSS 以及可访问性优先，并不是很难。任何中等能力的设计师和开发者都可以做到。问题主要存在于大规模的系统和在这个大站点中集成的第三方的内容。

为了保证我们站点的健康和交互能力，我们需要内容管理系统，因为它鼓励兼容，高可访问性。我们需要那些开发这些系统的人们认识到对以前妥协已经终结了，“强制的显示问题”不再是个可以接受的选择。并且我们需要那些希望为了向后兼容而愿意投资的客户。这样做会更有说服力，并且会带来财富。

现在不是没有说服力，财富则是另外一个问题。

在资金充裕的情况下，公司可以对研发、培训和长期规划进行投资。如果资金不足，则公司会主要关注的是如何缩减成本、裁员和减少工作流程，保证公司的生存。鼓励设计师和开发者学习兼容性的开发方法是很容易的，只要给他们展示能得到的好处就可以了。诱惑这些努力的公司为他们的站点进行长期的健康的投资，就像面对一次从黑暗走向光明的挑战。

4.7.3 用过渡方法拥抱标准

不是所有的标准兼容都需要采用 CSS 布局，并且不是所有希望获得 Web 标准与互动性的商业和组织都需要一步到位。第 2 章描述了一个达到标准兼容的过渡方法，这个方法可以使得老的和新的页面能够集成起来，并且迎合了那些公司和协会的需求，特别是对于那些已经安装了大量老的非兼容的浏览器的协会和组织，他们更偏向于缓慢地进行技术转化。

纽约公共图书馆就是这样一个组织，在 2001 年，在 Carrie Bickner 的领导下，纽约图书馆的分馆 (www.nypl.org/branch/) 已经采用了 XHTML 1.0 标准作为标记语言，在表现层上把过渡的基于表格的布局和 CSS 样式表进行了集成。这个图书馆也遵守美国的 508 条款能够提供给更多的有效的影响和服务。

同时，随着这个站点的标准化过程，这个图书馆发布了一个样式指南 (www.nypl.org/styleguide)，它由 Bickner 和你现在的 scribe 共同组成，解释了对于所有的图书馆分馆计划的标记和设计需求，同时也包括了一个关于 XHTML 和 CSS 的教程。

NYPL 图书馆并没有只把这个样式表指南放在内部网络，而是选择了在外部网络公开，希望帮助成千上万的设计师和开发者能够迅速应用 XHTML 和 CSS，并且鼓励其他组织支持标准。在撰写本书的时候，它已经在两个方面获得成功。

正如一开始就点燃了独立开发者的想像力之火一样，现在标准也在鼓舞着网站设计师和开发人员，他们能够正确看到，Web 标准和可访问性正是能够满足扩展用户和服务的关键。

4.7.4 W3C 的参与

过去 W3C 的策略是发布标准而不是推动标准，但现在 W3C 的这个情况已经中止了。在 2001 年，W3C 成立了一个质量保证组织 (<http://www.w3.org/QA/>) 来更好地与设计开发社区沟通，并且保证 W3C 的规范变得可用，得以正确的实现。W3C 还发布了一系列文档来解释和促进它的标准。

一个关于可访问性、基于标准的设计的 W3C 的文章草案 (www.w3.org/WAI/bcuse/benefits.html)，包括持续增长的市场占有率和用户影响，改进的效率（低成本），减少诉诸法律的概率，以及清晰的社会责任。这篇文章非常值得阅读、打印和共享。

如果 W3C 的文章中提到的好处听起来很熟悉，你可能已经很认真阅读过本书而不是随便翻翻而已。如果你觉得这个文章有些勉强的话，就将标准和可访问性介绍给你的用户以降低成本。如果这个优点不能引起那些公司的领导者的兴趣，他或者她应该不是和我们一样生活在同一个星球上。

4.8 执行概要

类似 XML 的标准已经改变了商务的面目，对标准的支持已经终止了浏览器大战，并且使得他们专注于可用性、可访问性及长期的生存等方面，同时形成了

即便是最顽固的竞争者之间合作的新领域。类似 CSS 和 XHTML 的标准，已经被主流的浏览器和专业的编辑工具所支持，已经更好改变了站点的设计方法，不仅仅应用于中立派和个人站点，而且也应用在政府、社会公共机构和没有经费升级的商业站点中。Web 标准已经获胜，而你已经确确实实拥抱它了！

现在让我们暂停欢悦的情绪并且开始工作吧，在第 5 章中会继续讨论这个问题。

Part II

第2部分 设计与构建

- 现代置标语言
- XHTML：Web 重构
- 紧凑而坚固的页面保证：
以严格和混合标记组成的结构
- XHTML 的示例：混合布局（第一部分）
- CSS 入门
- CSS 应用：混合布局（第二部分）
- 使用浏览器 第一部分：
DOCTYPE 转换和标准模式
- 使用浏览器 第二部分：
盒模型、bug 和工作区
- 使用浏览器 第三部分：排版
- 可访问性基础
- 使用基于 DOM 的脚本语言
- CSS 重新设计

现代置标语言

在第1部分“休斯顿，我们出问题了”中，描述了过时的 Web 设计方法所产生的可访问性和商业问题，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的美好前景。本书下面的部分将从概述转向具体的技术，最好的方式就是先从最基础的 Web 置标语言开始。

很多设计师和开发者都有思想上的畏惧。然而，那些曾经花了数个星期时间来学习 HTML 语法以设计一个专业站点的人们，难道就不能花一点时间学习更新的、功能更强的语言吗？难道学习一些如 PHP、ASP 或者 ColdFusion（参考“什么是 PHP”）等的服务器端的技术，就不如那些浪费宝贵时间在 HTML 表格和段落标记等入门技术重要吗？

答案是：更重要。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。就像在本书中讨论的 Web 标准，服务器端的脚本语言也能执行从界面中提取内容的类似功能。类似 CSS 的 Web 标准把设计师从无意义的影响带宽的表格元素的内容片断解脱出来一样，类似 PHP 和 ASP 的语言也把站点建造者从蠢笨、辛苦的手工制作每个页面工作中释放出来。

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大用处。如果那些动态页面不能在一些浏览器及设备上显示，或者不能在 10 秒钟内完成拨号网络加载，那么这个服务器端的技术就不可能完成那些你或者你的用户想完成的工作。简而言之，服务器端技术和标准缺一不可，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。

什么是 PHP

PHP (www.php.net) 是一个开放源码、通用的脚本语言，主要适用于 Web 开发并且可以嵌入到 XHTML 中。它利用了 C、Java 和 Perl 的语法并且非常容易学习。PHP (有些混乱的官方解释，PHP 就是：超文本预处理器 Hypertext Preprocessor) 有很多功能并且在一些用户中已经变得非常流行：当和 MySQL (www.mysql.com) 组合使用的时候，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 Apache 软件基金会 (www.apache.org) 的一个项目，这也是它得以流行的原因之一（另外一个原因是：它是一个挑错和调试工具）。开放源码开发者和独立的 Web 设计师沉醉于这个语言并且不断地开发可免费获得的 PHP 程序。比如 Dean Allen 的 Refer 程序 (www.textism.com/tools/refer/)，能够跟踪来访者从哪个其他站点链接来到你的站点的信息如图 5.1 所示，还有 Dan Benjamin 的 URL 清洁器 (Cleaner) (如图 5.2 所示 www.hivelogic.com/urlcleaner.php) 能够修复那些脚本语言，如 Cold Fusion 生成的错误的 Web 链接地址。

61

应用 PHP 脚本语言的威力。Dean Allen 的 Refer 程序 (www.bext-lan.com/tools/refer/) 能够获得来访者访问你站点前访问的站点链接，对于所有想使用它的设计者和开发者都是免费的。

Time	Date	From	To
3:28	2010-05-11	192.168.1.100	192.168.1.100
3:29		192.168.1.100	192.168.1.100
3:30	2010-05-11	192.168.1.100	192.168.1.100
3:31	2010-05-11	192.168.1.100	192.168.1.100
3:32	2010-05-11	192.168.1.100	192.168.1.100
3:33	2010-05-11	192.168.1.100	192.168.1.100
3:34	2010-05-11	192.168.1.100	192.168.1.100
3:35	2010-05-11	192.168.1.100	192.168.1.100
3:36	2010-05-11	192.168.1.100	192.168.1.100
3:37	2010-05-11	192.168.1.100	192.168.1.100
3:38	2010-05-11	192.168.1.100	192.168.1.100
3:39	2010-05-11	192.168.1.100	192.168.1.100
3:40	2010-05-11	192.168.1.100	192.168.1.100
3:41	2010-05-11	192.168.1.100	192.168.1.100
3:42	2010-05-11	192.168.1.100	192.168.1.100
3:43	2010-05-11	192.168.1.100	192.168.1.100
3:44	2010-05-11	192.168.1.100	192.168.1.100
3:45	2010-05-11	192.168.1.100	192.168.1.100
3:46	2010-05-11	192.168.1.100	192.168.1.100
3:47	2010-05-11	192.168.1.100	192.168.1.100
3:48	2010-05-11	192.168.1.100	192.168.1.100
3:49	2010-05-11	192.168.1.100	192.168.1.100
3:50	2010-05-11	192.168.1.100	192.168.1.100
3:51	2010-05-11	192.168.1.100	192.168.1.100
3:52	2010-05-11	192.168.1.100	192.168.1.100
3:53	2010-05-11	192.168.1.100	192.168.1.100
3:54	2010-05-11	192.168.1.100	192.168.1.100
3:55	2010-05-11	192.168.1.100	192.168.1.100
3:56	2010-05-11	192.168.1.100	192.168.1.100
3:57	2010-05-11	192.168.1.100	192.168.1.100
3:58	2010-05-11	192.168.1.100	192.168.1.100
3:59	2010-05-11	192.168.1.100	192.168.1.100
3:59	2010-05-11	192.168.1.100	192.168.1.100

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 可以同时运行在 Windows 和（大多数）的 UNIX 平台上。Apple 公司基于 UNIX 的 Mac OS X 操作系统也包含着 PHP 和 Apache 服务器，就像大部分 Linux 软件分发版本。



图 5.2

Dan Benjamin 的免费的 URL 清洁器 (www.hivelogic.com/url-cleaner.php)，用 PHP 开发，能够修复错误的 Web 地址链接

Microsoft 公司的动态服务页面技术，也称为 ASP (www.asp.net)，还有 Macromedia 公司的 ColdFusion (www.macromedia.com/software/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。在这三个主要的服务器端脚本语言中，只有 PHP 是免费的。一般的惯例是：ASP 经常被部署在一个全部为 Microsoft 产品的开发环境中，ColdFusion 一般和 Macromedia 的开发工具（包括 Dreamweaver 和 Flash）的集成使用，而 PHP 常常是那些叛逆者和独立开发人员的选择，然而惯例往往不一定正确。

另外一方面，类似 Yahoo 的业界领导者开始采用 PHP (www.theopenenterprise.com/story/T0E20021031S0001)，这说明了这个语言是强健和扩展性的，并且它的“免费”对于那些大公司也跟小公司一样具有吸引力。Yahoo 转向 PHP 和其他开放源码工具，也证明了：把通用惯例应用于像 Web 开发这样的快速变化的领域上往往是无效的。

上述三个主要的脚本语言都已经广泛应用于大大小小的网站。每种语言都有自己的优点，而且他们都能拥有大量的狂热爱好者。比较和评论这三种语言已经超出了本书的范畴。应该注意的是，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。ColdFusion 比较特殊，它好像违反这个长期存在的标准。这个可以通过使用 ColdFusion 中名叫 URLEncodedFormat（）的函数来解决。ASP 同样有个类似的

函数，它叫 `HTMLEncode`。通过这两种方式在输出 URL 之前传递函数，开发者们可以（也应该）能够避免这个问题。

Java 服务器页面 (JSP) 是另外一个动态技术，也是一个在大型企业中广泛使用的技术，它已经远远超出了本书的范畴。

5.1 垃圾代码的可耻秘密

我们在 Web 设计领域越成功，从事工作的时间越长，就越少注意到劣质代码造成的损失。在这个行业的前十年内，Web 设计就像是给满满一屋刚学会走路的都需要精心照顾的孩子喂饭一样。为了构造一个能够很好工作的站点，我们就必须给每个浏览器不同口味的食物。当今的浏览器都吃同样有营养的食物，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、牙齿腐烂，还导致那些大吃大喝的人们缺乏活力。同样，垃圾代码标记会经常地损害那些短期需求的用户和你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“99.9% 的网站都是过时的”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来代替那种认为 Web 代码只是个二流设计工具的想法。同时，我们还要学习一下 XHTML，这个为创建 Web 页面设计的标准语言，讨论它的目标和好处，并且探讨从 HTML 转化到 XHTML 的策略。

一个奇怪的事情就是，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 过渡方法中，如此的处理方法只是“不赞成的”，这就是说，如果必须使用，那么你可以使用。但是它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，严格地说，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“校验服务”（如图 5.3 所示）（如果你现在还不熟悉，那么你就要学习基于标准进行设计和构建 Web 站点，因而它会变得让你很熟悉的，请看阴影部分，“进行校验”）。

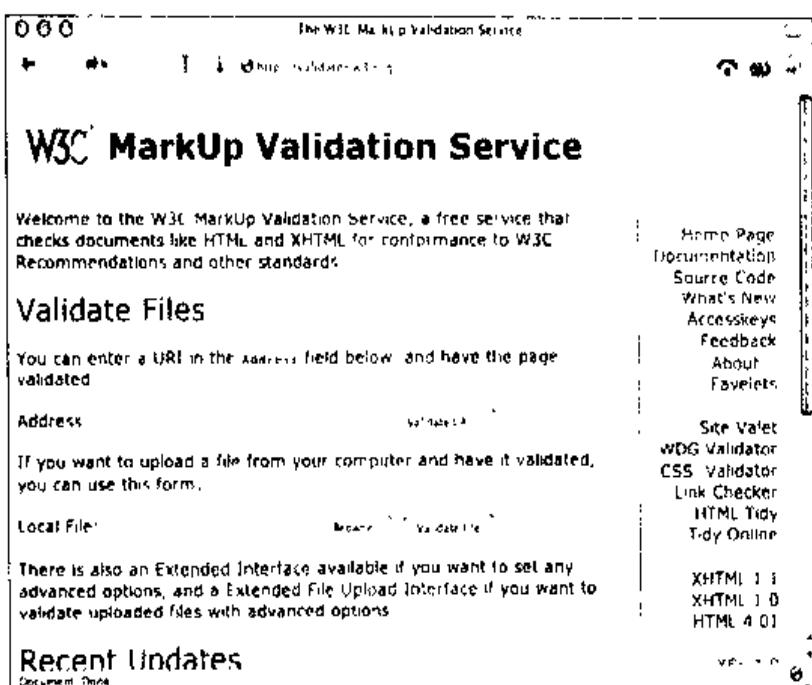


图 5.3

W3C 的免费校验服务 (<http://validator.w3.org/>) 正在被成手力的设计师和开发者们应用，以确认他们的站点能够符合 Web 标准。我们是否提过这项服务是免费的。

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/>)。所有这三种服务都是免费提供的。我们在第 2 部分“设计和构造”整个篇幅中会对它们及一些其他的工具（通常也是免费的）进行解释。

不论你选择 XHTML 严格的方式还是过渡方案，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 (`
`) 被您雪花般地撒开来用来模拟一个列表；`<h>`则被用来作为标题等级划分；透明的 GIF 图形，则用来生成空格 (`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，而着重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的颜色和对齐属性，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！

5.2 重新阐述了什么

根据 W3C 的官方解释,“XHTML (<http://www.w3.org/TR/xhtml/>) 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 的置标语言,并且看起来和 HTML 有些相像,只有一些小的但却重要的区别。虽然有些成熟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“XHTML: 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 类似。对于设计师和开发者,用 XHTML 设计 Web 和用 HTML 也基本一样。可能两者有些规则不同,并且 XHTML 还有一两个我们下面要讨论的新元素。

在第 4 章“XML 征服世界(和其他 Web 标准成功案例)”,我们讨论了 XML,也叫扩展置标语言 (<http://www.w3.org/XML/>),一个“超级”的、可以被程序员用来定制置标语言的语言。XHTML(扩展超文本置标语言)就是这样一种用 XML 定制的语言。XHTML 1.0 是最早的并且向前兼容的 XHTML 版本,从此对于浏览器和其他用户代理,有了容易学习的、最大兼容、最少麻烦的版本。

其他基于 XML 的应用程序和协议是非常多的,它们的流行部分是因为它们的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。类似的协议包括可伸缩矢量图像 SVG (<http://www.w3.org/TR/SVG/>)、同步多媒体集成语言 SMIL (<http://www.w3.org/TR/REC-smil/>)、简单对象访问协议 SOAP (<http://www.w3.org/TR/SOAP/>)、资源描述框架 RDF (<http://www.w3.org/RDF/>),以及私密标准平台 P3P (<http://www.w3.org/TR/P3P/>)。

所有这些标准(还有很多其他的标准)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的重要,而且没有一个是非常容易学习的。

为什么要用 XML 或者其他技术来“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,即使页面中 HTML 告诉浏览器不要结束表格元素,一些浏览器还可能拒绝继续显示这个页面。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，基于 XML 的语言和工具是目前 Web 表现的“金币”和通向未来的钥匙。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 是那么重要，为什么还要创建一个工作方式类似于 HTML 的、基于 XML 的置标语言 XHTML 呢？因为虽然 XML 是非常强大和深入的，但现在原始的 XML 数据还不能服务大多数 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，如图 5.4 所示。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（大多数）的简单特性。

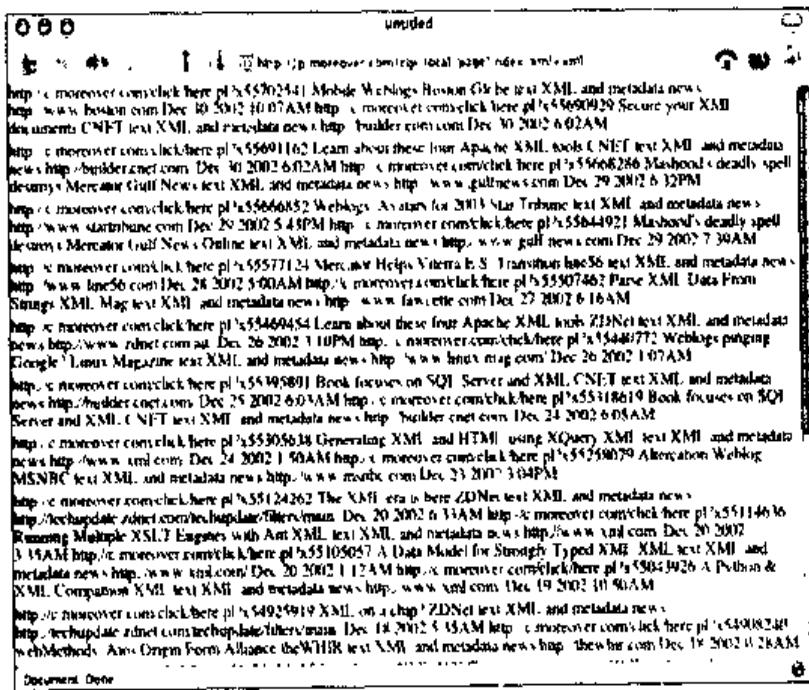


图 5.4

显示在一个现代浏览器中（这里是 Chimera）的一个 XML 样本文档 (http://p.moreover.com/cgi-local/page?index_xml+xml)。一些浏览器把 XML 显示为文本，另外一些则认为它是加上标签的文本。没有特别好办法。与之相比，实际上所有的浏览器和设备都能够知道怎么处理 XHTML，这样也使得它更有用处和更强大。

5.3 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，它可以显示在旧的和新的 Web 浏览器中，同样可以工作在大多数的 Internet 设备中，比如从 Palm 的 Pilots 到 Web 电话，到屏幕阅读器，这使得它变得轻便、实用和有效率。

XHTML 就像 HTML 一样容易学习和使用，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 年代就开始坚持 Web 设计和开发的老手稍微有点困难。

XHTML 是当前置标语言的标准（用来替代 HTML 4），而且它用来为 Web 内容设计严格的、合理的文档结构，并且能够和其他类似 CSS 和 DOM 的 Web 标准进行协作，还有能够很好地和其他现有的和未来的基于 XML 的语言、应用程序和协议进行交互。之后我们会简短地列出 XHTML 的很重要的好处。

5.4 什么样的 XHTML 适合您

在本章和贯穿本书介绍的都是我们会关注于 XHTML 1.0 和着重于 XHTML 1.0 的过渡技术，这种过渡能力是 XHTML 1.0 最大的好处，与现有开发方式有最好的兼容性，并且非常容易学习和进行转化操作。

许多标准的追随者们喜欢 XHTML 1.1 更严格的标准，虽然 XHTML 1.1 没有任何问题，但是与 1.0 有些不兼容，并且它使用了一个 MIME 类型可能会引起一些目前流行的浏览器进行错误处理。另外，从老页面转化到 XHTML 1.1 会比转化到 XHTML 1.0 需要花费更多工作，并且需要更多的重新考虑。所以对于本书大多数的阅读者来说，向 XHTML 1.0 过渡可能是近几年的最好选择。

在撰写本书的时候，XHTML 2.0 的规范草案已经提交给开发社团进行讨论。当它能够大范围流行的时候，它将把目前的 Web 开发方法推向更完美和更深入的境地。XHTML 2.0 故意设计成不向前与 HTML 或 XHTML 1.0 兼容。它放弃了一些常见的转化，包括 IMG 元素（用 OBJECT 来替代）、
标记（使用新的 LINE 元素来代替），还有历史悠久的 anchor 链接（会提供一个新的叫 HLINK 来取代）。在本书摆在书架上卖的时候，这些特性可能会改变，也可能会成为正式标准。

一些开发者欣喜地推崇 XHTML 2.0 的规范，然而另外一些则对它嗤之以鼻并且嘲笑之，其他一些则采用“等待和观望”的策略。还有很多 Web 专家则不关心这些，而是还在琢磨 Dreamweaver 中一些应用性选项。

现在还需要看看这些规范中有多少能变成最终的推荐标准。还有需要看看开发者和浏览器厂商是否会围绕着并坚持 XHTML 2.0 或者忽略它。因为这个提议的规范和最终规范还很遥远，并且当前没有任何浏览器支持它，它的出现相当有趣，并且和本书及我们的工作没有任何关系，这里我们再次推荐使用 XHTML 1.0。

最后，如果你觉得 XHTML 2.0 规范不向前兼容性会使你觉得 XHTML 是否又会有向后兼容问题的话，你应该知道现在还没有浏览器和设备厂商有放弃支持 XHTML 1 的想法。同样的道理，尽管我们在本书的第 1 章对 HTML 4 有些抱怨，但是也没有任何浏览器厂商不支持 HTML 4（在第 1 章中，我们抱怨过一些劣质的标记和不动脑筋的脚本，而不是任何 Web 标准，而 HTML 4 也是 Web 标准，只不过是一个过时的标准而已）。正确使用 HTML 4.01 规范制作的站点在未来几年内还是能正常工作的，那些用 XHTML 1 制作的网站也同样。在这两个标准（HTML 和 XHTML）之间进行比较，我们在下面总结了 10 个关键点。

5.4.1 切换到 XHTML 的 10 个最主要的理由

- (1) XHTML 是当前用来代替 HTML 4 置标语言的标准。
- (2) XHTML 是为和其他基于 XML 的置标语言、应用程序以及协议进行良好的交互工作的，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) XHTML 1.0 是通向 XHTML 未来版本的一座桥梁。XHTML 2 的标准草案到达推荐的状态了，想转化到它的话（如果你准备这么做），从 XHTML 1.0 会比从 HTML 转化更容易。
- (5) 老的浏览器能像 HTML 一样适应 XHTML，从这点来看，XHTML 没有什么有优势的地方，但是它也没什么缺点。
- (6) 新的浏览器都喜欢支持 XHTML（特别是 XHTML 1.0），并且很多都给 XHTML 特殊的待遇而不是给 HTML 4。这使得 XHTML 在很多情况下比 HTML 更有前途。
- (7) XHTML 能够很好地在无线设备、屏幕阅读器和其他特殊的用户代理上工作，就像它能够运行在传统的桌面浏览器上一样，在很多情况下 XHTML 可以使得我们不必为站点创建多个特殊的无线置标语言的版本（wireless markup versions），这样既可以吸引更多无线网络的浏览者，并且还可以付出较低的成本。我们并不肯定这种现象的起因和结果，但是的确很多 HTML 网站经常对于他们要同时处理无线网页版本、纯文本的版本，以及友好打印的页面版本感到沮丧，而大多数的 XHTML 网站却能够摆脱这个问题，他们只需要一个文档就能服务到所有用户（大多数情况下，当对这些多媒体信息进行适当的样式化，并且和他所采用的 CSS 适当的情况下一个文档才能服务所有人）。
- (8) XHTML 是 Web 标准家族（还包括 CSS 和 W3C 的文档对象模型（DOM））

中的一个部分，可以用来在跨平台、浏览器和设备上控制 Web 页面的行为和外观。

(9) 用 XHTML 可以帮助你去除书写表现层代码的坏习惯，并且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(如果你使用结构化的 XHTML 并且把它所有的或者大部分的显示采用 CSS 控制，你再也不用担心你的页面在 Netscape 和 Microsoft 的浏览器上产生差异，比如说添加了宽度的空的表格单元)。

(10) 使用 XHTML 技术还可以帮助您养成靠标记校验服务来测试页面工作的习惯，它可以减少您测试和调试的时间，从而更容易避免一些可访问性的错误，比如给所有的标记忘记了 alt 属性问题。如果想知道更多关于可访问性的信息，请看第 14 章“可访问性基础”。

5.4.2 不要切换到 XHTML 的最主要的原因

- (1) 你的老板钱多到无所谓的程度
- (2) 你喜欢针对任何可以想到的浏览器和设备创建多版本。
- (3) 你脑袋里面的小人告诉你不要去做(西方认为人脑里有一个天使、一个恶魔，分别代表人的善良和邪恶本性)。
- (4) 你不准备继续 Web 上的商务开发。
- (5) 你不知道 XHTML 的规则。

幸运的是，对于第 5 条原因，我们还是能做一些工作的，可以看看下一章。

XHTML: Web 重构

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？因为在你掌握了这一章简单、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。不是说你将用新的标签代码代替过去几年的，我们的意思是你会以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。

“除了炫目的 Flash 外，惟一的真实有效方法”是我们放弃的另外一个标题，因为看起来太复杂了。而这一章说的是 XHTML 是如何真正地将 Web 重新结构化的。所以，最后就用了“XHTML：Web 重构”这个标题。

在这一章里，我们将学习 XHTML 的入门知识和探究表现标记和结构标记之间的机制和关联。如果你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，当他们打开第 6 章的“知识宝箱”，他们会惊奇地颤抖。是的，说颤抖也许夸张了一点，你可能只是说：“嘿，这些对我来说是新知识”。

6.1 转换到 XHTML：规则简单，容易上手

只要你稍加注意一下，规则简单，容易上手，就会发现从传统的 HTML 转换到 XHTML1.0 是快速和平滑的（我们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 DOCTYPE 和名字空间（Namespace）

XHTML 文档刚开始的一些元素告诉浏览器如何解释它们及校验服务——如何一致地校验它们。所有这些一开始的元素就是 DOCTYPE (“document type”的简写) 声明。这些便利的少量的元素信息用来说明你用的 XHTML 或者 HTML 是什么版本。DOCTYPE 总是写在所有标记的前面，这原因只有 W3C 委员会成员才知道。

为什么要一个 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 声明校验服务，现代浏览器根据你定义的 DTD 来描绘你的标记。在转换过程中，这些信息告诉那些校验服务和浏览器如何操作页面。

DOCTYPE 声明是一个兼容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，否则你的标记和 CSS 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。如果你没有注意，结果会让你大吃一惊。在第 11 章 “DOCTYPE 转换和标准模式” 中，我们将解释 DOCTYPE 对不同浏览器的影响，包括 IE 和基于 Gecko 的浏览器，如 Netscape，Mozilla 和 Camino。

XHTML 1 提供了 DTD 的三种选择和三种可能的声明：

过渡的 (Transitional) —— 最宽松的 DTD，它宣称的目标是“自己活也让别人活”(看下一节“哪一种 DOCTYPE 适合你”)

严格的 (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。

框架的 (Frameset) —— 20 世纪 90 年代最流行的布局方式，同样地，也可以在你的设计中使用。

哪一种 DOCTYPE 适合你

上一节我们介绍了三种类型的 DTD，其中过渡期 DTD 是最接近于我们熟悉和喜爱的 HTML 的，就是说，它是唯一的还容忍表现层的标记、垃圾元素和属性的 DTD。

HREF 链接标签的 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，或者你不想这么做但你的客户却坚持——过渡期

DTD 是唯一允许你使用目标属性的 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 上, 要在新窗口打开一个链接, 你需要写 JavaScript, 你也需要确认链接能在不支持 JavaScript 环境下工作。不管你是否要在新窗口打开链接, XHTML 过渡方法会给你最低限度的惊讶。

XHTML 1.0 过渡方法同时容忍在表格单元使用背景颜色, 以及类似地, 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)。与之对比, 如果你把它声明为 XHTML 1.0 过渡型 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。

简而言之(呵呵, 也许已经说太多了), 对于那些正在向现代浏览器过渡的设计师来说, XHTML 1.0 过渡方法是理想的 DTD。

“对于过渡期的设计师和开发者, XHTML 1.0 Strict 是最好的选择”这种说法会引起争论, 就像“加入海军可以摆脱牢狱之灾”的争论一样。从臃肿的 HTML 4 跳跃到严格的 XHTML 1.0 Strict, 标记是为了结构而不是视觉效果, 这也许就是你正确的选择。但是为了这章的目的, 我们将使用 XHTML 1.0 过渡方法, 下面是 DOCTYPE 声明。

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

Frameset DOCTYPE 是用在那些使用<frameset>元素的文档上的。实际上, 你必须在<frameset>文档里使用这样的 DOCTYPE。

DOCTYPE 声明必须放在每一个 XHTML 文档最顶部, 在所有代码和标记之上, 放在<head>元素、<title>元素和 meta 元素, 以及样式表和 JavaScript 链接之前。当然, 也在你的内容代码之前。简而言之, DOCTYPE 声明在所有代码之前。

紧随 DOCTYPE 其后是 Namespace

紧跟在 DOCTYPE 声明之后是一个 XHTML 名字空间 (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 的一个名字空间是收集元素类型和属性名字的一个特定的 DTD，名字空间声明允许你通过一个在线地址指向来识别你的名字空间，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。

使用 DOCTYPE 和名字空间后，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 或者 [Webstandards.org](http://webstandards.org) 查看源代码，拷贝和粘贴到你的文档顶部。

6.1.2 声明你的内容类型

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编码语言，可能是 Unicode、ISO-8859-1 或者你需要使用的。

如果你对编码语言不太熟悉，或者 ISO-8859-1 不能正常工作，不用担心。我们将在这第一章的后面讨论这个问题（看后面的章节：“编码语言：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 XML prolog 开始，就像一个 XML 声明。当使用 prolog 时，它放在 DOCTYPE 和名字空间声明之前，它的作用是指定 XML 的版本和声明编码语言。

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

```
<?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，编码语言是 ISO-8859-1。惟一的新知识就是使用了一个“?”标记开始和关闭。

不幸的是，许多浏览器，甚至来自 W3C 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。

事实上，当站点不能正常工作，浏览器本身并未受到处罚，忍受痛苦的是你的访客。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，也许会使浏览器出错。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/Windows 遇上 prolog 会发生的事情）。

幸运的是，有一个解决方案。在棘手的 prolog 的地方，你可以在文档中的 `<head>` 标签中插入一个 Content-Type 元素来指定编码语言。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" />
```

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，你可以使用 Unicode 并在你的标记中，插入如下的 Content-Type 元素。

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

同样地，你可以访问 zeldman.com, alistapart.com 或者 webstandards.org 查看源代码并拷贝粘贴到你的文档中。

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拷贝到他们模板的顶部。但他们看起来也非常快乐地活着，创作着。

从一个令人讨厌的标题开始，简单涉及一些简短易懂的页面，你已经通过了这一章。恭喜你！可以休息一会儿，吃点蛋糕吧。

6.1.3 用小写字母书写所有的标签

不像 HTML，XML 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 W3C 和设计团体提供的免费标记校验服务，请看第 5 章）。

让我们来看一个典型的 HTML 元素：

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

你将重新定义 TITLE 元素，也将重新定义保密协议页面，不然除了你的法律部门没有一个人能够阅读到这份页面。将这个元素转换到 XHTML 非常容易，就像切换大小写的按钮：

```
<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 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。

流行的 HTML 编辑器，像 BareBones BBEdit，Optima-Systems PageSpinner 和 Allaire HomeSite 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：

```

```

```

```

```

```

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。

要小心使用大小写混杂的属性名字，如果你使用像 Macromedia Dreamweaver 的所见即所得编辑工具，或者使用像 Adobe 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`，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"
```

Tidy 时代

到目前为止，建立有效的 XHTML 页面最简单的方法是手工直接写代码。虽然常常对老页面充满感情，但它们真的需要重新设计了。找一个合适机会开始重新设计，你不得不手工移植到 XHTML。免费工具 HTML Tidy（如图 6.1 所示）能够帮助你快速地将 HTML 转换到有效的 XHTML。

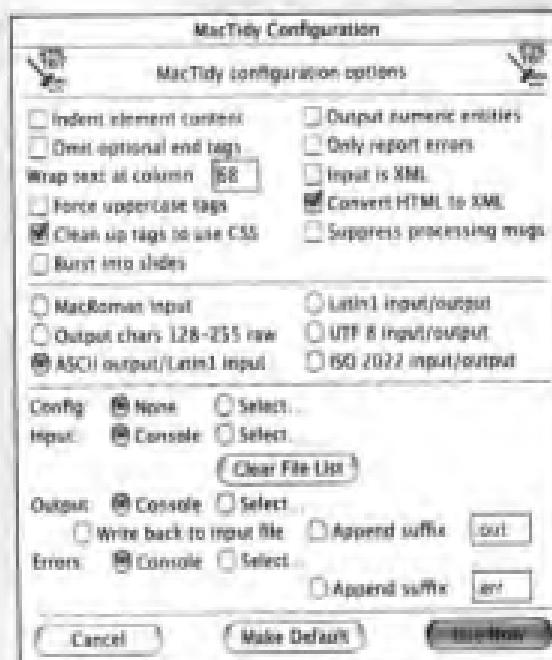


图 6.1

它非常可爱，价格更可爱。免费的 HTML Tidy (<http://www.w3.org/People/Raggett/tidy>) 能把你的页面从 HTML 特化成 XML。注意设置：把 HTML 转换到 XML。它体贴的是，Tidy 为每一个操作系统准备了不同版本（下面显示的是 Mac OS X 版本）。

Tidy 是“标准发烧友”Dave Raggett 开发的，现在作为一个开放源代码软件被 Source Forge (<http://tidy.sourceforge.net/>) 维护。虽然个别人由于兴趣建立了独立的版本，比如 Terry Teague (http://www.geocities.com/terry_teague/tidy.html) 开发了 Mac OS 版本（如图 6.1 所示）。

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件。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的许多时间。

6.1.4 给所有属性值加引号

在 HTML 中，你可以不需要给属性值加引号，但是在 XHTML 中，它们必须被加引号（例如，height=“55”，而不能是 height=55）。

Ok，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，你应该这样做：

```

```

如果你使用 escaped 字符来避免双引号和单引号次序混乱，还可以这样做：

```

```

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" />
```

注意：W3C validator 不得不同时处理 HTML 和 XHTML，它的解析器将无法识别这样的错误。任何基于纯 XML 的解析器将会捕获这种错误。

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：

```

```

如果你需要在属性值内含一个单引号，可以使用‘’，就像这样：

```

```

给属性值加引号在 HTML 是随意的，但是我们很多人还是使用它，所以在

转换到 XHTML 时候不再需要再做处理。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，这些站点将不得不重新加上这些引号。

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。

6.1.5 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>
<input type="checkbox" name="shirt" value="medium" checked>
<hr noshade>
```

必须被赋予一个值，属性值必须和属性名称一样。

```
<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，但还是养成习惯去做吧。

6.1.6 关闭所有的标签

在 HTML 中，你可以选择打开许多标签，如 `<p>` 和 ``，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图中一样地工作。

6.1.7 空标签也要关闭

在 XHTML 中，甚至“空”标签也要关闭，如 `
` 和 ``。在标签尾部

使用一个向前的斜杠“/”来关闭它们自己：

```
<br />

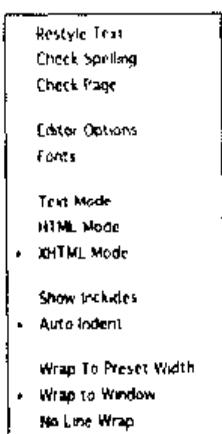
```

注意不要拉掉在 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，没有火箭科学那么深奥。

这并不需要很大的工作量，近来的编辑器，像 BBEdit、PageSpinner 和 HomeSite，如果你告诉它们你将工作在 XHTML（如图 6.2 所示）中，编辑器都会自动在空标签后面加上空格和 “/”。可视化 Web 编辑工具（如 Dreamweaver 和 GoLive）也同样这样做。

图 6.2

打开 Optima 系统的 PageSpinner 编辑器的下拉菜单，设置为 XHTML 模式，它会自动使你的元素名称和属性值变成小写，并且用空格和斜杠关闭空标签（www.optima-system.com/pagespinner/）



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包含一个 alt 属性，一个可选的 title 属性：

```

```

现在，这就是好的 XHTML。

6.1.8 不要在注释内容中使 “--”

-- 只能发生在 XHTML 注释的开头和结束，也就是说，在内容中它们不再有效。

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

6.1.9 把所有<和&符号编码

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

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

必须被标记为这样：

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

当遇到没有编码的符号，W3C 校验服务将用一个警告信息让你离开；而纯 XML 解析器将产生一个致命的错误。

注意：即使你从来没有这样编码过，我们也要推荐用>来编码>，这样可以使其他人更容易阅读。

让我们复习一下已经学过的 XHTML 规则。

6.1.10 执行概要：XHTML 的规则

- 以正确的 DOCTYPE 和名字空间开始
- 使用 META 内容元素声明你的内容编码语言
- 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

看上面执行概要的列表，XHTML 的规则看起来足够少和简单。事实上也是少而简单。

6.2 语言编码：无趣，很无趣，真的无趣

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。

6.2.1 Unicode 和其他编码设置

XML、XHTML 及 HTML 4.0 文档默认的编码设置是 Unicode(<http://www.w3.org/International/0-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 是一个全面的编码设置，“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。

即使 Unicode 是 Web 文档的默认编码，开发者也可以自由地选择其他编码设置，也许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，或者它来自哪里？说实话，你没有问本书这个问题，但本书需要一个跳转，那是我们转移话题的最好方法。

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）图形等字符集。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 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) 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！

将你的字符集映射到 Unicode

不管你选择什么编码语言，将它映射到 Unicode 标准，都必须声明你的编码语言，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编码语言：

- 一个服务器管理员可以通过 Web 服务器上返回的 HTTP 头部设置编码语言。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 头部多做一点思考。
- 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。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。

6.3 结构康复——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。这本书也将改写“豆腐烹饪方法”，比如用黑紫豆浆果做豆腐蜂蜜派。味道好极了！如果你用奶酪芝士代替豆腐就更好了，还要放糖，大量的糖，还有黄油和鸡蛋——对，不要忘记鸡蛋。

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。

6.3.1 用理性代替样式来标志你的文档

记住：为了最广泛的可访问性，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理性。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。

不是每一个站点都能够放弃 HTML 表格布局。W3C——CSS 的发明者——直到 2002 年 12 月还没有转换到 CSS。而且，即使最坚决的标准纯粹论者也不能总是将结构和表现分离开来——至少在 XHTML 1 中不行。但是，将结构与表现相分离（数据与设计相分离）是一个理想的趋势。我们就能够从混杂的过渡型的布局中大步前进，获得好处。下面是一些技巧，帮助你开始在结构上做更多思考。

6.3.2 大纲里的颜色

在语法学校里，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，我们可以抛弃大纲的呆板的框框限制，大胆投入到一个个性表达的领域。（也许我们的小册子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 前，我们不能提交这样受欢迎的布局，但是今天，我们只要基于文档结构就可以提交良好表现，而且不用支付额外的设计费用。

当我们编写页面文本，或者转换已经存在的文本文档到网页时，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``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

一些人在写标记时，已经养成习惯，希望字体大点，就用`<h1>`，希望在前面加个点符号，就用``。就像在 3.4.1 节中讨论的，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，``的意思是圆点，`<blockquote>`的意思是“缩进排列文本”。我们大部分人都乱写 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>
```

我们必须抛弃幼稚思维，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体（标准粗细），`<p>`文本能够变成巨大的、粗体的，``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。

从今天开始，我们要使用 CSS 来确定这些元素的外观。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。``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。

CSS 从结构中完全抽象出表示，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，所有标题的 6 个级别 (`h1~h6`) 可以看起来一样，如果设计师需要这样设计：

```
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、无线设备和邮件新闻组中保留文档结构。

在 XHTML 章节中展示的 CSS 技术，并非想说明我们超越了自己，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。结构元素应该只用于传达结构，而不要强迫用于显示。

宁应用结构化元素代替无意义的垃圾

因为我们已经忘记，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。我们很多人已经习惯写标记时不含结构。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 />
```

考虑使用一个命令或无序列表代替：

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

你或许会说“但是``显示的是一个圆点，我不想用圆点”。在前面的章节提到过，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。

所以使用`list`元素来作为列表的标记，同样，宁愿用``代替``，用``代替`<i>`等。默认情况下，多数桌面浏览器将把``当做``显示，把``当做`<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``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`<i>`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``意味着什么呢？）。此外，表现导向的元素，像``和`<i>`，很可能在将来的 XHTML 版本重蹈标记的覆辙。使用结构化的元素替代它们是安全的。

6.4 视觉元素和结构

“Web 标准”坚持的不仅仅是使用的技术，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 可能像早期的 Web 技术一样被滥用。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。罗嗦一句，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。

在前面 6.3 节中的指导能帮助避免结构过度复杂。但是我们对有关品牌视觉元素该做些什么呢，例如站点导航条，典型地包含一个标志。这些元素能结构化吗？它们必须结构化吗？

第一个问题的答案是“是”。包含导航条的视觉元素能真正地被结构化。例如，在前面的章节中提到过，CSS 能制造一个有序或者无序的列表，如一个完整

的、完全用 push 按钮和鼠标滑过效果的导航条显示。

第二个问题的答案是：像导航条这样的视觉元素在混杂的、过渡型布局中不需要被结构化。如果那些布局在它们主要的内容区域避免了冗长，使用好的结构；如果他们的 XHTML 和 CSS 都是有效的，每一个努力都使网站更容易接近，那么你已经完成了向标准的过渡，没什么可羞愧的（嘿，也许你在其他方面有点羞愧，但肯定不是你建造网站的方法。）。

在下一章，我们将比较好的代码书写和差的书写之间的区别（不管那个差的书写是否使用 XHTML 和 CSS），并讨论如何在非结构化的、混杂的、过渡的布局中创建好的、干净的、简洁的标记。

紧凑而坚固的页面保证：以严格 和混合的标记组成的结构

不管怎样，你都不要跳过这个章节。这个章节的内容会大大提高你的技能，讲述怎么减少你的 Web 页面中不需要的冗余代码，以及怎样能更加深你对标记和设计之间差异的理解。本章的思想非常容易理解，但是同时兼顾提高站点性能和设计、生产品和升级网站的简便性。

在本章，我们会展示怎么书写合理的、紧凑的代码，从而用来降低 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。我们通过把表现层的代码从 XHTML 页面中去除来达到这个目的，并且将学习怎么避免很多对站点不好的普遍的行为习惯。

这些不好的行为习惯害苦了很多在 Web 上的站点，特别是那些把 CSS 与表格布局混合在一起的站点。这些做法都是毫无意义而且很不恰当的，甚至是一些非常熟练的设计师也会犯这样的错误，更何况其他人。在手工编码的站点或者是采用可视化工具（如 Dreamweaver 和 GoLive）帮助下生成的站点中，都同样存在这个相同的问题。

在本章，我们将会对这些普遍的错误做法进行命名，从而使您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。我们还将和独特的标记符 (`id`) 属性及基于标准设计的“即时贴”交上朋友，而且还将展示它怎么可以让你书写极度紧凑的 XHTML，而不管你采用的是混合的，还是纯 CSS 的布局。

7.1 所有的元素都必须是结构化的吗

经过第 6 章“XHTML：Web 重构”的讨论，我们知道如果在过渡站点中的

导航元素常常是非结构化的，也是被认可的，一些组件并不需要结构化。但是，在 CSS 布局中更多引入类似的非结构化元素是危险的。在第 6 章我们所讨论的“标题，段落，目录”则是可以结构化的。

这些导航元素是仍然可以被结构化的。也就是说，它们可以通过通用结构化元素和特殊的结构化属性，使这个设计中最大的角色结构化。这并不是为了理论而去做，而是为了获得减少带宽占用和维护站点的现实利益。

在 zeldman.com 的每日报告（如图 7.1 所示）中用了 CSS 布局，用了 XHTML 结构化。并且它的代码里面几乎所有的元素都是结构化的，包括从列表到段落到你能想到的表示地址的地址标记。

图 7.1

在 zeldman.com 中的每日报告中用了 CSS 布局，用了 XHTML 来结构化。在顶层的导航图像都是非结构化的，是吗？



在第 6 章描述的场景中，这个页面顶层的导航图像是非结构化的。它们仅仅是带链接的图像，并且放置于 XHTML 的块级别元素（div）中，一些灵活的设计师会对这个元素赋予一个唯一的标志（id）。接下来我们会定义这些术语，并要看看这些组件集成起来创建一个结构化的站点，而且还能减少页面大小，以及不需要对表现层代码进行重新排序就能控制布局。

7.1.1 div、id 和其他助手

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 XHTML，更明智地使用 CSS，可以通过标准文档对象模型（DOM）

对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”中定义了这些元素和其他两个重要的 HTML/XHTML 元素：

DIV 和 SPAN 元素，联合 id 和 class 属性，提供了一个把结构添加到文档的通用机制。这些元素定义了内置的内容（SPAN）或者块级别（DIV），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并体验 (<http://www.w3.org/TR/RFC-htm140/struct/global.html#h-7.5.4>)

什么是 div

用短语“division”解释“div”是最恰当不过的。当你把一组链接放在一起，这就是文档的一块，正文也可以是一块，页面底部的法律声明又是另外一块，等等。

特殊结构的通用机制

所有的 HTML 开发者都对一些普通的标记很熟悉，比如段落标记 p 或者 h1，但是对于 div 可能就不太熟悉了。若要理解 div 元素，W3C 中的一个短语，即“一个增加结构的通用机制”是关键。

在 zeldman.com 的例子中，主导航图像放在一个 div 中，因为它们都不是一个段落、列表或者任何其他结构化元素的一部分。但是在一些大型的或者更特殊的情况下，这些图像要扮演一个结构化的角色，即主导航条。为了理解这个角色，包含这些图像的 div 被赋予了一个名为“primenav”的惟一标志，它是“primary navigation（主导航条）”的缩写。

你可以使用任何其他名字。“Gladys”或者“orangebox”等，都是很好的，是 XHTML 规则所允许的。但是有意义的名字（比如表达所包含的元素的功能）是最好的。当您的客户想看到一个蓝色，而你却把这部分命名为“orangebox”是不是很奇怪。如果从现在开始，只有 6 个月时间重新修订你的样式表，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，同样，您会感到原来的做法很愚蠢。

另外，精心构思的结构化 id 标签，如“menu”或者“content”，或者“searchform”可以帮助你记住什么代码还没有设计和创建您期待的良好结构的页面。不管你使用纯 CSS 布局还是混合的 CSS/表格布局，如果你养成了习惯给一些页面核心组件（比如导航、内容和检索区域）命名，你就会抛弃从页面表现层进行思考和开发的习惯了。

id 与 class

`id` 属性对于 XHTML 并不是新的，同样，`class` 属性以及 `div` 和 `span` 元素也不是。它们都是 HTML 原来的元素，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` 为“`content`”，就不可能有另外 `div` 或者其他元素拥有同样的名字）。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`）。下面的代码能够帮助你弄清楚 `id` 和 `class` 的区别：

```
<div id="searchform">Search form components go here.</div>
<div class="blogentry">A web log entry goes here.</div>
```

在这些例子中，`div id="searchform"` 用来标出页面中包含检索表单的区域模块，而 `div class="blogentry"` 则用来指出网站日志的每一项（我们所熟知的 `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。如果这个 Web 网站日志可以不通过 `div` 就能实现——比如可以使用原始的标题和段落结构来代替，这个效果会更好。在 `zeldman.com` 中的每日报告就调用了一个 Web 网志（尽管不是我们所制作的），它的日志条目都是采用普通的 `h3`、`h4` 以及 `p` 元素进行标记的。

即时贴理论

把 `id` 属性设想成一个即时贴，有助于我们对它的理解。我们在冰箱上贴上一个即时贴来提醒我们要去买牛奶。一个放置在电话上的即时贴会提示我们要给以一个未付钱的客户通电话。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来付钱的客户）。

`Id` 属性类似于在你代码中的特殊区域的标签，会提示你哪个区域需要特殊的处理。为了执行这个特殊的处理过程，随后你可以在样式表文件中写入一条或者多条规则，或在 JavaScript 文件中书写一些代码来引用这个特殊的 `id`。例如，你的 CSS 文件可能有特殊的规则来对那些 `id` 为“`searchform`”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“`menubar`”的表格处理的规则。（如果您继续本书后面的内容，你会知道一个赋予 `id` 为 `menubar` 的表格用来作为一个菜单条。在第八章“XHTML 的示例：混合布局（第一部分）”和第十章“CSS：混合布局（第二部分）”，我们会使用类似表格和 `id` 创建一个站点。）

当一个 id 属性值被用于一个特殊设置的 CSS，它称为 CSS 选择器，还有很多其他的方法能够创建一个选择器（看第九章“CSS 入门”），但是 id 是最容易和通用的方法。

id 的强大功能

id 属性有着令人难以置信的强大功能。id 属性可以完成下列的功能：

- 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、最小化了的 XHTML 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（或者为了向前兼容而和它共存）。
- 作为从基于 DOM 的脚本中特殊元素的引用的方法。
- 作为一个声明的对象元素名字。
- 作为通常目的流程处理的工具（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，这个时候用来标明区域”）。

Id 的规则

一个 id 值必须用一个字母或者下划线开头，它不能用一个数字进行开头。W3C 的校验服务可能不会捕捉到这个错误，而一些 XML 处理器可以。同样，如果你想把 id 在 JavaScript 中以 document.idname.value 的形式使用，就必须用合法的 JavaScript 变量去命名，就是说它必须用一个字母或者下划线开头，然后可以跟随字母、数字和下划线。空格和连字符（-）都是不允许的。由于一些老 CSS 实现（在一些浏览器中）中的限制，在 class 和 id 名字中使用下划线也是个不好的主意。

最后，对于那些发烧友来说，他们会发现还可以用 Unicode 转换的数字作为一个 id 或者 class 名字的开头。但是没有人会那么做。

强大功能带来更大的责任

我们这里讨论的元素和属性经常被滥用和误用，增加页面四倍大小而不是减少它，把 XHTML 中所有结构化的信息都去除了，替代了对它的通用结构元素，并增加原架构。当你阅读完本章后，你会认识到这些滥用的形成，并且知道怎么去避免。

7.1.2 勇敢去做最少的

现在我们已经讨论了一般目的的 XHTML 元素（特别是 div 和 id），让我们看看 zeldman.com 上的例子（如图 7.1 所示）。这四个人的菜单图片被一个普通的、唯一的、结构标记为 "primenav" 的 div 所约束。

```
<div id="primenav">[Linked images go here.]</div>
```

这个方法和其他更多的类似技术的区别在于：在这个块级别的元素中不包含任何表现层的代码——没有宽度，没有高度，没有背景颜色，没有水平或者垂直对齐，等等。所有老式布局结构中的东西，在我们的块级别的元素都没有。

那么浏览器又怎么知道去哪些地方放置这些图片呢？

我们可以稍微回顾一下早前关于 CSS 选择器的讨论。"primenav" 仅仅用来作为一个选择器。在这个站点的样式表中，一个 CSS 规则定义了 "primenav" 这个特殊的块级别元素没有边缘和填充，换句话说，它周围没有被空格围绕。因为<div id="primenav">是这个站点 XHTML 源文件中第一个可视化的元素，支持 CSS 的浏览器会把它放置在页面的左上角。

"primenav" 区域中的四个菜单图片一个挨一个地排成行显示。没必要用一个表格来放置这些图片的位置。同样，也不需要对每个图片都分配一个唯一的 CSS 定位规则。在 XHTML 文件中，图片列出的顺序就是它们在页面中显示的顺序。

通过元素顺序的逻辑性，我们就避免了表格布局的混乱和冗长的样式表（参见下面关于“classitis”和“divitis”的讨论），通过一个可靠的方式显示出站点的表示层，因为它们不包含任何多余的代码或 CSS。

下面是 zeldman.com 的代码，为了更清晰演示，去除了一些 JavaScript 代码，并且一些图片和路径名字都简化了：

```
<div id="primenav">
  <a href="/"></a>
  <a href="/"></a>
  <a href="/glamorous/"></a>
  <a href="/classics/"></a>
</div>
```

注意一下图片的高度和宽度这些属性，虽然它们是有用的，但是严格意义上是不需要的。同样需要注意，感谢 CSS，边界属性也是完全没必要的，只不过是我们为了一些老浏览器而把它放在这里。如此一来，我们的代码可以更干净和简单，如下：

```
<div id="primenav">
<a href="/"></a>
<a href="/"></a>
<a href="/glamorous/"></a>
<a href="/classics/"></a>
</div>
```

比较和对比一下前面紧凑而清晰的代码和列在下面传统的表格布局版本：

```
<table border="0" cellpadding="0" cellspacing="0" width="500">
<tr>
<td valign="top" width="150" height="100" bgcolor="#339999">
<a href="/"></a>
</td>
<td valign="top" width="140" height="100" bgcolor="#339999">
<a href="/"></a>
</td>
<td valign="top" width="110" height="100" bgcolor="#339999">
<a href="/glamorous/"></a>
</td>
<td valign="top" width="100" height="100" bgcolor="#339999">
<a href="/classics/"></a>
</td>
</tr>
</table>
```

这里没有任何争论，但是紧凑的代码和结构化的思考不仅仅限于 CSS 布局，它们同样可以强化基于表格的布局。

7.2 混合布局和简洁的标记：要做什么和不要做什么

害怕学习 CSS 布局或者没有能力在一些实际项目中使用它，这不是一个逃

遵 Web 标准的理由 集成改进的表格布局和基于 CSS 的结构化的、可访问的 XHTML，你就创建了一个混合的、过渡的、可以正常工作的方法。

CSS 正在发展中，浏览器也正在学习去支持 CSS1 和 CSS2，并且在有些布局中，把一些基本元素放在简单的 XHTML 表格中会很有成效。表格不像那些过时标记一样愚蠢，即使使用了一些非结构化组件的布局，也还是要比那些 2003 年才建立的冗余代码的华丽站点优秀。

提醒

我们曾经说过，一些混合站点中的界面元素可能不是结构化的，但并不是说结构化的标记语言对这些站点是没有用的，段落还是能够标记为段落，列表也能够被标记为列表，等等。这本书讲的并不是用陈旧的方法创建菜单或者其他元素，主要讲的是，不管是否结构化，任何元素都必须由清晰、紧凑的标记和干净的 CSS 组成。简洁的标记是我们本章要讲述的内容。

7.2.1 给坏习惯命名

接下来，我们将会看看许多站点使用的很不灵活的建站方式，并且解释为什么使用这些方法达不到他们的预期效果。我们还要给那些经常使用的、难以忍受的技术坏习惯贴上特别标签。阅读本节不仅仅能够让你在工作上抛弃这些坏习惯，而且能够帮助你发现你的同事或者开发商工作中的错误。

当你阅读完本节，把它的简单规则牢记在心里后，在你的同事或者开发商试图摆脱某些愚蠢代码时，你就可以告诉他们使用本章所介绍的做醒目、合适的描述标签。你的同事或者你的开发商将对这种方式有一个全新的认识，同时对你将有一个新的认识，最重要的是，你会在他们面前树立威信。

事实上，我们为这些实际中存在的坏习惯命名并不是为了取笑某些人，这仅仅是为了标示，从我们的工作中消除坏习惯，另外，也希望它们能够帮助你。

7.2.2 在超文本标记中的普遍错误

来看一下这样一个典型的、基于表格设计的站点（如图 7.2 所示），当访问者从一个页面进入到另外一个页面时候的时候，它采用一个改变状态的菜单（如图 7.3 所示）来标明访问者的位置（如图 7.4 所示）。这个站点的 logo 很明显是一个图形元素，一般都是 GIF 图像。菜单的名称（EVENTS, ABOUT, CONTACT 等）同样可以是图像，或者它们也可以是简单的超文本链接。如果采用超文本链

接方式，老的 Web 设计师可能把垃圾代码（如 font 标记）堆到页面文件中，控制页面的文本外观（大小、外观、颜色），还会使用一些过时的属性来控制对其边界属性和包含菜单标签的表格的底色。聪明的设计师会使用 CSS 来代替，但如果错误的使用，可能效果还不如原先方式好。



图 7.2

i3Forum 会议站点的设计比较（在第 8 章和第 10 章，我们会建造这个站点）

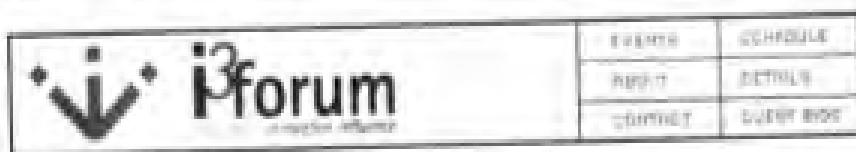


图 7.3

当访问者访问首页的时候，这个菜单就会显示出，Logo 采用白色的底色进行高亮



图 7.4

当访问者移动到 EVENTS 页面时候，EVENTS 标签就加亮了

即使采用 GIF 图像代替超文本时候，设计师也会想办法区分当前位置菜单的表格和其他菜单表格的外观。比如，这些菜单（如图 7.3 和图 7.4 所示）可能有一些细的黑边框，而其他在页面中的表格却没有边框。在一个设想的最差情况下，设计师会把这个菜单表格整个放入另外一个表格中，目的只不过是想创建一个黑的边框效果。这就是我们回溯到 1997 年所做的事情。现在的设计师可能会结合使用过度的样式表和 CSS 来区分菜单表格和其他的表格，这种方式并没有比老的方式更好。

Classitis：标记的麻疹

在这个页面中，我们怎么能够让当前位置的导航表格单元和页面中的其他表格显示得不一样呢？或者怎么让这个导航条表格单元中的链接显示的和页面中的其他链接不一样呢？而且还不能采用那些不赞成使用的表现层标记和属性，比如 bgcolor 和 font，即使您可能已经这么用过。我们不需要得到这样的代码：

```
<td align="left" valign="top" bordercolor="black"
bgcolor="white">
<font family="verdana, arial" size="2"><b>
<a href="/events/">Events</a></b></font></td> etc.
```

但我们也想在每一个需要特殊处理的元素中都附加一个 class。我们可能看过很多如下的例子：

```
<td class="whitewithblackborder"><span class="menuclass"><b>
<a href="/events/">Events</a></b></span></td> etc.
```

甚至我们遗憾地说，可能看到的是如下的例子：

```
<td class="whitewithblackborder"><span class=
"menuclass"><font class="menufontclass"><b>
<a href="/events/">Events</a></b></font></span></td> etc.
```

我们称这些标记为 classitis。在那些被 classitis 折磨的站点中，所有要处理的标记都被它自己膨胀的、垃圾 class 所包围。Classitis 就是标记的麻疹，这就意味着对任何页面都增加了不需要的内容。这个苦恼最早来自于早期对 CSS 不完全支持的浏览器和很多对 CSS 一知半解的设计师。

哎，许多设计师还没有成长起来，对 CSS 还是孩童般的误解，可能是因为他们刚好转向学习其他技术，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样式，并且他们“学习”CSS 是通过这个工具所产生的代码。Classitis 就和我们原有的标记方式一样糟糕，好的标记代码很少需要使用它。

可视化编辑器和 Classitis

即使是最好的、最成熟的可视化编辑器，也会一样产生不需要的 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，当你使用这种编辑器工作的时候，你要创建 5 个都使用 11px Verdana 字体的段落，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 GoLive 是非常成熟、功能强大的，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。

Divitis 的失望

在较好情况下，classitis 被复制到其他多个的结构化标记代码中：

```
<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>
```

上面的例子就是一些成熟且与标准兼容的可视化编辑工具可能生成的代码（请看“可视化编辑器和 Classitis”）。有时，在另一些更严重的情况下，classitis 会更恶化，我们称这种情况为 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="/gothere/" class="warning2">go here</a>!</div></div></div>
```

这里，我们没有看到任何的结构，只有许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。

Classitis 和 divitis 就像一个业余音乐家演奏的不必要的音符，如一个给歌手或者独奏者提供伴奏的高中吉他演奏者。Classitis 和 divitis 就像小说中不必要的修饰词语，以及花园中的杂草。

从你的代码中无情地切除那些 classitis 后，将会看到一个臃肿的 Web 页面缩减了一半的大小。避免 divitis，你会发现能够书写干净的、紧凑的、结构化的代码，而且能够正常地工作在一个文本浏览器中，就像在你喜爱的桌面浏览器一样。严格地去做，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。

7.2.3 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（在 zeldman.com 上的导航图像）。我已经抓住你的大谎言了，你是个坏家伙”。

实际上，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` 来替换那些完好的（或者更适当的）元素的时候，`Divitis` 就出现了。如果你写了一个段落，它应该通过“`text`”标记而不是通过一个 `div` 的 `class` 来标记。你的最高级别的标题应该为`<h1>`，而不是`<div class="headline">`。想看看区别吗？

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 的浏览器，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，但是这些不能破坏使用 `div` 的布局（请看第 3 章“推广标准的困难”中的“默认样式的缺点”）。

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，这么做的成本是非常高的。这样做也是无法让智能电话、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，这是不值得的。

另外一个典型的 `divitis` 的例子是，设计师中了“表格都是不好的，CSS 都是好的”的毒（请看第 4 章“XML 征服世界，和其他 Web 标准成功案例”），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，而且文档更加难以编辑。

7.2.4 热爱 `id`

如果表现层的 HTML 方法被淘汰了，`classitis` 和 `divitis` 也被我们放弃了，那我们怎么能够在一个结合 CSS 的表格的混合区域里面把独立的设计规则运用到导航区域呢？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：

```
<table id="menu">
```

接下来，在书写样式表的时候，你就可以创建一个“`menu`”的选择器，并且关联一个 CSS 规则，用来告诉表格单元、文本标签和所有其他元素该怎么去

准确显示。如果你选择用超文本链接来构造菜单，那么可以通过如下最简单的一条来达到：

```
<td><a href="/events/">Events</a></td>
```

“还有什么要做的吗？”你可能会这么问。是的，除了其他表格以外，没有任何“其他要做的”了。通过对这个表格进行“menu”的标记（或者您自己选择的其他的合适名字），并且通过下面在样式表文件中对“menu”进行的 CSS 选择器的编写，你就可以去除本章中讨厌的 classitis 和 divitis。

没有 classitis，没有 divitis，不需要陈腐的``标记，不需要对每个`<td>`标签附带一些多余的、占用带宽的表现层的高、宽、对齐和背景颜色等属性。只需要一个数字标签（标记“menu”的`id`标记），你就可以在一个分离的样式表内为干净的、紧凑的标记代码进行特别的表现层处理。

CSS 可以控制这些表格的每一个单元的外观，包括背景颜色（以及背景图片等）、边框处理（如果可能的话）、空格（填充）、水平和垂直的对齐，以及翻转效果。CSS 的表格单元翻转效果通常包括改变背景颜色和边框颜色，或者二者同时改变。这些翻转效果在当前大多数浏览器中都能够得到很好的支持，而那些不支持的浏览器则不会破坏页面的外观。同样，CSS 可以控制一个链接的外观，包括字体、大小、颜色及众多的变量参数。

关于图片怎么办

集成一个惟一`id`、紧凑的 XHTML 以及一个样式表文件的方法，是不是也能同样应用于那些使用图片的基于表格的菜单，而且和使用超文本链接一样呢？很高兴你会这么提问！当然同样应用于图片，工作方式丝毫不差。设计这样的表格只要使用那些它们必需的元素，应用到你的图片元素中，并且确认每个图片元素都包含一个可用的`alt`属性。

事实上，你不需要 JavaScript 的参与，就可以使用透明的 GIF 图像和 CSS 规则来创造一个精细和令人吃惊的翻转效果。但是那样会远远超出我们的能力，并且我们也不希望那么去做。

因为本章我们只是进行命名，我们已经对集成最少 XHTML 和智能 CSS 的`id`驱动组合方式进行命名，如果主要使用超文本菜单时候，我们称为“干净的文本方法”，如果使用图像驱动的菜单，我们就称之为“干净的图片方法”。如果本书介绍的方法出自一个自私贪婪的集团，那么这些方法可能会有版权、商标、专利及让你花费更多的许可费用。还好我们不是，你可以免费享受这些方法，并且可以随意地用你喜欢的名称来称呼它们。

另一个值得一提的重要一点是：我们将会理解怎么做可以让表格布局更智能化，并且更少的损害。

7.2.5 消除冗余的表格单元

让我们看看 Marine Center 的站点（如图 7.5 所示），它是在 2003 年设计和开发的，是为那些早期通过网络收集珍贵的鱼和珊瑚虫的顾客服务的站点。请注意，这个站点的布局分为几个任务导向的区域。例如，一个在页面右边的给热心购买者的区域是非常有用的，可以让访问者加入站点的邮件列表，并提供了检索和浏览的方法。左边内容区域对于那些新到这个站点的人是很有用处的，它能提供很多漂亮的照片，并且引导这些访问者阅读每月的“鱼故事”。

图 7.5

菜单上的鱼：Marine Center 站点 (www.themarinecenter.com) 提供了珍贵的网络收集的鱼和珊瑚虫，采用了一个混合的标准兼容布局



类似如此的布局一般把每个组件都放到它自己的表格单元内，比如我们在第 2 章“根据标准设计和制作”中介绍的 Gilmore 站点（如图 2.3, 2.4 所示）。在一个典型的用 Photoshop 创建的图片或者其他布局元素，通过可视化 Web 编辑器完成的布局里面，最大的图片会放置在它自己的单元格内，同样，小的鱼的图片和“鱼故事”的图形标题等也是这样。其他包含透明空格的 GIF 图片则用来控制各个区域之间水平或者垂直的间隔。

许多单元格会被引用，并且产生危险的依赖关系。比如，如果站点的内容编

辑放入了一个星期的介绍，该介绍有很多行字，这个最大的鱼的图片就会被挤到版面的下方，并且它的下移会导致右边出现我们不期望看到的垂直空白区域。站点也可能会占用更多带宽，不是因为它包含图片，只是因为它需要下载复杂的表格，而每个表格都关联着表现层属性。

尽管 Marine Center 站点使用了混合的布局技术，但它还是通过限制最低程度的使用表格来避免通常会产生问题。表格的基本网格把页面分成左边和右边的两列，在列中嵌入的表现元素、空白及对齐，都通过专门的 CSS 控制。我们将在第 10 章解释这种 CSS 方法。而本章的意图，主要是让大家理解尽可能不使用表格的意义。当你开始使用最少的表格代码和 CSS 后，就会发现任何曾经烦恼的问题不过就是如此。

7.3 过时方法的展示

为了参考的目的（并且你可能使用了其中某些方法），我们在这个章节中会将这些过时的方法按照在 Web 中出现的时间顺序一一介绍。

7.3.1 镜像的时代

在最早的商业 Web 站点设计中，图像镜像经常被用来实现一个导航条，比如在图 7.3 或者图 7.5 中的顶部和底部区域。早期的图像镜像由五个部分组成：

- 一个 Photoshop 文件（或者 Canvas 或者其他的图像编辑程序生成的文件），只是一张用户界面图片，以 GIF 或者（后来的）JPEG 格式保存。
- 一个镜像文件包含着每个“热点”区域，并且每个区域都跟随着一个 URL 链接。
- 一个 CGI 程序，一般用 Perl 编写并且安装在服务器的特定目录，它用来解释用户的鼠标单击这个镜像文件链接的响应。
- 在页面源码中对这个图像元素添加一个“ISMAP”属性。
- 在这个图像元素周围，一个超文本锚用来指向 CGI 程序的位置。

这个 HTML 代码一般如下所示：

```
<A HREF="/cgi-bin/imagemap/image.map"><IMG SRC=
"/images/image.gif" ISMAP></A>
```

这就是在 20 世纪 90 年代中期经常使用的服务器端图像镜像，为什么这么命名呢？这是因为站点服务器上的 CGI 脚本完成了用户的单击与 URL 的映射工作。当然，设计师也做了很多工作。图像镜像设计不是轻松的事情，但是设计师

若想创建他们所期待的可视化效果，这是惟一途径。

服务器端图像镜像很快就被客户端的图像镜像所代替，为什么这么命名呢？是因为这些单击的翻译处理工作“都是在客户端”上完成的，也就是说，是在访问者的浏览器上，而不是在服务器上。客户端图像镜像不需要一个独立的镜像文件，而只要在页面添加适当的代码，它们可能像如下的代码：

```
<map name="navigation">
<area shape="rect" coords="0,400,75,475" href="index.html">
<area shape="rect" coords="401,500, 425, 525" href=
"events.html">
...
</map>
```

客户端的图像镜像相对而言更简单，所以，服务器端的图像镜像当然就被抛弃在路旁了，这个进程多么伟大！

镜像和它的不足

对于那些使用图像镜像的开发者，如果要让他们的访问者保持在站点的层次内（如图 7.3，图 7.4 所示），除了为每个页面都创建一个单独的导航图像文件，别无选择。并且，每个访问者都必须为每个页面下载一些新的大量图片。如果开发者更聪明一些，就可能会对每个页面的镜像文件做更改，从而使得当前页面中某部分的图片是不能够被单击的。

为了达到这些效果，就必须占用过多的带宽，在 1994 年—1996 年间，拨号网络是最普遍的，而用户经常对速度感到不满，可访问性运动的萌芽，使得人们注意到这些用户的挫折，因为这些图片经常被标明是坏的。

今天大多数人已经不再使用图像镜像了。当然，还是有一些早期的专家、顾问和他们的追随者还对图像和图形设计持敌对态度，这就有点类似因为汽车吓了你的马，你就对汽车产生轻视。但是我们并不是要和类似不易改变的观点进行争论（除了那些对 Web 标准的误解和误传）。

没有结构就没有可访问性

图像镜像能够对页面图像的外表和动作进行处理，但是他们只能在打开图像功能的图形桌面浏览器上工作，并且他们需要用户物理移动（单击鼠标）和正常的视力（能看到这个图片）。虽然图像镜像可以通过 alt 属性增加更多的可访问性，但是大多没有这样做。

对我们的目的更重要的是，图像镜像没有传达出结构的意义。它们只是简单的图片，如果你有恰当的设备就可以单击它（包括物理或者人的设备）。非结构

化的 HTML 就这样开始了。

7.3.2 切割

图像镜像的主要缺陷在于，它们不能通过缓存重复使用的图片组件而达到节省带宽的能力。设计师们就开始了在 Photoshop 中进行切割，并且用 GIF 或者 JPEG 格式输出这些小图片，然后用 HTML 表格定位它们（我不清楚本书的读者是不是有复习的习惯，如果你有，我们再次建议你再看看第 2 章的图 2.3 和图 2.4）。

切割方法也不是结构化的，但是它节省了用户和服务器的带宽，它是通过把重复元素保存在用户的硬盘的浏览器缓存里面来做到的，并且这个方法通过对图片的 alt 属性可以具备更高的可访问性（通常也是这么做的）：

```
<a href="/events/">  
    
</a>
```

因为“切割”很容易创建美好的外观，而不必像图像镜像一样浪费大量的带宽，所以设计师和开发者很自然地倾向于使用这种方法。

切割时代

早期，我们使用 Adobe 的 Photoshop，然后通过矩形选取框这个工具，手工选择每个图片区域，然后拷贝粘贴到另外一个独立的新文件中。我们应用第三方插件把每个文件输出为 GIF 格式。然后手工书写 HTML 表格代码，并把这些图片组合起来。所有这些我们只能自己动手。

接着许多事情发生了。

首先，Netscape 扩展了它在 1996 年发明的 JavaScript 对象模型，可以让图片根据用户鼠标的行为进行切换响应，于是这种图片交替技术就诞生了。一些人学习使用 JavaScript 在手工编写的切割布局代码中包含这种效果，其他人则把这些代码拷贝和粘贴到他们的站点中。很多人两者都做过。

当时，Photoshop 也加强了它对 Web 的支持能力，为了达到这个目标，Adobe 公司对它附加了一个快捷程序：ImageReady，最开始的时候它是独立于 Photoshop 的，而现在它集成在 Photoshop 里面。通过 ImageReady，你能够通过 Photoshop 向导，让这个程序为你完成切割的功能。ImageReady 甚至还可以生成对应的 HTML 表格代码，在后来的版本中它还可以生成图片交替的 JavaScript 代码。

后来，Macromedia 的 Fireworks 的 Web 图像制作工具可以做同样的工作（现在仍然在做），而且 Macromedia 和 Adobe 公司也随即提供了可视化的 Web 编辑

工具，不管你是否会手工编码，应用这些工具都可以轻松地切分表格和制作图片交替效果。

现在开发人员拥有了两个好处——一个可视化的令人愉快的效果，并且所有的繁重的工作都由计算机来完成（它可不像人类，会对重复的工作感到疲惫）。

7.3.3 保留导航布局

HTML 表格（或者 XHTML 表格）包含 GIF 图片和 JavaScript，依旧是 Web 设计和开发的一个定式。一些标准的拥护者们不太喜欢这种方式，因为它们是非结构化的，并且一些可访问性顾问也看不起这种做法，他们轻视任何和图像设计有关的东西。

但是，就像我们在本章中想尽量去展示的（并且在后来的三章还会有更详细的解释）一样，这样的表格标记对于那些需要进行外观处理的混合的、过渡性的布局还是个恰当的工具。它们的主要缺点就是，当重新改变架构时，或者更新它的外观的时候，就会有大量的工作要做。这个缺点会使设计师和站点拥有者很烦恼，而不会对访问者有所伤害。

表格不仅仅能够达到视觉效果，而且跨浏览器表现的性能也很稳定，而且可以用紧凑的可访问性高的代码标记来开发，并且比图像方式更少地浪费带宽，这些都是在 90 年代大多数设计师探讨得到的。

7.3.4 多余的冗长的表格

为了节省带宽、增加可访问性并且让被搜索引擎（它不能读取图像）检索到，在 20 世纪 90 年代中期，一些 Web 设计师放弃使用 GIF 文本，而开始通过在 HTML 中对嵌入导航标记来切割布局。在那个时候 CSS 还没被浏览器支持，但是一些专利的 HTML 浏览器“扩展”却可以。下面我们将给出的例子，它会比使用它代替的图像方式浪费更多的带宽：

```
<!-- Begin outermost table. This creates the black border
effects. -->
<TABLE WIDTH=80% BORDER=0 CELLSPACING=1 CELLPADDING=0>
<TR>
<TD WIDTH=100% VALIGN=TOP BGCOLOR="#000000">
<!-- Begin actual navigational content table. -->
<TABLE WIDTH=100% HEIGHT=100% BORDER=0 CELLSPACING=
1 CELLPADDING=0>
<TR>
<TD WIDTH=25% HEIGHT=50 VALIGN=TOP BGCOLOR="#FF9900">
```

```
<FONT SIZE=2><BR><BR><FONT FACE="GENEVA, ARIAL,  
HELVETICA"><B>  
<A HREF="item1.html">Menu Item 1</A></B></FONT><BR><BR>  
</FONT></TD>  
<TD WIDTH=25% HEIGHT=50 VALIGN=TOP BGCOLOR="#FF9900"><FONT  
SIZE=2><BR><BR><FONT FACE="GENEVA, ARIAL, HELVETICA"><B>  
<A HREF="item2.html">Menu Item 2</A></B></FONT><BR><BR>  
</FONT></TD>  
<TD WIDTH=25% HEIGHT=50 VALIGN=TOP BGCOLOR="#FF9900">  
<FONT SIZE=2><BR><BR><FONT FACE="GENEVA, ARIAL,  
HELVETICA"><B>  
<A HREF="item3.html">Menu Item 3</A></B></FONT><BR><BR>  
</FONT></TD>  
<TD WIDTH=25% HEIGHT=50 VALIGN=TOP BGCOLOR="#FF9900">  
<FONT SIZE=2><BR><BR><FONT FACE="GENEVA, ARIAL,  
HELVETICA"><B>  
<A HREF="item4.html">Menu Item 4</A></B></FONT><BR><BR>  
</FONT></TD>  
</TR>  
</TABLE>  
<!-- End actual navigational content table. -->  
</TD>  
</TR>  
</TABLE>  
<!-- End outermost visual effects creation table. -->
```

这些代码标记就像天空上的乌云一样，它们是置标语言发展过程中的一种退步，但是它却被很多大型的商业站点使用，包括那些行业的领导者（对比一下我们将在第 8 章和第 10 章创建的例子）。更极端的例子就是在他们数据库记录中也保存着大量的表现层处理的代码。那就是说，即使是一个干净的、结构化的重新设计的网站，从数据库中得到旧的冗余代码还是会破坏这个新架构。

使用这些技术的组织只能靠巨大的手工劳动和成本生存在数字“石器时代”，因为这些数据在它不断升级站点的过程中是没有前途的。这样的组织或早或晚都将会面临挑战。如果你恰好在这样的组织中工作，我们希望这本书能鼓励你尽快地接受挑战。

7.3.5 劣质 CSS 登场

在 1997 年，当 MSIE3 开始提供部分 CSS1 的支持时，那些被 W3C 设计语言的承诺引起兴趣的人们，就开始用相同的、非结构化的、冗余的、滥用的代码构建图像表格布局。

当时，我们并不是采用能够被缓存的、外部链接的、独立的样式表，而是采用和老的 font 和 bgcolor 属性一样浪费带宽的嵌入样式表方式（下一节“重回 1997 年”将会举例）。为了支持“非 CSS 兼容的浏览器”，我们经常到处粘贴 font 标记和其他冗余代码，更加剧了布局对带宽的浪费。通常是没有经过对文档结构的考虑，我们就那么做了。

CSS 真正的威力是能把文档结构和表现层分离开来，但是它不是在浏览器中就能完全解决的，更重要的是必须存在于大多数专业的 Web 设计师的脑海中（很多情况下，我们就没这么想过）。设计师没有转向 CSS 只是因为他们还没准备好。

对于前面我们讨论过的 classitis 和 divitis，它们的前辈比 classitis 和 divitis 更浪费，因为它把所有的 CSS 样式都内嵌了，从而完全没有利用到可以节省带宽的样式表文件指向，并且不能够应用全局样式表从而适用于全站。我们会挖掘出早在 1997 年的那个时代的 CSS/XHTML 代码片断，给你展示它们是怎么样的，用来帮助你避免像那样去设计网页。

重回 1997 年

下面的代码是一个展示这类不好的 CSS（还结合了不好的代码）的绝佳例子，这些例子在 1997 年我们经常书写。如果你使用那个时代的一些可视化 Web 编辑工具来写作 CSS，那么通常也会产生这类 CSS：

```
<font color="#FFFFFF">|</font>
 &nbsp;
 &nbsp;
<a style="color:#FFFFFF;text-decoration:none;" href=
"/isapi/gosearch.asp?target=/us/default.asp" target="_top"
onmouseover="this.style.color='#FFCC00';"
onmouseout="this.style.color='#FFFFFF';">Search</a>&nbsp;
```

令人惊讶的是，你可能经常看到这些代码，而且不是在 1997 年为了达到设计目所做，不是在学生作业中，也不是在一些小型商业站点中。它恰恰是在 2003 年 1 月 7 日你看到的微软公司的首页里面（如图 7.6 所示），并且它还是经常更新的。

因为冗余不仅仅对代码有害，对于本书也同样有害，所以我们就不再解释这些代码的错误。如果你还不明白，那么我们之中任何一个人都不能完成这些工作。

Microsoft 是 W3C 的一个成员，也是对 CSS 和 XHTML 规范的关键贡献者，还是能够兼容他们站点的垃圾 HTML 和 CSS 的浏览器的开发商。重新再看一遍 4.4 节“协作的新时代”，并且回顾 20 世纪 90 年代最差的标记和 CSS 方法。



图 7.6

今天你想去哪里？2003 年 Microsoft 的首页还使用着以前的代码和 CSS，仿佛回到了 1997 年 (www.microsoft.com)。

幸运的是，1997 年后期或者是 1998 年前期，大多数设计师都从这种愚蠢的方法中走出来了。不幸的是，大多数人又转向了另外不好的设计（classitis 和 divitis）。但是乌云已经散去了。

7.3.6 继续前进

表格布局可以被构建得更好，并且更精简，有更好的访问性及标准兼容，而 CSS 布局能够做得更多：减轻你的工作量和服务器的负载，分离结构和表现，以此来增加站点的扩展能力和应用，并且给予可视化设计师的一个新鲜的工具来释放他们的创造力。在接下来的三章里，我们巩固至今为止所学的知识，并且开始我们用 CSS 创建一个紧凑的、兼容的站点之旅。

XHTML 的示例：混合布局 (第一部分)

本章以及接下去的两章，将为一个较小的单元。在本章中，我们将要卷起我们的袖子去应用迄今为止所学到的 XHTML 知识，去完成一个实际的设计项目。我们将要创建部分结构化、过渡性的、且完全符合标准的站点。在第 9 章“CSS 入门”，我们将会为初学者和中级学习者介绍 CSS 的基础知识。最后，在第 10 章“CSS 应用：混和布局（第二部分）”中，我们仍将学习怎么样应用 CSS 去完成我们的项目。这种“通过实践来学习”的方法正如学习游泳的时候就必须要去寒冷的深水区（尽管我们常常将此比喻为去巴黎学习法文的情况），一样是由浅至深的。在这个项目中，我们将学习如何使我们的代码（进而，网站）具备可访问性，以便于达到要求。

8.1 本章所使用的过渡方法的好处

在这章中，我们将要用手工制作一个混合的、过渡的布局，在其中结合使用传统的表格布局和结构化的文本标记及增强的可接近性。我们在这个项目中所运用的技术以及在这三章中解释的内容适用于学校、公共机构，还适用于想达到以下目的的所有小的公司、机构：

- 在有限的预算中管理超大容量的目录。
- 满足广大的网页浏览器，以及支持多种电脑配置。
- 节约访问者们（和他们自己）的带宽。
- 用可靠的、有效的、容易执行的发布方法对网页实现向标准化的过渡。

8.1.1 用样式表代替 JavaScript

在这三章之后，我们将会为 i3Forum 网站（如图 8.1 所示）建立一个标准兼容的模板。我们可以到 <http://i3.happycog.com/> 上的 Happy Cog 演示服务器去浏览最终的模板。同时，还可以到 <http://i3forum.com/> 去浏览用这些模板所产生的网站成品。

图 8.1

这就是最终完成的模板。在第 8 章到第 10 章我们将详细解释过程



在这章中，我们将明确我们要用的标记。在第 10 章中，我们将加入 CSS 来控制颜色、尺寸和元素相关的位置（为了学习 CSS 语言，我们将会在第 9 章暂停一下）。在另一方面，我们将采用 CSS、图片和 JavaScript 来实现菜单的图片交替效果。图片和 JavaScript 并没有什么不好的地方，但是如果用 XHTML 文本和 CSS 代替的话，我们就可以节省带宽，使得网站在多种环境下更加容易被访问，比如说对于单纯浏览屏幕的读者、文本浏览器、没有桌面浏览器者（比如说 PDA 和智能电话使用者及不支持 CSS 的浏览器）。

8.2 基本方法（概述）

i3Forum 版面是为了表现一个清新的、强大的品牌而设计的，它的 XHTML 相当简洁明确。这个版面采用了两个 XHTML 表格和一个导航的菜单，并通过 CSS 得到了加强及控制。第一个表格提供了导航菜单，第二个表格提供了内容（如图 8.2 所示）。



图 8.2

我们在这章中将要建立的模板，采用了关闭 CSS 和开启边界的方法。注意：菜单和内容部分之间的略粗的线是两个表格的结合处。

在这个表格中运用的 XHTML 将会在后面介绍。但是你可能已经遇到了一些初级问题。一般来说，一个版面仅用一个表格，这个表格运用了行和列控制不同的单元格。如果我们用 Adobe ImageReady 去切割 PhotoShop 文件，用来设计站点，ImageReady 将会把整个网页放在一个表格中。那我们为什么要用两个表格呢？

8.2.1 分割表格：CSS 和可访问性的优点

如果你没有看第 7 章中对于“div、id 和其他助手”的讨论，也许应该回去大致地浏览一下，以便于进一步的学习。将我们的版面分成两个表格允许我们更好地运用 id 属性，以达到以下目的：

- 我们将在第 10 章中创建的 CSS 流线性。
- 增强可访问性。
- 根据每个工作目的，对每个表格做出结构性的标记。通过这些标记，便于我们日后重新访问这些代码，或者用 CSS 控制的 divs 来代替表现层的 XHTML。

表格摘要元素

此外，把版面分成两个表格还允许我们为每一个表格分别添加摘要属性，例如：

```
<table id="nav" summary="Navigation elements" ... etc. >
<table id="content" summary="Main content." ... etc. >
```

对于普通的桌面上的浏览器，比如 IE 和 Netscape，摘要属性是隐形的，但是盲人所使用的屏幕浏览器软件是可以理解摘要属性，并且可以大声地读出它的数值。在我们的例子中，屏幕阅读器会说“导航因素”和“主要内容”。设计良好的界面允许用户忽略不感兴趣的表格。编写这样的表格摘要对于那些错过导航链接的用户来说，是一个后备的、增加可访问性的策略。

页面结构和 id

我们根据工作的目的（导航或者内容）为每个表格指定了一个 id 属性值。这样做可以允许我们以后为整个表格编写紧凑的 CSS，避免 classitis 和 divitis（参见第 7 章有关讨论与定义）。

8.2.2 什么是忽略导航，以及为什么忽略

正如名字所暗含的一样，忽略导航的链接允许访问者绕过导航，并可以通过一个类似于锚的链接直接进入到内容表格。

`id` 的 “`content`” 属性值提供了一个锚让我们链接：

```
<div class="hide"><a href="#content" title="Skip navigation."
accesskey="2">Skip navigation</a>.</div>
```

对于广大的视力正常的网络用户来说，忽略导航并不是必要的需求，他们能够通过迅速浏览，忽略不感兴趣的部分，直接查找并阅读网页中感兴趣的部分内容。

但是对于盲人用户来说，他们习惯在一个线性的基础方式上用屏幕阅读器体验网络，也就是说，一次只用一个链接。这些用户在每次打开你的网站时都要忍受不变的菜单朗读。对于这一部分用户来说，这是失败的。忽略导航可以让这些用户避免这些问题。

忽略导航还可以让用户在使用不支持 CSS 的 PDA 浏览器和网络电话时，避免冗长的翻页。虽然忽略导航这个方法并不完美，但是仍可以给身体上有障碍的用户带来益处（参考后面的部分“AccessKey：好消息，坏消息”）。

忽略导航和 Accesskey

我们的忽略导航链接让视觉障碍者或使用不支持 CSS 浏览器的访问者可以直接跳到第二个表格中的内容。第二个表格的 `id` 属性名（锚链接）是 “`content`”：

```
<table id="content" ...> etc.
```

在类似不可见或不支持 CSS 的环境下，页面（如图 8.3、图 8.4 所示）顶上的链接很有效。你同时需要建立一个规则，以便在支持 CSS 的浏览器中忽略忽略导航的链接。这些将在第 10 章中学到。（如果你比较急，我们在这里也提到了。）

```
.hide {  
    display: none;  
}
```



图 8.3

在一个不支持 CSS 的浏览器（或一个关闭了 CSS 但是支持 CSS 的浏览器）下，导航链接非常清楚地在网页的最上部分



图 8.4

当我们关闭 CSS 后，在上文中的隐藏的忽略导航就显示出来了

由于这个 CSS 规则，所以打开 CSS 支持模式的现代浏览器的使用者将看不到忽略导航链接。当然，大多数使用者是不需要看到此链接的，因为他们不需要

忽略导航的功能。忽略 CSS 的屏幕阅读器将会愉快地阅读 `div` 的内容，因而有视觉障碍的访问者可以避免浏览生闷的链接。

当然也有例外。当一些行动不便的用户运用支持 CSS 的浏览器访问网站时，可能希望忽略导航部分，并且直接进入到内容部分。大部分行动不便的用户可以一次性地看到整个网页（视障用户除外），这些用户可能使用键盘或者其他辅助设备进行导航，他们跳过不想要的导航链接，可能是一个麻烦的事情。

如果这些用户没有办法在他们的浏览器中看到这些忽略导航，我们应该怎么样帮助他们呢？`accesskey` 属性提供了一些选择，就算当忽略导航链接在浏览器上是隐形时也可以正常工作的。这个方法并不完美，我们将在下面介绍有关这方面的内容。

accesskey：好消息，坏消息

HTML/XHTML 的 `accesskey` 的属性使人们可以用键盘代替鼠标对网页实现导航。为一个元素指定一个 `accesskey`，你可以很简单地声明它，就像先前的 XHTML 的摘要一样，我们把这个属性用粗体显示在下面：

```
<a href="#content" title="Skip navigation." accesskey="2">Skip  
navigation</a>.
```

在此代码中，我们设定了忽略导航链接并且将它的值定义为 2。因此，访问者可以很简单地通过键入 2 从而达到忽略导航的目的。这种做法通常会增强网页的可访问性，这个标记代码也很容易写，对网页的视觉设计也没有影响。在这个例子中，这是有好处也有坏处的。

访问者怎么知道在他们的键盘打入 2 就可以忽略导航呢？多数浏览器不会显示这些 `accesskey` 值，也很少有浏览器这么做。

accesskey 和 iCab

只有 iCab（如图 8.5 所示）Mac 浏览器，可以显示出 `accesskey` 值。大部分网页用户并不是 Mac 的用户，而且大部分 Mac 的用户也并不使用 iCab。更糟糕的是，当忽略导航通过 CSS 被隐藏的时候，iCab 也不会显示出 `accesskey` 值。iCab 仍然不能给 CSS1 很好的支持，W3C 的第一个 CSS 推荐被发表于 1996 年。虽然 iCab 是一个非常有意思的浏览器，而且支持 HTML 4，也给人深刻的印象（我们并不是开玩笑：iCab 对 HTML 4 的支持是极好的），但是它还是不能完美解决 `accesskey` 问题。



图 8.5

世界上所有的浏览器，只有 Macintosh 的 iCab (<http://www.iCab.de/>) 可以显示我们的 accesskey 的值 2，使得使用者可以只要键入 2 就可以忽略导航。

对于 accesskey 的两个设想

大多数可能从 accesskey 属性中获益的用户根本无法知道哪个字母或者数字是他们应该键入的，所以他们根本无法从中受益。正因为如此，在你的设定中加入 accesskey 是有点不切合实际的。

如果 W3C 可以将 accesskey 推荐为通用功能（并且设计师和开发者遵循这个推荐），用户就可以知道他们应该键入什么了。那将会是件很好的事情。

作为选择，浏览器制造商增强了对 accesskey 的支持。如果希望显示 accesskey 值，用户可以在“参数设置”→“可访问性设置”中打开选项。Windows/IE 提供了一个可访问性选项，这些选项可以让用户忽略网页上的字体大小，也有一个总是显示 accesskey 值的选项。

我们必须承认，希望 W3C 在短期内对 accesskey 快捷键的设想进行标准化过于理想化。同时，我们也感觉到让主流浏览器开发商浪费时间和资源维护一个总是可见的 accesskey 值也过于理想化。我们仍然会继续用 accesskey。一些用户可能会通过查看源代码的方式查看页面中的 accesskey 值，然后再用正确的键去导航。我们希望这些设想可以快点成真。

8.2.3 附加的 id 属性

在我们前面的网页中，为导航表格每个单元格指定了一个独一无二的 id 属性名字和 id 属性，两个单元格将使这种方法更明了。

```
<td width="100" height="25" id="events"><a href="events.html">Events</a></td>
<td width="100" height="25" id="schedule"><a href="schedule.html">Schedule</a></td>
```

同样，我们还可以为内容表格的两个主要区域指定惟一的 id 属性值，就像下面例子中的 sidebar (`id="sidebar"`) 和 primary content (`id="primarycontent"`)。接下去，这样可以使内容清楚地转移，方便地从内容表格中剥离。下面的 id 属性和值已经用粗体标出：

```
<table id="content" etc.>
<tr>
<td width="200" id="sidebar">
Sidebar content goes here.
</td>
<td width="400" id="primarycontent">
primary content goes here.
</td>
```

为了更好地度量，我们在导航表格的第二级增加了一个 id 属性名字。因此，导航按键的第二行就赋予了一个 id 值：

```
<tr id="nav2">
```

正如你可能期望的一样，导航第三行也赋予一个 id 值：

```
<tr id="nav3">
```

多少 id 是合适的

后面的两个 id 属性名 (nav2 和 nav3) 不被版面设计的目的所需要，但是它们可能在以后的重新设计中会被用到。需要添加它们还是不要呢？若要添加它们，只需在 XHTML 增加一些字符；若不去添加，我们也许也可以获得同等的好处。

如果，在最终的站点中，导航位于一个服务器端包含的、独立的文件（或者由 PHP、JSP、ColdFusion、ASP 管理的一个独一无二的记录），顾客可能很容易地随时编辑那些文件，通过调整独立文件去改变整个站点。如果顾客计划使用服务器端技术，那么在其中包括 nav2 和 nav3 是非常愚蠢的。但是，如果没有服

务器端技术被运用，而且菜单设定被手动地在每个页面上进行重复，那么比较安全的做法是：在页面中设置 nav2 和 nav3，并且避免在未来重新设计中的潜在的发现并移除错误。这就是我们之前所做的。

8.3 开始的标记和结束的标记是对应的

在这页还有接下去的几页中，你将会发现我们网站的第一个标记是从 `<body>` 开始，到 `</body>` 结束。所有随后的设计都可以通过 CSS 来调整，一个好处就是你可以尽可能多地用样式表代替表象的东西，即使这里示例的混合代码网站也是如此。为了节约篇幅，我们删掉了可爱的 `body` 内容，用简单的文本来替代。

同样注意到，在标记中，我们用相对的图片链接路径 (`img src="images/logo.gif"`) 代替绝对链接路径 (`img src="/images/logo.gif"`)，因为我们工作在本机而不是在服务器上。结束标记也采用绝对路径（绝对链接比相对链接更加可靠，因为就算是文件位置变化，链接也不会被破坏。举例来说，如果我们将 `/events.html` 移动到 `/events/index.html`，绝对链接 `/images/logo.gif` 仍然会工作。同样，绝对 URL 可以帮助我们在一些老的服务器中避免 CSS 的 bug，这些 bug 一般是由于浏览器误解了样式表中的相对路径引起的）。

从技术角度上说，结束标记与开始标记略微不通，它在开始标记的前面加了一个“/”。大多数用户并不在意，但是总会有读者拿着本书中的代码与网站源代码进行对比。

你可以到网站直接看源代码。我们前面的项目被归档在 <http://i3.happycog.com/>

8.3.1 导航标记：第一个表格

接下来的部分是导航部分。为了提高阅读兴趣，我们提前告诉你这一部分的标记，虽然这是有效的，但我故意提交了一些不纯的标记，看看你能否找出这些错误。

```
<body bgcolor="#ffffff">
<div class="hide"><a href="#content" title="Skip navigation." accesskey="2">Skip navigation</a>.</div>
<table id="nav" summary="Navigation elements" width="600" border="0" align="center" cellpadding="0" cellspacing="0">
<tr>
```

```

<td rowspan="3" id="home" width="400"><a href="/" title="i3Forum home page."></a></td>
<td width="100" height="25" id="events"><a href="events.html">Events</a></td>
<td width="100" height="25" id="schedule"><a href="schedule.html">Schedule</a></td>
</tr>
<tr id="nav2">
<td width="100" height="25" id="about"><a href="about.html">About</a></td>
<td width="100" height="25" id="details"><a href="details.html">Details</a></td>
</tr>
<tr id="nav3">
<td width="100" height="25" id="contact"><a href="contact.html">Contact</a></td>
<td width="100" height="25" id="guestbios"><a href="guestbios.html">Guest Bios</a></td>
</tr>
</table>

```

8.3.2 陈述、语义、纯度、错误

你对 Web 标准的“发烧”达到什么程度呢？你能在上面的 XHTML 中找出错误么？其实第一行就犯了一个错误——在 body 的元素中使用了过时的 bgcolor（背景颜色）属性。就算是在非 CSS 服务器下，那个页面的背景色也应该是白色的（#ffffff）。如下：

```
<body bgcolor="#ffffff">
```

这样的守旧标记的写法会把我们立刻踢出标准的大门之外（呵呵，比流星的速度还要快）。毕竟，CSS 允许我们指定 body 的背景颜色，W3C 推荐的是使用 CSS 来达到这个目的，而不是用 HTML 或者 XHTML。在那些标准发烧友看来，我们使用 bgcolor 属性是一种错误。

一本过渡时代的过渡书籍

对于那些每个星期花几个小时在 W3C 的邮件列表上争论有害的表现标记的发烧友来说，我们在这里所做的也是不好的、有害的。其实，当我们用表格代替表数据表格容器，并在表格中指定宽和高及设定图像边距为 0 的时候，我们已经犯了错误。事实上，在一些人的眼里，整章都是一个错误。一些标准发烧友可能

根本不重视这本书。坦白地说，在他们看来，我们应该教你怎么样编写语义标记，而不应该告诉你有时候可以采用表格来布局。

他们的说法也不为过，但问题是现在到未来几年（直到设计师使用的是支持 XHTML 纯语义的浏览器和 CSS、SVG 的未来版本）这种做法还不现实。这是一本过渡时代的过渡书籍。“Web 标准”并非一成不变的法律条款，而是充满选择和判断的道路。以我们的观点来看，这两类人是一样的：要求在服务器上保持绝对纯度并且用网络标准在标准环境中尽可能地同主流作对的人们，没有听过或彻底地反对结构化标记和 CSS 的人们。他们都是极端。

宽容老浏览器

为什么我们不顾视觉效果而用 `bcolor` 属性？我们创建混合型网站时并没有假设我们的用户用的是什么浏览器。对于一个不支持 CSS 的浏览器来说，如果设置了除白色以外的任何背景颜色，透明的 GIF logo 图像的边缘就可能产生天使光环一样的杂点。没有用户希望他们的 logo 看起来很劣质，即使是在老浏览器上。

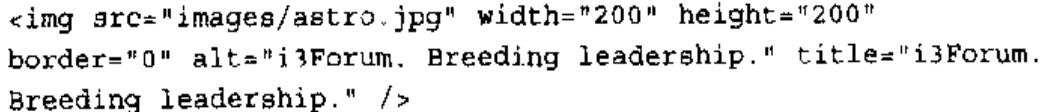
一个曾经流行的、不支持 CSS 的老浏览器将灰色设置为它的默认背景色。然而我们的 logo 图片不是基于灰色，而是基于白色背景。如果我们并没有通过 XHTML 的 `bcolor` 属性设置背景颜色，我们的 logo 在这样的浏览环境中看起来会非常糟糕。

事实上，你可能不关心你的网页在 2.0 或 3.0 浏览器上看起来会是什么样。同样，你可能也不关心它在 4.0 浏览器上看起来会是什么样。一旦你的老板和顾客却不是。在这章中所用的语义不纯的技术并不是为了在所有的浏览器上得到相同的视觉体验。在一个不支持的 CSS 浏览器中，我们的设计看起来不会比图 8.3 好多少。

我们在这个网站使用的表格及 `bcolor` 是为了向你展示：这样的折中办法可以用在 XHTML1.0 过渡期中，并且网站是有效的。我们这样做同样暗示：关于任何将标准引入你的工作的努力，即使是采用妥协折中（但是现实）的办法（使用部分表现层标记）——都是值得做的。

8.3.3 内容标记：第二个表格

紧跟导航之后的是内容表格，写过 HTML 或 XHTML 的人都应该读懂。两件事值得关注：紧凑简洁的标记和 `id` 的使用。

```
<table id="content" summary="Main content." width="600"
border="0" align="center" cellpadding="0" cellspacing="0">
<tr>
<td width="200" id="sidebar" valign="top">

<h2>Subhead</h2>
<p>Text</p>
</td>
<td width="400" id="primarycontent">
<h1>Headline</h1>
<p>Copy.</p>
<p>Copy.</p>
<p>Copy.</p>
<p>Copy.</p>
<div id="footer">
<p>Copyright © 2003 <a href="/" title="i3forum
home page.">i3Forum</a>, Inc.</p>
</div>
</td>
</tr>
</table>
</body>
```

在第 9 章，我们将会介绍 CSS 基础知识。而后在紧接着的第 10 章中，我们将会运用 CSS 为我们的混合网站添加视觉控制功能，并使之更加华丽。

CSS 入门

在第 8 章“XHTML 示例：混合布局（第一部分）”，我们学习了创建一个混合过渡方式的标准兼容的站点，在第 10 章“CSS 应用：混合布局（第二部分）”，我们将用 CSS 样式表来完成这个站点。在此之前，我们必须先了解一些基本知识，这就是学习本章的目的。本章我们将快速学习 CSS 的基础语法，了解一些 CSS 基本概念，最后还会介绍一种不同于教科书中所介绍和大多数设计师所使用的 CSS 设计方法。即使你对样式表非常熟悉，也建议你浏览一遍本章的知识。

9.1 CSS 概述

W3C 把 CSS 定义为“一种对 Web 文档添加样式的简单机制”（例如：字体、颜色、间距），(www.w3.org/Style/css/)。概述中一些被省略的细节值得关注：

- CSS 是 Web 标准化布局语言，它可控制颜色、版式设计，以及元素、图像的尺寸和布局
- 正如本章所示，虽然 CSS 精确并且功能强大，但是对于手工书写来说还是比较简单的。
- CSS 的适用性：仅仅一个 CSS 文件就能控制包含上千页面和几百兆字节的整个站点的外观。
- 其创建者（W3C）的意图是以 CSS 取代 HTML 表格式布局、帧和其他外观程式。我们将在下一章中看到，对于混合过渡方式的布局，CSS 也是相当高效的。
- 下一节“CSS 的优点”将会描述，纯 CSS 布局与结构式 XHTML 相结合能帮助设计师把表现从结构中分离出来，使站点的访问及维护更加容易。

9.1.1 CSS 的优点

CSS 具有以下这些实用性的优点（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（见第 1 章“99.9% 的网站都是过时的”里的“过时的标记：网站的成本”）。
- 缩短设计和开发的时间，能在几小时内创建像第 8 章和第 10 章里面示范的站点，而且其中有一部分时间我们在写作第 8 章和第 10 章。（当然这种时间上的节省仅限于开发，内容上花的时间仍然一样）。
- 缩短更新和维护的时间：
 - ◆ 当文档更新后，人们无需再担心复杂的表格、字体标签，以及其他旧式的布局元件，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 1 到两个规则，整个站点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。
- 从代码标记里清除一些或全部的表现元素以提高访问能力。

CSS 相关书籍介绍

《CSS 神珍指导》(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。别以为它小就用处不大，实际上它是 CSS1 最清晰完整的指导书。

如果你喜欢跟着设计示例学习，“Eric Meyer on CSS”将使你获益匪浅(Eric Meyer: New Riders, 2002)，这本书有很多第 10 章和第 12 章里讨论的设计方案，非常实用。

作为世界上领先的 CSS 专家之一，Eric Meyer 是 Netscape Communications 的一位权威作者，CSS-Discuss 邮件列表的共同创建者，也是本书的技术编辑之一。我们极力推荐 Eric Meyer 的这两本书，一本阐明 CSS 语法，另外一本教我们如何创造性地使用 CSS。

9.2 样式解析

我们将在这一节介绍CSS的主要支架部分，本书不是完善的CSS指导手册，CSS指导手册内容更多，虽然我们希望CSS指导手册能小到可以放进口袋——见前面的补充说明“CSS相关书籍介绍”。

下面开始解析样式，我们将通过在第10章的应用学习更多CSS知识，并且在本书的后半部分继续学习CSS。

9.2.1 选择器、声明、属性和值

样式表由一个或多个规则定义组成，这些定义控制被选择的元素将如何被显示。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，而在大括号内的color:red;就是声明。

声明相应地也是由两部分组成的：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

代替英文单词red，我们用红色的十六进制值#ff0000写出（网页颜色）：

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

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，下面两种方法中都可以：

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

零值后需跟百分比符号

注意使用RGB百分率值时，甚至当值为零时也要带百分比符号，在其他CSS情况下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百分率时，零值要求带百分比符号。坚持在RGB后显示百分比符号是CSS无法更改的事实，探究其原理，早已无法考证。

我们开始剖析样式表并不是说我们要抱怨矛盾或者提出一堆异议，只是我们已经身陷其中无法自拔，当我们继续用颜色例子讨论我们的 CSS 入门知识，偶尔抱怨几句，也请您谅解。

9.2.2 多重声明

不指定背景的颜色，只指定颜色是不够的，反之亦然：

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

属性值为 transparent 是为了用来防止一个定义的背景覆盖另一定义的背景：

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

注意 一个定义可以由不只一个声明构成，可用分号来区分不同的声明。

注意！会坏大事！透明背景和 Netscape 4

警告：Netscape 4.x 在很多地方把透明理解成黑色，如果你使用 background : inherit; 而颜色呈现为黑绿，那是因为 Netscape 4 不能识别 inherit 字符串，即使很明显，它不是十六进制值，也将其强制转为十六进制分析（IE /Windows4.0 以前的版本也会出现这样的情况，在此不举例了）。

问题在于：无论你说明与否，背景的颜色默认值都为 transparent，如果 Netscape 4 用户是你网站的主要访问者，你可能不想声明背景颜色。如果这么做，W3C 的 CSS 校验系统将发出警告，但是警告总比严重损坏网站要好得多。再次说明，只有当你支持相当多的 Netscape 4.x 用户时才值得这么做。

分号的作用

这里我们再次谈到矛盾和例外情况，任何声明的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 设计师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。

9.2.3 空格和不区分大小写

大多数样式表本身含有不只是一个定义，而多数定义又含有不只是一个声明。多重声明比较容易理解，使用空格将使得编辑样式表更容易：

```
body {  
    color: #000;  
    background: #fff;  
  
    margin: 0;  
    padding: 0;  
    font-family: Georgia, Palatino, serif;  
}  
  
p {  
    font-size: small;  
}
```

用不用空格对 CSS 在浏览器上的显示没有影响，与 XHTML 不同，CSS 是不区分大小写的，当然也有例外。当与一个 HTML 文件联用时，类别和 id 名称是区分大小写的，myText 和 mytext 是不相等的。CSS 本身不区分大小写，但文档语言也许就区分。详情见 <http://devedge.netscape.com/viewsource/2001/css-class-id/>

9.2.4 字体选择和默认值

网页设计师为整个站点指定字体，如下：

```
body {  
    font-family: "Lucida Grande", Verdana, Lucida, Arial,  
    Helvetica, sans-serif;  
}
```

注意多重名称字体 ("Lucida Grande") 必须直接用 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。美国的设计师可能对此感到很为难，因为他们已经适应了把逗号放在引号的里面。

字体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。依次类推，这就是每种字体按一定的顺序列出的原因。

按顺序调用以适应不同的平台

顺序很重要。Lucida Grande 字体在 Mac OS X 里可以找到，而所有现代的 Windows 系统、Mac OS X 和旧版本的 Mac 操作系统里都有 Verdana 字体，所以如果把 Verdana 字体列在第一位，OS X Mac 就会显示 Verdana 字体而不是 Lucida Grande 字体。

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）。

UNIX 用户用 Lucida，旧版本的 Windows 用户用 Arial，Helvetica 可用于旧版本的 UNIX 系统，如果所列出的字体都不能用，则默认的 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。

并非完美的科学

没人会认为 Lucida、Verdana、Arial、Helvetica 在美观、X 坐标和合适性方面都相等（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、显示器质量、gamma 值，以及操作系统，它们的设置混乱而且不相同，因此保持相同的视觉效果完全不可能的。我们只是在条件允许的情况下尽可能保证最佳视觉效果，不同的用户可能感觉看到的效果差不多。

9.2.5 群选择器

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, li, dl, dt, dd {
    font-size: small;
}
```

这种能力可以帮助你解决旧版本浏览器不支持 CSS 继承的问题。

9.2.6 继承和它的不足之处

根据 CSS，子元素从母元素继承特性，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 将被指定为 Verdana 字体，否则指定为一般 sans-serif 字体。

所有子元素

每个CSS定义都有继承性，如果定义的规则对最上级元素（这里主要指body元素）起作用，则对其子元素（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一个定义，整个body的子元素都将显示为 Verdana 或者一般的 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。

但对于在浏览器竞争最激烈的时候产生的浏览器不起作用，因为那时候支持标准不是制造商优先考虑的事情。例如，Netscape 4 就不支持这个功能，因为 Netscape 4 忽略继承，以及作用在body元素的定义（IE/Windows 直到 IE6 都有相关问题，使在表格中的字体样式被忽略掉了）。这样，母元素也就不起作用了。

“Netscape 4 友好”

幸运的是，你可以用我们称之为“Netscape 4 友好”的冗余原则来弥补老版本浏览器未能理解继承的不足：

```
body {  
    font-family: Verdana, sans-serif;  
}  
p, td, ul, ol, ul, li, dl, dt, dd {  
    font-family: Verdana, sans-serif;  
}
```

4.0 浏览器也许不能理解继承，但能理解群选择器，有效的 Verdana 字体能被所有浏览器接受，这种冗余（创建冗余定义）会占用一些带宽，但你还是希望使用它的。

顺便说一下，Netscape 4 不是唯一不支持继承的老版本浏览器，只是它是惟一的一个到今天还有少数忠实用户的浏览器。

继承是问题的根源吗

如果你不希望每个子元素都继承被指定为 Verdana、sans-serif 字体，例如，你想要用 Times 字体显示段落，怎么办呢？没问题，为 p 创建一个更详细的定义（在下列代码中用粗体突出表示），它将优先于母元素定义而执行。

```
body {  
    font-family: Verdana, sans-serif;  
}  
td, ul, ol, ul, li, dl, dt, dd {  
    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。

9.2.7 内容（派生）选择器

你可以根据元素表现的内容来决定元素的样式，从而避免过度的class，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“内容选择器”，因为内容决定其调用的定义。在CSS2里称为派生选择器，无论名称如何，作用都是一样的。

若要指定当出现在列表项时，标记为``的文档显示为斜体而不是黑体，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 affect 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>

```

看看下面的CSS：

```

strong {
    color: red;
}
h2 {
    color: red;
}
h2 strong {
    color: blue;
}

```

以及对标记的影响：

```

<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>
```

你也许不会用内容(派生)选择器来精简代码，也不会把列表中的黑体字文档指定为斜体，但你也许会用这些选择器创建复杂的设计功能，例如给一个普通的XHTML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，而更可能的是用内容id或类别选择器来制造这些效果。

9.2.8 id选择器和内容id选择器

在现代的布局中，id选择器经常用在内容选择器里：

```
#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或表格单元，虽然也可能是一个表格或是一些其他的块级元素，甚至可能是一个内联元素，如``或``。这样的用法有些古怪，也是非法的——不能把`<p>`嵌入``中，但不考虑那些使用过的元素，它就是该页中惟一使用id属性为 sidebar 的元素。如果不记得原因，可以看看第7章。

一个选择器，多种用法

即使标记里标记为 sidebar 的元素在每页只出现一次，id选择器在需要时可以被多次用做内容选择器/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，均被单独定义。

选择器可以单独作用

`id` 选择器不需要依靠内容，它可以单独地使用：

```
#sidebar {
    border: 1px dotted #000;
    padding: 10px;
}
```

根据这个定义，`id` 属性为 `sidebar` 的页面元素内容将有黑色点状边界，边界粗细为 1 像素，边框距（内部空白）为 10 像素。

9.2.9 类别选择器

把类别属性进行分别定义在样式表中的效果很好，在 CSS 里用一个点表示类别选择器：

```
.fancy {
    color: #f60;
    background: #666;
}
```

任何类别的属性为 `fancy` 的元素的文档将呈现橙色 (#f60)，背景呈现为灰色 (#666)。`<h1 class="fancy">Boy Howdy!</h1>` 和 `<p class="fancy">Yee haw!</p>` 都调用这个颜色方案。

如 `id` 一样，类别选择器也可用做内容选择器：

```
.fancy td {
    color: #f60;
    background: #666;
}
```

在上面的例子中，那些类别名称为 `fancy` 的较大元素里的表格单元文本都显示为橙色，背景显示为灰色（类别名称为 `fancy` 的较大元素可能是表格的一行，也可能是一个 `div`）。

可以根据其类别挑选元素：

```
td.fancy {
    color: #f60;
    background: #666;
}
```

上例中类别名称为 `fancy` 的表格单元显示为橙色，背景为灰色。

```
<td class="fancy">
```

你可以把类别名称 fancy 仅仅指定给一个表格单元，也可以指定给很多表格单元。这样这些表格都显示为橙色，并且背景显示为灰色。没被指定类别名称 fancy 的表格单元将不受这个定义的影响。使用类别名称 fancy 的段落和其他元素都不会显示成橙色，也不会有灰色背景。这种效果仅限于标记为 fancy 的表格单元，因为我们是这样书写定义的（用 td 元素来选择类别名称 fancy）。

9.2.10 组合使用选择器创造精致的设计效果

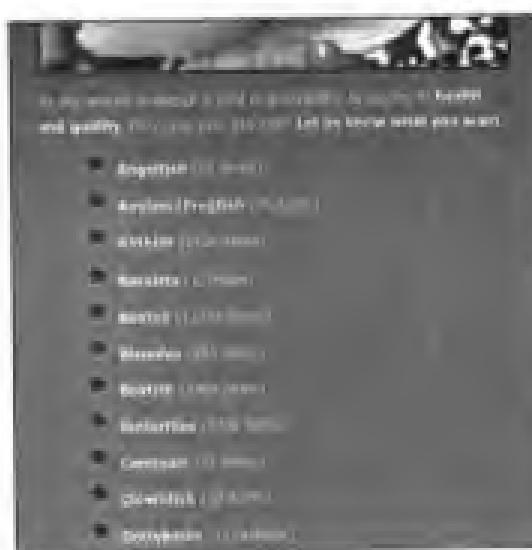
类别、id、内容选择器可以组合使用，从而制造出精致或突出视觉的效果。Happy Cog 制作的站点 The Marine Center 是用一个符合其商标形象的鱼的图像代替普通无序列表前沉闷的黑点。

图 9.1

类别、id、内容选择器可以组合使用，从而制造出精致或突出的视觉效果。站点 The Marine Center (<http://marine.happycog.com/>) 用更符合其商标形象的鱼图标代替了大多数无序列表前沉闷的黑点。

下面的 CSS 规则告诉类别属性 inventory 的无序列表，显示图片而不是黑点（在老式浏览器上显示不出鱼的图标而是显示一个圆点）。

```
ul.inventory {  
    list-style: disc url(/images/common/lister2.gif) inside;  
}
```

这里简写了它调用的标记：

```
<ul class="inventory">  
    <li><a href="/angelfish">Angelfish</a> (67 items)</li>  
    <li><a href="/anglers">Anglers/Frogfish</a> (35 items)</li>  
    <li><a href="/anthias">Anthias</a> (5526 items)</li>  
    <li><a href="/basslets">Basslets</a> (15 items)</li>  
</ul>
```

IE/Windows 和 Opera/Windows 用户还能获得一种附加的功能(并非刻意的), 站点先显示普通的圆点, 然后渐渐变成鱼的图片, 这很像 Flash 或 JavaScript-like 做出的动画效果, 但这仅仅是因为巧合, 是 Windows 系统中的 IE 和 Opera 浏览器读取并显示网页元件的次序的结果, 在其他的浏览器上, 用户只会看见鱼的图标, 不会看到这种转变的过程。

使用其他类别属性的无序列表是不会显示这样的鱼的图片的, 如果这个类别属性被用于其他对象而非无序列表, 也不会出现鱼的图片。

鱼的图标也会出现在网站的“立即购买”栏中(如图 9.2 所示)。这都是使用类别名称 `cartier` 的内联元素(间距)的作用。

```
.cartier {  
    padding-left: 5px;  
    background: transparent url(/images/common/cartfish.gif)  
no-repeat top left;  
}
```

图 9.2

鱼图标也出现在站点的“立即购买”元素中, `` 标签对制造这种效果没有影响, 看前面的文字说明可了解其制作过程



浏览 <http://marine.happycog.com/> 服务器上的资源, 你将学到更多的技术, 网站上存有站点的原始模板(如图 9.3 所示)。

继续学习

下一章我们将学到更多的 CSS 语法, 但现在先来考虑一个问题: 什么地方该插入 CSS? CSS 能嵌入 XHTML 吗? 还是它是一个独立的文件, 为什么?(提示: 继续往下看。)



图 9.3

在你的照片和 CSS 样件的图标之间，人们能浏览到网站的内容。即使“404 not found”错误页也能使访问者觉得只看见了很多负

9.3 外联、嵌入、内联样式

可以用样式表的三种方式中的任意一种指定网页：外联、嵌入或内联，我们从最好用的说起。

9.3.1 外联样式表

外联样式表（CSS 文件）是一个和它控制的 XHTML 页面分别存在不同的地方的文本文档。XHTML 页面通过文档顶端的一个超链接调入那个 CSS 文件，或者通过把 CSS 文件导入样式元素（也位于文档的顶端）来使用 CSS 文件。样式表的超链接如下：

```
<link rel="StyleSheet" href="/styles/mystylesheet.css"
      type="text/css" media="all" />
```

@import 命令用于输入样式表中，如下：

```
<style type="text/css" media="all">
@import "/styles/mystylesheet.css";
</style>
```

或者这样写：

```
<style type="text/css" media="all">
@import url("/styles/mystylesheet.css");
</style>
```

功能强大，成本低廉

无论是链接还是导入，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缓存中，它依然是动态的，能控制一个、几十个、几百个、乃至成千上万站点上的页面，而不需要再另外下载，极为方便又功能强大。

作为浏览器选择器的链接和导入

实际上所有支持 CSS 的浏览器都支持链接的方法，包括 CSS 支持不是十分优良的老式浏览器。相反，@import 法只在 5.0 及以上的版本上有效，所以设计师可以用 @import 法对 4.0 浏览器隐藏样式表。

但这并不意味着对老式浏览器隐藏所有的 CSS，比如在第 2 章和第 4 章中所讨论的那样，事实上是有一些老式浏览器能“获得”@import，而有一些浏览器却不能。

因为一个站点上可以用多个样式表，设计师可以把基本的样式放进一个链接的样式表，而把复杂的样式放进一个导入的样式表。链接的基本样式表在所有浏览器（包括不理解 CSS 的浏览器）上都是可视的，并提供基本设计元素，如使用的字体、文档、背景、链接颜色等，导入的表的样式只对一些更符合标准的浏览器有效。更多新版本的浏览器可以理解这些样式，而老式浏览器则分不清楚 CSS 定义。

你的 XHTML 可能可以书写如下：

```
<!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>
    <title>Your Title Here</title>
    <link rel="StyleSheet" href="/css/basic.css" type=
      "text/css" media="all" />
    <style type="text/css" media="all">
      @import "/css/sophisto.css";
    </style>
    <meta http-equiv="Content-Type" content=
      "text/html; charset=ISO-8859-1" />
  </head>
```

用这种分层的方法，你能支持所有的用户，而不用对任何人隐藏内容，也不必烦恼浏览器的检测（这种技术将在下一节“从嵌入样式到外联样式：双表格法”中详细介绍）。

只有当你使用外联样式表时才能获得两个好处：大大地节省用户和服务器的带宽，支持各种各样浏览器的性能。我们在第 10 章设计完成 i3 论坛站点时，将

使用外联样式表的这些优点（关于 i3 论坛有专门的教科书系列）。

9.3.2 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，设计师可在 XHTML 页面的 head 位置嵌入样式表，使用的样式元素如下（用粗体突出显示）：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>i3forum</title>
    <style type="text/css">
      <!--
      body {
        background: white;
        color: black;
      }
      -->
    </style>
    <meta http-equiv="Content-type" content="text/html; charset=
iso-8859-1" />
  </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也必须重新下载。为什么设计师还要用嵌入式样式表呢？以下是几点原因：

- 站点可能只由一个页面构成。
- 用户还在使用 IE3 访问站点。IE3 是最早开始支持 CSS 细节的浏览器，它不支持外联样式表。
- 设计师已经用外联样式表控制整个站点，但仅仅为一个页面必须额外写定义。那么创建一个嵌入式样式表就非常有必要了，实际上，这是惟一能达到一定视觉效果的方法。当我们在下一章设计 i3 论坛样式表时，我们将为站点的每个页面添加嵌入式样式表，用来控制当访问者从一页单击进入另一页时突出显示菜单项中的文字“you are there”。
- 设计师在不断地修改样式表，需要立刻看见修改后的效果，这是创建嵌入式样式表的另一个很好的理由。

设计站点时，在你要设计的页面<head>中插入嵌入式样式表是非常有意义的，如果你对你的设计很满意，把样式复制到外联 CSS 文件中，删除标记中的嵌入式样式。

在下一章描述的制作过程中，我们将使用嵌入式样式表，因为它能提供最快最简单的方法（修改一个定义，用浏览器重新读取这个网页，看看能不能获得你想要的效果）。如果 CSS 嵌入样式表的作用正如我们预期的那样，就可以把它从 `<head>` 里删除，在服务器的/css/ 子目录里把它存为外联 CSS 文件，建立链接，以节省完成后的站点上大量的用户带宽。

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;">
```

如你所知，内联样式并不节省带宽——内联样式会增加每个使用它的页面所占的带宽，这方面和 `` 标签一样浪费。

我们不应该首先就考虑用内联 CSS 来定义网页的样式，那样就本末倒置了。但是如果谨慎地使用，内联 CSS 会是一个很有帮助的工具。内联样式只是使 CSS 更加完善。

9.4 “最合适方案”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 divs 代替结构元素，如 h1, h2, li, p，以及其他一些我们再也不想使用的方法。如果站点在老式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——在他们手头上最差的浏览器上设计，但是代价太大。这个方法现在已经很少使用了。

代替的方法是把 CSS 写进嵌入式样式表，在可靠的浏览器上预览，如 Mozilla、Chimera、Netscape 7、IE6/Windows、IE5/Mac 或 Opera 7（仅列举少数优良的浏览器）。用这个方法，你可以制作出可访问性高的、占用很少带宽的、兼容标准的网页。

如果满意你的设计结果，就让这些页面在其他性能优良的、适应兼容标准的浏览器上测试。应该在所有浏览器上的效果都一样，如果不一样，你就要再做些修改，可能有一个 CSS 定义书写得不正确，但是符合标准的浏览器能够理解你的意图，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。

9.4.1 从嵌入样式到外联样式：双表格法

就算你的站点在所有适应的浏览器上都工作正常，并且看上去也不错，也不要浪费时间把它放到老式浏览器上去测试，你应该把 CSS 规则复制到名为 basic.css 的新文件里，再把文件上传到服务器的/css/目录里，然后从网页中删除嵌入式样式表，再把它和文档顶端链接起来：

```
<link rel="StyleSheet" href="/css/basic.css"
      type="text/css" media="all" />
```

清空缓存，退出后重新启动浏览器，再次测试，确保从嵌入式过渡到外联 CSS 时不会不小心删掉了一些定义。

在不兼容标准的浏览器上测试和支持

现在你可以在你想要支持的不兼容标准的浏览器上进行测试了，当然，还是有一点机会在比较差的浏览器上显示更佳效果的。建立一个新的空白文档，命名为 sophisto.css，对破旧的浏览器来说，将那些复杂性可能要求过高的 CSS 规则进行删改，然后把这些复杂的规则复制到 sophisto.css 中，随后把它们从 basic.css 中删除（名称可以是任意的，我们比较喜欢用 sophisto 和 basic，当然可以使用任何更清楚明白的名字）。

将 sophisto.css 上传到你的/css/子目录里，用 @import 把它链接到第二个外联样式表：

```
<style type="text/css"
       media="all">@import "/css/sophisto.css";</style>
```

清空老浏览器上的缓存，上传网页，再看看现在这样的显示是否可以接受，如果不可以，就要从 basic.css 里移动一些定义到 sophisto.css 中，最后你的站点看起来就和在标准浏览器上的一样了，而且在这个非标准的破旧古董上还显示得相当漂亮。

例如，Hillman Curtis 和 Happy Cog 联合设计的 The Fox Searchlight 网站(www.foxsearchlight.com)，就是用双表格法实现的，在所有的浏览器上呈现网站的大部分面貌，面对不能处理复杂样式的 4.0 浏览器，它能隐藏其样式表。虽然站点结构没那么紧凑，但看上去效果也还好，而且方便了使用。最佳方案法使我们摆脱了浏览器版本的限制（见 foxsearchlight.com 网站，并在各种各样的浏览器中做出了比较）。

9.4.2 相对及绝对文件路径

创建初始 CSS 时，我们用的是相对图像文件路径，因为我们是在本机上制作，而不是在演示服务器上制作的。在第 10 章，创建的最终样式表将使用绝对（而不是相对）文件路径，以避免老式浏览器对 CSS 里相对链接的错误理解。

事实上，老式浏览器会混淆 CSS 文件里的相对链接，所以我们要在新版本浏览器上预览，而不是老式浏览器。如果你是离线设计网页，并因此使用相对文件路径，那么老式的（先在老式浏览器上测试）浏览器将自动修补那些相对路径，导致图片不能显示。这样，你可能花上几个小时寻找样式表里的“错误”，而实际什么错误都没有。

9.4.3 最佳方案和双表格法的优点

最佳方案法和双表格法使我们摆脱了浏览器的限制，不必写一大堆没用的代码就能创建一个风格鲜明的商业性混合布局。在满足用户条件的情况下，我们给用户提供最佳的视觉体验，再也不用向用户抱怨他们使用的浏览器的局限性，再也不用在用户的强迫下选择使用 CSS 或 XHTML。

双表格法不是语义标记，不是纯 CSS 布局，更不是支持网页标准的先锋，也绝非设计现代网站的惟一方法，但是它是在过渡期内的一个好方法，能够将标准的好处带给你，正如 As Klaatu 对全世界人们所说的“它并不完美，但是它已经自成体系，它是有用的。”

双表格法不仅用于混合设计，在把纯 CSS 布局释放到各种各样支持 CSS 的浏览器上的时候，也同样有效。无论是用于哪种情况，目的都是一样的：在浏览器或设备最大允许的程度上为每个访问者创造愉快难忘的体验。

我们现在准备用已经学习的 CSS（以及更多）来完成在第 8 章开始创建的站点的布局。在第 10 章中，我们将正确而缜密地完成这些，加入我们吧！

CSS 应用：混合布局（第二部分）

在第 8 章“XHTML 的示例：混合布局（第一部分）”中，我们为 i3Forum 网站编写了混合式标记，结合使用比如<h1>、<h2>和<p>这样的结构元素和非结构元件（用 XHTML 表格布置基本布局格式），用表格摘要、accesskey 和忽略导航使站点在非传统浏览环境里的可访问性更高。

在本章中，我们将使用 CSS 实现支持品牌的设计效果，使站点更具吸引力，无需依赖 GIF 文档、JavaScript 图像滚动效果、间隔用的透明背景的 GIF 像素图片、嵌套表格单元结构或其他以前教授过的又老又旧的网页设计知识。

图 10.1 显示的是首页模板，它是在我们第一遍写完样式表后的效果。和所有的设计一样，用 CSS 进行设计是一个重复的过程。本章里，我们分两次完成 CSS：第一次处理 90% 的需求，第二次纠错和最后润色。



图 10.1

CSS 的第一遍写完后，元素、尺寸、字体、颜色都在适当位置上，但背景没有填满右边菜单“按钮”，所以还要做些修改。

10.1 准备图片

虽然站点是用 Photoshop 设计的，但这里主要不是讲 Photoshop。图 10.2 显示的是建立整个站点的 6 个图片，三个用做前景：宇航员照片、“良种”狗照片和用在菜单条的左上角透明的 GIF 标志。剩下的三个为背景。

Arrow.gif 是源自标志网页背景图像，作为水印放在背景中。Bgpal.gif（如图 10.3 所示）由单色像素和透明单像素交互构成，用于菜单的背景色。Nopat.gif 是纯白色背景，在鼠标划过菜单时，它能代替 bgpat.gif 图片以实现 CSS 滚动效果，也可以通过给每个页面一个内嵌的样式表的方式，指明访问者所在的当前位置（见“不必要的图片”）。

不必要的图片

严格来讲，在站点创建过程中并不需要六个图片，Nopat.gif 纯白背景图片（如图 10.2 所示）是多余的，CSS 规则指定的 background: white 将产生一样的效果（稍后本章将采用这样的 CSS 规则而不用 nopat.gif）。

图 10.2

站点用了 6 个图片，3 个做背景，不需要再用 Photoshop/ImageReady 的切割特性

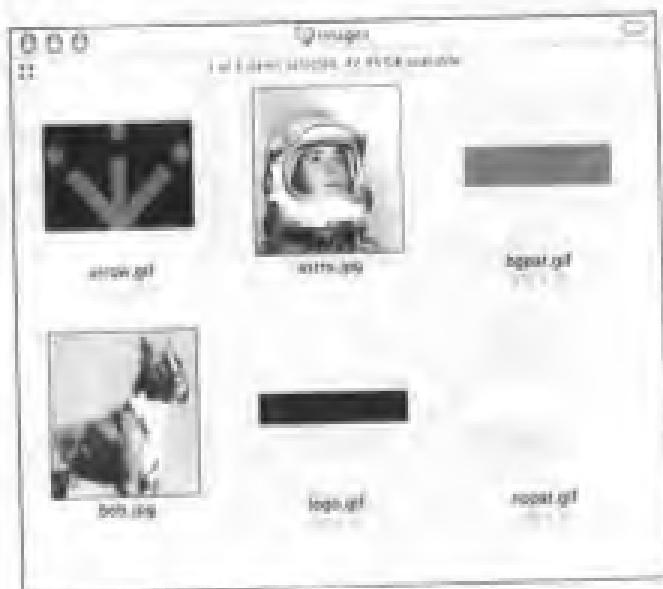
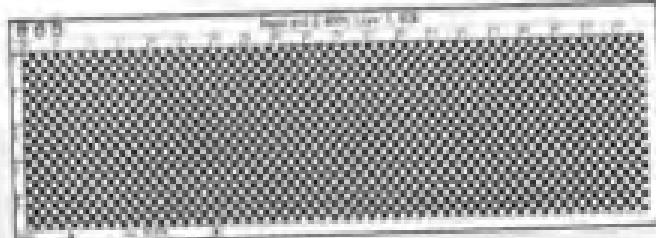


图 10.3

交互像素 GIF 背景图片被放大了 800%，插入黑色背景代替透明像素



稍后你将看到如何用 CSS 实现背景图片交替。如果你希望将水印或有纹理的背景交替，那么就需要两张图片，所需的 CSS 规则将在下一节中介绍。

10.2 设置基本参数

就像前面所说的，准备好图片，开始用 CSS 设置基本设计参数。我们知道网页字体应该是黑色，背景将是白色，文本将采用不同尺寸的 Georgia 字体（或为 serif），标志的水印会定位在内容的背景上，所以必须设置一定的空白属性值，而不能用间隔的透明 GIF 图片或附加的表格块。让我们来试试看。

10.2.1 总体样式，关于缩写和边距的更多内容

在第一个定义中，我们设置了基本页面颜色和上下边距：

```
body {  
    color: #000;  
    background: #fff;  
    margin: 25px 0;  
    padding: 0;  
}
```

每个调用这个规则的文档都将有白色的背景和黑色的内容。第 9 章“CSS 入门”里讲过，在缩写的 CSS 中描述了颜色（#000 是#000000 的缩写；#fff 是#ffffff 的缩写），缩写只能用于代替成对字符，比如#fco 等于#ffcc00。没有成对字符就不能缩写，如#93C7a 这样的非 Web 安全颜色代码就不能采用缩写形式。

缩写与顺时针顺序

第一个规则也设置了上下边距为 25px，左右边距为 0。下面是这个规则的缩写形式：

```
margin: 25px 0 25px 0;
```

完整写法如下：

```
margin-top: 25px;  
margin-right: 0;  
margin-bottom: 25px;  
margin-left: 0;
```

在 CSS 中，值是按照顺时针的顺序指定的：12 点（上边距），3 点（右边距），

6 点（下边距），9 点（左边距）。

若要设置上边距为 5px，右边距为 5px，下边距 10px，左边距为整个宽度的 30%，写法如下：

```
margin: 25px 5px 10px 30%;
```

当上下的垂直边距相等（在这里，比如说是 25px），左右的水平边距也相等时（比如说是 0），如下写法可以节省一些用户带宽：

```
margin: 25px 0;
```

第 9 章中曾提到过，0 值后不需要跟一些计量单位，0px 和 0cm、0in 甚至 0ba 亿万英里其实没有区别（CSS 里没有亿万英里这个单位，就算有，当值为 0 时也用不着）。

最后，我们将设置边框距定义为 0，适应 Opera 字体的需要，Opera 字体使用边框距而不是边距来设置负槽。

10.2.2 隐藏与块

下一步，我们将用两个简单的规则做一些有用的事情：

```
.hide {
    display: none;
}

img {
    display: block;
    border: 0;
}
```

第一条规则创建了一个名为 `hide` 的类别，用来隐藏支持 CSS 的浏览器上的元素或对象，这在第 8 章中已经学过，我们用 CSS 特性隐藏现代浏览器上的忽略导航链接，而对于文本浏览器、屏幕阅读器、PDA 和智能电话来说，不支持 CSS 的浏览器的用户为可显示的。（需要提到的是，一些 PDA 的浏览器也部分支持 CSS，但效果和在 3.0 和 4.0 桌面浏览器上的一样差，希望你读到此书时它们已经改进了。）

图片规则

第二条规则更有用，问题也更少。第一行：`display: block;` 指定页面上的每个图片都作为块级元素而不是内联元素呈现。如果你不熟悉这些术语，举两个简单的例子你就能明白。段落就是块级元素，而`<i>` (*italic*) 标签是内联元

素，块级元素存在其自身的“盒”中，后面跟一个暗藏的回车。内联元素是流的部分，后面不跟随回车。

告诉浏览器把图片按块级元素处理，就不需要在图片的前后写出
或<br clear="all">或类似的语法，也不必一定要把这些图片固定在自身的表格单元里，才能保持布局的间距要求（你将在 11 章“使用浏览器 第一部分：DOCTYPE 转换和标准模式”中学到更多相关内容，如果现在就想了解，可以参阅 <http://deverge.netscape.com/viewsource/2002/img-table/>“图片、表格和神秘的间隙”）。

因为它看起来太简单了，以至于很多人不加思索就跳越过去。但是明确指定一个元素为 block 或 inline 状态的这种工具，具有极其强大的功能。例如，通过 CSS 被制成块级元素的普通链接可以转变成按钮。在下一条规则中，用另外一个选择器，就能给存在于布局特殊部分的图片增加特定的垂直空白，只用几行 CSS 语法就能实现。要是用标记的话，就要用到一大堆的切片、方块、表格单元嵌套和 GIF 间隔图片。

再明确一遍，我们将告诉你：怎样用几行标记代码和少许 CSS 规则代替一个由 50 个表格单元和许多剪辑图片才能构成的布局。

接下来，使用 border : 0; 声明关闭图片的边界，这样就不必在标记里写 border="0" 了。（如果你很在意站点在非 CSS 浏览器上的效果，就必须在标记里写 border="0"，至于原因，可参见第 8 章内容）。

10.2.3 给链接加上颜色（推荐用伪类属性）

在外部 HTML 里，我们通过给 body 元素指定 vlink="#cc3300"，控制链接的颜色。在现代网页设计中，我们可以不修饰 body 而采用 CSS。为了达到更好的效果，CSS 给常见的链接添加一个“鼠标停在上方”的状态。至于访问过的状态和已激活的状态，我们在 20 世纪 90 年代就已经学过了。使用 CSS，不仅能改变链接的颜色，还能做很多其他的工作。

CSS 称这些锚（链接）状态为伪类选择器。在 CSS 的思考方式里，“真”类属性是用 class 的属性来明确指定，伪类依赖于用户或浏览器的活动状态（:hover, :visited）。下面的元素也是伪元素（:before 和 :after），在任何情况下，都可以使用下面四条规则来控制链接的颜色和其他元素（如图 10.4、图 10.5 所示）：

```
a:link {  
    font-weight : bold;
```

```

        text-decoration : none;
        color: #c30;
        background: transparent;
    }
a:visited {
    font-weight : bold;
    text-decoration : none;
    color: #c30;
    background: transparent;
}
a:hover {
    font-weight : bold;
    text-decoration : underline;
    color: #f60;
    background: transparent;
}
a:active {
    font-weight : bold;
    text-decoration : none;
    color: #f90;
    background: transparent;
}

```

图 10.4

链接颜色为暗红色，粗体，无下划线。CSS 定义：a-link rule

Sign up for the i3Forum Update newsletter and watch this space
for what comes next.

图 10.4

当用户的鼠标箭头停在链接上方时，它的文档颜色以橙色变亮，出现下划线。使用的 CSS 定义：a-hover rule

for the i3Forum Update newsletter and watch this space
for what comes next.

关于链接和伪类别选择器的更多内容

前四条规则中，你惟一觉得陌生的可能是文字修饰属性。当它的值为 `none` 时，链接无下划线；当值为 `underline` 时，链接有下划线，就像 20 世纪 90 年代中期的所有链接一样。当值为 `overline` 时，直线出现在文档的上面而不是下面。你可以将上划线和下划线组合使用，如下：

```
text-decoration: underline overline;
```

效果如何呢？看上去被链接的文档好像是装在一个有顶有底但是没有边的盒子里。我们了解到两个网站的设计师（也包括我们自己）不只一次用过这种效

果。在离开我们最可爱的链接之前，还有两点需要注意。

用 LVHA 帮助记忆

记住这些：一些浏览器会忽略一条或更多的锚元素伪类规则，除非它们早已按次序列出来，比如链接的状态：链接、已访问的链接、鼠标停在上方的、激活（LVHA），若要改变这四个状态的次序是很危险的。记住次序的口诀是“LoVe-HA！”（很多人很头痛这个）。如果你想知道次序为什么这么重要，请访问 <http://www.meyerweb.com/eric/css/Link-specificity.html>，那里解释得很清楚。

IE/Windows 的伪欺骗

注意即使最新、最人性化的 Windows Internet Explorer（直到写此书时）对于鼠标停在上方和激活伪类别属性也都产生问题，单击 IE/Windows 的返回按钮后，很可能你会发现鼠标最后激活的那个链接还处在“鼠标停在链接上方”的状态，而且你已单击的链接还处在激活状态。你肯定会很讨厌这样的小问题。

也许会因为激活的颜色而受骗，如果你给 `a : active` 伪类属性指定背景图片的话，IE/Windows 会错误地理解它们。如果你的访问者包括大量的 IE/Windows 用户，就要完全避免使用 `a : active`，或者你可以像我们一样不选择它来做一些极富创造性或挑战性的工作。

无论链接状态的冻结是不是 bug，两亿 IE/Windows 用户可能现在已经习以为常了，很多人以为网络就是这样工作的。

10.2.4 其他一般元素示意图

下面显示的代码组中，我们看到了我们的好朋友，“Netscape 4 友好”规则（第 9 章），它被用来告诉浏览器整个网站的字体为 Georgia 或一个可选择的 serif 字体（如图 10.6 所示）。

```
p, td, li, ul, ol, h1, h2, h3, h4, h5, h6 {  
    font-family: Georgia, "New Century Schoolbook",  
    Times, serif;  
}
```

Georgia 是由获得 Type Directors Club (TDC) 奖章的 Matthew Carter 设计的微软屏幕显示字体，就算字形很小也清晰易辨，几乎在所有的 Windows 和 Macintosh 系统里都能找到它。New Century Schoolbook 存在于大多数的 UNIX 系统里，Times 存在于 Paleolithic 计算系统里。如果以上字体都没有，我们还有

一个好朋友，最普遍的 serif 字体。

图 10.6

用一条指定多重选择器的单独规则来指定字体（p, td, li, ul, ol, h1, h2, h3, h4, h5, h6），由额外的规则和选择器控制大小和空行属性

Welcome to i3Forum – a celebration of inspiration, innovation, and influence.

Industry leaders don't spend Monday through Friday looking forward to the weekend. They're self-starters and highly motivated doers who like to mainline what's hot. We created i3Forum for them.

Once a year, members of this select group gather to discuss their work and the things that move them. It's all about sharing ideas and sparking new initiatives. The next i3Forum event will be held in Laguna, California on 69 April at the Chateau Grande.

If you're drawn to the kind of people who invent fire, then share insight about how and why they developed it—if you seek the spark of being among other original thinkers at least once a year—then i3Forum might be for you.

Sign up for the i3Forum Update newsletter and watch this space for what comes next!

在下面的规则中，我们告诉浏览器，标题字的字号应该稍大于用户的字号默认值。当没指定基本字号时，浏览器将采用用户字号默认值（1em）（如果用户字号默认值为 12px 或 48px 也没关系，浏览器的字号仍然会是 1em）。为了使标题字的字号稍大于默认的字号，我们将其设置为 1.15em，但保持一般字体权重（非粗体）不变：

```
h1 {
    font-size: 1.15em;
    font-weight: normal;
}
```

因为前面已经定义过了，所以不需要再告诉浏览器，h1 应该被指定为 Georgia 字体。

现在我们用 html 选择器给 p 样式添加更多的细节。除了 html 页面本身，该页面上的所有元素都是 html 子元素。我们很容易写成 p 而不是 html p。看看下面的规则及之后的说明：

```
html p {
    margin-top: 0;
    margin-bottom: 1em;
    text-align: left;
    font-size: 0.85em;
    line-height: 1.5;
}
```

在上述规则中，我们创建了顶部没有空白（这样可以使它们很好地紧贴在标题和副标题的底部）、底部空白为 1em（为了避免相撞）的段落，字号（0.85em）稍小于用户字号默认值（用早先用于 h1 的推理，但方向相反）。

10.2.5 关于字号的更多内容

正如我们在上面的规则中所做的那样，指定相对字号是比较明智的。特别是应该指定的字号比用户默认的稍小一点，因为用户可能设置的字号默认值很小，这样，你的文档使用小字号对用户来说就太小了。

例如，用户设置的字号默认值为 11px 的 Verdana，你的文档字号就可能只有 9px 或 10px，对于阅读量大的读者来说看这么小的字是很不舒服的。用户可以使用自己浏览器的控制字号部件来调整布局，但有些人觉得这样做很麻烦。另外，一些人甚至不知道文档的字号是可以调整的。

绝大多数系统出厂时的字号默认值都很大，0.85em 这样大小的字看起来很舒服。如果用户的视力不好，则可以把字号设置的比字号默认值大得多，这样的字对他来说还是不错的，他会看见比正常字号稍小的字。但是如果 Windows 用户选了“小”字号作为浏览的默认字，或者如果 Mac 用户把字号设置成 12px/72ppi，我们的文档看起来就会太小了，使用户的眼睛发痛，或者更有可能是干脆很快地退出网站。

我们可为文档大小选择一个像素值：

```
font-size: 13px;
```

用简捷的 CSS 创建主要部分：

```
font: 13px/1.5 Georgia, "New Century Schoolbook", Times, serif;
```

与 em 做字号的单位不同，对于所有的浏览器和操作平台，以像素为单位的可靠性为 99.9%。如果对于特定用户字号单位像素太小的话，他可以通过使用每个现代浏览器上的文本缩放或 Page Zoom 自己进行调整。但是有一个浏览器没有这种功能，真不幸，那就是 IE/Windows 浏览器——目前大多数用户使用的浏览器。

具有特别的讽刺意义的是，文本缩放（Text Zoom）是 2000 年早期在微软浏览器（IE5/Mac）上发明的，但这个极其有用的特性却还没在 Windows 版本实现。

这就意味着，你如果想用像素为单位安全地控制字号，就要冒视力不佳的 IE/Windows 用户不能阅读你的文档的危险。但如果你想要避免用 em 做单位所带来的问题，比如我们在本章建立的站点上做的那样，就会使已经把字号默认值缩

小了的用户很沮丧，但出厂的字号默认值对大多数人来说实在是太大了。

总之，无论你怎么做，总会使一部分人失望。我们曾经设计了一个不设置字号的网站，原以为这下所有人都满意了，但恰恰相反，网站收到了上百封信抱怨文档的字号“太大”，有关字号问题的详细内容见第 13 章。

关于行高的疑问

再看看现在正在讨论的问题——规则中的行高：

```
line-height: 1.5;
```

`line-height` 是 CSS 对行距的称呼，行高为 1.5 与行距为 150% 意思是一样的，行高也可以标记为 1.5em，但是没有这个必要。

使用 CSS 之前，我们只能通过使用段落标签的非结构用法（见第 1 章里 [suck.com](#) 上的讨论，“99.9% 的网页是过时的”）：要么使用`<pre>`，要么仅仅依靠每行文字中间的 GIF 间隔图片，强迫文档插入有绝对宽度的表格单元，以得到行距，然后祈祷将不可见的间隔像素图片无缝地下载下来。如果不动，访问者就会看见破损的 GIF 占位符图片，而不是看到漂亮的行距。不过 CSS 已经永远地解决了这些问题。

左对齐问题

最后，回到前面的定义上来。可能你会奇怪，为什么我们要指定文本的对齐位置为左对齐。答案很简单，如果不这样指定，IE6/Windows 的 bug 就会使文档居中，IE5/Windows 上不会发生这样的错误，其他浏览器也不会，这种行为是随机的。很多在 CSS 里并没有说明左对齐的元素在 IE6/Windows 里也能正确地显示为左对齐，但有些则不能。你不知道哪些会正确显示而哪些又不会，所以一律把它们指定为左对齐，从而避免出现这样的问题。

设置页脚

现在，用 CSS 术语，你就能看懂下面的一条小型规则：

```
#footer p {
    font-size: 11px;
    margin-top: 25px;
}
```

不用解释你也知道，它用了 footer 的惟一 id 属性作为选择器，footer 里的所有段落字号被设置成 11px，顶端都有 25px 的空白。

浏览器知道使用什么字体，因为页面定义中已经指定过了。

10.2.6 设置页面分界线

下面我们要为设置基本页面分界线书写一套规则。在 CSS 文件里，把许多定义放在一起以便编辑和重新设计，在规则前我们添加了注释：是因为需要提醒自己，或者向后来修改我们样式表的同事说明这套规则的作用。这里的注释和 HTML 里的注释是一样的，但和基于 C 语言编程的稍有不同。

注释的后面，我们有一条规则，是用来指定主要内容区域的左边和上面的空白为 25px 的（如图 10.7 所示），另一条规则在底部和 id 属性为 content 的表格中间设置一个不重复的图形背景图片（arrow.gif）（如图 10.8 所示）。



图 10.7

导航区和主体内容之间的垂直间距，以及工具条图片区和文档主体内容之间的水平空白，都是由一条指定主要内容选择器的单行规则来控制的。不需要间隔 GIF 图片或表格单元编排

图 10.8

从标志中截得的返回箭头图片，控制着内容区，并由于用于内容选择器的背景声明，从而用做它的水印。因为选择器包含整个表格，箭头可以横跨两个表格单元（工具条和主要内容）。效果很简单，但用以前教授过的方法要达到这样的效果，要么是不可能，要可能也十分困难

通过给表格单元指定背景图片来取得这样的效果，即使可能也会很难。因为背景图片必须横跨两个表格单元，这样的话，我们必须把背景图片切割成一片一

片的，把每片指定给不同的表格单元，还得指望所有的切片都排列整齐。

HTML 中的背景图片标签被默认地重复显示，而且没办法避免这样的重复。所以，我们必须做出两个严格和包含它们的表格单元一样高的透明图片。希望用户不要调整文档的大小，要不然表格单元的高度就会超出对齐线了。

除此以外，每个页面的高度都必须相同，这样可以限制访问者在每个页面中能增添或减少的文档数量，访问者对此并不欢迎。

使用 CSS 后，我们不用再考虑以上那些问题了，与我们书写表述它们的时间相比，阅读和理解规则的时间要少得多。

```
/* Basic page divisions */
#primarycontent {
    padding-left: 25px;
    padding-top: 25px;
}
#content {
    background: transparent url(images/arrow.gif) center
    bottom no-repeat;
}
```

下面是我们给附栏建立的规则：

```
/* Sidebar display attributes */
#sidebar p {
    font-style: italic;
    text-align: right;
    margin-top: 0.5em;
}
#sidebar img {
    margin: 30px 0 15px 0;
}
#sidebar h2 {
    font-size: 1em;
    font-weight: normal;
    font-style: italic;
    margin: 0;
    line-height: 1.5;
    text-align: right;
}
```

第一条规则说明，id 属性为 sidebar 的唯一元素内的段落内容将右对齐，字体为斜体，有大小为字高度（0.5em）一半的上边距（空白）。如果觉得以上那条规则有些眼熟，是因为在第 9 章对 CSS 的 id 选择器讨论时，它曾经出现过。

第二条规则指定 id 属性为 sidebar 的唯一元素内的图片，上边距为 30px，

下边距为 15px，左边和右边都没有额外的空白（如图 10.9 所示）。这条规则我们曾在本章的“图片定义”那一节间接提到过，现在做详细的解释。



图 10.9

通过给 sidebar img 指定上下边距值，可以有经过仔细挑选的、工具条图片上方和下方的垂直空白值。标签为 sidebar 的及 div 内的图片将遵守这些值，但其他地方的图片也会受到这些值的影响

使用这样的规则，我们再也不需要用多重空表格单元和间隔 GIF 图片制作空白了（你能想像吗，其他的媒体强迫设计师花费大量时间仅仅是为了制作空白，我们已经不再用那种方式建立网站了）。

本节的第三条规则使 h2 标题看上去更像杂志的标题而不像 HTML 标题（如图 10.10 所示）（特别是在 Windows Cleartype 和 Mac OSX Quartz 这种平滑的文本环境里）。它指定 sidebar 里的文档适中字号（1em）、一般权重、斜体、右对齐且具有与本页面中的其他文档相同的行高值。



图 10.10

附栏标题标记为二级标题（h2），然而看起来像杂志标题而不像 HTML 标题

10.3 导航元素：第一个步骤

到目前为止，我们做得都很好。我们书写的每条规则都用符合标准的、最好情况的浏览器测试，并显示出我们所期望的效果。正确地使用导航条很重要，在下面的第一个步骤里，我们把注意力集中在一些期望的特性上：

```
/* Navigation bar components */
table#nav {
    border-bottom: 1px solid #000;
    border-left: 1px solid #000;
}
```

上述规则说明，`id` 属性为 `nav` 的表格在底部和左边（不包括顶部和右边）制造一个 `1px` 的黑色实心边界效果。

```
table#nav td {
    font: 11px verdana, arial, sans-serif;
    text-align: center;
    vertical-align: middle;
    border-right: 1px solid #000;
    border-top: 1px solid #000;
}
```

不用解释你也知道，定义指定 `11px` Verdana 为菜单文档的优先字体，还添加了前面定义里忽略的对边界元素的设置（也就是右边和上面的元素）。如果我们告诉表格在四条边都制造边界效果，那么表格单元必须给上面和右边的边界增加一个像素，这样看起来效果不好。

前面的规则也说明文档在每个表格单元里水平居中，在每个表格单元的中间垂直对齐，很像我们以前学过的外观编程 `td valign="middle"`（这里似乎应该这么写，但是在第二个步骤，我们要把它从定义中删除）。

```
table#nav td a {
    font-weight: normal;
    text-decoration: none;
    display: block;
    margin: 0;
    padding: 0;
}
```

前面的规则中，你可以看出我们让链接怎样响应行为，你还能看出我们是用一系列复杂的内容选择器来完成这个工作的。多重部分选择器的意思是：下面的定义只指定表格单元内的链接，并且此表格的唯一标志符为 `nav`。

正如我们在讨论“图片定义”时提示的一样，我们也用 CSS 的 `display: block` 声明，把低级别的 XHTML 链接转变成完整填充它们表格单元的块级元素（至少希望它们能完整填充表格单元）。

```
#nav td a:link, #nav td a:visited {
    background: transparent url(images/bgpat.gif) repeat;
    display: block;
```

```
margin: 0;  
}  
  
#nav td a:hover {  
    color: #000;  
    background: white url(images/nopat.gif) repeat;  
}
```

最后的两条规则用内容和 `:link` 选择器控制链接、已访问的链接和“鼠标停在链接上”的伪类属性，用交替像素背景颜色图片[见图 10.3]填充前两个类别属性，用纯白色的背景图片填充鼠标停在链接上/图片滚动状态（见“不必要的图片 II：这次它是个人的选择”）。

不必要的图片 II：这次它是个人的选择

还记得我们在前面的补充说明“不必要的图片”里说过，纯白色背景对于执行来说确实不必要吗？的确，纯白色背景对这次执行来说确实是不必要的。

不要用这个：

```
#nav td a : hover {  
    color : #000;  
    background : white url(images/nopat . gif) repeat ;  
}
```

我们应该书写这样的规则：

```
#nav td a : hover {  
    color : #000 ;  
    background-image : none ;  
}
```

把图片从鼠标停在链接上的状态 (`background-image : none;`) 中删除，将制作出和纯白色背景一样的效果，并且少用一张图片，从而节省了一点带宽。在本章的后部分，当你为个人主页制作“you are here”效果的时候，我们就不用再依赖纯白色的 GIF 背景了。

因为 CSS 背景图片的交换棒极了，而且你也许希望利用它们的强大功能完成自己的项目，所以本章我们用它们制作了导航条图像滚动效果。

再看看图 10.1，它显示出了我们在设置网站样式时第一个步骤上所有正确的和错误的做法，除了导航条，我们想要的都有了：logo 没问题，背景颜色完整地填充到了（如图 10.11 所示）图片里，鼠标经过链接时显示的图像滚动效果（如图 10.12 所示）和我们的期望一样。

图 10.11

启用 CSS 图像滚动效果，通过给 id 属性为 nav 的表格内的表格单元指定链接。已访问的链接及鼠标停在链接上伪类属性来实现。这里我们看到的是一个菜单图形的默认状态



图 10.12

CSS 图像滚动效果，第二部分：当访问者的光标停在菜单图形上时，背景变为白色。看，我们没用 JavaScript！（并不是哪里出错了）



但是默认（链接、已访问）状态的背景图案（见图 10.1）只填充了每个右边菜单项的一部分，而我们是要它填充整个空间，仅填充一部分是不够的。我们首次尝试用 CSS 方法最终解决这个问题，但随之又产生了新的问题。

10.4 CSS 导航条：在第二个步骤的第一次尝试

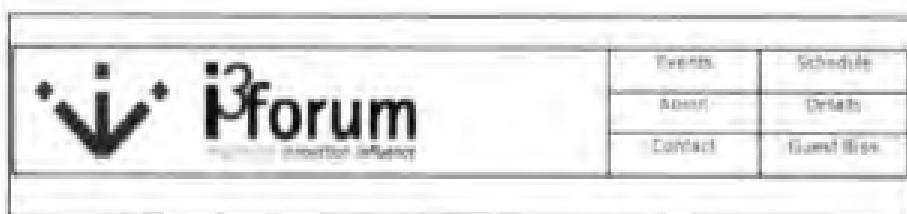
在第二个步骤里的第一次尝试中，我们指定了链接效果的尺寸：

```
#nav td a:link, #nav td a:visited {
    background: transparent url(images/bgpatt.gif) repeat;
    display: block;
    margin: 0;
    width: 100px;
    height: 25px;
}
```

正如我们预期的那样，右边的按钮得到完整的填充，但是标志却被弄乱了，标志背景也变成了 100×25pixels（如图 10.13 所示）。100×25 对于小按钮是合适的值，但是却不适合标志（应该是 400×75）。

图 10.13

前进一步，后退两步：指定 nav 伪类的尺寸能完整地填充标志的背景尺寸，但也破坏了我们前面设置的垂直对齐。文字内容现在变成了紧贴每个单元的顶部



这种修改也破坏了我们前面设置的垂直对齐，元素在它们的单元里是垂直对

齐的，但内容却不垂直对齐。在图 10.13 里可以看到，“按钮”文档紧贴在每个表格单元的顶部而不是在中间垂直对齐。在所有的 CSS 兼容浏览器上都呈现这种紧贴顶部的效果。这不仅仅是个 bug，而且是个出乎意料的脱离 CSS 布局模式的行为。

值得庆幸的是，在第 8 章建立标记时，我们给每个表格的单元都指定了惟一的标志符。`home` 为包含标志单元的 `id` 属性。我们能不能书写另一套规则来制约用以填充 `100×25` 按钮的定义呢？当然是可以的：

```
td#home a:link, td#home a:visited {
    background: transparent url(images/bgpat.gif) repeat;
    width: 400px;
    height: 75px;
}
td#home a:hover {
    background: white url(images/nopat.gif) repeat;
    width: 400px;
    height: 75px;
}
```

这些新的规则能正确地填充标志，使站点近乎完美，但是它还是没有实现我们想要的 `vertical-align: middle`，我们将在最后一个步骤来修正这个问题。

10.5 CSS 导航条：最后一个步骤

在最后的一个步骤中，我们实现了所有想要的效果：

```
/* Navigation bar components */
table#nav {
    border-bottom: 1px solid #000;
    border-left: 1px solid #000;
}
table#nav td {
    font: 11px verdana, arial, sans-serif;
    text-align: center;
    border-right: 1px solid #000;
    border-top: 1px solid #000;
}
table#nav td a {
    font-weight: normal;
    text-decoration: none;
```

```

        display: block;
        margin: 0;
        padding: 0;
    }
    #nav td a:link, #nav td a:visited {
        background: transparent url(/images/bgpat.gif) repeat;
        display: block;
        margin: 0;
        width: 100px;
        line-height: 25px;
    }
    #nav td a:hover {
        color: #f60;
        background: white url(/images/nopat.gif) repeat;
    }
    td#home a:link img, td#home a:visited img {
        color: #c30;
        background: transparent url(/images/bgpat.gif) repeat;
        width: 400px;
        height: 75px;
    }
    td#home a:hover img {
        color: #f60;
        background: white url(/images/nopat.gif) repeat;
        width: 400px;
        height: 75px;
    }

```

看看发生了什么变化，我们把 vertical-align: middle 指令删除了。然后把指定按钮为 25px 的那行定义也删去，用下面这行代码来代替：

```
line-height: 25px;
```

行高已经被设置成了 25px，这能使文档准确地定位在每个按钮的垂直中间位置。只有 CSS 大才能说清楚为什么用这个方法比另外那个方法效果好得多。关键是，这个方法的确起作用。

10.6 最后一个步骤：外联样式和“*You Are Here*”效果

要把站点交付给客户，还需要两个步骤。第一，我们必须把嵌入样式转换成外联 CSS 文件，并删除嵌入样式表，这点在第 9 章已经说明过了。然后创建一个“*You Are Here*”效果（如图 10.14、图 10.15 所示），帮助访问者了解他正在

浏览哪一页。记住：我们不会更改标记，我们需要采用 CSS 来改变效果，同时也不能给导航条增加额外的类。



图 10.14

在“事件”页面模板上的“You Are Here”效果



图 10.15

在“关于”页面模板上的“You Are Here”效果

“You Are Here”效果很容易实现。因为我们已经删除了嵌入样式，每页模板就要通过链接一个如下的嵌入样式表来获取它的CSS数据：

```
<!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>i3forum</title>
    <link rel="StyleSheet" href="/css/i3.css" type="text/css"
      media="all" />
```

我们需要做的全部事情就是用样式元素给每一页都添加一个嵌入样式表。该样式表只包含一条规则——转变该页面“菜单按钮”普通外观的规则。

在“事件”页面上，嵌入规则按如下写法，重要的选择器用粗体突出表示：

```
<style type="text/css" media="screen">
  td#events a:link, td#events a:visited {
    color: #c30;
    background: #fff;
  }
</style>
```

注意：按照之前的补充说明“不必要的图片 II”里的方法，我们用的是background : #fff；而不是nopat.gif来制造纯白色背景的高亮效果。

在“关于”页面上，嵌入规则的写法如下：

```
<style type="text/css" media="screen">
  td#about a:link, td#about a:visited {
    color: #c30;
    background: #fff;
  }
</style>
```

站点的每个页面都包含一条这样的规则。这样，不需要改变一行标记，每个页面都能告诉访问者“You Are Here”。你也许会问，“为什么从一个页面到另一个页面时不需要更改标记呢？为什么不为‘You Are Here’提示创建一个thispage类别属性呢？”这样做也是可以的，而且经常有人这样做。

但是，从一个页面到另一个页面时不改变导航标记，使得通过用户端一遍又一遍地插入相同的数据成为可能，这是一个对小型或中型站点（我们现在正在创建的就是中型站点）都很实用的方法。

在www.zeldman.com上，我们用这个技术逐页修改纯CSS导航条的“You Are Here”提示，而用SSI技术给每个页面插入相同的XHTML数据，但这是另

外一个网站和另外一回事了。下一章我们将更深入研究 CSS，学习浏览器 bug 和工作区，揭示浏览器做的正确（而我们却并不需要）的行为，讨论阻止我们使用 Web 标准的一些难处理的元素。

使用浏览器 第一部分： DOCTYPE 转换和标准模式

现代浏览器是怎么“知道”你创建的是一个向后兼容的站点呢？它在支持那些用过时方法创建的站点的同时，怎么能正确显示你的标准化站点呢？

答案就是，现代浏览器用了我们的老朋友 DOCTYPE。我们在第 6 章“XHTML：网站重构”中已经介绍过 DOCTYPE 了。DOCTYPE 是把标准模式（符合 W3C 规范的网站就是标准模式）和向前兼容的 Quirk 模式（这样命名是因为老式的站点支持各种各样不兼容浏览器的 Quirk 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。

有趣的是，基于 Gecko 浏览器，如 Mozilla 和 Netscape 上的标准模式和 Internet Explorer 上的标准模式工作方式稍有不同，这些细微的差别将对你的布局产生无法预计的深刻影响。为了解决潜在的问题，后来的 Gecko 浏览器（诸如 Mozilla1.01 和 Netscape 7）都增加了第三个模式，它的工作方式更像 IE 标准模式。（更妙的是，Opera 7 也支持了 DOCTYPE 转换功能，即便大家还不太清楚它是怎么工作的。接下来本书会讲到这个功能。）

Gecko 浏览器把类似 IE 的第三个模式称为“近乎标准”模式，这个名称不是意欲贬低 IE 的标准模式——至少我们不认为这是贬低——但它揭示了：在 Gecko 专家的眼中，我们大多数人从浏览器上得到的视觉行为与真正的 CSS 规范标准是不一样的。

本章叙述了各种各样的模式是怎样工作的，并提供一个简单的方法使你的站点实现期望的视觉效果，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。如果你已经把一个过渡式站点（表格加 CSS）转变成了 XHTML，你就会发现，转变后的一些浏览器改变了你的布局，你可能很奇怪为什么浏览器会

改变布局呢，怎么做才能改回来？本章将向你详细说明。

“DOCTYPE 转换的传奇故事”一节详细解释了为什么大多数浏览器需要一个联系纽带，从而实现标准模式和 Quirk 模式间的转换，为什么采用初级的 DOCTYPE 作为转换纽带。如果你想知道这些东西是从哪里来的，以及为什么它们是这样的，继续看下去就会有答案。如果你希望跳过背景介绍，直接看技巧和技术，那你可以跳过几页，直接从“控制浏览器性能：DOCTYPE 转换”一节开始学习。

11.1 DOCTYPE 转换的传奇故事

20世纪90年代后期，处于领先地位的浏览器制造商认识到，对于设计和创建网站的客户来说，对 Web 标准的完全精确的支持尤为重要，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，不要损坏老的非标准的网站。

毕竟，Netscape 和 Explorer 的已有版本说服了大批设计师学习他们特有的 Quirk 模式，包括对 HTML 标记的私有扩展，以及错误执行 CSS，还有浏览器制造商自己定制的脚本语言。微软和 Netscape 愿意对 Web 标准提供更好的支持，但前提是不损坏现有的、价值上亿的网站，否则对于浏览器制造商来说，等于自杀。

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 的早期版本开始部分支持 CSS1 时产生了一些错误，如盒模型（第 12 章“使用浏览器 第二部分：盒模型、Bugs 和工作区”中将详细说明），第一个方案总是很不完善的，于是有人建议微软公司在 1997 年初开始支持 CSS。

微软早期对 CSS 盒模型的错误理解产生了一个问题：几万名设计师已经“学会”了被 IE4 和 IE5 使用的错误盒模型，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能更精确地支持盒模型，那么那些已完成的设计肯定要土崩瓦解了，这样的话，客户、网站建造者和用户都不会高兴的——怎么办呢？

11.1.1 接受标准或拒绝标准间的转换

在微软和 Netscape 未同意制造更精确、完全支持标准的浏览器之前，有个无名英雄已经解决了这个问题——处理用非标准方法建立的网站。那个英雄就

是用户界面技术专家 Todd Fahrner，他是 W3C 的 CSS 和 HTML 工作组的投稿人，还是 Web 标准组织的共同创建者。你会发现 Fahrner 这个名字在本书里到处可见。

1998 年初，在一份晦涩的 W3C 邮件发送清单上，Fahrner 给浏览器制造公司推荐了一种转换机制，这种机制可实现打开或关闭标准适应呈现之间的转换，他建议根据 DOCTYPE 的存在与否来控制接受和拒绝的转换。

如果网页的标记以一个 DOCTYPE 开始，这就说明设计师可能知道 Web 标准，并且已经做出努力以适应 Web 标准。浏览器应该根据 W3C 的规则分析这样的网页，相反，不存在 DOCTYPE 的网页不能通过 W3C 的校验测试 (<http://validator.w3.org>)，而且这样的网页极有可能是使用老方法创建的，浏览器应该区别对待网页，也就是说，他们应该用老式的非适应性的浏览器显示这类网页的方式显示这些网页。

但还是存在一个问题：没有兼容标准的浏览器。如果 DOCTYPE 转换是向前及向后的兼容的关键所在，那么全世界的用户就还要等上两年（但你甚至不用等两分钟，看下一段）。

11.1.2 放弃开关转换

2000 年 3 月，微软公司推出了 IE5 Macintosh 版，它是由微软工程师和 W3C 的发烧友 Tantek Celik 开发的，它的表现层引擎对网络标准（包括 CSS1、XHTML 和 DOM）提供相当精确和近乎全面的支持。IE5/Macintosh 采用文本缩放以提高可访问性，它是第一个用 DOCTYPE 转换来实现 Quirk 模式和标准模式之间的转换的浏览器。

当那些忙于自己的革新并准备推出他们自己的适应标准的浏览器的开发商，以及在基于 Gecko 的浏览器上进行开发的工程师们，他们一看见这种方法时（DOCTYPE 转换和文本缩放）就明白这是两个非常有用的功能。因此继 IE5/Mac 之后不久推出的 Netscape 6、Mozilla、Chimera 的 Gecko 浏览器，都包含了 DOCTYPE 和文本缩放功能。通过 Gecko/Mozilla 表现引擎，使得支持严格具体的 Web 标准成为可能 (<http://www.mozilla.org/newlayout/>)。

IE6/Windows 加入兼容标准的行列后，也同样支持 DOCTYPE 转换，并增加了一个 DOM 特性，通过一个已有的 Web 文档，控制是否按照标准模式进行显示（<http://msdn.microsoft.com/workshop/author/dhtml/reference/properties/compatmode.asp>）。

不要在 Opera 上转换

7.0 版本之前，Opera 软件公司的 Opera 浏览器不是根据这些定义显示的。无论该网页怎样被设置，它总是尝试用兼容标准模式来呈现网页。7.0 以前的 Opera 版本也不支持 W3C DOM；这样，在老版本 Opera 浏览器上的基于 DOM 脚本的标准兼容网站就不能正常工作。好在 Opera 用户是一个自选的群体，当有新改进的 Opera 版本推出时，他们就立即将它下载下来。前面提到过，现在 Opera 7.0 包含 DOCTYPE 转换功能了，虽然公司并没有说明它是怎样工作的（它的工作方式和 IE 的 DOCTYPE 一样，还是和 Mozilla/ Netscape 的一样，又或者用第三种方式工作呢？我们还要等等再看）。

11.2 控制浏览器性能：DOCTYPE 转换

前面说过，大多数现代的浏览器用是否存在特定的 DOCTYPE 来实现严格兼容标准、容错的、向前兼容的不同模式之间的转换。浏览器的实现和细节方面有差异，1998 年，当适应性标准浏览器在开发商眼中还微不足道时，DOCTYPE 转换功能的要点就出现在 Todd Fahrner 所做的大纲后面。其工作原理的简单概述如下：

- 一个包括完整 URI（完整的网址）的 XHTML DOCTYPE 告诉 IE 和基于 Gecko 的浏览器以标准模式呈现网页，根据 W3C 的规则执行 CSS 和 XHTML。一些完整的 HTML 4 DOCTYPE 也触发标准模式，从现在起，我们将要讨论的一些网页就是这样的，在标准模式里，浏览器假设你知道你自己在做什么。
 - 用一个不完整或过期的 DOCTYPE，或根本就不用 DOCTYPE 时，同一浏览器进入 Quirk 模式，这种模式假定你（也许是正确的）书写的都是过时的、残缺的标记，并且含有针对某种浏览器细节的非标准代码。在这种情况下，浏览器尝试用向前兼容的方式分析你的网页，假设你的 CSS 显示效果和在 IE4/5 上显示的一样，并使用浏览器私有的 DOM。
- 要控制浏览器采用哪种模式，我们要做的就是插入或者完全省略一个 DOCTYPE，就这么简单。

11.2.1 萨拉姐妹的三个模式

我们在本章的后面再解释原因，最新的基于 Gecko 的浏览器版本在三个模

式之间实现转换：Quirk 模式（前面描述过的）、近乎标准模式（在 IE 上等于标准模式）、标准模式（和 IE 的标准模式在一两个关键的细节上有细微的差别）。同样，在早期浏览器版本上触发标准模式的 DOCTYPE，变为触发后来的浏览器上近乎标准模式的 DOCTYPE。

没有经验的人看这段可能会觉得非常迷惑和头痛。在下面几页，你会觉得更加混乱。尽管如此，三种模式还是合理的、重要的。用各种各样的 DOCTYPE 转换方式，不仅可以保证良好的、简单的 workaround 及页面正确的显示，而且还保证了像 Opera5/6 这种根本不用 DOCTYPE 转换的浏览器也能正确地显示页面。

在我们探讨浏览器差异的好处之前，先来看看浏览器有什么共同的特点。

11.2.2 完整和不完整 DOCTYPE

使用 DOCTYPE 转换的浏览器会寻找完整的 DOCTYPE，也就是说，它们寻找包括完整网址（如 <http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>）的 DOCTYPE。下列的 DOCTYPE 是很多现代浏览器采用的触发标准模式：

```
<!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 过渡方法。但这里我们用严格的 DOCTYPE，是为了避免出现一大堆的错误警告，错误警告总是来得又快又多。我们想要说的是：使用一个就像上面这样完整的 DOCTYPE 将能触发标准的模式。

在我们编写开发的很多程序中，总被默认地插入不完整的 DOCTYPE，从而触发向前兼容的 Quirk 模式而不是触发我们想要的标准模式。例如，很多制作工具插入 DOCTYPE 的方式如下（这个是它们从 W3C 网站上获得的）：

```
<!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网址。其实，它是链接到 W3C 网站的 DTD 文件的一个相对链接。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上任何东西。因为 DTD 文件存在于 W3C 网站而不是你的网站上，因为

URI 是相当不完整的，所以浏览器只能以 Quirk 模式显示你的网站

(当前浏览器会尝试查找并下载 DTD 文件吗？不会。它们仅仅识别自己数据库中的不完整 DOCTYPE，然后进入 Quirk 模式，这意味着 w3.org 上用这个相对于 URI DOCTYPE 的网页也会以 Quirk 模式呈现吗？看起来像是这样。)

再看看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。把这个 URI 复制粘贴到任何网络浏览器的地址栏里，这样你就能在网站上阅读到 XHTML1.0 严格的 DTD 了。因为在 DOCTYPE 末尾的 URI 里，指出了网站上的有效位置，支持 DOCTYPE 转换的浏览器认为这个 DOCTYPE 是完整的，将全部以标准模式呈现你的网页。

另外还有个新方法，无论 DOCTYPE 是否包含一个完整的 URI 地址，只要存在 XHTML DOCTYPE 中，就提示 Internet Explorer 进入标准模式，不包含完整 URI 的 XHTML DOCTYPE 在技术上是无效的。在下面的讨论里，我们提倡你在 DOCTYPE 里用一个完整的 URI 地址。(如果网页是无效的，那为什么还要转变成标准模式呢？)

IE6/Windows 的 Prolog Quirk

每条规则执行起来都会异常，即使是完整的 XHTML DOCTYPE，如果还包含可选的 XML prolog，IE6/Windows 就会返回到 Quirk 模式。这就是我们在第 6 章里建议你略过 prolog 方式的原因。

11.2.3 完整 XHTML DOCTYPE 的完整列表

下面列出的 XHTML DOCTYPE 是完整的，在支持 DOCTYPE 转换的浏览器上能触发标准或近乎标准模式。什么是近乎标准模式呢，我们马上就会讲到。

XHTML 1.0 DTD

XHTML1.0 Strict (严格方式) —— 在所有支持 DOCTYPE 的浏览器上都触发完全标准模式，但对于 7.0 以前版本的 Opera 和 6.0 以前版本的 IE/Windows 没有影响。

```
<!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) —— 在相应的 IE 浏览器 (IE6+)

Windows, IE5+/Macintosh) 上都触发全面标准模式。在第一代基于 Gecko 浏览器上 (Mozilla 1.0, Netscape 6)，也触发全面标准模式，在更新版本的基于 Gecko 浏览器 (Mozilla 1.0, Netscape7+, Chimera0.6+) 上，触发近乎标准模式。对于 7.0 以前版本的 Opera 和 6.0 以前版本的 IE/Windows 没有影响。

```
<!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)——在相应的 IE 浏览器 (IE6+/Windows, IE5+/ Macintosh) 上都触发全面标准模式。在第一代基于 Gecko 的浏览器上 (Mozilla 1.0, Netscape 6)，也触发全面标准模式，在更新版本的基于 Gecko 的浏览器上 (Mozilla 1.01, Netscape7+, Chimera0.6+)，触发近乎标准模式。对于 7.0 以前版本的 Opera 和 6.0 以前版本的 IE/Windows，则没有影响。DOCTYPE 转换也许会影响框架的呈现方式，但如果真的影响，还没有浏览器制造商会文档化这些变化。

```
<!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 转换的浏览器上都触发全面标准模式，对于 7.0 以前版本的 Opera 和 6.0 以前版本的 IE/Windows 没有影响。

```
<!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的定义代码（谁会喜欢呢），尽管去 A List Apart 的“用正确的 DOCTYPE 巩固你的站点”里复制粘贴那些代码 (<http://www.alistapart.com/stories/doctype/>)。

执行概要：在 IE 和 Gecko 浏览器上的 XHTML DOCTYPE 触发器

让我们回顾一下目前为止所学的内容：

- 在 Internet Explorer (IE6+/Windows, IE5+/Macintosh) 的相应版本和基于 Gecko 的浏览器的早期版本 (Netscape 6 Mozilla 1.0) 上，任何完整的 XHTML1.0 或 1.1 DOCTYPE 都会触发标准模式。

- 完整的严格的 XHTML DOCTYPE 在以前和后来的任何版本的基于 Gecko 的浏览器（Mozilla 1.01+、Netscape7、Chimera0.6+）上效果都是相同的。
- 在后来版本的基于 Gecko 的浏览器上，完整的 XHTML1.0 过渡式和框式支架 DOCTYPE 都触发近乎标准模式。
- 相对 URI 地址或缺少 URI 地址的不完整 DOCTYPE，在所有支持 DOCTYPE 转换的浏览器上触发 Quirk 模式，还会生成 CSS 校验错误，但不会生成 XHTML 校验错误，因为 W3C 的 XHTML 校验系统有一个 bug，或是因为 XHTML 和 CSS 校验系统有差别。关键在于你用的是哪个，只要记住不完整 DOCTYPE 触发 Quirk 模式和 CSS 校验错误即可。
- 很多制作工具默认的是，插入不完整 DOCTYPE，这样就触发了 Quirk 模式并且产生 CSS 校验错误。

Web 标准组织的作者和个体成员已经和主流工具的制造商取得联系，告诫他们有必要默认插入完整的 DOCTYPE。我们也已经要求 W3C 在其网站（www.w3.org）上公布完整 DOCTYPE 的快速查找列表。同时，为了保证你的站点触发标准模式，必须手工编辑创作工具自动生成的 DOCTYPE。

DOCTYPE 转换：缺陷存在于细节中

DOCTYPE 转换不仅限于 XHTML。前面提到过，一些完整的 HTML DOCTYPE 也触发标准模式。例如，在 IE 存在一个完整的 HTML 4.01，且严格 DOCTYPE 的情况下，也触发标准模式（最近推出的基于 Gecko 的浏览器则触发近乎标准模式）。HTML 4.01，且严格 DOCTYPE 代码如下：

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

但与 4.01 的不同的是，在 IE 和基于 Gecko 的浏览器上，一个完整的 HTML 4.0 DOCTYPE 却触发向前兼容的 Quirk 模式。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，以避免出现这些矛盾。

要想详细了解另外一些在最近推出的基于 Gecko 浏览器上如何触发标准、Quirk 和近乎标准模式的 DOCTYPE，请参阅 <http://devedge.netscape.com/viewsource/2002/almost-standards/> 上的“Gecko 的近乎标准模式”。如果你觉得那些内容太多太复杂，也可以不看。只要我们用 XHTML 1.0 过渡方式、严格方式或者严格的 CSS，大多数人就不需要考虑这么复杂的问题。

11.3 感谢浏览器的多样性（或至少学会接受）

基于 Gecko 的标准模式与 IE 的标准模式在一些细节方面有所不同，如果你并不希望出现这样的差异，那么它可能令你很不愉快。特别是在第一代 Gecko 浏览器上显示混合布局（CSS 加表格）时，这种差异尤为明显，令人讨厌。

从 HTML 4.01 过渡式转到成 XHTML1.0 过渡式之后，除了那些在第 6 章里描述过的标记以外，其他的标记不发生任何改变。设计师希望他们的布局看上去一直是一个样子，所以他们只会修改少许标记的标签而不会改变整个设计。然而他们的布局在早期版本的 Gecko 浏览器（诸如 Netscape 6.0 和 Mozilla1.0）上，看上去和其他浏览器上有很大的差别。如果没有转换成 XHTML 严格方式，那些布局在新版本和老版本的浏览器上的显示效果都会各不相同。发生这种变化是因为一个基于标准的原因，恢复它的方法也很简单。

Gecko 里的图片间隔问题

图 11.1 是 www.zeldman.com 网站的每日报告，是几年前网站用当时的 CSS 结合表格的过渡技术做出来的。

在把 HTML4.01 过渡式转换成 XHTML1.0 过渡式（从完整的 HTML4.01 DOCTYPE 转换成完整的 XHTML1.0 过渡式 DOCTYPE）以后，站点看上去和在 Windows 和 Macintosh 系统里的 Internet Explorer 上是一样的。但放在 Gecko 浏览器上时，在表格里，控制站点的导航菜单的图片间出现了不必要的间距（如图 11.2，图 11.3 所示）。



圖 11.1
这个老式的混合表格+CSS 有间。
(www.zeldman.com) 使用了触发标准模式的完整 DOCTYPE。在把 HTML 转换成 XHTML 后，网站在 IE 中看起来和原来一样。
(www.zeldman.com/daily/1002a.html)

图 11.2

采用本章描述的 CSS 方法之前，在早期 Gecko 浏览器上，网站的导航表格外有不必要的间隙。



图 11.3

这张是放大的截图，可以更清楚地看到那些间隙。并不是早期的 Gecko 浏览器显示这个表格的方式有误，而是它们呈现的效果无法预料，不尽人意。



一条（或两条）规则把它们连接起来：用 CSS 清除间隙

这种间隙的出现有一个合理的解释（见“Gecko 标准模式里的基线和垂直空白”）。探讨这个原理之前，让我们先解释一下如何在 Gecko 和 IE 浏览器里把导航条恢复成其理想的状态（如图 11.4 所示）。它用到的只是两条 CSS 规则：

```
img {display: block;}  
.inline {display: inline;}
```

图 11.4

两条简单的 CSS 规则清除了 Gecko 里不必要的间隙。一切都没问题了。在后来的 Gecko 浏览器采用的近乎标准模式里就不存在这些间隙，也不需要 CSS 禁制。但我们建议最好还是使用 CSS 规则，原因将在下面说明。



第一条规则标记浏览器把图片作为块级而不是内联元素处理，这样的话，间隙被闭合（如果你不记得块级元素和内联元素的区别，请看第 10 章）。但如果我们要以内联元素显示一些图片，该怎么办呢？用第二条规则（.inline）。这条规则创建了一个类，使图片能以内联元素显示，标记书写如下：

```
<img class="inline" ... etc. />
```

还有一些其他的方法可用，比如，一些设计师更愿意书写这样的规则，当表格单元只包含一个图片时，指定图片作为块级元素显示：

```
td img {display: block;}
```

仅仅采用这么简单的一条规则，就能使每个表格单元只含一个图片的导航菜单以块级元素显示，而不产生间隙。网站上的其他图片以内联元素显示。这种二

者选其一的方法优点是只用一条规则，而不需要用两条，节省了一点带宽，也省去了给一些图片增添类的必要。

不同的方法对不同的布局作用也不一样。要根据你的设计挑选方法，建议 CSS 初学者先用一种方法，不行再换另外一种，可能你自己还能想出第三种方法。

近乎标准模式的起源

若要创建前面出现的那些 CSS 定义，根本不需要花费多少时间，把它们并入你的设计，从而不必再担心浏览器之间的差异。但是当图片间隙问题刚出现，而且似乎只出现在把 HTML 转换成 XHTML 之后时，一些设计师认为是 XHTML 出了故障，另一些则断定是 Netscape 6 和其他的 Gecko 浏览器不够完善，或者是有 bug。其实 Gecko 给未指定样式的图片加一个空白属性，是一个特性，而不是一个 bug（见“Gecko 标准模式里的基线和垂直空白”）。然而这不是设计师们最初理解的特性，当然也不是他们要求的特性。

为了给处在过渡期的设计师提供更好的服务，Gecko/Netscape 的工程师们创建了一个与 IE 标准模式行为相同的近乎标准模式，也就是说，在近乎标准模式里，不会出现意想不到的间隙。因为间隙主要产生在过渡式布局里，于是工程师们测定 XHTML 过渡式 DOCTYPE 将触发 Gecko 的近乎标准模式，而严格的 DOCTYPE 会继续触发 Gecko 更严格的标准模式（总之，如果你写的是严格的表现性标记，也许就不能把图片粘贴到表格单元里）。

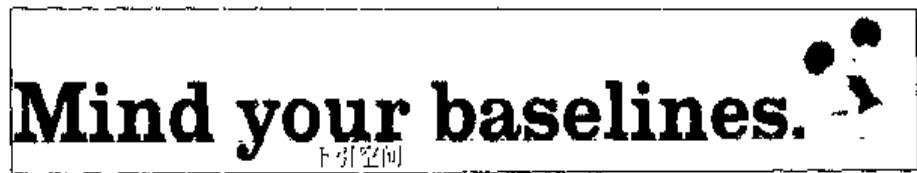
Gecko 标准模式里的基线和垂直空白

正如前面一节“Gecko 里的图片间隙问题”所阐述的一样，IE 和 Gecko 浏览器的标准模式对图片元素与网页暗格的关联处理有所不同。在真正的标准模式里，除非你在样式表里已经把图片指定为块级元素，否则 Gecko 把图片作为内联元素处理。例如，文档位于基线上，给笔画伸向基线以下的小写字母（比如 y, g, j）留出空间。所含元素的尺寸和字体（如包含内联图片的一个段落的文档尺寸和字体）决定基线的尺寸和定位。

图 11.5 显示位于基线上的文档和图片。在 Gecko 的标准模式中，内联图片的下方总有空白，这是为了给所包含块的下行字母留出空间，不论这个块实际是否包含文档。想要了解更多，可参见 Eric Meyer 在 Netscape DevEdge 里的“图片、表格和神秘的间隙”，虽然是写在近乎标准模式出现之前，但是与它有很大的关联 (<http://devedge.netscape.com/viewsource/2002/img-table/>)。

图 11.5

所有基线上的文档为下行字母留出了空间。在 Gecko 的标准模式里，所有包括图片在内的内联元素与其所含元素（这里是只有一个句子的段落）共享基线



近乎标准模式的局限性

如果 Gecko 用户定期升级到最新版本，近乎标准模式将解决过渡式布局中的图片间隙问题。很多 Gecko 用户的确总是升级版本，但一些 CompuServe 用户就不总是升级版本（只有当服务提供商给他们寄去一个新安装盘时他们才会升级他们的浏览器）。这样的话，定期书写的“一个（或两个）空格”定义把它们连接起来：用 CSS 清除间隙”里那样的 CSS 定义就很有意义。这样做还能防止现在和将来在非 IE 和非 Gecko 浏览器上出现无法预计的情况，这些浏览器对图片和其他内联元素的处理，可能与 Gecko 处在标准模式时一样严格。另外，它可以使你脱离表象标记和与之有关的标准化浏览器行为，帮助你从 CSS 角度更多的思考。

11.3.2 从“差异永存”到“@#\$! This \$#@\$.”

无论是不是你期望的那样，正确的行为是一回事，而 bug 和错误又是另外一回事。在第 12 章，我们将学习浏览器常见的 bug 和怎么修复它们。我们还将学习 CSS 布局中一些极其重要的知识。

使用浏览器 第二部分： 盒模型、bug 和工作区

“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 XHTML 的代码方式不是为了去赢得什么荣誉，而是为了我们的站点能在今天或明天，甚至往后的十年里，能够在桌面浏览器、文档浏览器、屏幕阅读器及手持设备上很好地运作。同样地，我们用 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），而是为了保证我们的站点在 Netscape 4.0 和今天的浏览器里看上去一样，不走样，不再让表象标记在非 CSS 环境里妨碍用户体验。

兼容标准的浏览器开发商从痴心妄想状态转向理性的、可接受的设计策略，如果网站主要还是依赖 Netscape 和微软 4.0 浏览器显示，那么多数的站点就不可能发展，不可能获得向后兼容性。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，继续推广他们私有的技术，那网络作为一个开放式平台的前景就堪忧了。

幸好，在过去的几年推出的主流浏览器和一些小浏览器都还能被称为是“兼容标准”的，只是一些浏览器的兼容性比另外一些要好一些。解决不兼容性是本章要阐明的内容。

12.1 盒模型和它的不足之处

随着 1996 年 CSS1 的推出，W3C 公司建议网页上的所有对象都放在一个盒中，设计师可以通过创建规则来控制这个盒的属性，无论对象是一个段落、列表、标题、图片，还是如

这样的一般块级元素。如果它

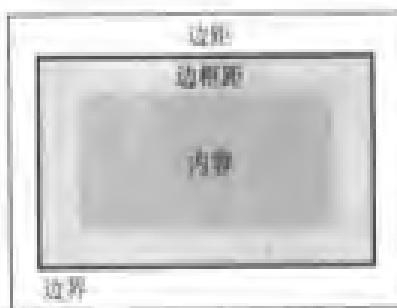
能通过标记插入一个网页，就存在于一个盒中。

图 12.1 举例说明 CSS 盒的四个区域：内容、边框距、边界和边距。内容（在我们的表中为暗灰色）是最里面的区，下面是边框距（浅灰色）、边界（黑色）和边距（白色）。你应该很熟悉这些区。在第 10 章“CSS 应用：混合布局（第三部分）”里，当我们为 i3 论坛站点创建一个混合布局时，已经为各种各样的页面元素设定了边界、边框距和边距，当书写如下规则时，我们也为内容区指定了尺寸（内容值以粗体突出表示）：

```
td#home a:link img, td#home a:visited img {
    color: #c30;
    background: transparent url(/images/bgpac.gif) repeat;
    width: 400px;
    height: 75px;
}
```

图 12.1

CSS 盒的四个区：内容、边框距、边界和边距（我们加深边距的外边缘颜色，以便于你看得更清楚）



在上面那个定义中，内容的宽度为 400 像素，高度为 75 像素，注意我们没写成这样：

`content: 400px;`

也没写成下面那样：

```
content-width: 400px;
content-height: 75px;
```

在 CSS 里没有这样的规则。按照名称指定边界、边距和边框距的值，但不这样指定内容区的值。刚开始学习和使用 CSS 时你也许会认为 `width: 400px` 是指定给整个盒（除边距外）的。毕竟，页面布局编程就是起这样作用的。设计师会这么考虑，用户也会这么理解。如果你创建两个含有并列 `div` 的 CSS 布局，给每个都指定一个为访问者浏览器窗口 50% 的宽度。当增加边框距和边界的值时，你可能期望这两个 `div` 还是保持原值。但 CSS 并不是这样作用的。

同样，如果你是一个在标准化出现初期，支持 CSS 浏览器的制造商，像定义显示的那样你会错误地把 `width: 400px` 指定给整个盒（除边距外），而

不是仅限于指定给内容区。特别是你已经在执行 CSS1 时，规则才刚被制定出来，所以很容易犯这样的错误。我们将在几段后再讲到这个话题。

实际上，CSS 盒模型相对你所期望的一般感觉、图形设计标准，以及早期浏览器实现，会更复杂，在一些方面甚至更加麻烦，但是它能显示比你所期望看到的更多。

12.1.1 盒模型怎样工作的

根据 CSS，可以给四个区（内容、边框距、边界和边距）中的每个区指定值，这些值是附加的。把内容、边框距和边界的值全部加起来，就是盒子的总宽度（如图 12.2 所示）。如果内容宽度是 400px，边框距每边是 50px，边界的每边是 2px，那总宽度将是 504px（400 内容 + 2 左边界 + 50 左边框距 + 50 右边框距 + 2 右边界 = 504）。



图 12.2

盒模型工作时，把内容、边界、边框距的值加在一起就是它的总宽度

CSS 不在乎你选什么值作为主要值，也不在乎你选什么样的值来组合和匹配。例如，在显示页面时，你可以指定内容值为访问者浏览器窗口的 67%，边距为 5em，边界为 1px。总和怎么算呢……很难计算。这将取决于访问者浏览器窗口的宽度和文档的默认尺寸。

关于盒模型还有更多需要解释的，一方面是：垂直相邻元素的上边距和下边距相互叠加（形成加深或是加强效果，有关更详细的内容请看：<http://www.w3.org/TD/REC-CSS2/box.html>）。另一个方面是：当盒为空时，尺寸值没有意义。如果你指定空盒占据页面高和宽的 100%，而实际它只占 0%，原因没有意义。如果你企图用背景颜色（不是确切的数据）填充两个块级格布局方式不同。如果你企图用背景颜色（不是确切的数据）填充两个块级元素的方法来实现颜色效果，这种方法在框架和 4.0 浏览器上的表格布局里是不起作用的，但在 CSS 里不起作用。从这个意义上说，你很难把外观和结构区分开来，因为既不存在数据也不存在外观。

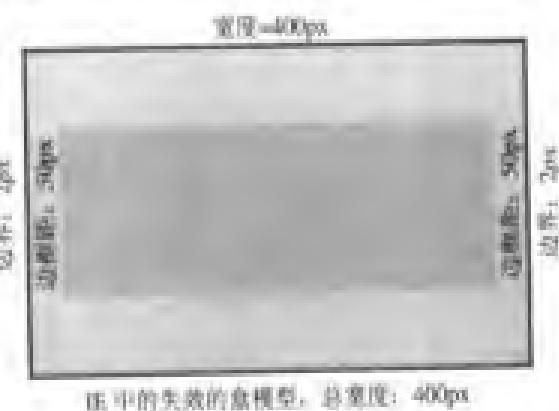
12.1.2 盒模型怎样失效

如果你留意过，就会对盒模型在诸如 IE4 到 IE5.5/Windows 浏览器上的早期 CSS 执行中怎样失效有所了解。在这些浏览器和 5.0 以前版本的 IE/Mac 上，它却将仅指定给内容区的宽度和高度值指定给了整个盒（边距除外）。

图 12.3 显示的是操作中失效的盒模型。再看看前面的例子，在 IE4-5.5/Windows 上，若内容的宽度为 400px，边框距为每边 50px，边界为 2px，总宽度仍然是 400px 而不是正确的 504px——内容区域被缩小成 296px（ $400 - 50 - 50 - 2 - 2$ ），边框距和边界值则增加了一些。布局显示的效果和意图差距越大，内容区就被缩得越小。

图 12.3

在 Internet Explorer4.0 到 5.5
Windows 浏览器上失效的盒模型：
 $400+2+50+50+2=400$ ，错误的算法。
但也许是个好想法



直到写这本书时，成千上万的人还在用 IE5.X/Windows，还有上万设计师在学习错误的 IE5.X/Windows 盒模型，并将其应用在他们的设计中，致使在正确理解盒模型的浏览器上（IE6/Windows，IE5/Macintosh，Netscape 6，Netscape 7，Mozilla，Chimera，Kmeleon，Safari，Opera 5，Opera 6 和 Opera 7）出现失效的布局。

理解盒模型，书写正确规则的设计师在我们刚才提到的浏览器上能制造出期望的效果，但也会在 IE5.5/Windows 和成千上万人还在使用的更老的浏览器（我们前面可能说过的）上得到糟糕的、甚至不可用的布局。

其实有几个解决的办法，最常用的一个方法是一个微软工程师想出来的。在我们修复失效的盒模型之前，让我们来看看它的一些优点。

修复失效的盒模型

IE4/5.X/Windows（也在所有的 IE/Macintosh5.0 以前的版本）上的盒模型按照 CSS 规格肯定是无效的、错误的，但这并不是一个愚蠢的错误。在某些方面，失效的盒模型比 CSS 里正确的规则更易书写，也更易于使用。CSS 盒模型能很

好地作用于使用绝对像素值的布局，但其复杂性使得它不直观、令人厌烦，甚至当你试图用它创建适应访问者显示器的基本百分比值（“动态”）的布局时都会起反作用。

设想一个简单的两个单元格设计，其左边和右边的栏宽加起来为访问者显示器宽度的 100%。用 HTML 表格设计这样一个布局相当容易，但用 CSS 就很困难。为了更新一个 CSS 里的两栏动态布局，你必须创建两个 div，还要用 CSS float 属性把一个 div 定位在另一个的旁边：

```
#nav {  
    width: 35%;  
    height: 100%;  
    background: #666;  
    color: #fff;  
    margin: 0;  
    padding: 0;  
}  
  
#mainText {  
    width: 65%;  
    height: 100%;  
    color: #000;  
    background: #fff;  
    margin: 0;  
    padding: 0;  
    float: right;  
}
```

这里我们页面由两个区域组成：一个出于导航的目的，另一个是为了主文档，正如其名称所示，float : right; 说明主文档栏漂浮在导航栏的右边，导航栏被指定为占访问者浏览器窗口宽度的 35%。主文档区占余下的 65%，在它们之间，两个页面分界线应该占窗口的 100% ($35+65=100$)。

图 12.4 表示，只要没有使用边框距和边界，这个方法就能起作用。边界值是有选择性的，取决于设计，但边框距值对于防止晦涩难看的平铺式文档（如插图里所示）极为重要。当设计师把边框距和边界值添加给一个像下面这样的基础并列式布局时，网页就会土崩瓦解。在一些浏览器上，元素相互交叠（如图 12.5 所示），在另外一些浏览器上，因为总数字加起来超过了 100%，布局对于显示器来说就太宽了，所以不能得到完全的显示，读者必须水平滚动才能看到全部内容。而且无论用的是多大的显示器，布局总是对于显示器来说太宽了。

如果你用百分比值设置边框距和宽度，情况就会好些，所有的值加起来后符

合访问者浏览器窗口的大小。但是要把百分比值和用长度单位表示的边框距值正确地相加是根本不可能的。为了纠正这个问题，CSS3 包含了一个建议性的“盒尺寸设置”属性，以便设计师为特定的设计确定一个设置尺寸的模式。如果 CSS 盒模型和在 IE/Windows 旧版本上的错误执行程序一样工作，我们就不需要使用盒尺寸设置属性了。当然，直到今天，我们的浏览器里也没有这样的一个属性。

图 12.4

用 CSS 很难采用弹性百分比布局以适应访问者显示器。这个布局定位两个动态 div，并列存在。
! http://www.zeddrum.com/ponent/a1a/boxmodel1blue.g/



图 12.5

添加边框距和边界值后，布局才崩瓦解。在一些浏览器上（如此处所示），元素相互交叠。在另外一些浏览器上，网页大大宽于屏幕。



在缺少盒尺寸设置属性的情况下，设计师一般采用不给 divs 指定边框距值，

而在那些包含 divs 的元素里插入子元素，并且把边距值指定给这些子元素的方法，通过这个方法制造想要的“边框距”效果。这个方法的问题是：会生成无关的非结构性标记元素，它们仅仅是为了纠正布局的问题（第 16 章“CSS 重新设计”里的“勇敢的四百”元素，可以被认为是这样的元素，因为它只有一个作用——纠正格式化问题）。增加过多这样的元素，使你原本简洁的标记变成像我们在第 7 章“紧凑而坚固的页面保证：以严格和混合的标记组成的结构”里所警告的那样。

除了给带宽增加些字节，用创建和嵌套非结构性元素来解决布局问题还有什么害处呢？实际上也没什么大害处。

在失效的 IE4/5/Windows 盒模型下，你能创建混合使用长度和百分比值的动态布局，而无需嵌入不必要的 divs 或者伤透脑筋。我们不想说 IE4/5/Windows 是适应标准的集中体现，因为它的确不是。我们想说的是错误的盒模型也有它的用处。

有一个创建混合长度和百分比单位 CSS 动态布局的方法，但是人们都不愿意用，而且它最多也就是个不准确的方法。诀窍是只创建一个 div，把网页当成别的网页用，通过漂浮属性在它们之间协调，因为加起来不可能等于总尺寸，所以要估计、制造数字，这些数字加起来不真的等于 100%。

当 2001 年 2 月，A List Apart 杂志从 HTML 表格转换成 CSS 布局时，包括笔者在内的三个设计师一起来考虑怎么样转换。除了笔者，其他两个设计师是专家，他们是 Fahrner（为 CSS 规则撰稿）和 Gelik（为 CSS 规则撰稿并在 IE5/Mac 上建造了 Tasman 呈现引擎）。最初的基于表格式设计（如图 12.6 所示）主要是由一个动态内容区域和以英石计算宽度的一个右手边工具条构成的两栏布局，我们三个人努力实现在上一段中用非直观方式创建的 CSS 里基本相同的效果（如图 12.7 所示），总宽度为加起来大约等于 100% 的值的总和。创建一个混合了动态和固定元素的布局应该不会太困难。

我们必须承认，现在做这样的设计比以前容易多了，因为设计师可以照搬或者修改作为典型示例的 A List Apart 和其他 CSS 站点的技术，并将其用在自己的设计中。CSS2.1 解决了一些这样的问题，而 CSS3 里的建议解决了更多这样的问题。值得我们注意的是，直到写这本书时，浏览器主要还是支持 CSS1 和一小部分的 CSS2。CSS1 盒模型的复杂性对于那些想创建特定类型动态的人来说，仍然具有重大意义，而对于设计师来说却是反直观的思维方式。

2002 年 DOM 专家 Peter-Paul Koch 接受访问时这样解释：

“逻辑上，测量一个盒是从它的一个边界到另一边界，任何有形的盒都是这么测量的。往里面装一些东西，这些东西显然要小于盒。任何人测量盒的宽度都会测量盒子两个边（边界）之间的距离，没人会去测量盒子里面的东西大小。创建含有内容的盒的网页设计师关心的是盒的可视宽度、边界与边界之间的距离。边界才是网站访问者的视觉提示，而不是内容，没有人会为内容的宽度感兴趣。”

<http://www.netdiver.net/interviews/peterpaulkoch.php>

图 12.6

A List Apart 帖子 (www.alistapart.com) 最初是用适应访问者显示器的表格单元自动排版的

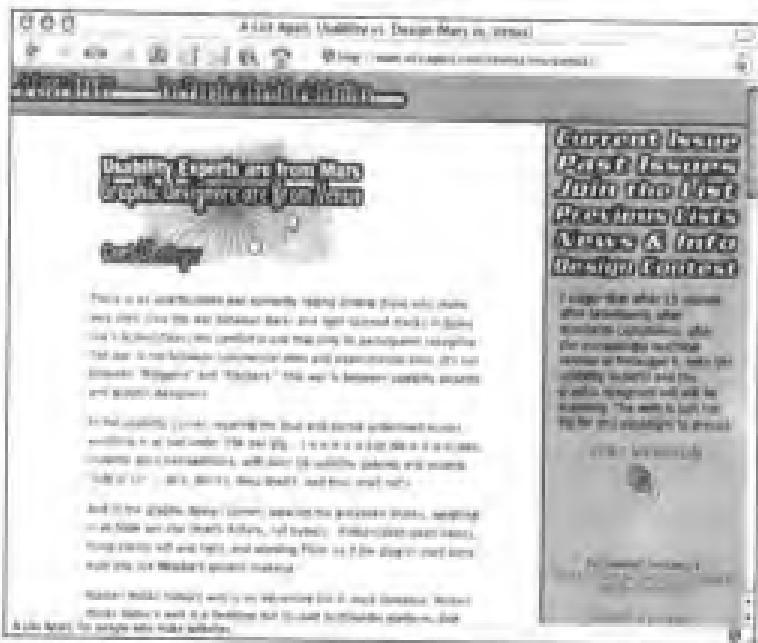


图 12.7

在 2001 年 2 月，在对站点进行 CSS 重新设计时，三个 CSS 设计师一起（其中两个是公认的专家）思考怎样用样式表（style sheet）来实现相似的外观和风格



无论是否过于复杂，现有的规则就是我们必须使用的规则。我们不要求浏览器制造商改进规则，只是要求浏览器能更准确更完整地执行这个规则，所有主要

和大多数中等规模的浏览器制造商也确实是这么做的。

我们怎样才能创建一个布局，使它在那些错误理解盒模型的浏览器上像在正确理解盒模型的浏览器上一样作用呢？

12.1.3 盒模型黑客程序（Box Model Hack）：使 CSS 更安全

Tantek Celik——这个名字在本章已经出现过——提供了解决不正确的盒模型实现中产生的方法，这个方法现在已被广泛采用。他的盒模型黑客程序（Box Model Hack）利用 IE5.x/Windows 里的 CSS 分析程序 bug，指定一个适合 IE5.x/Windows 的伪宽度值，然后再用真实值覆盖这个伪值（如图 12.8 所示）。在下面从 <http://www.tantek.com/css/Examples/boxmodelhack.html> 摘录的例子中，内容区的真实值为 300px，但 IE/Windows 误认为内容区的值是 300px 减去了 100px 的边框距和边界之后的值。

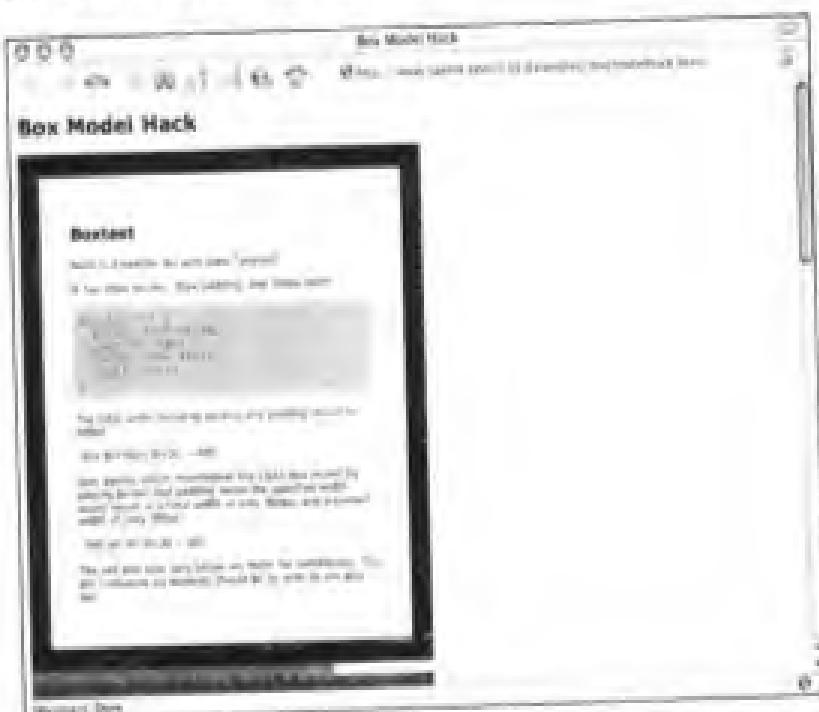


图 12.8

Tantek Celik 的盒模型黑客程序解决了盒模型的问题。对于解决其他浏览器问题也很有用。
(<http://www.tantek.com/css/Examples/boxmodelhack.html>)

让我们先来看看你可能会为所有的浏览器所书写的不正确的 CSS 规则，如下：

```
div.boxtest {  
    border: 20px solid;  
    padding: 30px;  
    background: #fffcc;  
    width: 300px;  
}
```

正确理解 CSS 的浏览器将创建一个总宽度为 400px ($20 + 30 + 300 + 30 + 20 = 400$) 的盒。但老式的 IE/Windows 浏览器会从内容区的 300px 值里减去边框距和边界的值，结果盒宽变为 300px，内容区则变为 200px ($300 - 20 - 30 - 30 - 20 = 200$)，所以我们给它们指定一个 400px 的伪值（我们实际想要的总尺寸），然后把两行 IE4/5Windows 不能理解的包含 CSS 选择器的规则混入其中（IE 不能理解的两行规则用粗体突出表示）：

```
div.content {
    width:400px;
    voice-family: "\}\\";
    voice-family:inherit;
    width:300px;
}
```

在最后的定义中，我们赋予其真实值——300px。IE5.x/Windows 遇到不能理解的声音属性时就不再继续阅读定义，但更为适应的浏览器会继续阅读，并执行其真实值。

在 Opera 浏览器（至少为 Opera 7 以前的版本）上产生了另外一个问题，虽然它们也能阅读 CSS2 选择器和盒模型，但是产生像 IE/Windows 一样的解析 bug。所以在 Opera 浏览器上也要使用伪值，可是我们并不想这样。Tantek 用创建附加的“Opera 友好”规则来解决这个问题：

```
html>body .content {
    width:300px;
}
```

Opera 浏览器可能在前面的规则中忽略了真实值，但在附加规则中，Opera 浏览器会尊重真实值，因为这个真实值被提出来具体指定，在 CSS 定义中，更为具体的定义比一般的定义优先执行（就像#menu p 定义就比 p 定义优先）。

从 2000 年下半年推出以来，盒模型程序已经被上万的网站采用，以促进 CSS 布局，做法就是给老式的浏览器指定一个其失效盒模型所要求的伪值，而给更适应的浏览器提供真实值。解释它比使用它麻烦得多，我们还是经常使用它吧。

注意，当边框距和边界值为零时不需要使用 Box Model Hack，用不用盒模型 Hack 取决于设计的敏感性和特定项目的需求，当边框距值很小，设计相当“松散”时也不需要，你不用担心浏览器之间几个像素的差别引起盒大小的变化，这点变化是可以忽略的（还有一些设计中，你可能被选加起来小于 100%的一些值所欺骗）。

浏览器的隐藏和搜寻

盒模型黑客程序（也称为 Tanek 方法）依靠一个解析 bug 使老版本 IE/Windows 正确支持盒模型。就像第 9 章“CSS 入门”的“外联样式表”一节所讨论的，样式表的@import 方式链接会使 4.0 及更老版本的浏览器忽视样式表。在这些及其他的一些方法中，我们用标准方式支持标准模式。我们所做的一切都是为了使有局限性的浏览器适度地忽视其不能理解的定义，为此我们不仅要使用以前的老式方法（例如浏览器搜寻和代码分支），还要让基本标准尽其所能发挥作用。下面是更多对不同性能浏览器隐藏各种各样的方法，见 http://w3development.de/css/hide_css_from_browsers/summary/

12.2 IE/Windows 上的空格 bug

看了那么多复杂的理论，现在让我们来看看一个用不太复杂的方法就能解决的简单问题——IE/Windows 上的空格 bug。图 12.9 显示的是空格 bug 对混合布局的严重破坏（和第 11 章“使用浏览器第一部分：DOCTYPE 转换和标准模式”里的图 11.4 比较一下），它对纯 CSS 布局也具有同样的破坏性。



图 12.9

这次出现了另一个显示问题，其根源是 IE/Windows 上的空格 bug。为了使你看得更清楚，图片已经放大。和第 11 章的图 11.4 的正确显示比较一下。

下面的两个标记代码具有相同的功能，但是由于代码里的空格作用不同（或没有作用），它们在解析 XHTML 里的空格有错误的浏览器上的显示是不一样的：

```
<td></td>
```

和下面这个定义显示不一样：

```
<td>
  
</td>
```

第二段标记代码里有空格，它可能会使你的网页上出现多余的间隙，如图 12.9 所示。下面这个定义可能也会产生间隙，下一个标记（没有空白属性）……

```
<td><a href="#foo"></a></td>
```

在你浏览器上的显示和下面这个在功能上相等的定义的显示不一样：

```
<td>
  <a href="#foo">
    
  </a>
</td>
```

为什么会出现这种情况呢？空格 bug 是 Netscape Navigator 的一个知名问题，其历史可以追溯到 3.0 版本。微软决定制造与之竞争的浏览器时，仿效了 Netscape 的许多行为，很不幸，也包括一些 bug。截至写这本书的时候，IE/Windows 浏览器甚至 IE6 都还在效仿 Netscape 的那些老 bug。解决这些 bug 的方法就是从你的标记中删除空格。

CSS 布局设计中的空格 bug 与混合式 CSS/表格布局中的空格 bug 一样。看下面这个列表（出自 zeldman.com 2003 1 月版）：

```
<div id="secondarynav">
  <ul>
    <li id="secondarytop"><a href="/about/" title="History, FAQ, bio, etc.">about</a></li>
    <li id="contact"><a href="/contact/" title="Write to us.">contact</a></li>
    <li id="essentials"><a href="/essentials/" title="Vital info.">essentials</a></li>
    <li id="pubs"><a href="/pubs/" title="Books and articles.">pubs</a></li>
    <li id="tour"><a href="/tour/" title="Personal appearances.">tour</a></li>
  </ul>
  ...</div>
```

CSS（从现在开始提供一些段落供参考）把列表转换成用户可单击的“按钮”，如图 12.10 所示，但 IE/Windows 标记里的空格使得按钮的间距和定位紊乱。为了解决 IE/Windows 的 bug 问题，必须清除空格：

```
<div id="secondarynav"><ul><li id="secondarytop"><a href="/about/" title="History, FAQ, bio, etc.">about</a></li><li id="contact"><a href="/contact/" title="Write to us.">contact</a></li><li id="essentials"><a href="/essentials/" title="Vital info.">essentials</a></li><li id="pubs"><a href="/pubs/" title="Books and articles.">pubs</a></li><li id="tour"><a href="/tour/" title="Personal appearances.">tour</a></li></ul> ...</div>
```



图 12.10

把 CSS 声明指定给无序列表来创造导航“按钮”（放大，嵌入）的视觉效果。列表的标记必须将结构澄清，用以避免 IE/Windows 上的空格 bug。

这种标记比使用空格的正常标记更难编辑，也更难看懂，但只要我们想在浏览器上正确显示，网站就别无选择。删除空格后，文件也被削掉了一或两个字节。我们把制造这些菜单按钮效果的 CSS 重新书写如下，可参见 <http://www.zeldman.com/c/sophisto.css>。

```
#secondarynav ul {  
    list-style: none;  
    padding: 0;  
    margin: 0;  
    border: 0;  
}  
  
#secondarynav li {  
    text-align: center;  
    border-bottom: 1px solid #066;  
    width: 100px;  
    margin: 0;  
    padding: 0 1px 1px 1px;  
    font: 10px/12px verdana, arial, sans-serif;  
    color: #cff;  
    background: #399;  
}  
  
#secondarytop {  
    border-top: 1px solid #066;  
}
```

```

.secondarynav li a {
    display: block;
    width: 96px;
    font-weight: normal;
    padding: 1px;
    border-left: 2px solid #6cc;
    border-right: 2px solid #6cc;
    background-color: #399;
    color: #cff;
    text-decoration: none;
}

.secondarynav li a:hover {
    font-weight: normal;
    border-left: 2px solid #9cc;
    border-right: 2px solid #9cc;
    background-color: #5aa;
    color: #fff;
    text-decoration: none;
}

```

在第 16 章，我们将用几乎一样的 CSS 标记和规则进行了重新设计。想知道更多关于怎样把结构列表转换成图像导航元素，可参见 Mark Newhouse 的“CSS Design: Taming Lists”(http://www.alistapart.com/stories/taming_lists/)。

12.3 IE6/Windows 上的“漂浮” bug

本章前面解释了 CSS “漂浮” 特性是怎样使一个元素紧靠另一个元素横排。例如，下列规则将使惟一 id 属性为“`maintext`”的 `div` 漂浮在工具条或其他右边元素的左边。

```

#maintext {
    float: left;
}

```

“`maintext`” 里的文档最好简明扼要。IE6/Windows 的一个程序 bug 会删减漂浮在 `div` 里的较长文档，使读者看不到完整的文档，同时还会造成滚动条的消失。读者必须刷新网页，并快速连续按两次 F11 键才能看到全部的文档，恢复浏览器的滚动条。可是没人愿意每登录一个新网页就按两次 F11 键，太麻烦了。

这个问题还影响到一些 IE6 用户，但中文用户很少受其影响。首次报告发现

这个 bug 是在 2001 年，微软工程师们已经努力修复，但直到 2003 年，这个 bug 依然存在。

也许微软工程师们在实验室里不能复制这个 bug，也或许他们找不到原因所在，但是一些独立的设计组织似乎找出了问题的根源。

固守原值

很明显，IE6 在计算块级元素的高度方面出现了问题。<http://dhtmlnirvana.com> 的 Eddie Traversa 发现 IE6/Windows 存储它对站点的一个网页的计算结果，然后把这个计算结果错误地指定给其他网页。比如说，如果第一页上的“maintext”div 的高度是 300px，第二页上的高度是 1400px，第二页只会显示为 300px，而不是 1400px。用户每登录一个网页都要清除存储的初始值，人工地修复这个 bug。

用脚本修复

知道了漂浮特性有 bug，也知道了 bug 产生的根本原因，下面我们告诉你解决的方法。<http://www.youngpup.net> 的 Aaron Boodman 编写了一个小小的 JavaScript 程序，重申漂浮块特性的存在，使网页正确地自动换行和显示。下面是 Aaron 的完整程序：

```
if (document.all && window.attachEvent)
    window.attachEvent("onload", fixWinIE);
function fixWinIE() {
    if (document.body.scrollHeight
        < document.all.content.offsetHeight) {
        document.all.content.style.display = 'block';
    }
}
```

如果你愿意，可以从 <http://www.alistapart.com/styleswitcher.js> 上复制粘贴这些 JavaScript 程序。这儿行定义使你能无所顾忌地用 float 属性创建 CSS 布局，在你的设计中，用漂浮 div 名称替换 content。例如，你的网页上含有 id 名称为“maintext”的 div 里的漂浮内容，则你应该编写代码如下：

```
if (document.all && window.attachEvent) window.attachEvent
    ("onload", fixWinIE);
function fixWinIE() {
    if (document.body.scrollHeight <
        document.all.maintext.offsetHeight) {
        document.all.maintext.style.display='block';
    }
}
```

另外一个避免 bug 的方法是使用 CSS 绝对定位来模仿漂浮，这个方法我们将在 16 章中介绍。

12.4 Flash 和 Quick Time，期望的对象

很多人在网站中插入 Flash 和 Quick Time 电影这样的多媒体对象，但在各种各样的浏览器和平台上，往网站里插入多媒体对象并没有一个可靠的适应标准方法。

12.4.1 可嵌入对象

当初，Mosaic 和 Netscape 浏览器的制造商想到可以让设计师在网页里插入图片时，为浏览器特别创建了一个标签，借此“扩展”HTML。W3C 公司不赞成这种做法，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标签，却没有网站支持 W3C 的元素。

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 再次建议使用标签把这些内容嵌入网页。但是 Netscape 又发明了标签，于是所有的浏览器都支持 Netscape 的标签。

在 Netscape 和微软看来，顾客希望网络能成为一个丰富的多媒体空间，浏览器制造商必须通过革新来满足这样的要求。

在 W3C 看来，浏览器制造商忙于创建他们自己的标签而不去使用已经非常完善的标准。如果没人关心公开有用的标准，W3C 的会员创建它们还有什么意义呢？在随后的几年里，W3C 会员证明了他们的建议是正确的。

W3C 建议的重要性

W3C 的第一个举动就是在所有官方的 HTML 规则中避免包含标签，尽管几百万网站和几十万设计师在使用标签。HTML3.2 中没有标签，HTML 4.0 或 HTML 4.01 中也没有标签。因为 XHTML1 是建立在 HTML4.01 的基础上的，所以 XHTML 里也没有标签。这样，任何使用标签的网站就不能作为有效的 HTML 或 XHTML，一直到现在还是如此。

这下，几百万嵌入多媒体的网站都被 W3C 规则认为是非法的，因为 W3C 一直都不认同元素。接着 W3C 又清除了 XHTML2.0 原来规则中的低等

级元素（见第 5 章“现代标记”里的“什么 XHTML 适合你”），如果你想要插入图片、Flash、QuickTime，那就只能用`<object>`

问题是，虽然几乎所有现代的浏览器都能支持`<object>`，但是老式的浏览器却不能。而且设计师也不会仅仅因为 W3C 反对他们所使用的标签就不嵌入 Flash 或其他形式的多媒体内容。如果一个设计师既使用网络标准又必须嵌入丰富的多媒体内容，那该怎么办呢？

12.4.2 折中方法：支持标准的同时嵌入多媒体

2002 年 11 月，dreamweaverfcver.com 的 Drew McLellan 和网络标准项目开发组联合进行试验。在给 A List Apart 杂志的一篇文章里，Drew 摒弃了在网页里嵌入 Flash 时常用的臃肿的非法标记，以简洁适应的 XHTML 取而代之 (<http://www.alistapart.com/stories/flashsatay/>)。他摒弃了所有非法的`<embed>`元素，并把下面这个笨重的 IE 样式标记：

```
classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
```

用兼容标准标记改写如下：

```
type="application/x-shockwave-flash"
```

当你插入一个电影时，Flash 生成的 HTML 如下：

```
<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/x-shockwave-flash"
         pluginspage="http://www.macromedia.com/go/
         getflashplayer">
</object>
```

这个 HTML 不但臃肿而且非法，但在很多安装了 Flash 插件程序的浏览器上很有用。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，而且能在 Netscape 和 IE 浏览器上播放 Flash 电影。但是因为一个 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，但是要是 Flash 文件很大，那问题就大了。

Drew 采取以一个小 Flash 电影下载一个更大电影的方法解决 IE/Windows 的连续播放问题。看来嵌入多媒体的适应标准方法最终被发现了。Drew 的 Flash Satay 法在每个浏览器和平台上通过测试之后，A List Apart 刊登了那篇文章。

对象故障

遗憾的是，在文章登出后不久，就发现了一个新问题，一些访问者看到的是空白文档区而不是被嵌入的 Flash 内容（如图 12.11、图 12.12 所示）这种情况主要出现在 IE/Windows 浏览器（5, 5.5 和 6 版本）上，但是大部分的 IE/Windows 用户还是可以看到正常的 Flash 内容。

图 12.11

Satay 法使用 100% 合法的标记嵌入 Flash 内容，但在一些浏览器上却很奇怪地失效，如图所示。图为安装 Flash 程序后的 Mozilla (SuSE Linux 8.0) (<http://www.alistapart.com/stories/flashsatay/>)



到底有多少用户受到影响呢？与文章相关的论坛很快就充满了错误报告 (<http://www.alistapart.com/stories/flashsatay/discuss/>)。

根据公布的浏览器上对象执行程序的对照表 (<http://www.student.oulu.fi/~sairwas/object-test/results/>)，被认为能理解并正确执行 `<object>` 元素的 IE/Windows 和其他用户代理有时不能显示 Flash，令人费解。如 XHTML2 所示，当 `<object>` 被用来嵌入图片时，IE/Windows 也产生相似的问题。Drew 的 `<object>` 法对大部分人来说都是有用的，但仅仅大部分是不够的。一些设计师采用 `<object>` 法，因为大部分时间，`<object>` 法在几乎所有

的浏览器上都正常工作，但对于另一些设计师，嵌入丰富多媒体内容的需要和达到标准，二者是不可调和的。



图 12.11

已经安装了 Flash 3，标记也合法，但有些用户还是看不到 Flash 文件

用 JavaScript 的 document.write 来欺骗校验系统

为了同时实现嵌入 Flash 内容和通过 XHTML 校验，一些设计师用 JavaScript 浏览器检测和 document.write 来隐藏非法的<embed>标签，以通过 W3C 的校验检测系统：

```
<!-- 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" ...
```

这种技术是有用的，在所有支持 JavaScript 的浏览器（除非用户认为“JavaScript 是安全隐患”，并关闭了 JavaScript）上都能正确地显示 Flash，也骗得 W3C 的校验系统认为网页标记是合法的。

欺骗校验系统和达到适应性不是一回事。如果你的客户要求在适应标准的同时，可靠地嵌入多媒体元素，当然这种要求不太可能出现，JavaScript 法可以帮你满足这两个要求。但我们建议你最好用最简单的语言跟客户和老板解释清楚前面描述过的问题，并接受这个事实（至少是目前）——你的网站中嵌入多媒体的那部分代码是非法的。

嵌入 Flash 或 QuickTime 内容的同时，不能满足 XHTML 适应性只是一个很小的问题。随着浏览器处理<Object>标签能力的不断提高，我们终将能够大胆

地往具有良好适应性的网站里嵌入多媒体内容。

12.5 一个平凡的工作区世界

事实上现在没有一个浏览器是完美的，甚至有一些根本谈不到完备，但是 *workaround* 开始帮助我们实现标准的承诺（创建一次，随处发布）。*workaround* 使我们能自由地改善站点的内容、设计和可用性，而不必把很多时间浪费在私有的、无出路的技术上。但不是每个人都对 *workaround* 满意，无论是实践还是它的益处。一部分人认为，如果浏览器不能全面或正确的支持特定的 W3C 规则，对浏览器用户来说就太糟糕了，或者就应该避免使用标准。这个观点的问题很多，包括以下这些：

- 如果你限定自己使用所有浏览器都能完美、精确支持的规范，就根本无法创建这样一个网页。即使是 HTML 3.2 也不能得到所有浏览器完整准确的支持。
- 为了达到标准，要花费很多的时间……时间在竞争激烈的市场里是很宝贵的。商业压力使工程师们在确定每个详细规则的细微差别之前就将其投放到市场上。他们经常在明知有 bug 的情况下必须推出软件（Netscape 6.0 和 IE 6.0 都有 bug，比如本章前面提到的 IE6 漂浮属性 bug，而制造他们的工程师也知道他们有 bug）。
- 有时规则在某些地方很含糊，还有一些规则公布之后又被更改了，CSS2 就是一个例子，在前几页我们说过，2002 年 CSS2 被修订为 CSS2.1，因为原来的版本中有一些含糊或不能执行的规则，整个 CSS 制作组都在不断地探索。同样，在 CSS1 的规则中，相邻字号的关键字之间的缩放比率因数为 1.5，但是 Netscape 4 和所有的浏览器都完全按照 W3C 认为浏览器应该执行的关键字缩放比率来执行，结果小字体太小，大字体又太大，后来这个规则就被修改了。
- 因为老式浏览器的 bug，我们大部分人应该得到补偿，特别是那些还在广泛使用的，如含有错误执行的 CSS 盒模型的 IE5.x/Windows。

请注意，本章提到的 *workaround* 都具有完美的标准兼容性，我们不需要写专门的代码，或写一大堆的标记以克服浏览器的 bug。我们将用标准方式来支持标准模式，比方说，在盒模型程序中，我们将用 CSS2 选择器来保证含有失效盒模型的浏览器能够正确表现我们的 CSS1 布局要求。从标记上删除不必要的空格，

努力创建可访问性高、易读的文档。

这些技术本身没什么错误，它使我们能使用标准而不用寻求完美的浏览器。随着浏览器的不断改进，老式浏览器的废弃，我们需要的 `workaround` 将越来越少，虽然我们可能还会用它们来保证网站在向前兼容时和向后兼容时看上去的效果一样好。下一章，我们将以检验设计的一个最基本方面，即对排版的控制来简要总结浏览器的优点和缺点。

使用浏览器 第三部分：排版

育以来，定位、颜色和排版都是设计中最基本、最主要的工具。印刷设计师花了几年的时间研究字体类型的历史和应用。他们学习区分不同字体的样式，例如 Arial 和 Helvetica 这两种字体的样式差别，对于没经验的人来说，它们的样式是一样的。当这些有传统教育背景的设计师转向 Web 设计时，由于他们的局限及 Web 与传统印刷具有截然不同的排版工具集，他们总是不如那些未受过传统设计教育的人得心应手。

13.1 字号问题

Windows, UNIX 和 Macintosh 根据不同的分辨率采用不同的默认字体——从像素化到轻微锯齿（在 Mac OS 9 中），这样的字体边缘会像 Photoshop 文本一样平滑。（比如在 Mac OS X 的默认 Jaguar，比如 Windows XP 里用的 Cleartype——用户必须打开 Cleartype 选项。）以前的``在不同的操作系统里指定的尺寸和外观都不一样，虽然主要的浏览器制造商一直努力建立一个标准的字号默认值，以减少或消除跨浏览器所产生的问题，但大多数 CSS 的字号指定方法还是会造成本不同的操作系统里的字号的不同。

13.2 用户控制

除了平台、过时的方法和 CSS 表现样式的不同造成的差异以外，网页的打印样式也不同。因为用户应该有权控制打印的效果，用守旧的方法是很难满足用户的这种需要的，其原因本章将会讨论到。传统的设计师也很难接受让用户控制的前提。更可惜的是，可以给用户控制权的 CSS 方法（em、百分比率、字号关

关键字)在跨浏览器和跨平台时都产生了问题,曾经有一段时期,这些问题被浏览器制造商们解决了,因为他们同意统一支持一个跨平台的标准尺寸。但是新浏览器很不明智地撤销了对那个标准尺寸的支持,使得在满足设计要求的同时,要满足用户的控制权更为艰难。

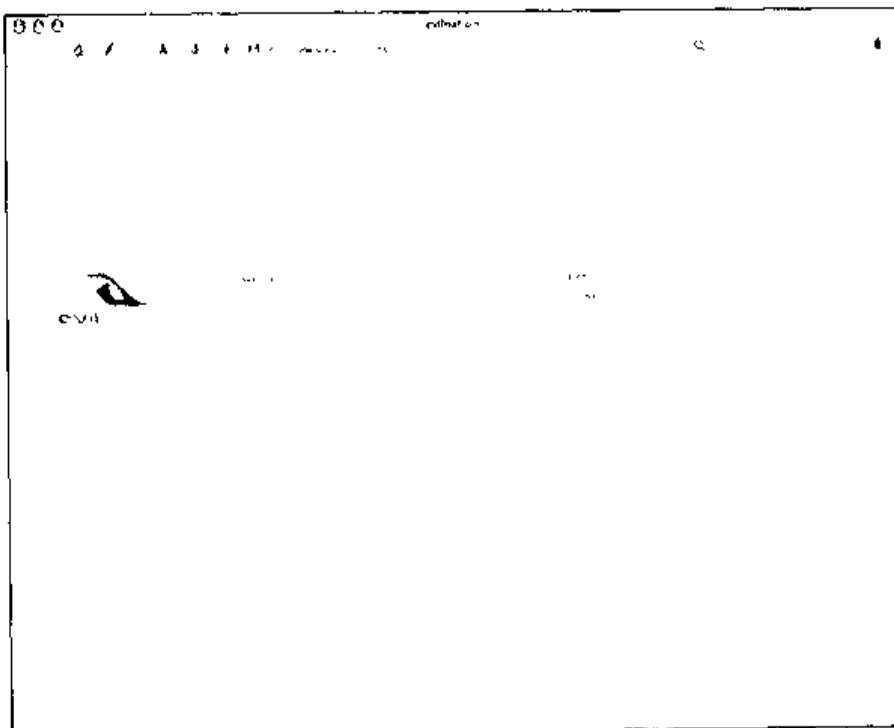
本章我们将讨论网页排版的历史和所产生的问题,学习两个通过 Web 标准设置文档的方法。两个方法都很有用,但没有一个是完美的。我们还会看一下更不完美的方法,它们也存在更多的问题。我们将涵盖这些技术的优点、缺点和有时候会出现的问题,你来决定哪一种方法更适合你的访问者。

13.3 守旧的方法

Web 的发明者和 W3C 的创建人 Tim Berners-Lee 认为他的创造是方便文本文件交换的一个媒介,所以他没有在 HTML 的结构语言里写进排版控制属性。在第 2 章“根据标准设计和制作”里的“跳出那个禁锢”一节中已经说过,网页设计师最初是用<tt>, <pre>, <blockquote>和语义上没有任何意义的段落标签来改变字体,实现定位的效果并模仿行距的。后来他们开始用文档的 GIF 或 Flash 图片,现在还有很多熟练的设计师在用这种方法(如图 13.1 所示),因为他们很难接受 CSS 和 XHTML 的限制,也很难在满足用户需求和设计两者中取舍。

图 13.1

该页面上的每个“字”(<http://www.evinformation.de>)都是一个 Flash 失量,而不是真正的文本,所以不可搜寻,不可访问,也不能被复制和粘贴。注重排版的熟练设计师很难接受 XHTML 和 CSS 的限制,也很难取舍用户的控制权。



直到 1995 年，随着商业性网站的不断涌现，设计师们把 HTML 的作用发挥到了极致，至少提供一些基本的排版工具。Netscape 给我们提供了指定字号的 `` 标签，你可以在用户默认 (``) 的基础上指定数字 (``) 或是相对数字。设计师们很快放弃了段落标签和其他的结构性元素，而改用组合使用 `` 和 `
` 标签来控制他们的布局。微软还给我们提供了 `` 标签。

这些标签也产生很多问题，主要是由平台之间的差异造成的。在 96ppi（每英寸像素）的情况下 Windows 预设默认的基本尺寸为 16px。Macintosh，很像印刷设计标准，以 PostScript 标准预设在 72ppi 情况下基本尺寸为 12px。在一个平台上合适的字号在另一个平台上看起来不是太大就是太小。以 Mac 为主的设计师认为 Windows 预设的字号很不科学，而以 Windows 为主的设计师则认为 Macintosh 的很不科学。

13.3.1 磅值产生差异

如 IE3 上的早期 CSS 执行程序，在很大程度上解决了跨平台所产生的问题。磅 (pts) 是一个打印单位，不是屏幕单位。设计师对磅这个单位应该很熟悉，很多人选它作为指定网页文档的单位。在 Windows 系列产品里，7pt 型的字高为 9px，这是能看清楚的最小尺寸。在 Mac 里 7pt 型的字高为 7px，很难看得清楚。

1997 年，微软公司网站 (Microsoft.com) 用 7pt 型字的文章（如图 13.2 所示）大肆宣传他们为 Windows 和 Macintosh 设计的新 IE3 浏览器的威力。但是这么大的字在 IE4.x、Macintosh 的 Netscape4 上根本看不清楚，不是浏览器出了什么问题而是平台之间的差异造成的。Todd Fahrner 随后发明了 DOCTYPE 转换（见第 11 章“使用浏览器 第一部分：DOCTYPE 转换和标准模式”），他在他个人的网站里张贴了图 13.2，注释说磅值对于屏幕设计来说是一个无用的 CSS 单位（尽管对于打印样式表很有用处）。

Fahrner 还指出用磅和像素值设置的文档尺寸不能通过浏览器内置的字体大小调节部件来调整，这倒不是因为 CSS 认为磅和像素是固定单位，其值不可改变（CSS1 并没有定义任何设置尺寸的方式为不可调整的设置方法），只是第一个支持 CSS 的浏览器制造商是这么设置的。用户可以用 em¹ 和百分比值重新设置字号，但是用它们设置字号使 IE3、IE4、Netscape 4 都出现可怕的 bug。Fahrner

1. em 是一种字号单位，相对于实际字号单位磅值 (pt) 来定义长短。例如，如果现在的字体大小为 12pt，那么 $1.5em=18pt$ 。

将很快提出一个解决这些问题的方法。

在图 13.2 中，惟一能在跨浏览器和跨平台时不产生差异的方法就是用不可调整大小的像素设置字号。5 年后的今天，惟一可靠的单位仍然是在 IE/Windows 里不可调整大小的像素。

图 13.2

在这个 1990 年中期的屏幕快照中，Todd Fahrner 评述了 CSS 试图砌作为指定屏幕的单位所产生的问题 (http://style.cleverchimp.com/font_size/paints/font_warts.GIF)。由于平台的差异，这么大的字在 Macintosh 上通常看不清楚的。



13.4 目前使用的标准尺寸，但是它又能延续多久

为了减小跨平台时产生的差异，Netscape、微软浏览器和 Mozilla 的制造商于 1999 年下半年召开会议，一起确定设置一个跨平台的 16px/96ppi 标准字号默认值。通过把相同的页面放在所有的平台上，浏览器制造商和用户就能避免跨平台时出现的尺寸差异和没用的、不可辨认的文档。

16px/96ppi 字号标准

在 1998 年的 W3C 的邮件发送单中，Todd Fahrner 就提出在每个 Windows 用法里使用标准化的 16px 字号默认值。2000 年，所有主要的浏览器制造商都采纳了他的这个建议。虽然 Fahrner 关心的是实用性（他想保证字的在线读取），但是下面所引用这些话可能会使你觉得奇怪。

CSS的创建人认为必须根据Web用户的“平均”臂长来定义一个像素的尺寸，事实并非如此，Fahrner在提出跨平台的标准字号的主张时，同时也包括了非常重要的臂长及可用性等问题，他写道：

在Mosaic之前，所有主要浏览器上的字号默认值都被设置成12pt，我提出要把字号默认值重新设置为16px...目前所用的12pt默认值在跨平台时产生巨大的差异，在Mac上，字号只有12px（逻辑上为72ppi），而在Wintel PC上，字号被默认成16px（逻辑上默认为96ppi），所有字号的调整都是相对于这些不一致的基数进行的。对于一个设计师，这就意味着惟一实现[跨平台时的一致]的方法就是用CSS像素单位设置字体大小，而用户是不能调节像素的，所以它不是最理想的单位……

在Mac浏览器上正确的纠正方法就是打破传统，设置“中等长度”的文档，字号的默认值为16px而不是12pt。虽然还是不能支持用户调整字号，但是一个始终如一的初始值至少使得可调整的字号值对于设计师来说不再那么成问题，因为任何的默认值变化，都是由于用户的优先设置而不是由于OS差异。

设计师一般认为16px作为基数太大，为什么要把它作为默认值呢？原因之一是纯粹为了方便，虽然Mac在网页设计领域很有代表性，但是它是只拥有少部分用户的平台。期望Windows/X11改变其默认值以匹配Mac的72ppi限度是不现实的。

1996年CSS1标准为“参考像素”提出了一个1/90英寸值，这个值是从臂长的0.0227°视角推断出来的，如果已知的物理分辨率与这个值相差太大，[用户代理]必须适当地调节像素的大小。一个1/90英寸参考像素使12pt的光栅转换成15px，而不是16.15px，当然15比12更接近16，然而，因为OS/UA目前不采用90ppi的逻辑分辨率（像素的大小调整执行程序也不采用这个值）……参考值应该被修改为1/96英寸。赋给参考用户一个长的臂长就很能保持0.0227°的视角。

设计师认为16px太大了，仅仅是因为他们已经适应了Mac把12px作为基数尺寸。<http://lists.w3.org/Archives/Public/www-style/1998Dec/0030.html>

两年后，第一代标准兼容性很高的浏览器采纳了Fahrner的建议。在Internet Explorer、Netscape、Mozilla、Windows和Macintosh平台上默认的文档尺寸都修改成了16px/96ppi，可读性的提高受到了全球用户的欢迎。

Fahner 的努力最终使 W3C 同意采用与 16px/96ppi 概念相关的标准参考像素尺寸，这个尺寸可以在 CSS2.1 工作图中找到。在下面的摘录中，你将注意到 W3C 公司仍然很关心读者的平均臂长问题，还好，他们说的只是一个手臂的长度：

有人建议把参考像素定为像素密度为 96ppi，离读者距离为一个臂长的装置上一个像素的视角，如果标定的臂长为 28 英寸，那么视角大概就是 0.0213°。

——CSS2.1 工作图, <http://www.w3.org/TR/CSS21/syndata.html#length-units>

虽然像素为 W3C 的标准默认尺寸扫清了道路，但是这两者间没有什么正式的联系。

Netscape 6+, Mozilla, IE5+/Macintosh 和 IE/Windows 为所有的用户提供相同的字号默认值的设置，只要用户不改变它的优先选择，在跨平台时，磅值、em、百分比值和字号关键字就不会发生变化，设计师和开发商对平台之间差异的疏忽不会再给用户带来不必要的损失，虽然在一些浏览器上，对百分比值和 em 的执行还存在 bug 和继承问题（后面几页将具体讲讲这些问题），但是最主要的障碍被清除了。

浏览器制造商不解释他们为什么这样做，网页用户也不去研究这些设计上的问题，用户如果更改了优先选择或者没领会要点，就不能获得使用标准尺寸的好处。

13.4.1 好工作会被随便一个单击而否定

因为觉得文档的尺寸太大不美观，一些 Macintosh 用户立刻把他们的浏览器改回到以前的 12px/72ppi 设置，这样做不仅否定了标准化的努力而且使他们自己再一次陷入了在很多网站上不能阅读文档的境地。在网络里，用户自己作主，即使用户并不清楚他们最感兴趣的是什么。

如果是设计师把设置“改回到以前的 12px”，那他的站点可能在 Macintosh 上看上去很好，但是在用户数目占优势的 Windows 上肯定出现问题。很多基于 Macintosh 的设计师用了 12px/72ppi，结果他们的站点出现问题。

当然，也有很多 Windows 用户发现 16px 的默认尺寸太大，他们不喜欢，所以经常把浏览器上的文档尺寸设置在标准以下。Windows 和 Mac 用户这样做对使用像素指定字号的站点没有影响，但对用 em、百分比值和 CSS 字号关键字的

网站来说问题就大了。我们将在本章稍后讨论这种情况。

13.4.2 探寻被遗忘的角落：对浏览器上变化的错误反应

把字号转换成低于标准尺寸的用户可能感觉更舒适，他们不是惟一没理解标准尺寸要点的人，很多网络专业人士也不得要领，部分原因是 Netscape 和微软并没有公布标准尺寸问题的关键所在。

特别是 Quirk Brigade 浏览器，让人们以为浏览器在外观和行为上本来就有很大的差别，而解决这些差异的方法就是联合使用浏览器检测系统、代码分支和标签。

2000 年以前，开发商用的是磅值，而不是像素来保证文档在几乎所有浏览器和平台上呈现的尺寸都一样，因为磅值在两个主要的计算平台上执行的方式差别很大，而且对于屏幕来说磅值没有任何意义。开发商们创建了多重磅值驱动样式表，用平台检测系统提供一个磅值驱动样式表给 Windows 用户，另外一个是给 Macintosh 用户¹。

蛇吞下它的尾巴：有条件的 CSS

2000 年，随着支持跨平台的相同默认字体尺寸和分辨率的标准兼容浏览器的推出，网页设计师可以摒弃浏览器检测系统、磅值驱动样式表和有条件的 CSS 了。但是 Quirk Brigade 浏览器却更新它们的脚本，匆匆推出新的有条件的 CSS。

扩展的脚本和有条件的 CSS 没有像预期的那样发挥作用，像在第 1 部分“休斯顿，我们出问题了”里所描述的那样，它们总是造成难读的，或者被奇怪地格式化过的页面。它们所提供的脚本和构思糟糕的样式表总是像一个有故障的机器人服务生。

大多数创造出这些没有可用性东西的开发商并不是愚蠢，他们都是经验丰富，薪水很高，资深的专业人士。可是他们的客户却要把大量的资金花费在这些复杂的，从不正常工作的网站及其昂贵的日常维护上，这并不是人们的愿望，但却是事实。

最后资金耗尽，网站所有者和访问者就看见了第 1 章里描述的那个废弃的网站。

如果我们明白浏览器支持的一般标准，明白最好的方法就是最简单的方法，我们就可以防止上述那种情况的发生（也就是说，给所有的浏览器指定相同的样式表，只在打印里用 pts）。

13.4.3 Chimera 和 Safari：性能优良，尺寸不佳

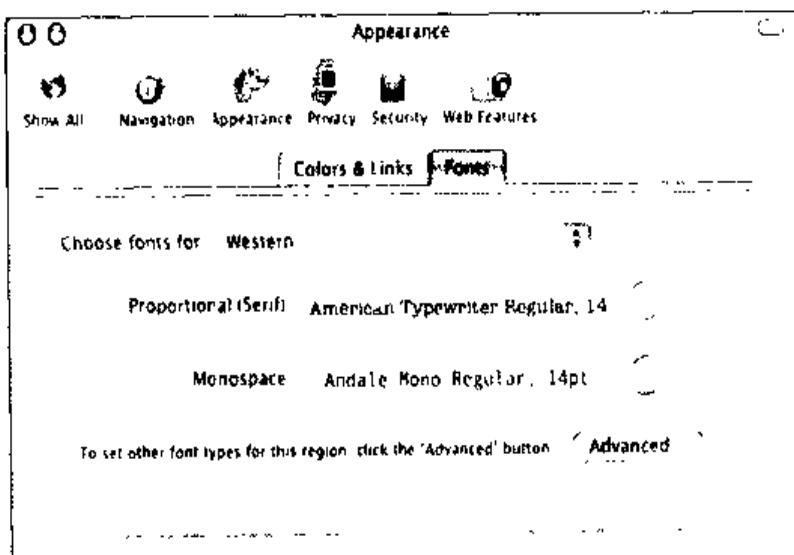
在本书即将出版的时候，为 Macintosh OS X 设计的浏览器投放市场，虽然在很多方面表现出色，但是少了和标准字体尺寸有关的一个功能。主要因为基于 Gecko 的 Chimera（Navigator）浏览器和苹果公司的主要基于 Kmeleon 的 Safari 浏览器在写这本书的时候都不是处在主导地位，但它们分布广泛，也有很多的支持者。这两个浏览器都为 CSS 和其他标准提供可靠的支持，特别是 Chimera 的支持非常出色，因为它结合了 Mozilla/Gecko 的标准兼容性，优美的文档呈现效果和先进的 OS X 用户特征，你读这本书时，可能 Chimera 已经出于法律的原因把名字改成了 Camino(<http://www.Mozilla.org/projects/camino/>)。我们将继续讨论像 Chimera 这样的浏览器。

现在的 Chimera 和 Safari 都没有把字号默认值设置成 16px，而是设置为成了 14px。既不是标准设置也不是 2000 年以前的 Macintosh 默认设置。制造商们似乎为了节省开支，给 Mac 用户设置了比标准小的字体，但是还不至于小得使文档看不清楚。像 Goldilocks，他们似乎认为 14px 很“合适”。

这么大的字号可能正是 Macintosh 用户所希望的。的确，两个浏览器的优先选择控制面板（如图 13.3、图 13.4 所示）都使得转换成 16px 有些困难，16px 没有作为默认尺寸的一种，虽然用户愿意的话，可以人工写入这个数字。一些用户也改变浏览器的默认字体和字号，但是极少人是基于对标准的充分认识而去修改默认字体的。这样，除非网页的文档已经被指定为像素值，OS X 的两个浏览器都将以默认的字体显示。

图 13.3

OSX 的主要基于 Gecko 的 Chimera（Navigator）浏览器提供极好的标准兼容性，但是它没有把字号默认值设置成 16px，优先选择控制面板也没有提供明显的方法能把字号转换成 16px（如果用户想转换）



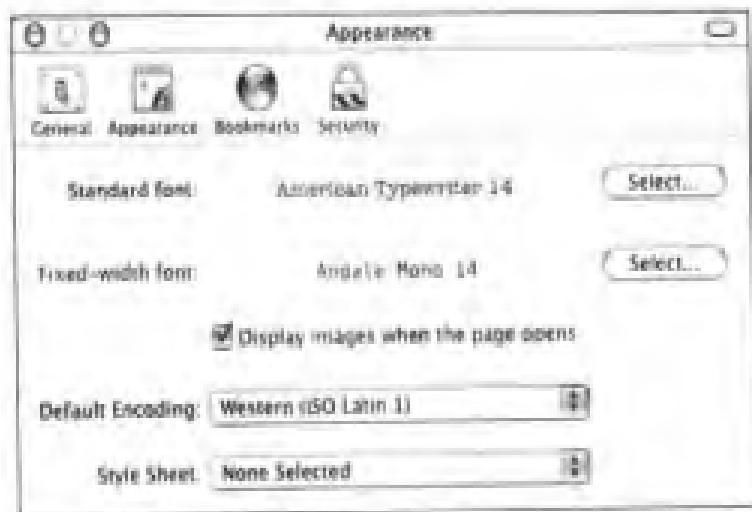


图 13.4

一直支持标准的苹果公司的 Safari 浏览器设置的也不是标准字号，很令人吃惊。和 Chimera 一样，其优先选择控制面板也不太可能引导用户选择 16pt 的标准字号设置，在 OS X 的两个浏览器里，当字号为默认设置时，Mac 用户必须再一次忍受小字号文档。

非标准尺寸的效果

图 13.5 到图 13.10 显示的是我们在第 8 章“XHTML 示例：混合布局第一部分”里创建的 i3Forum 网站，我们使用了 em 以保证用户在任何浏览器上都能重新设置文档的字号，即使是在非常顽固的 IE6/Windows 浏览器上。你会发现页面的高度有些变化，这是因为每个浏览器上的用户界面颜色深度不同。为了平衡发展成本，Opera 的免费版（如图 13.8 所示）里包括了付费广告的横幅，从而产生了大量的垂直间距。



图 13.5

我们在第 8 章到第 10 章创建的 i3Forum 站点使用 em 控制字号，这里看到的是在 Macintosh 平台上 Netscape 7 里的效果。（www.i3Forum.com）

图 13.6

这像是在 IE5/Mac 上的效果，虽然因为浏览器上的颜色差别，网页颜色看起来有点深，但是尺寸没有变化。



不考虑这些差别，很容易看出文档在 Netscape 7/Macintosh(如图 13.5 所示)、IE5/ Macintosh(如图 13.6 所示)、IE6/Windows(如图 13.7 所示)和 Opera/Windows (如图 13.8 所示)上的尺寸都是一样的。这说明浏览器和平台的检测系统和条件 CSS 为不同的平台提供不同的磅值。

图 13.7

当我们把它放到 Windows XP 和 IE6 上时，尺寸还是一样





图 13.8

同样，在 Opera 7 上，文档的尺寸也没有变化。在所有浏览器上设置相同字号默认值的好处应该已经很明显了，文档不但清晰可读而且在所有的浏览器上都可以调整其字号。



图 13.9

在 Chimera (Navigator) 浏览器上，文档看上去太小而且很难阅读，虽然很容易重新设置文档的字体大小。但比较麻烦——一些用户可能还不知道怎么改变字号。

你还能看到文档在 Chimera/Camino（如图 13.9 所示）和 Safari（如图 13.10 所示）里的字号比在其他浏览器上的小得多。文档的确太小了，这个问题在 serif 字体——如现在所使用的这个 Georgia——比在 sans serif 里要明显。sans serif 线条很清晰，即使字号很小也可以看得清楚。如果 Chimera 和 Safari 用户发现页面很难阅读，他们可以用文本缩放毫不费力地调整字号，解决问题。但不是每个用户都知道文本缩放，有些用户可能不知道怎么才能修改令人头痛的小字体。

图 13.10

与 Chimera 一样，苹果公司的 Safari 上显示的字号比默认的小。我们只能指望 Chimera 和 Safari 的工程师们能明白支持一般 16px/96ppi 字号默认值是明智的行为。



我们只能指望那些优秀的、有标准化意识的 Chimera 工程师们和同样天才的 Safari 技术人员们能明白支持一般 16px/96ppi 字号默认值是明智的行为，并在下一个更新版本中这样设置。（也许在你读这本书的时候，他们已经把默认尺寸转换成了 16px/96ppi）。在使用 em 相对尺寸时我们按照可访问性提倡者的一贯建议去做，但是效果不好，除非所有的浏览器都支持相同的默认字体尺寸。你将会看到它在其他情况下也不能很好地作用。

13.5 em 理论的失败

提高可访问性的拥护者和 CSS 的创建者一致认为 em 是发展的趋势，可惜 em 并不是。所有的讲课、书籍和文章都提倡用 em 指定字号。但 em 的完美理论在实践中却行不通，而且在浏览器上未能支持一般字号默认值。

em 在老式浏览器上就有问题，Netscape 4 忽视指定给文档的单位：em 和 ex。虽然这些单位用于指定行高时不出任何问题，IE3 把 em 当做像素处理，2em 高被错误地理解为 2 像素高，虽然现在几乎没有人还在用 IE3，但是问题依然存在。

同样，老式浏览器总是把使用 em 单位指定的嵌套元素的继承属性弄得乱七八糟。因为用 Netscape 4 的人越来越少，我们就不在这里详细地解释浏览器对用 em 指定的嵌套元素的错误处理了，只要知道会产生这些问题就行了（如图 13.11 所示）。

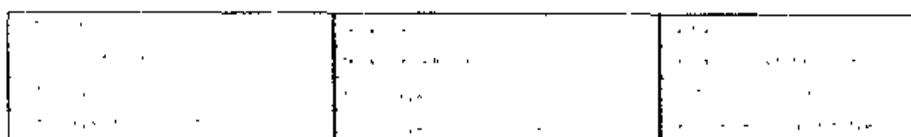


图 13.11

用 em 肯定不能使尺寸在跨平台、跨浏览器时保持一致性
(http://www.thenoodleincident.com/tutorials/box_lesson/font/)

13.5.1 用户的选择和 em 单位

使用 em 作为单位最常见一个问题就是用户常常缩小默认的字号，Mac 用户总是把他们的设置恢复成 12px/72ppi，Windows 用户在设置浏览器视图的时候把文档的尺寸设置为“小”而不是“中等”，这些改变使得字号小于 1em 的文档看上去比预期的小，而且很可能太小以致看不清楚。2002 年，CSS/DHTML 专家 Owen Briggs 在很多浏览器和平台上检验了每个现有的文档字号设置方法，从而发现了哪些是有用的，哪些是没有用的，在比较了 264 张屏幕快照后，他得出结论：用 em 是行不通的（如图 13.11 所示）。

只要你不把文档的尺寸设置的比用户的默认尺寸小，只要用户不调整他们的优先选择，em 就不会出任何问题。但是大多数设计师和客户喜欢小一点的字号，而且很多设计也要求用小字号，还有很多用户认为 16px 的默认尺寸不适合普通阅读，于是他们改变优先选择设置。当 em 用于设计网站时，设计师和用户调整过的设置就会发生混乱，造成文档很难阅读或是根本看不清楚。

如果你用 em（或百分比）做单位设置小字体，用户的优先选择设置会产生混乱，在你的显示器上看上去效果很好的字体，在你的访问者的显示器上却小得可怜，这样肯定不能增强网站的可访问性。

在 i3Forum 网站，我们把字号设置得比 1em 小一点点，为的是尽量减小可能出现的损失。只要客户允许，看上去美观，你可以选择在设计网站时，只在设置一般或比较大的字时用 em 单位，(<http://www.alistapart.com/stories/dao>) 这样能避免尺寸造成的可访问性问题。但是极少设计用 16px 或者更大作为默认尺寸——一些用户抱怨网站不美观就是因为文档的字号“太大”。如果这样都不能使人人满意，那么用 em 单位指定你的文档尺寸就更不可能使人人满意。

一些标准的支持者和可访问性的提倡者可能不相信我们所说的这些，“人们并不想接受太多的事”，说这话的人是对的。

那么人们想要什么呢？本章最后介绍的两个方法可能就是他们想要的，这两个方法看上去比我们前面介绍过的方法好用得多，但是它们也存在一些问题。

13.6 像素的作用

除了下面我们将要讨论的一些比较例外的情况，在 CSS 值中，像素是在新旧版本、兼容和不兼容标准的浏览器和任何平台上都能作用的单位。无论是在 Windows、Mac OS 或在 Linux/UNIX 上看，13px 都是 13px。以像素设置的文档在 Gecko/Mac OSX（如图 13.12 所示）、IE6/Windows（如图 13.13 所示）和 Opera7/Windows（如图 13.14 所示）上的效果都一样。

图 13.12

2003 年 2 月的 zeldman.com 重新设计 (<http://www.zeldman.com/>) 的开始部分。文档以像素设置，因此在 Mac OSX 的 Chimera 显示的是同一尺寸



图 13.13

如在 Windows XP 的 IE6 上一样……不仅如此……



像素不仅用于设置字体，也用于创建图片。如果我们要创建的图片尺寸为 200×200px，用 CSS 指定左边距为 100px，图像的大小与边距应该是 2:1，所有的用户都将看到这样的比例。我们把创建这种基于尺寸关系的性能称为像素制。基于尺寸关系不仅是格式设计也是定义式设计的一个组件，我们将在第 16 章“CSS 重新设计”中具体讨论它。

像素对于清晰的布局来说必不可少，如果我们设置标题高为 25px，左边距 100px，文档主要部分的字体为 11px，不考虑用户个人电脑的分辨率和显示器的尺寸，所有用户看到的尺寸都将相同。



图 13.14

不像理论家们所说的，Opera 总是缩小像素的值。文档在 Opera/T/Windows 上的效果相同。

13.6.1 最小的单位：它是相对而言的

分辨率为 1200×870 的情况下，一个像素在 22 英寸显示器上略显笨重，而在 17 英寸上又过于细小。因为显示器和分辨率的不同，CSS 把像素作为一个相对单位，从屏幕的一角到另一角，从一个屏幕到另一个屏幕，10px 还是 10px。像素总是一个特定屏幕的最小单位，新媒体设计的原子。于是大多数设计师和用户就认为像素值是一个绝对单位。所有的网络用户都熟悉像素——电视机出现以后他们就知道像素了——但都不太清楚它的本质。

如果 CSS 是抽象的理解像素（“离读者平均一手臂长的距离用 90dpi 的像素”），浏览器制造商可不是，他们和我们一样是这样理解像素的：像素是屏幕上最小的点。

事实上所有的浏览器都以相同的方式支持像素，所以用像素作为字体尺寸单位，你（你的用户）将能在几乎所有的浏览器和平台上看到预期的效果。但是，

像任何定义一样也有例外：一方面精致复杂，混浊度低，另一方面是不能处理数据，可被忽视。

像素在 Opera 上失效

一些 Opera 浏览器上显示的字号一直比指定的小，例如，你给文档指定的字号是 11px，Macintosh 的 Opera 5 上显示出的字号接近 10px。

Håkon Lie 是 Opera 的首席技术官员，也是 CSS 的创始人。大多数设计师都有些尊敬他，也因为他认为 CSS 应该抽象的（臂长）而不是以我们大多数人理解的像素是绝对的（“屏幕上最小的点”），也因此他被称为科学家。

有些人认为 Opera 把像素的值缩小处理是基于“平均臂长”的抽象概念，与其这样说，还不如说这是 Opera 的 Macintosh 版本上的一个 bug——关于这个 bug，Opera 似乎并不关心要如何去修复——这么说可能是对的，但是我们注意到 Opera 7/Windows 处理像素的方式和其他浏览器是一样的（如图 13.14 所示）。

假定 Opera 7 现在支持 DOCTYPE 转换，有些读者可能会想知道 Opera 7 是不是只在 Quirk 模式中与其他浏览器一样把像素作为绝对值处理，而在标准模式中又继续把像素作为抽象值处理，这些读者考虑过多了。图 13.14 显示的 Opera 处理像素的方式和我们所理解的一样，这个网页是合法的 XHTML1.0 过渡式 CSS 布局。我们倾向于认为 Opera 呈现的方式与其他的浏览器是一样的，只是理论不同。

总之，用像素指定字号时，大多数 Opera 用户都能获得想要的效果。除了一些用老版本 Opera 的用户可能看到的字号会稍微有点小。

当像素在某些 Netscape 4 中失效时

像素的可靠性还有一个例外情况，一些版本的 Netscape 4 错误理解像素值。在一两个平台上的升级版本中，有时还是会忘记怎样理解像素单位，到底是哪些版本会弄错像素值，我们不知道也不关心，大多数版本的 Netscape 4 都能正确地理解像素值。

Netscape 4 的像素值混乱问题与抽象的理论没有关联，它仅是一个 bug——用户可以通过更新成版本来避免出现这样的问题，或者可以直接更新到 2000 年以后推出的 Netscape 浏览器版本。很多人不愿意更新版本，是因为他们不了解兼容标准的浏览器的种种好处。

13.6.2 像素的缺点

前面说过，用像素也会出现一些问题，现在进行详细的阐述。在 IE/Windows

浏览器上，用户不能调整被设置成像素的文档的大小，如果你把站点的主要文档字号设置为 9px，很多人都会看也不看就退出你的网站，即使是 11px 字型对某些人来说也太小了，这主要是根据所选字体（例如 11px 的 Verdana 就比 11px Time 的适应性小），显示器的大小和分辨率，访问者的视力，前景和背景对比度，有无抽象背景这些情况而定的。对于一个视力为 20/20 的人，这么小的字根本就看不清楚，对于视力严重受损的人来说，情况就更糟。

也有用 32 英寸工作站显示器浏览网页的工程师给网站设计信箱写投诉信抱怨文档的蝇头小字。如果他真想解决这个问题，可以用支持文本缩放或 Page Zoom 的浏览器。如果他喜欢用 IE/Windows 浏览器，可以选择设置忽视字体尺寸（怎样设置将在后面几段详细阐述）。

他还可以书写一个如下的样式表：

```
html, body {font-size: 1em !important;}
```

但是他可能只是抱怨问题而不使用任何方法解决问题。

作为一个设计师，有责任为其用户解决问题，甚至是那些用特别高分辨率浏览网页的人的奇怪问题。

没有尺寸适合

本书出版时，文本缩放推出已经三年了。微软 Mac 浏览器的技术组发明了文本缩放，解决了像素的问题。但是现在文本缩放并没有成为 IE/Windows 浏览器的一部分。在我们游说、请求无效后，我们终于相信微软不会把文本缩放和其他的 IE5/Macintosh 用户友好功能加入到他的王牌浏览器中。

代替文本缩放，IE/Windows 提供了一个“忽视给网页指定的字体尺寸”的选项，在 Internet 选项里的常规标签下的可访问性选项里（如图 13.5、图 13.6 所示）。这个鲜为人知的特性可以让视力严重受损的人使用适合他们的字号浏览网页。

这个要么全有、要么全无的特性与文本缩放（2000 年后的 IE5+/Macintosh、Netscape、Mozilla、Chimera、Kmeleon 和 Safari）或者页面缩放（Opera 的所有版本上）不同。

当用户看不到字号设置的时候，文本尺寸调整没有任何意义。如果设计师书写的是结构式标记，损害还不算大，但是我们知道目前在网页的设计中用非结构式标记的占多数。如果尺寸被 divitis 和 classitis 关闭，那么文本尺寸调整就失去意义了。文本缩放和页面缩放可以帮助读者更方便地阅读内容。虽然 IE/Windows，特别是 MSIE6 是很不错的浏览器，但是隐藏的字号设置不能取代根据用户需求的放大或者缩小文本功能。

图 13.15

IE/Windows 里有一个选项，可以忽略网页指定的字号。用户必须先打开 Internet 选项，找到“常规（General）”标签，单击“辅助功能”

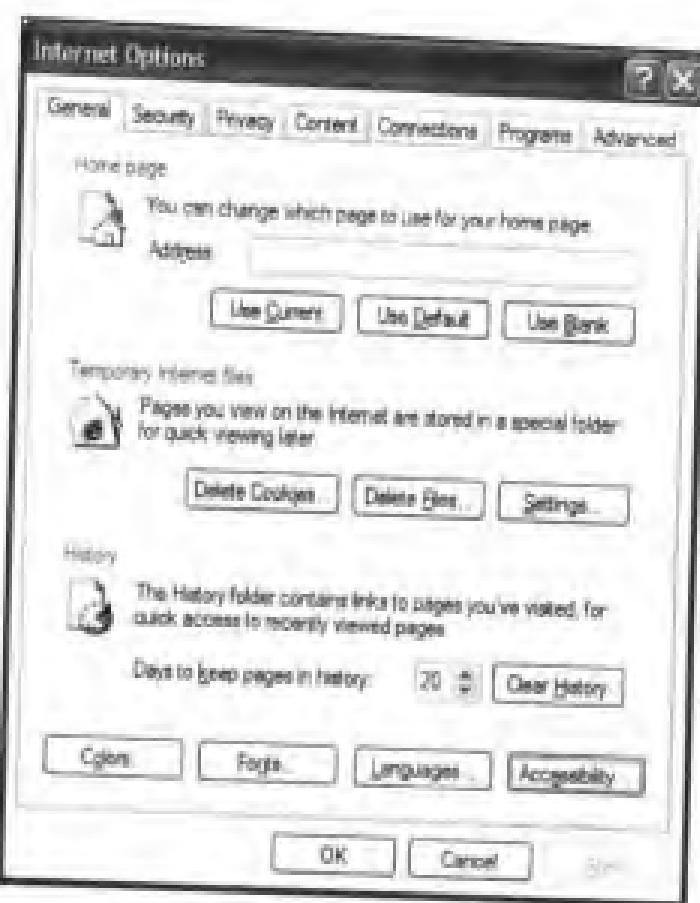


图 13.16

用户必须选中“不使用网页中指定的字号”，然后确认后退出对话窗



假如现在大多使用 IE/Windows，而 IE/Windows 不允许用户更改以像素设置的文档字号，你就得放弃使用这个可靠的跨平台像素。也许你不需要放弃像素，第 15 章“使用基于 DOM 脚本”中我们将探索一个新技术，能让设计师使用像素而不妨碍视力不好的 IE/Windows 用户。

同时我们提供另外一个方法，这个方法在任何浏览器上（甚至IE/Windows）都可以很方便地调整字体的尺寸，并能避免与em和百分比有关的大部分问题。

13.7 字号关键字法

很少人知道，也几乎没人用过，CSS1（后来，CSS2）提供7种字的尺寸，用以控制文档的大小而不存在像素的绝对论、继承、跨平台、用户使用em和百分比单位等问题。下面是7个关键字，买过衬衫的人都能理解他们的意思（<http://www.w3.org/TR/CSS2/fonts.html#value-def-absolute-size>）：

xx-small
x-small
small
medium
large
x-large
xx-large

13.7.1 为什么关键字比em和百分比好

如前面的13.5节“em理论的失败”里提到的一样，当你用em和百分比时，总是存在它们的值会相乘的危险，造成文档的字号太小或者太大。相反，关键字值就不会混合，甚至当元素相互嵌套时也不会。如果<body>是small、<div>是small、<p>是small，以及p存在于div中，而这个div存在于body中，这三个值不会相乘（而em和百分比值就会），结果文档还是能看得清楚，而且，结果仍然是small（不是x-small或xx-small）。em和百分比值会相乘，关键字值不相乘。

另外，至少是在Gecko和现代的IE浏览器上，xx-small不可能小于9px，这就意味着不可能会不可读。也许有些用户读的时候有困难，但这和不可读是不一样的。

和em一样的是，关键字是建立在用户默认字体的基础上的，而和em不一样的是，关键字值不会低于适当分辨率的最低值。如果用户的字号默认值是10px，x-small将是9px，xx-small也将是9px。显然，在这种情况下，x-small

和 xx-small 之间没有差别。

在不与 IE/Windows 的不允许调整像素大小性能冲突的情况下指定字号，并且不会造成不可读的细小字体，这听上去很不错。字号关键字值似乎平衡了可访问性和设计师控制的要求，那为什么会出现问题呢？一句话：浏览器。

13.7.2 关键字执行程序的首要问题

7 种字号关键字值应该被正确、统一地执行，但是事实并非如此。不幸的是，在第一个试图支持它们的浏览器上，它们被错误地执行，即使执行过程是正确的，结果也很糟糕。这使得随后推出的 CSS 标准做了修改。

Netscape 4 基本上不使用关键字，Netscape 4.5 及其高级版本、IE3 把它们呈现为不可读的尺寸。例如，Netscape 4.5 和 IE3 把 xx-small 呈现为 6px，比可读性的临界值小 3px。

Netscape 的工程师们遵循 W3C 早期的一个建议，就是每个尺寸应该是它下面那个的 1.5 倍。如果 small 是 10px，那么 medium——一个尺寸——应该是 15px，x-small 就应该是 6.6667px (10/1.5)，在原来的 1.5 倍缩放比率被证明行不通也不理想后，W3C 把缩放比率改成了 1.2，而证明 1.5 这个比率不可用，不理想的就是 Netscape 4 书写规则的精确重现功能。

Netscape 4.5 严格遵循了 W3C 的建议值，结果损失惨重。Netscape 4.x 很少正确地理解 Web 标准，这是其中的一次。（另外两次结果同样不好，Netscape 4 拒绝给 class 和 ID 名称加下划线，也拒绝让 class 和 ID 名称使用数字开头，在 CSS1 里这样做是完全正确的。）Netscape 正确理解标准时总是受到打击，而错误理解时总是得到鼓励，真为那些 Netscape 工程师感到有一点遗憾。

IE/Windows 上关键字被弄错

同样地，IE4/Windows，IE5/Windows，甚至 IE5.5/Windows 都以不同的方式错误地理解关键字，在这些浏览器上关键字和其呈现方式之间没有逻辑联系。small 就是中等大小，medium 比一般大些，等等。

开发出为 Windows 服务的 IE 的工程师努力做着正确的事情，记得本章开始就说到的 Netscape 的 7 种``标签吗？IE 的开发者们使 CSS 的 7 个关键字直接和 7 种 Netscape 的字号一一对应，在很多方面，这么做是正确的。

当然问题是尺寸和关键字不是一一对应的。在以前的浏览器上，``是默认值或是用户在其优先选择里指定的 medium 大小。在 Netscape 的扩展 HTML 标记中，``是预设了的，除非你指定另一个尺寸。

逻辑上默认值应该与 medium CSS 关键字相对应。遗憾的是在 IE/Windows 的最初方案中，``是与 small 相对应的，而不是与 medium 对应的，因为 small 是列表中倒数第三个值。

有进步，使用率却很低

最终 IE5/Macintosh, Netscape 6+, Mozilla 和 IE6 都正确地理解了关键字，但是那时还有几百户用户在使用 IE5 和 Netscape 4，它们不能正确地理解关键字。如果设计师正确地使用 CSS 关键字，则在 IE4-5/Windows 上不能显示（在 Netscape 4 上也不能显示）。如果设计师刻意误用关键字以达到在 IE4-5/Windows 上的正确显示，则在其他所有的浏览器和 IE4-5/Windows 随后的版本上都不能显示。所以我们大多数人很少使用关键字设置法来设置 CSS 字号。

13.7.3 关键字的时代：Fahrner 方法

Todd Fahrner 再次想出了一个解决的办法。在第 12 章“使用浏览器 第二部分：盒模型、bug 和工作区”里我们介绍过的 Tantek Celik 创建的盒模型程式，使得问题的解决成为可能。你可以在 A List Apart 的“尺寸问题”(<http://www.alistapart.com/stories/sizematters/>)里找到 Todd 法的演变过程和细节，你还可以在版本记录里细读 Todd 法的 Happy Cog 变量(<http://www.happycog.com/thinking/colophon.html>)。

该方法步骤如下：

- 用@import 命令（见第 9 章）对 4.0 浏览器隐藏样式。因为 4.0 浏览器不理解@import 命令，它们不使用包含 CSS 规则的输入样式表。让 4.0 浏览器用户以他们所选择的默认字体大小浏览文档。必要的话，给他们提供一个链接的外联样式表里的像素值。
- 使用盒模型黑客程序给 IE5.x/Windows 提供伪字号关键字值，对于适应性更强的浏览器则提供真实值，就像在第 12 章里我们用它为具有不同能力的浏览器提供边距、边框距和内容区的伪值或真实值。
- 添加一个更为具体的 CSS 规则，以保护 Opera 不出现一种 bug（第 12 章的“Opera 友好”规则里对盒模型黑客程序的讨论）。

该方法的使用方法

下面是一个简单的例子。我们希望段落里的文档字号为 small（比 medium 小一号）。标准兼容浏览器能提供我们所需要的值，IE5/Windows 则不能，我们必须用一些方法获得我们想要的值。如果我们告诉 IE 我们要 x-small，它会给

我们一个适应浏览器上的 small 值，下面是规则：

```
p {
    font-size: x-small;
    /* false value for WinIE4/5 */
    voice-family: "\}\\"";
    /* trick WinIE4/5 into thinking the rule is over
   */
    voice-family: inherit;
    /* recover from trick */
    font-size: small;
    /* intended value for better browsers */
}
```

像我们在第 12 章做的那样，现在我们增加一条“be nice to Opera”规则，因为这条规则的选择器比前面那条规则的选择器更为具体，Opera 会特别注意到这条规则，所以不至于采用只为 IE4/5Windows 提供的伪值：

```
html>p {
    font-size: small;
    /* be nice to Opera */
}
```

只要这条规则是在一个输入样式表里，Netscape 4 就不会注意到它，也就不容易搞混了。如果需要把这个尺寸指定给多重选择器，也很简单，参见 <http://www.happycog.com/c/sophisto.css> 上的样式表，你可以根据自己的需要修改。

美中不足

上面那个方法很好用，但是字号关键字还存在一个问题，这个问题和我们讨论过的困扰着 em 的问题差不多，也就是，任何比用户默认字体尺寸小的字体都可能太小，以至于难以阅读，特别是对于在阐述印刷问题时所说的那三种用户：

- 把视图和 Text Size 菜单设置为 small 而不是设置为 medium 的 Windows 用户。
- 把字体设置恢复为 12px/72ppi（或其他低于 16px 的值）的 Mac 用户。
- 最近推出的浏览器，诸如 Chimera 和 Safari，未能支持虽然在跨平台上很有用，但是比较大而且显得不美观的 16px/96ppi 的字号默认值的用户。

当把字号关键字，@import 和 Tantek 程序一起处理时能保护用户不遭受 em 所产生的最大损失，因为它们能保证字体不小于 9px。但是如果用户设置的默认尺寸比较小，那么 medium 字体也会很小，其他比 medium 小的字体就更小。

13.7.4 可用字体：持续问题

经过了 13 年，虽然网络作为媒体发展成熟，但是还是不能提供可靠有效的字型设置方法，同时满足设计师对控制的要求和用户对自由调整设计初始尺寸值的要求，如果 IE/Windows 像 IE5/Macintosh, Mozilla, Netscape, Navigator 和 Kmeleon 一样，允许用户调整以像素设置的文档大小的话，还是有希望出现这样的方法的，但是想要微软允许用户自由调整文档的大小，恐怕要等上很长一段时间。

这不是一个标准化的问题，文本缩放不是 CSS 参数，也不是其他 Web 标准化参数。没谁规定说用户一定要有权调整网页上的文档大小，常识是这样的，但是常识不是标准。

不要以为这只是标准化设计师的问题，对于那些根本不愿意考虑 Web 标准的人，损失更大，更容易产生难以预料的后果。

专门设计 Flash 的人面临比我们其他人更多的显示器分辨率问题和更大的可访问性挑战，甚至使用用户选择的内容菜单放大 Flash 内容时也不能保证可读性，特别是当 Flash 内容是被包含在框架中时，框架的边界会遮盖被放大的区域。

有时你可以使用像素和提供一个 DOM 驱动文档尺寸小部件来消除 IE/Windows 的限制。有时你可以用 Fahrner 的方法创建可访问的字号关键字，这样在任何浏览器上都能调整字号。

可访问性基础

可访问性和标准有很多相同之处，它们都是为了保证我们的网站拥有更多的用户，使他们更容易访问网站。可访问性与本书所讨论的其他标准联系紧密。20世纪90年代W3C建立了Web Accessibility Initiative (WAI)，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提供了三种访问标准等级，从最容易达到的（优先级1）到要求更多工作才能达到的（优先级2），最后是精通级（优先级3）。如本书中讨论的其他形式标准兼容性一样，一个等级的关键是其可访问性是一个连续统，而不是一个“要么全有要么全无”的东西。就算我们不打算将其转换到CSS布局，也能预先保护我们的网站，并用第8章“XHTML的示例：混合布局（第一部分）”，第9章“CSS入门”和第10章“CSS应用：混合布局（第二部分）”里介绍的混合式方法使其符合标准。只要花一点精力，任何人——甚至是那些新接触可访问性的人都可以掌握优先级1的功能，或至少基本上掌握它的功能。这样，以前不能访问我们网站的人现在也能访问了。

很多国家法律禁止拒绝残障人士的访问，一些国家通过建立网络可访问性法令把这条法律应用于新媒体，如美国的508条款。这些国家法令中的一些严格遵守WAI优先级1，另外一些则从中随机地挑选出一部分来遵守。而一些国家的法令甚至使WAI的成员感到迷惑，虽然他们花了几年的时间研究访问性问题。有一些法令很含糊，而另一些则非常实用。很多法令甚至被创建他们的团体所忽视，更不用说设计师会有多么迷惑了。

本章我们将对应用可访问性做一个实际的总体评价，还将讨论把可访问性融入设计中的工具，并指出这些工具的局限性。仅仅一章的内容是不可能涵盖整个可访问性领域的，如果我们说可以涵盖的话，肯定是针对可访问性专家。实际上很少有书籍能够阐明要点，有一些甚至不经意地加深设计师对可访问性的迷惑和

故意。但是有两本书很不错，以对设计师友好的方式关注于可访问性，我们可以向你推荐。

14.1 有关访问性的书籍

很多关于可访问性的书里面，作为示例的可访问站点外观都很粗糙，甚至是不实际的，还有一些不实用的建议如“不要指定字号”等。这个领域里的一些网站对设计很不在乎，很厌恶，而另外一些在开发商业性站点方面毫无经验。看了这些书的设计师可能就会认为可访问性无关紧要。

还有一些书对访性问题做了深入的研究，但是一般网络专业人士却没有推荐它们，因为它们主要是针对那些或多或少有些残疾的读者，这些书花了大量的篇幅解释如何选择输入法和评价各种用户代理的优点和缺点。这样的书不会给予设计师们从正确的网站可访问性的观念。

我们推荐下面这些书：

- 《建立可访问的网站》，作者 Joe Clark。

Joe Clark 的《建立可访问网站》是网络可访问性的书中写得最好也是最完整的：诙谐幽默、坚持己见和实话实说。在我们这个行业工作的人，大多人都会看最新的设计书和电脑方面的书，这些书很多不完整，或者完全看不懂。从 Clark 的书上你不仅可以学到你想学的东西，而且你读它时充满了乐趣，从书写可用的 alt 属性的基本知识到对很多媒体复杂性的说明，他都能清晰地引导读者，大到大图片，小到最小的细节，提供适合任何受预算和时间限制的可访问性策略。Clark 也同样关心外观问题，他告诉你怎样使设计既美观又具访问性。

- 《构建可访问的网站》，很多作家共同撰写。

包括 Jim Thatcher、Shawn Layton Henry、Paul Bohman、Michael Burk 在内的很多项目专家共同撰写了这本任务集中的书，该书像一本杂志文章的摘要，每个摘要都具体解释了访问性问题的一个特殊方面。这本书大篇幅地介绍了按钮操作的可访问性，测试软件的局限性，还讨论了包括大企业实现可访问性的切实可行的方法，提供详细的制作可访问的 Flash MX 技巧。

两本书都是针对有一定设计、建立、拥有或管理网站经验的读者而写的。

阅读这两本书可以帮助你推翻一些广为流传的错误观念，下面我们就来看看这些错误的观念。

14.2 普遍的错误观念

面对这些可访问性的说明，许多设计师、开发者及网站拥有者都可能滔滔不绝地说出一些无意义的关于服务客户的条条框框，而当接收到类似于 U.S.508 条款的可访性规则时，又常常会陷入窘境。

14.2.1 天才的参与

在很多场合，我们听到过很多备受尊敬的网络设计师用这样的废话回答用户关于可访问性问题：“我们的工作主要是针对我们客户想服务的大用户。可访问性只占市场的很小份额，我们客户不在乎失去一小部分的用户，我的意思是，我们的客户制造高清晰度，宽屏幕电视机，盲人是不会买这样的电视机的。”

实际上，如果网站能让他们阅读产品规格说明而使用在线定购服务的话，盲人也许会为视力正常的搭档或者家庭成员购买这样的电视机。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。要求提高可访问性的大部分是低视力、色盲和轻微近视的用户，他们很可能希望，也愿意购买高质量，大画面的电视机。

“盲人亿万富翁”

此外，网虫也可以说是盲人用户。在我们最近参加的一次会议上，一个发言者提出 Google 搜索引擎是网络最大的盲人用户，这个“用户”每天每分钟为成千上万的客户以搜索结果的形式提供建议。

另一方面，Google 的读者总数使搜索引擎很像一个盲人亿万富翁，有多少网站会对一个有着上亿身家的潜在用户说“不”呢。光看看美国的残障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。

可访问性问题不仅是视力受损用户的问题

可访问性问题不仅是视力受损用户的问题。运动机能受损的用户（部分或完全瘫痪）可能也想买个电视机，更愿意在网上购物，很多可访问性辅助功能主要是针对这部分用户的。可访问性辅助功能也帮助那些使用 Palm Pilot 或网络电话浏览网站，打算买高质量、大屏幕电视机的健全用户。

总之，那些说“盲人顾客不会买我们产品”的设计师、开发商和网站所有者大错特错，他们自己盲目地拒绝了本不需要拒绝的用户——包括上百万通过搜索

引擎找到他们网站的健全的用户，遗憾的是，他们不是惟一错误理解可访问性作用和服务的人。

一大堆模糊的观念

这些年，我们听到了很多网络专业人士说出的误导性的言论，包括以下这些：

“我是一个设计师，我制作漂亮的东西，我不能想像为盲人制作漂亮的东西有什么用。”（答案是：几乎所有的可访问性辅助功能对可视设计都没有影响，因为它们只存在于标记中，存在于网站的漂亮外观下面。网站可以既美观又具有很高的可访问性（如图 14.1 所示）。）

图 14.1

网站 (<http://www.spazzola.com/>) 遵守 U.S.508 条款可访问性方针。你能从外观分辨得出来吗？当然不能，这便是重点。



“这是我们客户的问题，如果他们不在 RFP（Request for Proposal）里提出这个问题，我们就不需要考虑这个问题。”一个全球最大最著名的网络代理商之一的主要开发商如是说，不久这个开发商破产了，公司的剩余部分被中亚的一家小公司购买。还有，他说可访问性只是客户的问题，这个观点也是错误的。

“508 条款？这是为政府制定的，可我们不是政府职员。”一个设计代理商这样说道，而他的客户已经取得了合法的授权，提供访问性服务。

“我们的一个委员正在调查，关于这个问题我们迟早会发布白皮书。”在第 508 条款颁布一年之后，美国政府机构的一位高级规划管理人员赞成我们对此事的看法。

“我们是一个 Dreamweaver 工作室，”一个设计师告诉我们，“完全可访问性问题是在更新的版本上处理的，不是吗？”（也是，也不是，Dreamweaver MX

提供非常多的可访问性辅助功能，但是你必须知道怎样使用它们，仅仅打开 Dreamweaver 是不能保证网站的可访问性的。)

14.3 法律和布局

当 U.S. Workforce Investment Act 的第 508 条款通过时，人们更加疑惑(注意：这里我们用美国的例子，从美国人的角度分析问题，但是在任何地方，无论地方法律如何规定，这条原则都是适用的)。

508 条款要求很多网站进行调整，以适应残障人士——从活动能力受限制的到广大视力受损者——的需求，并明确指出可访问性的含义。(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或不具吸引力的网页，“低档”的设计，其实不是这样的。

图片、CSS、表格布局、JavaScript 和其他的目前所使用的网页设计材料都能完全与 508 条款的适应性相兼容，只要多注意一些就行了。本章我们将研究可访问性和 508 条款适应性所必需的一些条件，还有怎样使用智能和可用的工具使你的网站完全符合要求。

14.3.1 508 条款的解释

508 条款(如图 14.2 所示)是 1973 年颁布的 Rehabilitation Act 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(Rehabilitation Act 的 1998 年修正案 [<http://www.usworkforce.org/wialaw.txt>])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、公用电话亭，以及其他用于传输、接收或存储信息的设备，也包括一些网站。

508 条款于 2001 年 6 月 21 日正式成为美国的法律，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。这条法律也应用于政府投资项目和任何采用该法律的州。美国的读者可以在线查看各州执行这条法律的情况。
<http://www.resna.org/taproject/policy/initiatives/508/508Stateactions.htm>.

总之，508 条款用于下列情况：

- 联邦部门和代理机构(包括美国邮政服务系统)。
- 可由提供服务的承建者交付使用。
- 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。

给每个人提供相等的可访问性

508 条款要求每个网站在其权限下为每个用户提供“相等的可访问性”，包括视力、听力受损的或身体有缺陷的残障人士等。

图 14.2

508 条款的官方网站遵守了它自己的方针 (www.section508.gov)，并不是所有的官方网站都遵守这个方针。而且开始 section508.gov 也没有遵守这个方针。在右上方的 ALA 影响的文档字体大小控制器用来弥补浏览器不能使用户调整字体大小的不足。见第 12 章，了解更多关于样式转换器内容。



这些网络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（见第 13 章“使用浏览器 第三部分：排版”中对 IE/Windows 里像素问题的讨论）。细小的导航按钮只有很小的“单击”范围，不方便一些身体机能有障碍的用户使用。闪光或闪烁的页面可能对患有癫痫病的用户产生生命危险。（当 David Siegel 写《网站制造杀手》这本书时可能还没想到这一点 [New Riders, 1997]。）

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、图片和表格布局，只要遵守一定的方针，也没有禁止在网页中插入如 Flash、QuickTime 这样的多媒体元素，这些方针我们将在这一章的稍后部分介绍。当然，大多数适应 508 条款（像大多数适应标准一样）的站点在新版本的浏览器上都比在老版本浏览器上漂亮，但这不是法律的问题，因为网络用户可以通过下载，自由地更新浏览器，大多数受欢迎的浏览器都可以免费下载。

遵守可访问性方针和标准兼容性不仅使更多残障用户能访问你的网站，还能帮助上百万甚至更多的，包括使用 PDA、网络手提电话、不知名浏览器和公用

电话亭的消费者访问你的网站——通过搜索引擎还能吸引更多的访问者。

更多的访问者，更多的阅读者，更多的用户，更多的成员，更多的消费者，这不是很好吗？为什么设计师，开发者和网站所有者要对 508 条款和类似的可访问性规定采取怀疑或敌对态度呢？因为他们对访问性问题的认识一直都是错误的，我们将努力消除这些错误的观点。

14.4 直逼可访问性的错误说法

下面列举的只是网络专业人士对于可访问性问题错误认识的一部分，我们的解释也受到篇幅的限制。

14.4.1 错误的说法：可访问性迫使你要为网站创建两个版本

如果你用 Web 标准设计，并遵循一定的方针，则你的网站对于屏幕阅读器、Lynx、PDA 和老式浏览器和现代的适应浏览器都具有可访问性。标准和可访问性在一个网络文件应该服务所有读者和用户这一点上是一致的。

如果你专门在 Macromedia Director/Shockwave 上设计，为了符合 WAI 的优先级 1 或者符合 508 条款，你就必须创建两个独立的版本。如果访问性已经是你的计划的一部分，那就用标准设计，把 Shockwave 用在最合适的地方。

Macromedia Director 即将推出的版本，将使开发商能创建可访问性更高的 Shockwave 文件，至于这些文件是否能完全符合 WAI 和其他访问方针的要求还不太不清楚。

14.4.2 错误的说法：纯文档网站能给每个人都提供相等的可访问性

适应性技术的发展经历了一很长的一段时间，可以使常规网页上的大部分内容完全具有可访问性，或者至少是部分具有可访问性，而布局不发生变化。（记住：这种改变只是发生在标记中，对网站的外观没有影响。）为残障的访问者提供纯文档网站，假定色盲的用户看不到，或者假定图片对活动受限的人没有用处，还设想这些用户没兴趣在你的商业网站上购物，或参加一个在线讨论论坛。总之，用过时了的纯文档的方法对任何人都没有帮助。不仅如此，创建和维护纯文档网页比给网站添加访问标签和属性的花费要大得多。

14.4.3 错误的说法：可访问性代价太高

根据第 8 章里描述的，创建一个忽略导航链接的代价是什么？或者书写一个

表格摘要的代价是什么？为页面上每个图片打出一个短短的 alt 文本的代价是什么？这些事都可以在几分钟内完成，你可能是以小时计费的，但是除非每小时的计费是几百万美元，否则添加大多数 WAI 优先级 1 和美国 508 条款要求的可访问性的费用都可以忽略不记，特别是这样做能保护你不受到反歧视的指控，要知道打官司可是要花费大量金钱的。

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，例如，给 Real 或 QuickTime 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标题花费就要多些。但是很少网站的内容要求这么具体细致的工作。

大型的网站内容的更新工作都是由非开发（编辑的）人员来做的，给新网页添加访问提示所要求的属性，就如更新内容管理系统一样简单。在动态网站上，就和重写生成网页的模板一样简单。给表格添加访问属性，给表格布局添加结构摘要，对整体模板做其他的、相似的调整都是一次性的投入，回报是更多的新用户和更少的麻烦。

我们总是浪费了大钱，节约了小钱

我们有一个这样的朋友，他总是在买 CD 但是从没有时间听，总是在买 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，但是他两年没画过任何东西。他收得到所有的有线频道，但他从不看电视，因为他总是去酒吧。他这种积极消费的生活方式唯一的缺点就是他左下方的白齿一阵阵的悸动，但是他牙痛了两个月却没钱看牙医。

你也许会说我们那位朋友的消费观点不正常，但是很多公司其实和他的消费态度毫无差别。

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、特定的脚本、特定的 CSS，甚至第 13 章里描述的特定 HTML 上。他们认为给 10 个不同的浏览器发送 10 个不同的表格不是在浪费金钱，其实用一个简单的 CSS 文件就能解决问题，根本不需要这么浪费。而让他们在可访问性上花上几美元，他们会说“太贵了”。

2003 年 2 月，MSN.com（如图 14.3 所示）在给 Opera 7 发送 HTML 和 CSS 时出现问题。这些条件文件能提高 Opera 上的显示效果吗？其实，他们使用的效果更差（<http://deb.opera.com/howcome/2003/2/msn/>）。更不必说，网站不是完全可访问性的，根据纯文字界面浏览器的测试（如图 14.4 所示）和 Watchfire 的 Bobby，一个在线可访问性校验系统：<http://bobby.cast.org/bobby/bobbyServlet?URL=http%3A//www.msn.com/&gl=wcag1~aaa>

微软不是仅有的在过时的 workaround 上浪费金钱的组织，在 workaround 上浪费金钱不仅不能给任何人带来好处，反而成倍地增加了网络发布向导的成本。微软也不是仅有的研究技术问题上花大手笔的花费，但确实在可访问性问题上斤斤计较的公司。

很多公司声称他们负担不起，即使是大部分基础等级的后台开发，流媒体和（经常是不必要的）复杂 JavaScript 脚本的可访问性的费用。如果那些花在浏览器检测、特定脚本、特定 CSS 和 HTML 上的开销被认为是经营开支的一部分，那么比这些少得多的提供访问性的成本也应该被认为是经营开支的一部分。



图 14.3

你能看出这个站点是否可访问的吗？(www.msn.com) 如果你是残障用户你就能够看出来，就像图 14.4 显示的那样



图 14.4

文字界面浏览器，Lynx，展示了 MSN 的一些访问性问题，包括搜索表格里没有空间可以插入文档，盲人“单击这里”的链接和毫无用处的图像对应参考。

14.4.4 错误的说法：可访问性迫使你创建原始的、低档的设计

这种说法也是不正确的。表格布局、CSS、JavaScript、服务器端技术（如 PHP）和其他现代 Web 设计元素与 WAI 优先级 1 和 508 条款适应性能完全地兼容，只需要多一些关注和判断。只要遵循有关的方针（本章的稍后讨论），你也可以使用基于插件的技术，如 Flash 和 QuickTime。所有 Happy Cog 在过去两年

里创建的网站都采用了 DOM 脚本, CSS 或混合布局, GIF 和 JPEG 图片等, 它们都很容易地通过了在线访问校验测试。

重要的是, 我们马上会发现, 通过在线访问校验测试, 不能保证你的网站真正具有可访问性或是适应性。软件能执行的测试有很多局限, 必须用人的判断来调节, 评价所有这类的测试。另外, 不能通过 WAI 优先级 1 或美国 508 条款测试表示, 网站不能适应可访问性要求。

14.4.5 错误的说法: 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

这种说法不正确。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, 这不是法律的原因, 而是因为网络用户能自由地通过下载来更新浏览器。在 Lynx 上的内容对于屏幕阅读者、PDA 和其他的用户设备必须是可访问的, 可视设计在一些这样的环境里根本不能显示, 更不用说在不同的浏览器上显示一样的效果了。以前谈到过的, 旨在使站点在所有浏览器和平台上显示相同的旧式方法是造成许多站点不可访问, 非法错误和难以维护的原因之一。

14.4.6 错误的说法: 可访问性“只是为残障人士服务的”

这种说法也不正确。诚然, 可访问性能提高(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 Palm Pilot, 网络智能电话和其他非传统浏览设备的用户——所占的市场份额正在不断增加。
- 暂时有损伤或残障的人(例如, 摔断了手腕的人)。
- 有可校正的视力问题的人, 包括年纪比较大的人(占人口的很大比率)。
- 临时访问不习惯的网站的人——例如, 通过机场或其他公共场所的公用电话亭或限定特性浏览器浏览网站的人。
- 想获得这类访问者的网站所有者。
- 想从搜索引擎获得好处的网站所有者(参见 14.2.1 节中的“盲人亿万富翁”)。

14.4.7 错误的说法: Dreamweaver MX/Watchfire 的 Bobby/其他一些可以解决所有的适应性问题的工具

不正确。这些工具能起帮助作用, 但是不能代替人类的判断, Watchfire 的

Bobby 可访问性校验系统(如图 14.5 所示)Cynthia Says™ Portal (<http://www.contentquality.com>)或者 Usable Net 的 LIFT (<http://www.usablenet.com/>)能帮助你测试 WAI 或 508 条款适应性问题。Dreamweaver MX 里出现的 LIFT 也能帮助你测试，并鼓励你制作更具可访问性的网站。但是这些测试和工具都不是万能药，也不是按钮操作的解决方法。它们只能辅助你想出切实可行的方法，识别每个设计具体的问题所在。

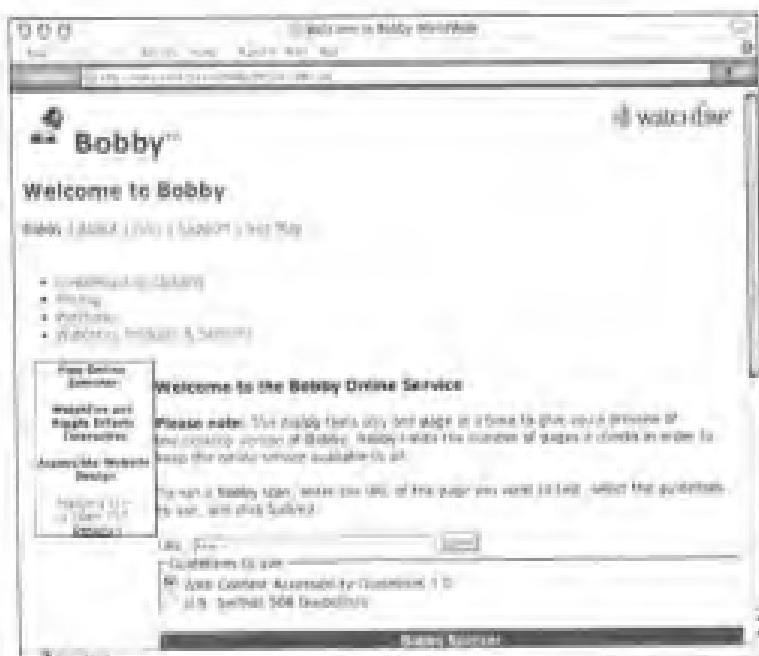


图 14.5

Watchfire 的 Bobby (<https://bobby.watchfire.com>) 是一个免费的可访问性在线校验系统。能帮助你使网页适应 WAI 或 508 标准，但就像所有的软件一样，Bobby 也有局限性，你必须用自己的判断力和常识评价它的报告。

14.4.8 错误的说法：如果是客户提出可以忽略可访问性，设计师就可以不理睬可访问性条款

这种说法是否正确还不清楚，我们说的是：还没有创建了不可访问网站的设计师，已经受到了法律的处罚的情况——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。无论客户是否告诉设计师“这样建造网站”，网络设计师某天都可能面临这样的处罚。告知客户（或老板）是我们的责任，而不是在明知是错的还去做同时责备他们。

14.5 关于可访问性的一点技巧，一个元素接一个元素

下面的内容提供了一些方法，使常用的网页元素与 WAI（或政府的）可访问性方针相适应。

14.5.1 图片

文档如果遗漏了 alt 属性, 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], 或者其他一些同样没有帮助的东西。看看第 2 章中所描述的可视性示例, 图 2.5, 缺乏 alt 属性的文档被标记为 WAI 访问错误或标记为 XHTML 校验错误。必须给 img 元素添加 alt 属性来描述每个图片的用途。

你的朋友, 无效的 alt 属性

对于一些无意义的图片, 例如间隔 GIF(倒不是还在使用的 GIF 间隔图片), 用 alt="" , 也写作 null alt 属性或 null alt 文档。对于一些无意义的图片如 alt="pixel spacer gif" 或 alt="table cell background color gradient" , 不要认为 alt 文本就是用户问题。给图片添加 null alt 属性仅仅是为了创造可视化的(无意义的, 非语义的)设计效果。

用 alt 属性给访问者传达意思

用 alt 属性给访问者而不是给你或是你的同事传达意思, 例如, 在一个也用做链接回首页的标志上, 使用 alt="Smith Company home page" , 而不使用 alt="smith_logo_lev3" 或是 alt="Smith Company logo" 。对于一个残障用户, 他对看不见的那个图片是不是一个“标志”不感兴趣, 告诉他单击这个图片就能回首页, 才更有意义。如果你觉得必要的话, 你可以书写代码如下:

```
alt="Smith Company home page [logo]"
```

不要用对生成 alt 属性的“有帮助”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 文档:

```
alt="smith_logo_32x32"
```

总之一句话, 不要让机器做人类的工作而实际上却……

不相信软件能做人类的工作

即使你的网页通过了 Bobby 的 WAI 或 508 条款可访问性测试, 也不要认为 alt 属性就正常作用了。每个图片使用 alt="mickeymouse" (或是 alt="") 的网页能够很顺利地通过这些测试, 没有什么软件能告诉你, 你的 alt 文档是不适合的。

Alt 属性提示的胡闹

当访问者的鼠标光标停在图片上方时，一些主要浏览器被误导显示 alt 属性为工具提示。虽然现在几百万的网络用户对此已经习以为常，但是出于很多原因这样显示是很可怕的。简单而言，alt 文本是一个可访问性工具，不是什么工具提示（title 属性才是工具提示的）。W3C 清楚地解释了 alt 文本只有当不显示图片的时候才为可见的：

alt 属性指定了当图片不能显示的时候就显示的替换文本……用户浏览器不支持图片的时候必须显示替换文本——这些浏览器可能不支持一些特定类型的图片，或者是浏览器被设定成不显示图片。

<http://www.w3.org/TR/RBC-html40/struct/objects.html#h-13.2>

没有浏览器应该要显示描述视觉正常访问者所能看见内容的多余 alt 文本。但是如 IE/Windows 就会显示（如图 14.6 所示），Netscape/Windows 更是会首先就显示多余 alt 文本。这不是你的问题，除非它误导你去写“创造性的” alt 文本，在为那些针对视觉障碍人士做的图片做解释失败的时候。



图 14.6

eBay 的标志的位置是显示一个可单击链接回首页的按钮，但事实上它并不是。（www.ebay.com）它的 alt 属性（“eBay logo”）是恰当的，但是把 alt 文本作为工具提示而显示像 IE/Windows 这样的浏览器上，显然是多余的。

背景图片没有 ALT

初学者经常问是不是要给 CSS（或 HTML）背景图片书写一个 alt 文本，答案很简单，不需要。实际上，即使你想这样做也不可能达到。在 CSS 里没有这样的 alt 属性（例如，没有 alt 属性可以指定给<body>标签里的背景图片）。

如果背景图片表达的是网页文档里没有表达的重要意思——例如，如果网页的文档只说了“他是一个正直的人”，CSS 背景图片是亚伯拉罕·林肯的画像——你可能想给 body 元素添加一个包含“总统亚伯拉罕·林肯”这样的文字的 title 属性。或者，如果使用了表格的话，可以添加一个 table summary 属性。

性。你也可以适当地插入一些说明性文本在<noscript>标签里，假设那些看不见图片的人是处于一个不支持 JavaScript 的环境里。

最好你放弃那些想法，直接在林肯图片的旁边写上这样的文字“他诚实吗？”

14.5.2 苹果的 QuickTime 和其他视频流媒体

要求用到插件的地方也要用到一个明确的链接。

如果你用一个图片链接到必需的插件程序，要确定图片有合适的 alt 文本。

为了提高 QuickTime（或 REAL）视频的可访问性，使用添加说明工具或如 SMIL 这样给音频提供描述文档说明的 Web 标准。见 A List Apart 的“用 SMIL 播放”(<http://www.alistapart.com/stories/smil/>) 中关于友好性设计的 SMIL 概述，可通过阅读 Joe Clark 的书来学习如何在 QuickTime 中添加说明。

给视频添加说明需要专业技术、时间和金钱，特别是专业技术。如果你的客户需要这个服务，一定要在产品开发费用和开发周期上说明这些。

WGBH Boston 网站对 QuickTime 和 Flash 可访问性做得很好(<http://main.wgbh.org/wgbh/access/>)，Joe Clark 推荐的 BMW Films 站点也一样做得很好 (<http://www.BMWFilms.com/>)。这些站点应该都能给你带来灵感。

14.5.3 Macromedia Flash 4/5

Macromedia Flash 4 和 5 提供有限的可访问性选项，如用音轨描述导航按钮的功能。如果你不能把它更新到 Flash MX，那么使用这些选项来提高 HTML 选择性，只要可能，最好使用 Flash MX。

14.5.4 Macromedia Flash MX

2002 年推出的 Macromedia Flash MX 大大加强了可访问性功能，包括屏幕阅读器兼容性，虽然大部分的辅助功能还只能在 Windows 环境里实现，例如 Flash MX 与 Microsoft Active Accessibility 相交互。

屏幕阅读器，有时也被错误地称为是“声音浏览器”或者“文档阅读器”，是能够大声朗读网页文档的浏览器。目前，两大屏幕阅读器，自由科学的 JAWS（如图 14.7 所示）和 GW 微电子(<http://www.gwmicro.com/press/flash.htm>) 的 Windows-Eyes，都可以理解 Flash MX 增强的访问性功能。

Flash MX 达到了 U.S.508 条款的一些要求，包括以下这些：

- 内容放大。

- 不使用鼠标的导航。
- 声音同步。
- 支持自定义颜色调色板。

Flash MX 的其他一些关键的可访问性功能，包括：

- 创建文本等同物的能力。
- 使活动事件制作不佳时产生的危害最小化的能力。
- 创建可访问的按钮、表格和标签能力。
- 在 XHTML 中，指定标签顺序，帮助非鼠标用户导航的能力。



图 14.7

自由科学的 JAWS，一种屏幕阅读器，与 Windows 浏览器一前一后的工具，帮助视力受损者阅读网页内容和访问 Flash MX 站点 (http://www.freedomscientific.com/fs_products/jaws/jaws.asp)。另外，自由科学的站点遵循 WAI 优先级 1 和 U.S. 508 条款，这种情况对于这样的公司并不多见。

在 Flash MX 中，标签顺序是通过 ActionScript，Macromedia 版本的 ECMAScript 来指定的。在随后的 14.7.2 节“保留标签：tabindex 属性”，详细解释怎样在 XHTML 里指定标签顺序。

如果你不学习怎样使用，并使其融入 Flash 制作过程，这些增强功能都是无效的。这些增强功能也不是很容易就能理解和执行的。Flash MX 并没有包括可访问性的所有方面。

关于制作可访问的 Flash，可参见 A List Apart 第 143 期里关于 Flash MX 的文章 (<http://www.alistapart.com/issues/143/>)。Joe Clark 和研究可访问性媒体的 CPB/WGBH National Center for Accessible Media 的设计师 Andrew Kirkpatrick 写的文章中介绍了 Flash 增强的可访问性的功能及其局限性——如果你还没有看过这些内容，可以看看本章开始向你推荐的那两本书。

补充的 Flash 适应性技巧

要求用到插件程序的地方也要用到一个明确的到必需项目的链接。

如果你用一个图片链接到必需的插件程序，要确定图片有适合的 alt 文本。

如果你用 JavaScript 来检测是否存在 Flash，最好为那些不能使用 JavaScript 的用户制定一个备份计划——就是说，目的明确的到必需项目的链接。你用 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。

要知道不管你多好多努力，还是有一些用户可能看不到你的 Flash 内容。

14.5.5 颜色

如果你用颜色来传达信息（例如可单击性），最好用其他的方法来加强它的这种功能（例如，粗体或者下划线链接）。如果你已经通过 CSS 关闭了下划线，考虑使用粗体字的链接，还要避免在非链接的文档里使用粗体字，以免患有色盲的用户分不清哪些粗体字文档是加了超级链接的，哪些是没有加链接的。如果链接了的文档和普通文档之间的区别很明显，色盲用户也可以分辨的话（如果文档为黑色，链接为白色），就不需要考虑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”这样的命令，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当心，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，他的书在很多书店都能买到。你还可以浏览以下网站 <http://www.vischeck.com/>，了解怎样向患有不同类型色盲的用户呈现你的网站。

14.5.6 CSS

在有样式表和没有样式表的两种情况下测试你的网站，以确保在每种情况网页都是可读的。不用担心关闭样式表时图形设计会发生变化，除非这些变化使网站不可用。

缺乏 CSS 时，用结构化标记传达意义

如果你使用结构整齐的 XHTML 制作，当样式表被关闭或不能用时，网页也能很好地显示（如图 14.8 和图 14.9 所示）。读者“获得”有 CSS 的结构标记或者没有 CSS 的标记，就像在第 13 章里描述的使用 IE/Windows “要么全有，要么全无”的字号访问特性时获得这些标记一样。我们不断地强调使用结构标记，避

免

is。像下面给出的那个标记，当用户可以使用 CSS 时也不起什么作用：

```
<div class="header">Headlines</div>
<div class="copy">Text</div>
<div class="copy">Text</div>
<div class="copy">Text</div>
```



图 14.8

作者的开启了 CSS 的适应浏览器上的个人网站 (www.zeldman.com)，你在第 13 章也已经见过了



图 14.9

关闭 CSS 时，侧标题和段落的关系仍然很清晰，因为站点是用简单的结构式 XHTML 制作的

不要相信只在一个浏览器上的显示效果

书写有效的 CSS，并在很多浏览器上测试。不要书写碰巧在某个浏览器上能正常作用的无效 CSS。不合适的 CSS 会使得网页不可读。不是每个网络用户都

知道怎样关闭 CSS，更别说怎样用用户的 CSS 覆盖 CSS 了，而且也不是每个浏览器都能支持用户 CSS。

设置文本大小时要注意

如果你使用的是 em 而不是 px（见第 13 章），别以为你是对的。避免使用基于像素的，不可调整字体大小的文档（见第 13 章），或使用 DOM 或者后台驱动的样式转换器，以免使用户在浏览器拒绝授权的时候，用户也能改变文档的字号大小。基于 DOM 的样式转换器将在第 15 章中详细介绍。

不要相信，再次重复，可访问性校验测试的结果

不要仅仅因为你的网页通过了 Bobby 的可访问性测试就认为你的 CSS 是“安全”的，因为字号为 7px 的，模糊的网页也能通过这些测试。

14.5.7 其他的脚本行为

要考虑当 JavaScript 关闭时，书写的代码保证链接也正常作用。在浏览器上关闭 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，但可能不能单击和完成其他用鼠标操作的功能，给这些用户提供一个可选代码，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。书写同一功能的两种代码给网页增加了一点字节。虽然多占用了一点带宽，但是吸引了更多残障用户的访问。

对于不能使用 JavaScript 的用户，用 noscript

[zeldman.com](http://www.zeldman.com) 上的每日报告对 form 元素使用 JavaScript，提供对上级网页的访问。出于简洁原因删掉一些元素，是书写简单代码的基本方法。你会注意到 onkeypress 和 onclick 都能被用做指向上一级网页的 URL：

```
<form action="foo"><input type="button" value=  
➥"Previous Reports" onclick="window.location=  
➥'http://www.zeldman.com/daily/0103c.shtml';" onkeypress=  
➥"window.location='http://www.zeldman.com/daily/0103c.shtml';"
```

```
  >/>  
  ...  
</form>
```

这对于那些使用支持 JavaScript 浏览器和不关闭 JavaScript 的用户是很好的，但是对于不使用 JavaScript 用户怎么办呢？在 noscript 标签里书写的简单的连接就可以满足他们的要求：

```
<noscript>  
<p><a href="/daily/0103c.shtml">Previous Reports</a></p>  
</noscript>
```

下面是完整的代码：

```
<form action="foo"><input type="button" class="butt" value=  
  =>"Previous Reports" onclick="window.location=  
  =>'http://www.zeldman.com/daily/0103c.shtml';" onkeypress=  
  =>"window.location='http://www.zeldman.com/daily/0103c.shtml';"  
  =>onmouseover="window.status='Previous Daily Reports.';  
  =>return true;" onmouseout="window.status='';return true;" />  
<noscript>  
<p class="vs15"><a href="/daily/0103c.shtml">  
  =>Previous Reports</a></p>  
</noscript>  
</form>
```

完全不熟悉 JavaScript 的人会感觉上面的代码有点令人害怕，特别是被印刷在书上的时候。但是有经验的人就知道都是很简单的代码。任何设计师都能写出这样的代码。虽然都是很基础的代码，但是能满足“典型”用户、行动不便者、失明者和非传统设备用户，还有那些使用支持 JavaScript 浏览器，但关闭了 JavaScript 的健全用户的要求。

避免或尽量少地使用工具生成的脚本

避免使用 Dreamweaver 或 GoLive 生成的脚本，这些脚本的浏览器和平台的支持性不是很好。有一点就是：在不知名浏览器上测试这样制作的脚本，将要关闭 JavaScript。

学习更多有关知识

脚本行为和可访问性之间的相互作用可以非常复杂，这个讨论已经超出了这章的讨论范围。想了解更多关于怎样使用 JavaScript，可以参见 Apple Internet Developer：“JavaScript 的使用”(<http://developer.apple.com/internet/javascript/>)、Webreference.com 和 scottandrew.com，在第 15 章我

们也会谈到一些。

14.5.8 表单

在本章的开始部分，我们极力推荐了两本书。《建立可访问的网站》用了整整一章的篇幅论述怎样创建可访问的在线表格。《构建可访问的网站》也用了整章来论述创建在线表格问题。由此可知，创建在线表格是一项必须要做的工作。

别担心，大多数要做的工作都比较简单直接，像把表格区域和适当的标签连接起来（例如，把一个搜索表格里的文档区域和“搜索”标签连接起来）。还有很多和这个差不多的工作，如果全都讨论的话就要超出本章的范围了。

创建了可访问表格后，放在 Lynx (<http://lynx.browser.org/>) 或 Jaws 上测试。Macintosh 用户还需要虚拟 PC (<http://www.connectix.com/products/vpc6m.html>) 或一个真实的 PC 来运行 Jaws，Linux 用户需要一个包含屏幕阅读器的 SuSE Linux 7.0 发行版本 (http://www.hicom.net/~oedipus/vicug/SuSE_b.linux.html)，或选择 Speakup (<http://www.linux-speakup.org/speakup.html>)。

14.5.9 图片热点连接地图

尽可能避免使用图片热点连接地图，一般都可以不用。一定要用的话，使用客户端图片要有 alt 文档对应。

14.5.10 表格布局

别头疼这个，根据第 8 章所学的，书写一个简单的表格摘要，使用 CSS 来避免用深层嵌套表格，避免使用 GIF 图片做表格间隔及其他垃圾。

无论你听过什么相反的意见，简单表格布局的使用不是造成访问性问题的主要因素，在 WAI 或 508 条款下也不是非法的。但是如果可以选择 CSS 来布局，那就选它。

14.5.11 数据表格的使用

识别表格标题，用适应的标记把数据单元和表格的标题单元连接起来，表格可能含有两个或更多逻辑层的行和列标题。在一个列出电影 “The Music Man” 演员的表格里，典型的表格标题应该是 “演员”，与它连接的表格单元应该包括 Robert Preston, Shirley Jones, Buddy Hackett, Hermione Gingold, 等等。

使用图像浏览器，近视的用户会看见 “演员” 及其下面姓名栏之间的连接。

但是对屏幕阅读器用户则要求书写额外的标记来连接表格标题和数据单元。

见 http://www.w3.org/WAI/wcag_curric/sam45-0.htm 上的资源了解 WAI 团体怎样阐明标题和其连接的数据单元之间的关系(如图 14.10 所示)。



图 14.10

WCAG 的网络可访问性的一个示例网页，展示了在复杂表格数据单元中结合标题的一种方式(http://www.w3.org/WAI/wcag_curric/sam45-0.htm)。

14.5.12 帧 (Frame), Applets

不要使用它们。

14.5.13 闪烁和闪光的元素

也不要用，从使用 FrontPage 模板以来，你可能就没有再使用过<blink>或<marquee>标签了，但是记住在 Flash 和 QuickTime 内容里也禁止使用闪烁和闪光元素。

14.6 所需工具

如果你用可视化编辑器制作网页的话，使用一些工具和插件程序可以更简单地实现可访问性。

• SSB Insight LE 和 GoLive

免费的 SSB Insight LE 插件程序能为 GoLive 自动识别很多（但不是全部）可访问性冲突。

<http://www.adobe.com/products/golive/ssb.html>

- **UseableNet LIFT 和 Dreamweaver**

本章前面说过, UsableNet 的 \$249 LIFT 为 Macromedia Dreamweaver 4 提供很多特性并能坚持可访问性。LIFT 自动识别很多(虽然不是全部)访问性冲突。LIFT 还有很多独立存在的版本, 最近推出的两个版本结合了 Nielsen Norman Group 建议的可访问性方针。

http://www.usablenet.com/lift_dw/lift_dw.html

- **Dreamweaver MX**

很多 LIFT 的功能都被并入 Dreamweaver MX, 包括一个内置的 508 条款校正检测器, 一个 508 条款参考向导, 还有为图片、表格和帧添加可访问性的一些工具。但记住: 没有什么工具可以解决所有问题, 或是提供替代你的判断力和经验的自动防故障装置。就像锤子和钉子, 工具只能帮助那些知道怎样使用它们的人。

- **LIFT 并入了微软的 FrontPage**

UsableNet 于 2002 年 7 月 9 日宣布把 LIFT 并入了微软的 FrontPage, 微软的 FrontPage 是广泛使用的制作工具: http://www.usablenet.com/frontend/onewebs.go?news_id=45

LIFT 不禁止 FrontPage 产生专门的、非标准标记, 但是像在 Dreamweaver 里一样, 并入 LIFT 使 FrontPage 用户能够检测访问性问题, 指导用户给图片、表格、帧添加可访问特性。

14.7 和 Bobby 一起工作

无论你用的是前面提到的产品还是你自己网站上手工书写的标记和代码, 下面你都要用到 Watchfire 的 Bobby 可访问性校验器:

<http://bobby.watchfire.com/bobby/html/en/index.jsp>

只要单击一下按钮, Bobby 就能为你检测任何网页的可访问性, 虽然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检查表来验证可访问性。不像 W3C 的标记和 CSS 校验系统, Bobby 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。你还得分析 Bobby 的输出结果, 这是其复杂之处。

14.7.1 理解检查表

“如果下列的 508 条款不能应用于你的网页，通过 Bobby 的 508 条款也可以”，Bobby 在其免费在线版（也有相应功能强大的零售版）完成对你的网页的评定后这样说。Bobby 永远不会说：“网页 100% 绝对适应 WAI 优先级 1 方针”或是 508 条款或是其他任何公布的评定规范。

Bobby 不是全能的，它是软件，软件依赖运算法则来检验常见问题。很多出现的问题是极其不能够理解的。

我们没有时间描述你检验网站的可访问性时可能遇到的每种情况，但是示例会告诉你怎样理解并应用 Bobby 生成的检查表。zeldman.com 的混合式（表格加 CSS）版本通过了软件的 508 条款检验，但是是否真正通过，是要看 Bobby 生成的检查表的分析结果的。

Bobby 的检查表包括以下项目：“考虑在表格控制、连接和对象中指定一个逻辑标签顺序。”

14.7.2 保留标签： tabindex 属性

XHTML `tabindex` 属性指定表格控制里 `tab` 键切换的导航顺序。如果你没有创建逻辑标签次序，依赖 `tab` 键切换（不用鼠标切换）的用户必须按你的 XHTML 源代码里所列出的次序从一个链接切换到另一个链接，这不是引导用户浏览你的网站的最有效方法，特别是当主体文档包含很多链接或是位于标记较前位置的冗长的导航。

如 `Skip Navigation` 和 `accesskey`（第 8 章中介绍的），`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跃到感兴趣的内容。忽略导航跳过很多链接，而 `accesskey` 提供链接到网页不同组件的命令键，`tabindex` 提供链接到网页不同部分的快捷方法——但是不像 DVD 的章节索引可以往前翻页，也可以往后翻页。

在商业性网站，创建下面要介绍的标签次序后，就会让真正的用户来测试。个人的或非营利的网站上，不需要这么做。如果不能让用户测试，可以建立一个用户说明，在此基础上创建一个切换次序，看看使用 `tabindex` 的访问者反应如何。

实现切换顺序

图 14.11 显示的是以前的混合式站点，它揭示了切换顺序的使用优先于

`tabindex` 的使用。当用户按下 Tab 键后，网站按照标记中所列的链接次序从一个链接缓慢地链接到下一个。这是默认行为。图 14.12 显示的是用 `tabindex` 修改后的次序，其中每单击一次 Tab 键都关闭一个到某处的链接。数字在屏幕上经过了加深处理以显示标签次序，在真实情况里他们是不存在的。

图 14.11

混合式的 Zeldman 站点是在使用 `tabindex` 修改切换次序出现之前创建的。用户单击 Tab 键后，网站按照标记中所列的链接次序从一个链接缓慢地链接到下一个 (www.zeldman.com)。数字不是网站上的，而是我们加上去的，为了显示默认的标签次序



图 14.12

修改标签次序后，按下 Tab 键可以直接链接到访问者想浏览的内容。数字经过了加深处理，用来显示修改后的标签次序。站点的可视布局并没有发生变化。不单击 Tab 键的人根本不会发现有些东西被修改了



在讨论所需标记之前（初学者都能写出的标记），让我们来看看我们选择的标签次序及选用它们的原因。虽然我们的选择不一定是最好的，但是它们向你阐明了怎样为网站制定一个周全的标签次序。

设想我们处在一个非鼠标用户的位置，会想以什么次序单击链接呢？我们得

出以下标签次序。

(1) 第 1 个 tab 连接到网页标题/Home 按钮，以便访问者重新登录网页，确认他们的网页位置，并返回首页。

(2) 第 2 个 tab 连接到设置默认字体大小和样式的按钮——DOM 驱动样式转换器将在下一章中介绍。这样做的目的是，允许用户调整字号或字体（或两个都调整）以提高可访问性，也可以通过调整字号、字体来改变网站的外观和风格。

(3) 第 3 个 tab 连接到一个按钮，其作用是允许用户选择字号以适应视力受损者，把默认文档字号设置为“smaller”（见第 13 章）的 Windows 用户的需要。

(4) 第 4 个 tab 连接到搜索表格输入区。通过先在访问者的链接串里指定搜索表格，省去很多不需要的链接。

(5) 第 5 个 tab 连接到搜索按钮，使用户能执行搜索。

(6) 第 6 个 tab 连接到屏幕快照区，把访问者转到页面底部的历史报告按钮（在“对于不能使用 JavaScript 的用户不用脚本”中讨论过）。历史报告按钮使访问者以与时间相反的次序登录以前的网页。

(7) 第 7 个 tab，也在屏幕快照区下方，使访问者不使用滚动条就能回到页面的顶端，对那些不能使用鼠标得用户特别有用。

除了这 7 个以外，还有一些常规的切换功能。

创建和测试

改变标签次序很简单，只要把 tabindex 值指定给任何我们想给予优先的项目就行了。例如，下面是提高访问性之前字号默认值按钮的 XHTML 的简单版本：

```
<form action="send">
    <input type="button" />
</form>
```

下面是重组后的同一按钮（惟一改变的元素用粗体突出显示）。

```
<form action="send">
    <input type="button" tabindex="2" />
</form>
```

它后面的一项标记为 tabindex="3"；再后面一项为 tabindex="4"，等等。

这种做法出现之后，我们用纯 CSS 布局和结构式 XHTML 对 zeldman.com

进行了两次重新设计，你可以浏览 <http://www.zeidman.com/daily/1002a.html>，了解我们修改后的标签次序。

大多数关于访问性的零星工作都很容易。对于典型任务，Bobby 建议为表格元素创建键盘快捷方式，并在关闭样式表后测试网页是否仍为可读、可用。如果你学了更多关于访问性的技术，就可以不用 Bobby 的这些建议。

Bobby 经常提一些无关的建议，例如，即使你的网站只用表格布局，Bobby 对表格这样反应：“如果这是一个数据表格（不仅用于布局），要识别表格的行标题和列标题。”同样，Bobby 会对任何网站说“如果你用颜色传达信息，请确定还有另外一种传达方式。”

这些意见很快就过时了，所以很多访问性专家根本不用 Bobby 或其他这类的工具，而只依靠他们自己的知识工作。但是对于非专业的用户，只要能容忍 Bobby 的，因为它还是很有帮助的。

Cynthia 说访问测试工具 (<http://www.contentquality.com>) 比 Bobby 测试的更多，而使用的时候更愉快。

在本节结束之前，再次提醒你，即使 Bobby 给出一个清单，你的网站很可能还是有可访问性问题的。忽视无关的清单上的项目，运用你的判断力和常识，多阅读可访问性书籍。

一个页面，两种设计

修改了标签次序后，站点的外观没有发生变化。但是在某种意义上，站点现在有了两个用户界面设计：一个是为用鼠标导航的传统图形浏览器用户；另一个是为那些在图形或非图形浏览器上用 tab 键切换的用户。两个设计共同存在，不要求特殊的“可访问性”网页版本，对大多数访问者看见的可视设计不做任何改变。（另一方面，如果你的颜色方案使文档对视力受损者呈现为不可读，就必须修改颜色方案，也许还要修改整个设计。）

14.8 为访问性做计划，你会受益良多

虽然现在还没有法律规定网站要提供可访问性，但是可能以后会有这样的规定。我们知道法律总是在不断修改中。现在就提高站点的可访问性可能为你节约将来重构时的昂贵花费，还可能使你免于受到反歧视的起诉。

让可访问性为你服务

提高网站的可访问性，吸引更多的新访问者，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进入，他们会成为忠诚的用户。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 和其他的搜索引擎上被搜索到。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。A List Apart, zeldman.com 和其他 Happy Cog 设计的大多数网站在搜索引擎的搜索结果上都居前几位。不是因为网站多么吸引人，而是因为网站使用的是增强访问性的结构式标记。

还有两个原因使我们必须提高网站的可访问性：

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使你超越了仅能对表面化的外观（外观和风格）做设计，而进入用户流，创意设计和常规可用性的领域。这些都是网络设计师、信息工程师、可用性专家考虑的问题，可访问性问题可以更好地建造网站和满足不同人。

提高可访问性和研究实现适应性的策略能提高你的开发技术，开阔视野。学习 WAI 特别是 508 条款（或其他法律规定可访问性标准）的方法将提高你作为专业网络设计师的价值，使你更具竞争力。这就是每个网站所有者，每个设计师或开发商要努力达到的目标。实现可访问性可以帮助你的访问者达到他们的目的，也帮助你达到自己的目的。

使用基于 DOM 的脚本语言

最初，Netscape 开发了 JavaScript。它工作得非常好。后来，Microsoft 推出了 JScript，但它和 JavaScript 不太一样，于是就产生了巨大的冲突。DHTML 中的框架更能对所有人产生威胁。随着文档对象模型（DOM）的诞生终于解决了这些问题。它首次出现是 DOM Level 1 规范(<http://www.w3.org/TR/1998/RFC-DOM-Level-1-1998-001/>)。它的确是个非常好的规范。因为 W3C 第一次能够给予 Web 设计师和开发者一个标准的方法，让他们来访问他们站点中的数据、脚本和表现层对象。

最近几年来，W3C 继续来更新 DOM 规范，并且在 Web 标准组织（WaSP）的压力下，大多数浏览器已经支持了绝大多数 DOM Level 1 规范内容，虽然他们在某些时候可能采取不同的方式去实现。（如果想看看你所喜爱的浏览器是否支持 DOM 的程度，可以访问 <http://www.w3.org/2003/02/06-dom-support.html>。）在这个章节中，我们将接触到 DOM，并且研究一些可以帮助我们完成某些有用任务的方法，比如对访问者的单击触发进行显示和隐藏的内容切换，或者提供一些个性化和辅助的选项，以及用来创建动态菜单。我们可以看到，上述的这些都不会很难实现或者很技术化。

15.1 初步接触 DOM

什么是 DOM？根据 W3C DOM 规范(<http://www.w3.org/DOM/>)，DOM 是一种与浏览器无关的，平台，语言中性的接口，能够允许“程序和脚本动态地访问和更新文档的内容、结构和样式。这个文档还可以被继续处理，并且处理的结果能够组合成最终的结果页面”。

用简单通俗的语言来说，DOM 使得你可以访问页面其他的标准组件（样式

表、标记元素和脚本)并处理它。如果你把你的 Web 页面比做一部电影, XHTML 就是个剧本作家, CSS 就是艺术导演, 脚本语言提供特效处理, 而 DOM 则就是整个电影的导演。

另外的好处是, 在这个过程中, DOM 驱动的交互不再要求服务器进行繁重的处理, 以及在 HTTP 请求中进行等待, 取代的是在客户端就能完成(就是说, 浏览者的计算机的硬盘中)。它甚至在 Internet 网络连接中断后还能继续工作。

15.1.1 使 Web 页面变得像应用程序的标准方法

虽然像标题这样的需求超出了本书的范畴, DOM 驱动的交互性的最令人兴奋的功能就是能够使你的 Web 页面模仿一些传统应用程序的动作。举个例子, 访问者可以通过单击表格的头部行就能改变表格中数据的排列顺序, 就像一个 Excel 表格或者在 Macintosh 计算机中的 Finder 程序(这个程序可以让 Macintosh 用户对他们计算机上的文件进行排序、拷贝、移动、更名、删除, 或者做其他操作)(如图 15.1、图 15.2、图 15.3 所示)。

图 15.1

DOM 能让 Web 页面的响应变得像桌面应用程序。在 Porter Glendinning 的例子中, 数据可以通过用户单击专辑标题就进行标题排序 (<http://glendinning.org/webbuilder/sortTable/>)。

Rank	Album	Artist	Price
1	Before Your Love/A Moment Like This	Clarkson, Kelly	\$4.49
8	Bounce [Digipack]	Bon Jovi	\$12.99
2	Home	Dixie Chicks	\$12.99
4	October Road	Taylor, James	\$13.49
3	Rising, The	Springsteen, Bruce	\$13.49

图 15.2

再次单击这个标题就能改变标题顺序。这个顺序的改变能够在原始的页面上马上显示出来。这个过程并不需要另外的一个“结果”页面。

Rank	Album	Artist	Price
3	Rising, The	Springsteen, Bruce	\$13.49
4	October Road	Taylor, James	\$13.49
2	Home	Dixie Chicks	\$12.99
8	Bounce [Digipack]	Bon Jovi	\$12.99
1	Before Your Love/A Moment Like This	Clarkson, Kelly	\$4.49

图 15.3

单击级别一栏就能对数据列表进行重新排序，仍然不需要重新加载一个新的页面或者进行一个 server/client 处理。查看或者要下载这个例子请访问 <http://glendinning.org/Webbuilder/>

在传统的 Web 页面上，像如此的交互能力需要一个客户-服务器的处理过程，后端的脚本和数据推送，并且当每次排序顺序的改变都要生成和加载一个新的新“结果”页面。但是如果采用了 DOM，所有的交互，不管用户是否连接到服务器，都可以正常地进行。在图 15.1 到图 15.3 所示的页面如果被下载后，就算几年过去了，这些数据仍旧能够被排序而不需要一个活动的 Web 连接。

同样，在购物车中的放入的物品能够在不通知服务器的情况下删除，直到用户想买一些其他的物品或者检索其他的产品。还有，从一种语言翻译成其他语言，只要一次鼠标单击就可以了（如图 15.4、图 15.5、图 15.6 所示）。DOM 通过处理用户行为中的数据来完成这些任务。它能够完成这些是因为在这个 Web 页面中的任何标准元素都能够被任何 DOM 兼容的用户代理程序访问（用另外一种方式说，DOM 处理标准 Web 元素，就像 ActionScript 处理 Flash 文件组件一样）。

图 15.4

在 David Eisenberg 的演示例子中，西班牙语到英语的翻译只要一次鼠标轻轻单击就可以了 (http://www.artistapart.com/book/es_comclicks.html)

虽然这些演示例子都是非常普通的，但是它们背后的意义却是非常深远的。多年来，那些希望创建前端 Web 应用程序的程序员们都必须使用 Java 或者 Flash。当然现在这些工具还是可以使用的，但是支持 DOM 的浏览器使得 Web 开发人

员拥有了创建多样化、功能强大的基于 Web 的应用程序的能力，而且完全是由本书所讨论标准技术架构而成的。

图 15.5

懂得一些西班牙语的英语读者对那些他们不认识的单词或者短语，可能希望能通过简单鼠标单击得到翻译结果。

The first source for obtaining the Linux system is the Internet itself, and it is where the latest versions and most current applications will be on many anonymous FTP servers. Another method, very frequently of interest to beginners or for those who don't want or might not be allowed to copy such a quantity of information over the net, is by means of commercial versions on CD-ROM.

Show full translation

cantidad
such a quantity

图 15.6

一些复杂的西班牙语短文可能需要全部的翻译，而不是单独的单词或者短语。所有的操作都在客户端完成，并不需要一个活动的网络连接，或者需要加载一个“结果”页面。

The primary source for obtaining the Linux system is the Internet itself, and it is where the latest versions and most current applications will be on many anonymous FTP servers. Another method, very frequently of interest to beginners or for those who don't want or might not be allowed to copy such a quantity of information over the net, is by means of commercial versions on CD-ROM.

Show full translation

Click any bold word or phrase at the left to see its meaning.

在本章中，我们将聚焦于那些内容站点或者电子商务站点所需求的一些简单的基于 DOM 的任务。但基于 DOM 的，结合 XML、XHTML、CSS 及 ECMAS 的开发将会影响 Web 的未来。

15.1.2 什么浏览器能支持它们正常运行呢

虽然有一些差异（差异可以参看后面的 15.1.4 节“深入的 DOM 细节”），下面所有的浏览器都支持 W3C 的 DOM 标准：

- Netscape 6 及更高版本
- Mozilla 0.9 及更高版本
- Chimera/Navigator 0.6 及更高版本
- IE5/Windows 及更高版本（IE6 以及更高版本）
- IE5/Macintosh 及更高版本
- Opera7（多说一些：早期的版本比如直到 Opera4 之类都不值得考虑，幸运的是大多数 Opera 的用户都是经常更新的）

- Kmeleon (丧失一些特性，有一些特性被关闭)
- Safari (基于 Kmeleon，所以它同样丧失一些特性及有一些特性被关闭)
- Konqueror (一些废话：虽然你不知道为什么，你不能改变一个文本节点的值，无论什么位置你都没办法做到)

15.1.3 不支持 DOM 的环境

是不是发现上面的列表少了一些浏览器？少了 IE4 (Macintosh 和 Windows 版本 0)。如果没有人再使用 IE4，那它缺乏 DOM 支持也就不成问题了。在第 8 章“XHTML 的示例：混合布局（第一部分）”讨论的 iCab 浏览器同样也没出现在上面的列表中。这也不会有大的关系。因为 iCab 的用户不能访问到大部分交互性脚本的网站，他们不会因为缺少访问你们站点 DOM 交互性能力而感到惊奇。

Netscape 4 同样不在上述的列表中。这个可能对一些用户造成真正的麻烦。但是，现在我们可以使用 DOM 检测技术让你给 Netscape 4 用户（或者其能对 JavaScript 进行响应而不支持 DOM 的浏览器）提供替代页面，而避免这些麻烦。

还有什么浏览器没在这个列表中呢？手持设备和 Web 智能电话还不支持 DOM，一些文本浏览器如 Lynx 也同样不支持。如果你需要对那些不支持 DOM 的用户进行赔偿，那么同样的原因你也必须对那些不支持 JavaScript 的环境用户进行赔偿。（你已经对那些非 JavaScript 用户进行赔偿了吗？）

要支持那些不支持 DOM 的设备，可以做如下操作：

- 使用`<noscript>`标记来提供一个替代的访问方式（比如，一个代替特殊按钮的文本链接）。
- 在 Lynx 上进行测试，就如我们在第 14 章中关于可访问性和 JavaScript 的讨论一样。
- 用`return false`返回链接比`javascript: "links"`方式，比如``要好，我们可以举个例子，下面来自于 zeldman.com 的代码用来激活一个样式转换器。它以一个真实链接（/about/switch/）开始，而用`return false`作为`onclick`触发的结束：

```
<a href="/about/switch/"  
  onclick="setActiveStyleSheet('default'); return false;"  
  onkeypress="setActiveStyleSheet('default'); return false;"  
  accesskey="w">  
  
</a>
```

理解 DOM 语法的浏览器将会忽略这个链接而调用函数。不支持 DOM 和不支持 JavaScript 的用户终端会跟随链接到达页面（如图 15.7 所示），这个页面解释这个切换，并告诉用户页面并没有出错。（在这个代码中引入的其他元素包括我们的老朋友：accesskey 和 onkeypress，它们已经分别在第 8 章和 14 章中分别介绍，用来使页面变得更加容易访问）。

图 15.7

同一个 JavaScript 事件开始链接，并用 return false 结束它，可以使你的站点支撑那些 DOM 的浏览器。并且能迎合那些不支持 DOM 的浏览器用户（<http://www.zeldman.com/about/style/>）



更大的问题不在于列表中缺少的那些浏览器，而在于列表中已有的那些浏览器。比如，Opera 4、5 和 6，认为它们能支持 DOM 而实际上它们不支持，它们将自己标志为 IE 浏览器的做法，也使得通过浏览器检测解决问题变得不可能。首先，我们不是狂热的浏览器检测爱好者，当我们认为有必要对浏览器发送异常条件的时候，我们更喜欢检测浏览器是否有支持能力，而不是靠用户代理标志字符串来判断。唉，Opera 5/6 总是盲目自信以为它能够“获得”这个 DOM，并且能够防止错误发生。我们会被从现在开始的一些段落（15.2 节“DOM 请不要让我伤心”）中说明的事实所沮丧。然而，过时的 Opera 版本还不是我们的唯一问题。

15.1.4 深入的 DOM 细节

虽然当前所有流行的浏览器都支持 DOM，但是它们并不是完全通过同样的方式来支持的，这对于那些想使用本章中这些技术的 Web 设计师们来说只是个小问题，而对于那些想使用 DOM 来开发复杂的基于 Web 的应用程序的人来说则是个头疼的大问题。对这些开发者幸运的是，Peter-Paul Koch 的 DOM 兼容性概述了这些 DOM 实现上的差别 (<http://www.xs4all.nl/~ppk/js/index.html?version5.html>)。如果因为频繁更新，这个页面被移到别的地方了，请直接访问 <http://www.xs4all.nl/~ppk/>。

不知疲倦的 Koch 先生还另外维护着免费的 W3C DOM 邮件列表，它的用户包含世界各地的成熟的脚本开发者 (<http://www.xs4all.nl/~ppk/js/list.html>)。用 W3C 做这个邮件列表的名字，并不代表着它是 W3C 的所有权或得到了 W3C 认可。它只是用来区别 W3C DOM 和其他 DOM 是不一样的，例如：用来清楚地表示，它不是一个讨论微软的 IE4.0/Windows DOM 技术的邮件列表。

除了上段落提及的资源外，Koch 先生还写了一份 DOM 的介绍 (<http://www.xs4all.nl/~ppk/js/dom1.html>)。阅读过这个介绍，你就不会再对比如“节点”或者“树结构”之类的词语或者短语感到迷惑。你可能还不知道它们是什么意思，但我们会逐渐地熟悉，正如把美国中西部地区命名为纽约一样。

David Eisenberg 的 DOM 系列则提供了另外一个舒适的通向 DOM 的道路 (<http://www.alistapart.com/stories/dom/>)。在浅显介绍文章和教程后，Eisenberg 介绍了怎么样显示和隐藏组件 (<http://www.alistapart.com/stories/dom2/>)，创建动态菜单 (<http://www.alistapart.com/stories/domtricks2/>)，以及更新页面内容 (<http://www.alistapart.com/stories/domtricks3/>)，比如从图 15.4 到图 15.6。

15.2 DOM 请不要让我伤心

基于 DOM 的脚本对于那些完全缺乏 W3C DOM 支持的浏览器来说并不是太好。Netscape 4 是我们最主要的问题。解决方法是转向支持 DOM 的浏览器，或者是给浏览器提供一个替代的内容或者 Web 页面。

有时，我们不得不这样做，是很令人叹息的。标准的目标就是对所有的用户

代理程序（浏览器及其他）能获得同样的内容。在 HTML 和 XHTML 中，我们已经能够正确地做到了。而 CSS 相关的内容也在第 9 章“CSS 入门”中介绍了，我们看到了“双样式表”的方法可以让我们给所有的浏览器发送同一个文件，而使用@import 方法可以让那些老浏览器访问得到那些它们不能处理的样式表。在第 12 章和第 13 章我们也接触到，盒模型黑客程序的研究可以使得 IE5/Windows 早期版本的 Opera 提供同样的样式，而不会产生在诸如 IE5/Windows 得到字号大小的错误。

在前面所有的这些例子中，我们已经解决了对所有的访问者提供同样的文件的问题，虽然有时候采用了对兼容问题的避过方式，然而对于 DOM 支持我们却不能采用同样的方法来进行。虽然 IE5/Windows 对 Box 模型处理不当，但它仍然是个支持 CSS 的浏览器并且能处理一个 CSS 文件。而对于 Netscape 4 来说，这是个完全没办法进行 DOM 兼容的浏览器，W3C 发布 DOM 规范正好是 Netscape 4 发布的时候。除非你是个预言家，你不可能支持还没有出现的规范。

我们不能期望一个非 DOM 兼容的浏览器能够使用基于 DOM 的脚本，就像我们不能期望小鸡会使用计算器一样。我们能做的就是，对 DOM 的知识进行测试，并且对那些不支持 DOM 的浏览器输出一个替代的页面，或者更好地是通过服务器端包含 (SSI)，活动服务器页面 (ASP)，或者其他中间件解决方案输出替代的内容。

DOM 如何工作

Web 标准组织的 Dori Smith 先生 (www.dori.com) 在 2001 年的时候就发明了 DOM Sniff，作为浏览器向标准升级大战的一个手段（见第 4 章“XML 征服世界[和其他 Web 标准成功案例]”）。它也可以用来做其他的事情，很多情况下可以代替复杂的浏览器检测脚本。

DOM sniff 工作非常简单。当你创建完一个合法、语法正常的页面，在你的文件中的<head>或者其他调用一个链接的 JavaScript(.js)文件的地方插入如下脚本：

```
if (!document.getElementById) {
    window.location =
        "http://www.somesite.com/somepage/"
}
```

somesite.com 代表你的站点，/somepage/ 代表那些比如 Netscape（或者其他非 DOM 支持的浏览器）能处理的替换内容。这个替换页面可能用 JavaScript 来模拟正常状态的“标准”页面，或者它包含更有用的，与脚本无关，而且能让

所有的浏览器都能处理的内容。

这个替代页面甚至可以什么都不做，只是建议用户去下载 Netscape 7 或者类似的兼容的浏览器。“现在升级”或者类似方式是过时的，并不能满足那些因为愚蠢的行业管理而不能升级的 Netscape 4 的用户。这只是个反对 Web 标准的组织在 2001 年发起的浏览器升级战役的合法借口。甚至在今天，虽然在雇员中使用古老的、不支持标准的浏览器的组织数量在减少，但是仍然还有这样的现象。

然而，如果一个基于 DOM 方式的应用程序对你的站点是非常重要的话，你除了建议访问者使用一个 DOM 兼容的浏览器之外没有别的选择。在这样的站点中，DOM 检测程序防止用户访问到他们浏览器不能处理的页面，而是给他们一个选项返回并且下载新的浏览器后享受到这些内容。

简单化动态站点：DOM 或者非 DOM

如果你已经创建了动态的或者根据条件变化的页面，并且你的页面组件会被用户代理所改变，你可以使用 DOM 检测程序来提供适当的内容给那些非 DOM 支持的浏览器，并且可以简化你的内容管理。原来你需要对 Windows 下的 Internet Explorer 5，Macintosh 下的 Internet Explorer 5，以及 Internet Explorer 6、Netscape 6 等浏览器都要提供对应形式的内容，现在你只需要简单地提供两种内容：DOM 或者非 DOM。

代码变量

一个 js 脚本文件，代码类似前面提到的。当你在单独的页面中插入这个脚本时，自然地，你可能会在<head>和</head>中嵌入它。在 XHTML 1 过渡文档中，脚本可能像如下：

```
<script type="text/javascript" language="javascript">
<!-- //
if (!document.getElementById) {
    window.location =
        " http://www.somesite.com/somepage/"
}
// -->
</script>
```

在 XHTML 语法严格的文档中，你可能会使用一个 js 文件，或者如下面所示。

```
<script type="text/javascript">
// <![CDATA[
if (!document.getElementById) {
    window.location =
        " http://www.somesite.com/somepage/"
```

```
}
```

```
// ]]>
```

```
</script>
```

它是怎么工作的

这个 DOM 检测程序是一个二进制的触发器，而且并不会比一个转换器更复杂。因为 `getElementById` 是一个 DOM 方法，支持的浏览器会正确地跳过它。相反，那些不能识别 `getElementById` 的浏览器与 DOM 都是不兼容的，并且能够重定向，因此，可以显示它们能够理解的页面。

JavaScript 的重定向会使回退按钮失效，这个对用户来说不是很友好。但你不需要在 DOM 检测成功后用 `window.location` 操作，你可以在 DOM 检测中使用你的用户最感兴趣的任何命令。

为什么要使用 DOM

你不再需要创建、测试并且持续升级和维护一个长的、复杂的浏览器检测脚本。我们不再需要测试用户浏览器代理标记字符串。我们只需要用脚本来测试它是否符合 W3C 规范，非常的整洁，非常的简单，并且非常的有效率。

什么时候会失效：对老 Opera 用户不是那么重要

像在本章前面建议的一样，这个 DOM 检测程序会对 Opera 的老版本失效，它总是认为自己能处理 DOM。它不是像正常支持 DOM 的浏览器一样获得这个链接路径，这些老版本的 Opera 会通过 `getElementById`，并且会由于一个它们不能正常处理的页面而失效。

怎么来解决这个问题呢？我们不能再求助于曾经流行的检测浏览器的方法，因为 Opera 默认状态下是把自己标记为 Internet Explorer。如果你再使用 DOM 检测程序来引导 Netscape 4 用户到另外的替代页面，我们需要在我们“标准”页面中做一件事情：需要包含一个说明链接到那个替代页面来告诉那些 Opera 用户们访问那个页面。这样的解决方法就像 Mack truck 一样完美，虽然这个比较对 Mack trucks 有些不公平。但它会中断所有访问者的思维，用难用的浏览器技巧来中断用户体验并且使你的站点变得不专业。不仅仅如此，一个“Opera 用户们请单击这里”的提示可能会错误地引导 Opera 7 用户离开那个其实它能处理的基于 DOM 的页面。

那么我们要怎么做呢？以我们的观点来说，我们不去处理他们。如果我们用 DOM 驱动的站点在 Opera 6 中出错了，我们相信这些用户会意识到是应该升级他们浏览器到 Opera 7 或者更高版本的时候了。现在，很令人悲哀！让我们把这

个孩子放到车上，看看她是怎么把车开上路的（我担心，我这个驾驶的比喻可能和运动的比喻一样糟糕）。

15.3 显示和隐藏

有很多情况下，你有可能希望在你页面加载的时候隐藏一些内容，而在用户的一些操作时候显示出，如图 15.8、图 15.9、图 15.10、图 15.11 所示。DOM 可以非常容易地进行内容的显示和隐藏处理。

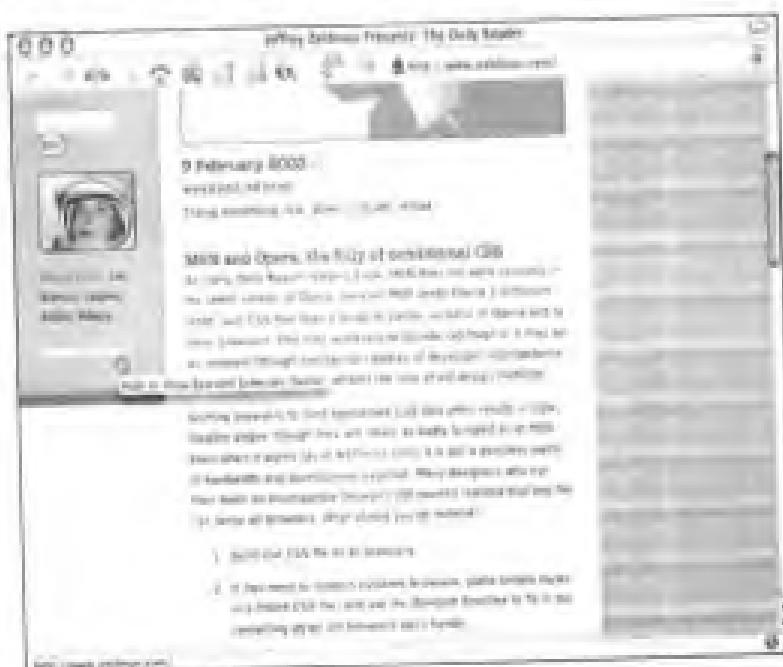


图 15.8

基于 DOM 的内容显示和隐藏是非常基本的。这个页面的选项条包含着一个刚开始是隐藏起来的链接列表 (www.zeldman.com)

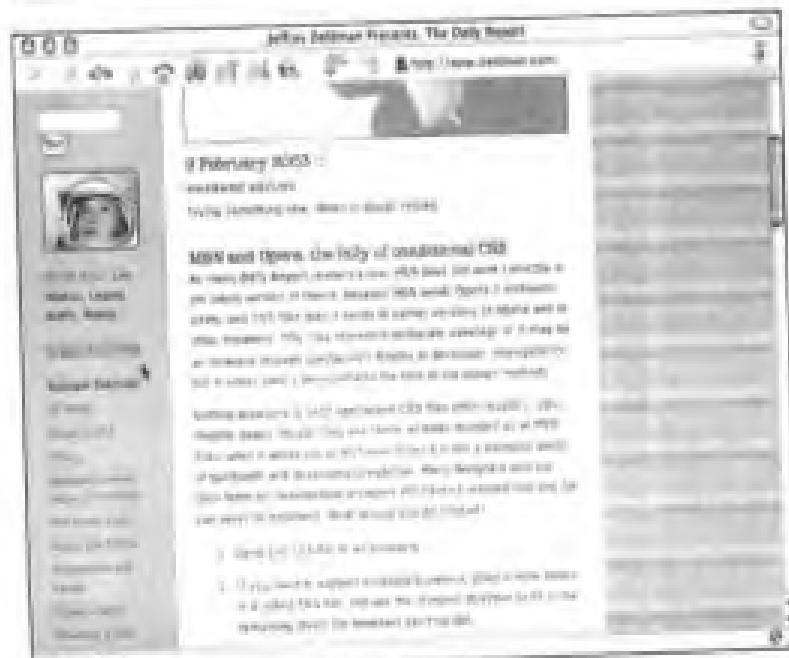


图 15.9

当“Toggle External”被单击时，列表就显示出来了

图 15.10

基于 DOM 的显示和隐藏处理还可以轻松地和其他效果处理结合 (<http://www.happycog.com/projects/>)。



图 15.11

在这里，显示和隐藏处理还附带了滚动效果。



在 zeldman.com 中，在主内容区域左边的一个选项条包含着很多到第三方站点的链接。这些链接可能在初始的时候被访问者忽略，因此在页面加载的时候，它们就会被隐藏起来，如图 15.8 所示。简单单击链接就能显示它们，如图 15.9 所示，就如访问者所期待的。再次单击就会再次隐藏它们。

这个技术只需要很少的组件。在站点的 JavaScript 文件 (<http://www.zeldman.com/zeldman.js>)

zeldman.com/jnu.js) 中，一个简单的函数使用到了我们的老朋友：`getElementById`，从而实现了切换界面显示的功能：

```
// toggle visibility
function toggle( targetId ){
    if (!document.getElementById){
        target = document.getElementById( targetId );
        if (target.style.display == "none"){
            target.style.display = "";
        } else {
            target.style.display = "none";
        }
    }
}
```

这个文件 (<http://www.zeldman.com/includes/outside2.html>) 包含着第三方站点的链接以及上述代码来切换它们的显示状态。

刚开始的代码段落是用来初始化切换功能函数的，并且用属性变量 (`outside2`) 来控制，同时提供了一个基本的提示信息（切换程序的扩充功能）以链接的 `title` 属性来提供给用户：

```
<p><a href="/" onclick="toggle('outside2');return false;">
➥title="Hide or show Relevant Externals (below).">Toggle
➥Externals</a>.</p>
```

接下来的定义列表的代码段落，里面我们看到了切换函数需要用到的 `id` 属性值 (`outside2`)，同时使用了嵌入的 CSS 规则（“`display:none;`”）来隐藏这些链接列表，直到显示状态被改变为止：

```
<dl id="outside2" style="display:none;">
<dt>Relevant Externals:</dt>
<dd><a href="http://www.somesite.com/">
➥title="Description.">Site name</a></dd>
etc.
```

在上面我们使用了嵌入的 CSS 规则（“`display: none;`”）来对链接列表进行隐藏，不过我们也可以不用这么做。在站点的外部样式表文件中给 `id` 值为 “`outside2`” 添加一个样式表规则就能使这些链接在默认状态下隐藏起来。这样的样式表规则可以是像这样子的：

```
#outside2 {
    display: none;
}
```

如果这个样式表规则是外部样式表文件的一部分，则对于列表定义的标记，只要简单写成如下样子：

```
<dl id="outside2">
<dt>Relevant Externals:</dt>
<dd><a href="http://www.somesite.com/" title="Description.">Site name</a></dd>
etc.
```

如果你需要一个元素是不可见的，但是还需要在你的布局中占据空间，可以使用 `visibility:hidden` 来代替 `display:none`。

把显示/隐藏和其他技术结合起来

我们刚刚讲解的例子是关于内容的显示和隐藏的切换，这个技术是 DOM 非常基本的功能。只要你想使用，这个技术都可以应用。在 Happy Cog 页面中（如图 15.10 所示），访问者会被引导到一个有 9 个整洁图标的页面中，每个图标代表着一个项目。当图标被单击的时候，如图 15.11 所示，关于这个计划的文本信息就显示出来，同时还附带着把图标滚动条上“加亮”。

我们在 Happy Cog 的主 JavaScript 文件 (`http://www.happycog.com/j/h.js`) 中使用了为 zeldman.com 设计的“显示/隐藏”功能函数。在 `http://www.happycog.com` 中的标记技巧是比较花哨的，但它的技术概念跟我们前面讨论的一样简单。比如，在顶行的图标中，我们通过在一个 `id` 为 `chcopy` 的元素中使用这个函数来切换文本的显示状态。下面是没有附带切换 JavaScript 功能函数的代码：

```
<a href="/projects/" onclick="toggle('chcopy');return false;">
</a>
```

下面则是另外一个，不过这次加上了附带的加粗的 JavaScript 代码，显示了用来切换名为 `ch` 的图片的代码：

```
<a href="/projects/" onclick="toggle('chcopy');return false;" onmouseover="changeImages('ch', '/i/p/cho.gif'); return
  true;" onmouseout="changeImages('ch', '/i/p/ch.gif'); return
  true;">
</a>
```

接下来我们更深入一些，我们继续加上隐藏内容的功能，代码现在看起来您

应该很容易理解了：

```
<div id="chcopy" style="display:none">
<h2>Title of this section.</h2>
<p>The first paragraph goes here.</p>
<p>More text goes here, followed by an opportunity for the
reader to invoke the toggle function once again, thereby
hiding the content he has just read: [<a href="/projects/"
onclick="toggle('chcopy');return false;">Close text</a>].</p>
</div>
```

你只会局限于滚动条和显示/隐藏功能的组合吗？呵呵，当然你肯定不会满足！你可以把显示/隐藏功能和滚动、弹出、滑动菜单或者吓人的人猩猩（如果你认为你网站的访问者喜欢）结合起来。如果你确实这样做，那我们真想面见你们的访问者。

15.4 动态菜单（下拉和展开）

DHTML 菜单有四个臭名昭著的特点：

- 开发者会回忆起夸大的市场承诺，低劣的性能和 1998 年“DHTML”的严重的不兼容性问题。
- 委员会除了成员的利益而不能在站点架构上多花功夫，从而使得不能给我们一个多层次的菜单，让我们能够访问到我们要查找的内容。
- 如果在我们的浏览器中“DHTML”菜单出错了，这会使我们不能访问整个站点或者其他内容。
- 可视化编辑器生成的 DHTML 菜单为了解决经常遇到的浏览器无关、跨平台的问题，导致了臃肿的、非结构化的、浏览器相关的代码，从而使页面代码加倍或者更臃肿（这就意味着增加了页面下载的时间）。

从目光短浅的私有方法，不以用户为导向的架构，效率低下的测试到代码的臃肿，DHTML 都是过时或者令人感到麻烦的技术。就像 Cousin Elmer's GIF 动画技术，DHTML 看起来就像 Web 上毫无意思的烦恼技术，就和在整个船舱中盘旋着的晕船的呕吐气味一样。

而 DOM 技术通过使用简单的、结构化的标记和有效的 CSS 使这些问题终结了。这里有个我的收藏文章：David Lindquist 的《使用列表来创建 DHTML 菜单》(<http://www.gazingus.org/dhtml/?id=109>)，用无序的列表创建了一个紧凑的 DHTML 下拉菜单（如图 15.12 所示）和可展开菜单（如图 15.13、

图 15.14 所示)。这个例子可以完全通过可下载的 CSS 和 JavaScript 文件获得，你可以通过定制而应用在你的站点中。

图 15.12

它看起来和其他的下拉菜单没什么两样，但是它并没有使用一些私有的、臃肿的、非结构化的代码标记而是使用了适当的列表标记、CSS 及 JavaScript (http://www.gazingus.org/html/menu_spdown.html)。我们可以下载这些源文件为我们所用。

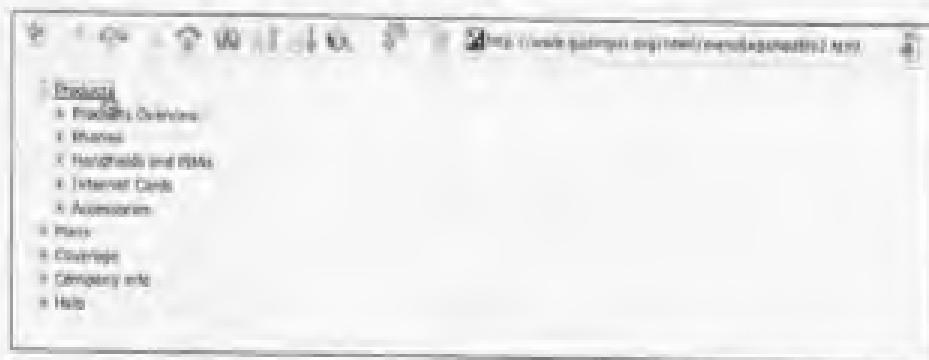
图 15.13

这个菜单树子能根据单击进行展开和收起 (http://www.gazingus.org/html/menu_spdable2.html)。



图 15.14

当用户单击的时候，这个菜单就会展开。再一次，结构化的标记、合适的 CSS 及一些基本的 JavaScript 脚本一起组合起来，通过 DOM 来提供了超越所有部分技术总和的效果。同样，你也可以把这些源文件应用下载到你的站点中。



15.5 样式切换器：增加可访问性，提供更多选择

在第 13 章中，我们讨论到，想不疏远任何所有你的潜在客户并传递 Web 类型是不可能的，并且遗憾的是，在 Web 作为媒体发展的 13 年中，像素仍然是最可靠的尺寸类型，而这个正是对于那些 IE/Windows 用户最大的麻烦。什么能够让你可以为你的用户提供可以选择的尺寸类型方法？什么能够让你在阅读的时候能够改变布局？

通过 CSS，你就能做到。CSS 能允许你不仅用一个默认(固定不变)的样式文件，还可以用另外可替换的 CSS 文件来管理一个页面，在需要加强可访问性

的重要性前提下，这个可更换的样式文件可以提供更大的字体或者高对比度的颜色配置（如图 15.15、图 15.16 所示）。或者它们都可以用来完完全全地改变一个站点的外观，这种需求有人就称为“用户个性化定制”。



图 15.15

纽约公共图书馆的在线的图片收藏站点是图书馆多个图像资源站点中的一个 (<http://digital.nypl.org/mmpeo/>)。它是用了一个样式表切换器来避免那些视力不好的用户访问上可能存在的一个问题

W3C 推荐浏览器能够提供一个方法可以让用户选择这些样式，一些基于 Gecko 的浏览器比如 Mozilla 和 Netscape 7 就能完成。但在本书编写的时候，还没有其他浏览器能够提供这个功能。从这个可切换样式表的功能能够使我们的 Web 页面获得更好的创造性和可访问性直至将来。但在 2001 年，Paul Sowden 通过利用像其他页面组件一样可替换的样式表能被 DOM 所访问的技术事实，解决了这个问题。

在 A List Apart (ALA) 的《可轮换的样式：基于替代样式表工作》一文 (<http://www.alistapart.com/stories/alternate/>) 中，Sowden 写下了一个 JavaScript 文件 (<http://www.alistapart.com/stories/alternate/styleswitcher.js>)，它使站点访问者单击一个链接、表单元素，或者任何其他可以交互的部件的时候，加载另外一个替换样式。在解释了在你们站点上怎么使用这个脚本后，Sowden 开放了这些脚本的代码。成千上万的设计师和开发人员已经在商业站点、公共社区及私有站点用来解决可访问性问题（如图 15.15、图 15.16 所示）。开发创造性效果，或者两者的集合（如图 15.17、

图 15.18、图 15.19 所示)。好啦, 阅读这个文章, 下载这个代码, 开始吧!

图 15.16

当用户选择扩大, 不仅仅字号会变大, 而且对比度也要高了。同时一个背景层(可能会对那些视力不好的用户产生混乱的感觉)也隐藏起来。

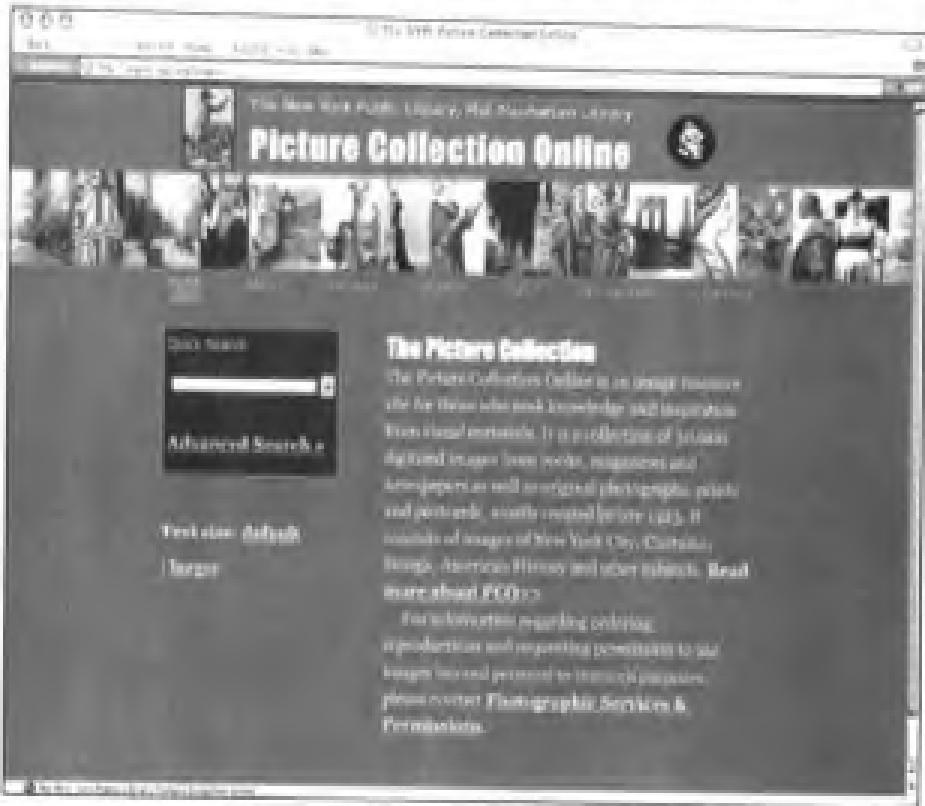


图 15.17

Eric Meyer 的关于样式表教学的个人站点中使用了一个 ALA 的样式切换器和其他效果的叠加功能 (www.meyerweb.com)。在这里我们可以看到站点使用了它的“natural”样式表(请注意背景图片和在顶部和右边的效果)





图 15.18

这里又是一个，使用了一个基本的样式表。去除了些元素——没有背景图片，没有顶部右边的特效（导航条已经从右边转移到了左边）。



图 15.19

这里用户又切换回了“natural”样式表，不过在“高级”模式中把字号从 11px 增大到 16px。

需要更多的 DOM 知识

本章仅仅提供了一个对 DOM 的接触：我们了解到 DOM 能做什么，而且在商业性、个人性和公开的社区站点中，它是怎么应用来处理一些简单的或者复杂

的工作。你可以使用 DOM 来创建变化多姿的 Web 应用程序，创造性的效果，增强的可访问性等。可是当前，我们还不知道有容易理解的书籍或者网站涉及这个主题。一定会有一本好书能向设计师们展示如何用 DOM 来完成那些令人惊异的工作的，我们也相信 New Riders 能把它带给你，那时候你就可以仰天长笑了。

CSS 重新设计

本章，我们将把所学的关于标记、CSS、可访问性和 DOM 运用到一种把表现和结构分离的网站重构上。我们会定义该 CSS，并且让它能够符合服务我们站点用户和使我们获利的目的。一旦我们对网站的外观和风格都满意了，就创建一个使用 CSS 和 DOM 的由用户选择的可替换的布局。

虽然本章所描述的可视化设计并没有什么特别之处，但是它的构建方法非常重要，因为我们将使用这个技术创建几乎所有的网站。前面几个章节讨论了标准问题，也告诉你怎样创建结合了经典和现代方法的混合式网站。本章将进一步讨论未来 Web 的典型设计方法。

这一章将探索用于创建 CSS 布局的技术——从制作网页的基础方法到更复杂的技术，例如用结构列表创建菜单按钮，不用 JavaScript，甚至不用 XHTML，`img` 元素就能生成类似 JavaScript 的图片滚动效果，另外还有一些特殊的技术。我们将讨论 Web 设计思考的方式，如基于格子的(`grid-based`)和基于规则的设计。

16.1 设定目标

在网站重构中，按照本书的特别约定，我们将重新构造 `zeldman.com`（作者的个人站点）的外观、结构及基本功能。无论你是想吸引很多阅读者还是只给自己或家庭成员欣赏，或者兼而有之，个人网站都是无价之宝，特别是当你用 CSS 和其他标准——这些并不想真正用于用户或机构内部发展——来做实验的时候。

个人网站可以帮助你认识到你作为一个设计师或实践者并不孤单。通过建造这样的网站，你会加入一群特别想互相帮助的面向标准的设计师和开发者。很多时候，我们在 `zeldman.com` 上说出遇到的 CSS，标记或 JavaScript 的问题，不久就有读者发来邮件提出解决的方法，同样，我们如果看到其他人遇到了问题，我

们也会建议解决的方法。

这种情况并不少见，任何把网络设计或开发问题公布在个人网站上的人都遇到过。去尝试它吧，你会喜欢它的。

16.1.1 突出个性

1995 年 5 月建立以来，zeldman.com 提供了教学软件、新闻和对于网络设计的见解。突出个性是要标新立异，其设计宗旨是：整洁、有最低限度、风格独特。我们这个重构必须保留这些特性。

16.1.2 最主要的 10 个目标

重构的额外要求并不特别，和你和我的网站都可能相关联，下面列出最重要的 10 个目标：

(1) 站点必须在非图形环境和在最新最好的浏览器，如 Netscape, Microsoft, Opera 上都是可用的。其内容和基本功能在任何浏览器和设备上都应该是有效的，其布局在任何理解 CSS 的浏览器上都能正常工作。

(2) 标记必须遵守有效的 XHTML1.0 过渡规则，还要避免使用表现元素，我们把结构和表现分开，就是说在使用 p 或 h1 的时候不会还使用 spans 和 divs，也意味着不会有间隔 GIF 图片，除了用于呈现表格式数据的表格以外不会有表格，不会有过时的或非法的属性，如 bgcolor 或 marginheight，不会错误使用<blockquote>来实现格式化，当结构化元素能创建想要的视觉效果也能表达意义时不会再使用
标签（如果你不太明白，可以再看看第 7 章）。

(3) CSS 必须有效，简洁，并尽量合理地排列。（见第 9 章和第 10 章。）

(4) 为了使我们的站点在任何可想到的浏览环境里，内容和功能都有效，必须努力实现完全的可访问性。很多想实现可访问性的人总是因失败而放弃。为了避免这样的情况发生，必须依照已接受的可访问性标准测试我们的网站。我们选了 U.S.508 条款（见第 14 章），任何标准都是可以用的，关键在于选择了一套方针就要遵守。

(5) 站点的外观和风格必须是突出的，但不能因此产生臃肿的代码标记，过于复杂的脚本和多余的图片，会浪费访问者和服务器的带宽。站点不能浪费资源但要保持风格（突出特性并不意味着过度的装饰）。它应该是对以前站点的发展，而不是完全地脱离了以前的站点，面目全非。

(6) 站点应该提供可视化的互动（其中一些应该是很有趣的），使站点富有生气。

(7) 只要条件允许，为使用文本浏览器和其他非传统设备的用户提供动态或相当于动态的元素（这也从另外一个方面证明了使用互动不会降低可访问性）。

(8) 站点应该为用户提供定制选项，同时保持自己的突出特性。

(9) 文档应该有趣、易读，应该使其成为站点的主要部分，因为站点是阅读站点，不是购物/单击/按钮操作的网站。正确地处理文档对于任何站点都是很重要的，特别是对那些像日报一样每天要被阅读的网站更是如此。

(10) 导航应该清楚、直观、明显。导航应该有一些可视化的装饰，但是必须也能在非视觉环境里工作。通过线性方式（如通过屏幕阅读器）访问网络的访问者应该能够直接跳过导航。如果使用结构化元素——比如，无序列表的组件——实现导航功能，那更好。

还有很多其他的目标。

任何 Web 设计和大多数站点设计项目都应该符合上述那些要求，它们应该都是很平常很普通的，不需要特别说明就应该达到的要求，就像不用特别说明，商店里的瓜果应该无虫害，车辆应该能制动，衣服应该能防寒保暖一样。

16.1.3 制作不同版本的方法

在本章的结尾，我们将创建一个默认的 CSS 布局（如图 16.1 所示）和它的一个可选择的版本（如图 16.2 所示）。访问者可以单击用户界面窗口小插件来选择。设计可选择版本就是把初始 CSS 文件用另一个文件名保存，并编辑它的值。改变了的颜色、字体和一个不滚动的图片放置在屏幕的底部，这些都只需要做些简单的工作。网站可以有多少个可选择的版本呢？你喜欢有多少都可以。



图 16.1

这里我们看到的是在其默认（白色）外观和风格下的站点完整的样貌（www.zeldman.com）

图 16.2

这是同一网站的另一个可选择版本的外观和风格。褐色在白底黑字的书里看起来显得很拘谨，但是在网页中效果好。



两种设计方法看起来都很简单，它们的确也简单，但是需要一整章的篇幅来阐明两种样式表中的每个定义和原理。我们主要是生动、有重点地阐述重要的内容，而不是把每个 CSS、XHTML 和可访问性的细微差别都列出来加以描述，也没有必要这样做，因为从第 7 章到第 14 章已经涉及了用于纯 CSS 重新设计的一些 CSS、XHTML 和可访问性辅助技术。

我们网站的经历

我们在本章讨论的修改后的外观和风格从以前的工作基础上修改得来的，并且重用了原来的 CSS 设计的一些组件。原来的 CSS 设计也是经过了很长时间的发展得来的，只是偶尔根据用户的反馈意见稍做修改。

设计商业性网站时，我们学习了突出特性，设想了用户的导航场景以及结构目标，并与各类专家一起制定网站制作流程、技术和商业的要求。然后我们花了几个月用 Photoshop、Illustrator 或 Freehand 设计和润色。

表现层后面就是修订工作。可是完成日期一拖再拖，范围不是扩大了就是缩小了，原本精神饱满的 Anglo-Saxonisms 也面目全非。接下来就是写代码和在演示服务器测试，不断地重复写代码、测试的过程，直到最明显的错误都得到修改。几个月以后，这个项目终于完成了。

从部分进展到整体

我们前面讨论过典型的用户驱动和品牌驱动的设计方法，但是对于这个项目，我们没有使用它们。在没有整体计划和比较文件的指导下，我们开始尝试制

作独立的网站组件，针对的是组件，而不是对整体进行设计。（先设计门把手，它会把你领进设计之门，最后，你还搞不明白到底是怎么做的，就已经设计出了一幢房子。我们从界面专家 Jeffrey Veen 那里获得组件设计的概念，他一直主张使用组件设计。）

我们决定做一个包含三个主要区域的布局。有这个前提，我们就不用去做什么总体计划，我们把重点放在对细节的设计上而不是整体设计，导航的窗口应该又什么样的外观呢？段落与副标题的位置怎样？如果我们需要连接到第三方网站，登录时是不是能看见这些连接呢？访问者可不可以选择关闭或打开这些连接呢？

我们用代码做大部分的设计，而不用 Photoshop。我们公开书写代码，不时更新站点，这些变化读者都可以看见。有些人认为我们脑子不正常，有些人很喜欢看到这些，有些对其中的一些地方提出批评，所有这些意见都有助我们想出新的方法。这样比本章讨论的计划和执行，我们将学到更多的技术。在某种意义上说，所有的设计工作都是这样进行的，但是大多数探索的过程是用户和客户都不了解的，他们只看到成果，没看到制作的过程。

访问 www.zeldman.com，你将看到最新、最正确的标记、CSS 和代码。你可以在这里看到样式表：<http://www.zeldman.com/c/wh.css> 和 <http://www.zeldman.com/c/or.css>。如同任何网站一样，zeldman.com 总在不断的修改中，可能你读到这本书时，某些设计方面已经被修改了。

16.2 建立基本参数

我们已经决定在模板中包含三个主要区域（如图 16.3 所示）：

- 位于顶端的商标条，可用做 Home 按钮，也包含主要的导航，我们把这个区称为“新菜单”。
- 左边的工具条包含二级导航、搜索、用户界面窗口小部件、广告和链接。“二级导航”可以用做这个区域的标签。
- 主要的内容区域包含的是网站的主要内容。让我们把这个区域命名为“主要内容”。

在 XHTML 页面中，在 DOCTYPE、命名空间、元数据后面，我们的第一个定义是这样：

```
<div id="newmenu"></div>
```

```
<div id="secondarynav"></div>
<div id="primarycontent"></div>
```

这三个主要区域的每一个最后都将填充不同的内容，但是首先，要设置每个区域的样式。在我们的默认（白色）样式表中，我们建立了布局的全局参数，把它们分组并置顶。我们从新菜单区开始，可以看到这个区在图 16.3 的顶端，并从左边延伸到右边：

```
/* Establish general layout parameters */
#newmenu {
    background: #e0861e;
    color: #000;
    border: 0;
    border-bottom: 1px dotted #000;
    margin: 0;
    padding: 0;
    text-align: left;
    height: 31px;
}
```

图 16.3
模板被分成三个区域



在学完第 9 章、第 10 章和第 12 章中的盒模型后，你应该知道怎样理解这些定义。我们来总结一下：区域高 31px，背景用的是非网络安全的橙色 (#e0861e)，内容被设置成左对齐，避免在一些版本的 Internet Explorer 上产生一个 bug，底部被设置成了点状边界的效果。我们不需要弥补盒模型的不兼容性，因为边距、边框距和边界都被设置为零。在 IES/Windows 上调整 1px，底部的 1px 点状边界就会侵入的上面的区域，因为这些浏览器不能理解在 CSS 盒模型里，边距、边

框距和边界的相互作用的方式。

16.2.1 安装选项条

二级导航或选项条区（如图 16.4 所示）的规则如下：

```
#secondarynav {  
    /* float: left; */  
    position: absolute;  
    left: 0;  
    margin: 0;  
    padding: 0 25px 25px 25px;  
    background: #bdcdcd url(/i/2003/sbbot4.gif) repeat-x  
    no-repeat bottom left;  
    border: 0;  
    border-top: 10px solid #95a580;  
    border-right: 1px dotted #000;  
    border-bottom: 1px dotted #000;  
    width: 151px; /* False value for IE4-5.x/Win */  
    voice-family: "\\"}\"";  
    voice-family: inherit;  
    width: 100px; /* Actual value for conformant browsers */  
}  
html>#secondarynav {  
    width: 100px; /* Be nice to Opera */  
}
```

要看的东西很多，但是你对其中的很多东西都已经很熟悉了。区域的总宽度为 151px（其中内容 100px，左右边距各为 25px，还有右边的点状边界为 1px）。我们用盒模型程式（第 12 章）给 IE5/Windows 赋值，用一个“对 Opera 友好”规则来避免 Opera 浏览器上发生一个分解 bug。还有很多工作要做，让我们从定位开始，进行深入的研究。

16.2.2 定位部分

定位部分的规则书写如下：

```
#secondarynav {  
    /* float: left; */  
    position: absolute;  
    left: 0;
```

执行定位的那一行用粗体突出表示；/* float : left ; */仅是对另外两行 CSS 要实现的效果做说明，或者，你可以这样理解，对它们要仿效的效

果做说明 (CSS 说明的书写方式和 C 编程语言的说明方式是一样的)。第一条规则把工具条从文档的普通流程中提出来 (`position : absolute ;`)；第二个 (`left : 0 ;`) 使工具条在菜单区的下方紧靠左边。

可以用其他的方法实现这样的效果吗？当然可以，实际上，在较早的版本里，我们使用的是以下的方法，不同之处用粗体突出表示：

```
#secondarynav {
    float: left;
    margin: 0;
    padding: 0 25px 25px 25px;
    background: #bdcdbd url(/i/2003/sbot4.gif) repeat-x
    bottom left;
    border: 0;
    border-top: 10px solid #95a580;
    border-right: 1px dotted #000;
    border-bottom: 1px dotted #000;
    width: 151px; /* False value for IE4-5.x/Win */
    voice-family: "\\"}\"";
    voice-family: inherit;
    width: 100px; /* Actual value for browsers that get
    CSS */
}
```

注意区域被指定为 `float : left`。这使得它漂浮在左边。下面解释用绝对定位而不用漂浮属性的原因。

float 产生的问题，999 部分

如在第 10 章中讨论的，在一些用得比较多的浏览器上用 `float` 属性会产生问题，包括不能实现漂浮的效果，在访问者从一个页面链接到另一个页面时，浏览器错误地计算漂浮区的高，使得内容区有点破碎。你可以通过 JavaScript 来弥补内容区，使其完整，但是这样做会产生可怕的浏览器 bug，至于是什么样的 bug，在此就不说明了。

而且当你把一个区设置为 `float`，如果用户的显示器比页面的总宽度小，就会损坏布局。用绝对定位来效仿漂浮，在大多数情况下都能解决这个问题。对基于 `float` 属性的 CSS 不协调的支持才是问题的真正所在。就像整个汽车都能正常工作，除了驾驶部分。

文件名的命名方法

我们总是热衷于节省带宽，所以努力使用缩短的目录名：用 /i/，而不用 /images/；用 /j/，而不用 /javascript/；用 /c/，而不用 /css/ 或是 /style/..

我们也把这样的方法用于文件名。例如，用于填充“创建颜色条”中的工具条底部的图片，被命名为 sbbot.gif 而不是 sidebar_bottom.gif。所用的字符越少，浪费的字节也就越少。我们不能保证百分百地这样做，但是我们要竭尽全力。

同样，在“具有 CSS 滚动效果的 Home 按钮”里，我们不使用 logo_banner.gif 指定默认的标志图片的状态，或 logo_banner_over.gif 指定图片的鼠标停在图片上或“在上方”状态，而使用 1b.gif 和 1bo.gif 分别指定这两种状态。文件名里出现的数字，显示版本，这样 1bo2.gif 就第二个版本。（“o”在 1bo2.gif 是“over”里的“o”，不是零。）

16.2.3 创建颜色条

我们来学习更多关于工具条的定义，注意我们通过 CSS border-top 属性给背景指定较浓的灰绿色 (#bdedb)。定位顶端的颜色块为实心的 10px#95a580（中度的灰绿色），使用 CSS border-top 属性你可以任意设置颜色块的高度，像下面这样：

```
border-top: 10px solid #95a580;
```

我们希望这个区的底部也有一个相应的 10px 的实心块，但是我们为制造 1px 的点状边界效果已经用光了 border-bottom 属性。（有一个 CSS 方法可用，但是据我们所知，只能用在 IE5/Macintosh 浏览器上。）

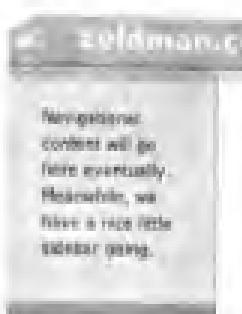
但是我们还有别的办法，可以用 Photoshop 创建一个大小合适的有颜色的矩形，命名为 sbbot（给工具条底部的），复制保存一张 GIF（sbbot4.gif），用做定位在左边底部的背景，并且只能横向复制（repeat-x），不能纵向复制（repeat-y）。下面是代码：

```
background: #bdedb url(/i/2003/sbbot4.gif) repeat-x bottom  
➥ left;
```

现在一个颜色和大小都合适的工具条就做好了，它的顶端和底部都有一层薄薄的点状边界和 10px 高的颜色带（如图 16.4 所示）。（我们花了很长时间达到图 16.4 那样的效果，不过任何优秀的 Web 组件设计都要花很长的时间。）

图 16.4

看看我，我是工具条！顶部的颜色带是用 CSS 的边界属性制作出来的。底部的颜色带暗一点，是用 Photoshop 创建的一个 GIF 图片。



颜色匹配问题

有趣的是，用 Photoshop 制作的下端的颜色条不能和通过 CSS 边界属性创建的上端的颜色条相匹配，即使你直接把 CSS 边界顶端的颜色（#95a580）复制粘贴到 Photoshop 的颜色选择器里也不行。因为这些相互冲突的技术，网络中的颜色就不可能匹配。下面是一些解决这个问题的方法：

- 所有浏览器在显示网页之前都要调整 gamma 值，你可能已经校准过你的显示器，而且把它设置为 sRGB (<http://www.w3.org/Graphics/Color/sRGB.html>)，一种 Internet 的标准默认色彩空间。但是浏览器不知道你已经调整了设置，它们还是会调节 gamma 值，每个浏览器改变 gamma 值的方式都和其他浏览器不同。
- 16 位的系统不能正确重现黑白以外的任何颜色，并且在 16 位系统里，浏览器对 CSS 颜色的显示有偏差，而对图片颜色显示又有相反的方向的偏差。参见 David Lahn 和 Hadley Stern 的说明。<http://hotwired.lycos.com/Webmonkey/00/37/index2a.html?tw=design>

我们将错就错，把这个颜色匹配问题当做一种设计创新，在底部的工具条挑选另外一种颜色——比较深的颜色看起来会更像“底部”。

16.2.4 内容里的间隔

创建主要内容区（图 16.3 中的一大块空白）很简单：

```
#primarycontent {
    border: 0;
    border-top: 10px solid #bdcdbd;
    padding: 0;
    margin: 0;
    margin-left: 150px;
    width: auto;
}
```

我们再次使用 CSS 创建 10px 高的颜色带（border-top:10px solid

#bdcdbd;)。虽然我们没有计划和比较文件，站点还是得到与实际一致的设计元素。我们关闭了所有的边界，创建一个左边距用以告诉文档区紧接在工具条结束的地方 (margin-left: 150px;)，并关闭其他所有的边距和边框距。通过一个小定义 (width: auto;)，告诉主要文档区填充所有的水平空间，从浏览器显示器的左边缘到右边缘。

盒中之盒

如果希望整个文档自动回行以符合浏览器窗口的宽度，我们应该设置基本参数。但是我们并不喜欢铺满整个屏幕的文档，读者也不喜欢。文档应该有趣、易读，这是我们重新设计的第 9 个目标。

很多书籍和杂志的设计师都知道，使文档易读的方法之一就是把文档放置在一个既不太宽也不太窄的栏里，它的宽度与工具条的宽度相结合应该适合最小的显示器。

你可能认为只要编辑 primarycontent，将其调整到 400px 宽就能达到目的了，其实这样做是不行的。因为如果这样做的话，一个浏览器会一直把文档挤到页面的左边，而另一个浏览器可能坚决要把文档放在工具条的前面。我们将创建 400px 宽的文档占位，并把它放进主要文档区。指定文档占位的 CSS 代码如下：

```
#bravefourhundred {  
    margin: 0;  
    border: 0;  
    padding: 15px 25px;  
    width: 450px; /* False value for IE4-5.x/Win */  
    voice-family: "\}\\";  
    voice-family: inherit;  
    width: 400px; /* Actual value for conformant browsers */  
}  
  
html>#bravefourhundred {  
    width: 400px; /* Be nice to Opera */  
}
```

知道怎样理解上面的规则，你就能识别实际值 (400px)。Tantek 程序使老版本浏览器正确地显示宽度，以及随后的“Be nice to Opera”规则。来看看标记，我们插入“brave four hundred”内容占位如下：

```
<div id="primarycontent">  
    <div id="bravefourhundred">  
        Content will go here.  
    </div>  
</div>
```

16.3 基于规则的设计

我们已经用 CSS 和标记打好了根基，这个布局在本世纪推出的任何浏览器上都能正常工作，它的内容在任何时期，任何用户代理上都能显示。我们开始用示例文档和图片填充模板，添加 CSS 定义来控制选定的标记元素的样式，以及这些元素之间相互作用的方式。我们的方法虽然简单，但是已经超越了“文件比较”驱动的思维模式，进入一个更新、更网络化的设计理念。总之，我们在朝着基于规则的设计方向发展。

我们不会阐述样式表里的每条小规则（你可以在 <http://www.zeldman.com/c/wh.css> 上自己研究这些规则），但是把其中的一个作为示例来解释基于规则设计：

```
p {  
    margin-top: 0;  
    margin-bottom: 1em;  
    font: 11px/1.5 Verdana, Trebuchet, Lucida, Arial,  
        sans-serif;  
}
```

这里的关键元素是值为 0 的上边距和值为 1em 的下边距（也就是说来，基于文档大小第一个结束符）。如果我们没有指定下边距的值，浏览器就会猜测我们的意图，它们不会设置段与段之间的垂直间距。

我们的目的不仅仅是避免网络浏览器上产生的不可预见行为，还有建立定义控制组件级外观和风格。基于规则的设计是创建组件级设计的一种技术。把上边距设置为 0，副标题就会定位于段落的顶端（特别是如果把副标题的下边距值也指定为 0 时，我们将在下一条规则中这样指定）。副标题的下边距值被设置为 1em，段落间的间距和段落与副标题之间的间距就被区分开来。

CSS 布局是由这些小决定组成的，使用它们，就能不依赖费解的外观标记和其相关的带宽占用而实现控制。

创建很多这样的小规则也可以使我们不必在标记代码中书写很多 class 属性，例如，前面出现的 小小的段落规则使我们免于书写以下的代码：

```
<p class="notopmargin">
```

这些小举措也能节约带宽。给各种各样的页面元素书写规则时，我们加入了关于这些元素怎样相互作用的指令代码。例如，创建一条规则，不仅给每个图片

指定一个 1px 的黑色边界，还指定在每个图片的下方插入一个 10px 的空白。使用 `#d` 选择器，我们可以为布局的 A 区的图片创建一种边界和间距，而为 B 区创建完全不同的边界和间距。在 CSS2 里，我们甚至可以根据它前面的元素改变元素的定位：

```
p+p {  
    text-indent: 2em;  
    margin-top: -1em;  
}
```

这个规则说明的是如果一个段落前面还有一个段落 (`p+p`)，第二个段落的第一行要缩进 2em，一般下边距规则将被暂停执行。这种规则在本书中经常出现（如图 16.5 所示）。几乎所有情况都能使用这样的规则：

```
img+h3 {  
    margin-top: 15px;  
}
```



图 16.5
打印风格的段落的缩进不必使用类选择器就能实现

这条规则告诉浏览器在任何紧随图片后的 `h3` 标题的上面留 15px 的空白。举例说明，在某个新闻网站里，一个 `h3` 标题紧随一个置顶的图片之后，上方需要留出 15px 的空白。虽然还不是每个浏览器都支持这类的 CSS 选择器，但是不理解这类规则的浏览器只是会忽视它，不会产生什么危害。大多数的基于规则设计就不必借助部分浏览器不能理解的 CSS2 选择器也能完成。基于规则设计就是很多这样相互关联的小规则的总和。

让我们来看看重新设计更多的重要的问题。

16.4 具有 CSS 滚动效果的“返回首页”按钮

前面，我们已经创建了 31px 高的“新菜单”区，用导航修补过后（本章稍后会讲到这个问题），我们发现它可以完全被放进工具条。“新菜单”区用来干什么呢？我们将把它用于有品牌的“返回首页”按钮（如图 16.6 所示）。

图 16.6

看这两个相似图片中的一个将被用做创建具有 CSS 滚动效果的有品牌的“返回首页”的按钮



“返回首页”按钮将响应用户的鼠标活动，在第 10 章阐述过的图片滚动效果将用 CSS，而不是 JavaScript 来实现。图片将通过使用 CSS 背景属性来插入，而不用常规的标签。

用 Photoshop 创建两张相似图片，其中一张如图 16.6 所示。在默认图片里，标志文档为米白色 (#fde8f2)。当访问者鼠标停在标志上方时，标志文档变为纯白色 (#fff)。这种效果在书中复制不出来，所以我们只给出一张图片而不是两张。两张 GIF 图片的背景都是透明的，以便#newmenu 的 CSS 背景颜色能充分地显示出来。（记住：如果你想在匹配 Photoshop 里的颜色，颜色就会失效。）

在标记中，我们把有惟一标志符“bannerlogoban”的

插入新菜单区：

```
<div id="bannerlogoban"></div>
```

指定样式如下：

```
/* Image-free logo banner with rollover */
#bannerlogoban {
    margin: 0;
    padding: 0;
    border: 0;
    width: 600px;
    height: 31px;
    background: url(/i/2003/part2/lbo2.gif) no-repeat;
}
```

宽度为 600px 以匹配图片的实际宽度。高度为 31px 以匹配图片和所含元素（“新菜单”）的高度。背景图片定义在鼠标停在图片上方时 (lbo2.gif) 预载图片，所以当用户的鼠标活动触发 CSS “交替”效果时就不会有迟滞。

16.4.1 Fahrner 的图片置换 (Fahrner Image Replacement, FIR) 方法

虽然我们的标志/Home 按钮已经实现了鼠标响应的图片交替效果了，但是按钮不能被单击，不能实现任何功能。下一步要做的工作需要费点脑筋理解，它是 Todd Fahrner 首先倡导的。首先我们要创建一个名为 alt 的类别，它的惟一目的就是使之在 CSS 环境里不可见：

```
.alt {  
    display: none;  
}
```

下面我们要写第二组规则，用来指定链接到主页的锚链接：

```
#logoban {  
    display: block;  
    padding: 0;  
    border: 0;  
    margin: 0;  
    background: url(/i/2003/parts/lb2.gif) no-repeat;  
    width: 600px;  
    height: 31px;  
}  
a#logoban:hover {  
    background: url(/i/2003/parts/lb02.gif) no-repeat;  
}
```

第一条规则在非鼠标停在图片上方的状态加载图片 (lb2.gif)。第二条告诉浏览器，当鼠标停在图片上方的时候，显示鼠标停在图片上方的形式。

接下来构建所需的标记，由里至外。先创建要在非 CSS 环境中显示的一些文档：

```
Zeldman.com. Web design news and entertainment since 1995.  
ISSN #1534-0309.
```

再把文档放进一个类属性为 alt 的间距元素，对支持 CSS 的浏览器来说，文档将被隐藏：

```
<span class="alt">Zeldman.com. Web design news and  
entertainment since 1995. ISSN #1534-0309.</span>
```

文档将在屏幕阅读器和 Palm Pilot 上是可见的，但是在支持 CSS 的浏览器上则看不到。下一步，把文档放入一个链接回主页的锚链接中，为这个链接指定 "logoban" 的 id 属性：

```
<a id="logoban" href="/" title="Zeldman.com. Web design news and entertainment since 1995."><span class="alt">Zeldman.com. Web design news and entertainment since 1995. ISSN #1534-0309.</span></a>
```

这样我们有了一个可工作的链接。在支持 CSS 的环境里，它显示为一个具有滚动效果的图片。在非 CSS 环境里，它仅仅是被链接了的文档。现在把这些整个放进“bannerlogoban”规则中，这条规则是在本节开始部分创建的：

```
<div id="bannerlogoban"><a id="logoban" href="/" title="Zeldman.com. Web design news and entertainment since 1995."><span class="alt">Zeldman.com. Web design news and entertainment since 1995. ISSN #1534-0309.</span></a></div>
```

看上去有很多的标记，但是这比原来写 `image` 元素、JavaScript 预载、JavaScript 鼠标激活链接脚本，以及一般存在于 `image` 元素内部的 JavaScript `onmouseover` 和 `onmouseout` 事件处理器，等等需要的代码的字节小得多。

下面是完整的标记代码，包括“新菜单”的父元素：

```
<div id="newmenu"><div id="bannerlogoban"><a id="logoban" href="/" title="Zeldman.com. Web design news and entertainment since 1995."><span class="alt">Zeldman.com. Web design news and entertainment since 1995. ISSN #1534-0309.</span></a></div></div>
```

我们可以把 `bannerlogoban` 的规则添加到 `newmenu` 的规则定义中，这样我们就可以节省一个步骤和 DIV (`bannerlogoban`) 层。但是，如果在将来我们可能要给 Home 按钮的右边（或左边）添加一些东西，我们只能这样做。

Fahmer 图片置换方法 (FIR) 的额外用途

Fahmer 在“看，没有 `IMG` 标签”里介绍的技术有很多种用法，例如，可以用它在自适应布局的中心插入一个非常宽的图片。当访问者扩宽他的浏览器窗口时，图片的多余部分会显现出来。我们用这个技术创建了三个横幅的外观，而且，不仅在图片上，在图片的边界上也有图片交替效果（如图 16.7 所示）。

图 16.7

制作网站的其他部分时会再次用 Fahmer 方法，创建三个横幅的影像，图片上有滚动效果，边界上也有独立的滚动效果。



刚起步时的 CSS 就像上面那个示例里的一模一样，只是多一条创建边界的规则。但不是所有的浏览器都能处理这类的 CSS 方法。在 Opera 6 上产生一个问题。

题，在 Netscape 7 上会产生另外一个问题。为了修复这些问题又可能在 IE 5/Macintosh 浏览器上产生新的问题，这样下去，总存在需要解决的问题。

笔者把 Porter Glendinning 和 Fahrner，还有笔者自己的方法结合在一起，写出了如下规则：

```
/* Banners without img elements, thanks Todd and Porter */
#banner1, #banner2, #banner3 {
    margin: 10px 0 0 0;
    padding: 0;
    width: 100px;
    height: 25px;
}

#banner1 {
    /* Opera uses this background for the rollover effect. */
    background: url(/i/bans/hc100bano.gif) no-repeat 1px;
}

#banner2 {
    /* Opera uses this background for the rollover effect. */
    background: url(/i/bans/alal00bano.gif) no-repeat 1px;
}
#banner3 {
    /* Opera uses this background for the rollover effect. */
    background: url(/i/bans/zeldmix2.gif) no-repeat 1px;
}

#hcban, #alban, #wtban {
    display: block;
    padding: 0;
    border: 1px solid #000;
    background: url(/i/bans/hc100ban.gif) no-repeat 1px; /* start hiding from macie*/
    background-position: 0px; /* stop hiding */
    width: 100px;
    height: 25px;
    voice-family: "\}\\""; /* Need we explain? */
    voice-family: inherit;
    width: 98px;
    height: 23px; /* Actual values to overlap borders */
}

html>body #hcban, html>body #alban, html>body #wtban {
    width: 98px;
```

```

height: 23px; /* Be nice to Opera */
}

#alban {
background-image: url(/i/bans/ala100ban.gif);
}

#wtban {
background-image: url(/i/bans/zeldmix.gif);
}

a#hcban:hover {
background-image: url(/i/bans/hc100bano.gif);
border: 1px solid #ffc;
}

a#alban:hover {
background-image: url(/i/bans/ala100bano.gif);
border: 1px solid #ffc;
}

a#wtban:hover {
background-image: url(/i/bans/zeldmix2.gif);
border: 1px solid #ffc;
}

.alt {
display: none;
}

```

下面是页面中调用它的标记:

```

<div id="banner2"><a id="alban"
href="http://www.alistapart.com/" target="eljefe"
title="A List Apart, for people who make websites.">
<span class="alt">A List Apart</span></a></div>

<div id="banner1"><a id="hcban" href="http://www.happycog.com/"
target="eljefe" title="Happy Cog Studios. Web design and
consulting."><span class="alt">Happy Cog Studios</span></a>
</div>

<div id="banner3"><a id="wtban" href=
"http://w3mix.web-graphics.com/" target="eljefe"
title="Remix the W3C."><span class="alt">WThRemix</span></a>
</div>

```

这个方法在本世纪推出的任何浏览器上都能正常工作。更重要的是，它可以在任何浏览器和 Internet 设备（无论是新的还是老的）正确表现内容（如果样式表被链接了，这种技术在 JAWS 屏幕阅读器上就会失效，JAWS 屏幕阅读器只阅读浏览器上呈现 CSS 后的可视内容，但是如果输入样式表，就可以在 JAWS 上看见文档了）。

这个练习的代码已经全部展现出来了。以我们的观点来看，用这种方法来创建边界图片交替效果并不理性。因为只要用 Photoshop 修改颜色就能达到同样的效果，而且占用的带宽更少。但这种方法是值得学习和尝试的。本章里讨论的所有 CSS，可以访问实际正在用的样式表还可以复制、粘贴、修改它们，以满足你自己的需要。

16.5 CSS/XHTML 导航条

现在我们来填充图 16.4 中未完成的工具条。例如，我们可以给它填充导航以方便访问者查找。从概念上说，导航条只是一个链接列表。我们把这些链接写成一个标准的无序列表：

```
<ul>
<li id="secondarytop"><a href="/" title="The Daily Report. Web
design news and info.">daily report</a></li>
<li id="glam"><a href="/glamorous/" title="My Glamorous Life:
Episodes and recollections.">glamorous</a></li>
<li id="classics"><a href="/classics/" title="Entertainments,
1995&#8211;2002.">classics</a></li>
<li id="about"><a href="/about/" title="History, FAQ, and
suchlike.">about</a></li>
<li id="contact"><a href="/contact/" title="Write to us.">
contact</a></li>
<li id="essentials"><a href="/essentials/" title="Info for web
designers.">essentials</a></li>
<li id="pubs"><a href="/pubs/" title="Zeldman&#8217;s books
and articles.">pubs</a></li>
<li id="tour"><a href="/tour/" title="We may be coming to your
town: personal appearances.">tour</a></li>
</ul>
```

为了避免出现第 12 章中讨论的空白 bug，我们必须清除空白，像下面这样：

```
<ul><li id="secondarytop"><a href="/" title="The Daily Report.
Web design news and info.">daily report</a></li><li
```

```

id="glam"><a href="/glamorous/" title="My Glamorous Life: Episodes and recollections.">glamorous</a></li><li id="classics"><a href="/classics/" title="Entertainments, 1995&#8211;2002.">classics</a></li><li id="about"><a href="/about/" title="History, FAQ, and suchlike.">about</a></li><li id="contact"><a href="/contact/" title="Write to us.">contact</a></li><li id="essentials"><a href="/essentials/" title="Info for web designers.">essentials</a></li><li id="pubs"><a href="/pubs/" title="Zeldman&#8217;s books and articles.">pubs</a></li><li id="tour"><a href="/tour/" title="We may be coming to your town: personal appearances.">tour</a></li></ul>

```

是不是很难阅读呢？但是只要我们关心站点在 Windows 的 Internet Explorer 上的外观，我们就别无选择。

16.5.1 添加样式

我们匆匆做出了一个列表，现在要把它转换一下：

```

/* Create buttons */
#secondarynav ul {
    list-style: none;
    padding: 0;
    margin: 15px 0;
    border: 0;
}

#secondarynav li {
    text-align: center;
    border-bottom: 1px solid #000;
    width: 100px;
    margin: 0;
    padding: 0;
    font: 10px/15px Verdana, Lucida, Arial, sans-serif;
    color: #000;
    background: #e0861e;
}

#secondarytop, #tertiarytop {
    border-top: 1px solid #000;
}

#secondarynav li a {
    display: block;
}

```

```
font-weight: normal;
padding: 0;
border-left: 1px solid #000;
border-right: 1px solid #000;
background: #e0861e;
color: #000;
text-decoration: none;
width: 100px; /* False value for IE4-5.x/Win. */
voice-family: "\}\\";
voice-family: inherit;
width: 98px; /* You get it. Good value for compliant
               browsers. */
}

html>#secondarynav li a {
    width: 98px; /* Be nice to Opera */
}

#secondarynav li a:hover {
    font-weight: normal;
    border-left: 1px solid #000;
    border-right: 1px solid #000;
    background: #c90;
    color: #fff;
    text-decoration: none;
}
```

现在我们来逐一看看每条为菜单按钮而服务的规则：

```
#secondarynav ul {
    list-style: none;
    padding: 0;
    margin: 15px 0;
    border: 0;
}
```

在这条规则里，我们告诉列表不要显示样式（list-style : none ;）还设定边框距、边界和右边距的值为 0，给整个导航在顶端和底部指定了 15px 的空白。下一条：

```
#secondarynav li {
    text-align: center;
    border-bottom: 1px solid #000;
    width: 100px;
    margin: 0;
    padding: 0;
    font: 10px/15px Verdana, Lucida, Arial, sans-serif;
```

```

color: #000;
background: #e0861e;
}

```

在前面的步骤里已经设置了父元素（ul）的样式，现在来设置了元素—列表项—文档居中，明确地设定了宽度（这里没有边框距或水平边界值，所以不会，也不能正确理解盒模型的浏览器的混淆宽度的这一情况），告诉每个列表项在其底部生成一个实心黑色的线条。当把这个线条添加给稍后描述的其他项时，就会产生一个盒或者说一个按钮轮廓的效果。

最后，注意行高值，这个值列在字号的后面，是列出的第一个值。（Font: 10px/15px 是 CSS 的简略形式。它和 font-size: 10px; line-height: 15px 是相等的。）用这条规则告诉每个列表项，或是“按钮”占据的高度为 15px，把文档放置在垂直方向的中间位置（因为这是行高的工作方式）。虽然这条规则很简单，但是起的作用很多：

```

#secondarytop, #tertiarytop {
    border-top: 1px solid #000;
}

```

上面那条规则把一个黑色边界附着在其惟一 id 属性为 secondarytop 或 tertiarytop 的一个元素的顶部。在这里我们只关心 secondarytop，再看看标记，我们把标记简化重新书写如下：

```

<ul><li id="secondarytop"><a href="/" title="The Daily Report. Web design news and info.">daily report</a></li>

```

可以看到第一个列表项有一个 secondarytop 的 id 属性。除了那些指定每个列表项样式的规则，这条规则也依赖一个附加的定义告诉它在顶部添加一条黑色线条（1px 的边界）。换句话说，第一个“按钮”将有一个顶部和底部。如果给每个列表项都指定一个上边界和下边界，线条会互相破坏，效果就不好了。

填充颜色：美观和可访问性

下面的这条规则还不够完美：

```

#secondarynav li a {
    display: block;
    font-weight: normal;
    padding: 0;
    border-left: 1px solid #000;
    border-right: 1px solid #000;
    background: #e0861e;
    color: #000;
}

```

```
text-decoration: none;
width: 100px; /* False value for IE4-5.x/Win. */
voice-family: "\\"}\\";
voice-family: inherit;
width: 98px; /* You get it. Good value for compliant
               browsers. */
}
```

上面那条规则中，指定了锚链接的样式，并告诉链接以块显示，而不以内联显示。这个执行结果和在 #secondarynav li 里设置的行高相结合，那个“按钮”就被特定的颜色填充（增加美观的角度），整个按钮也可被单击了（这样能提高可访问性，使视力受损和近视用户受益），而且“按钮”看起来真的像按钮了。

最后给按钮创建左右边界，这样按钮的填充工作就完成了，我们还以一贯使用的手法使 IE5/Windows 不去修补盒模型：

```
html>#secondarynav li a {
    width: 98px; /* Be nice to Opera */
}
```

下面是按钮所需的最后一条规则，制造一个鼠标停在上方或者滚动的效果：

```
#secondarynav li a:hover {
    font-weight: normal;
    background: #c90;
    color: #fff;
    text-decoration: none;
}
```

出于可访问性和美观的原因，使用前面创建的链接定义里的 display : block 指定“按钮”为可单击的，链接规则串联了更多的在这里显示的具体的停在鼠标上方规则。这个规则告诉文档颜色和文档背景颜色的变化对应于用户的鼠标光标移动。不必重复左边界和右边界指令，因为会单像素的黑色边界效果会从前面介绍的链接规则一直持续下去。

为了效果更好，像在第 10 章里创建混合布局时一样，我们把特定网页嵌入样式插入页面的<head>部分，使相关按钮呈现“you are here”效果：

```
<style type="text/css">
<!--
#secondarytop a:visited {
    font-weight: normal;
    background: #c60;
```

```

    color: #fff;
    text-decoration: none;
}
-->
</style>

```

如图 16.8 和图 16.9 所示，可以看见每日报告“按钮”里的“you are here”效果，还有它的深色背景（#c60）和反向突出的文档（#fff）。另外，还可以在图 16.9 中看到用户激活的图片滚动效果。

图 16.8

CSS 翻新了基本无序列表，但是还有更妙的



图 16.9

这些都是无处不在的滚动效果。对一般用户来说，它们只是标准的导航按钮，但实际上，它们是已经被指定了样式的导航添加的标记



一个技术上的小提示：如果用 XHTML1.0 严格方式来制作，围绕样式表的<!--注释就会产生问题，还好我们用的是过渡式。如果把网页作为 application/xhtml+xml 处理，这些注释也会产生问题，我们还是把它作为 text/html 来处理为妙。如果要理解为什么这些元素在严格式 XHTML 环境里会产生问题，见 <http://devedge.netscape.com/viewsource/2003/xhtml-style-script/>。

当导航条、横幅、搜索界面和其他一些小部件结合后，结果就是你在图 16.10 中看到的那样。

16.6 最后的加工

关于布局还有很多内容，即使是这么长的一章也不可能完全讲完，比如包括搜索表格的、用来增强可访问性的 form 元素，忽略导航、tabindex 和 accesskey，以及一些小部件（在图 16.10 的中间可以看到）的设计。还有像 16.11 展示的可选样式，图 16.12 展示的“隐藏”的外部链接列表，这些设计的

代码调用方法在前面的章节（第 15 章）中都有介绍。

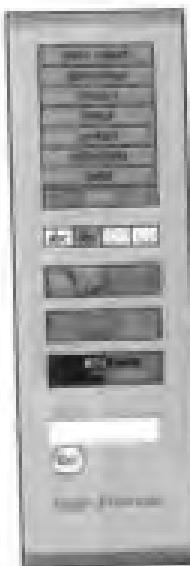


图 16.10
最后工具条被完全填充

设计完成后，不可避免的，还有一个工作要做——排除程序中的错误。虽然几乎没什么错误，因为浏览器能够理解我们所使用的 CSS，但是还是会有一点问题。

例如，在苹果公司的 Safari 浏览器上，把白色的布局转换成橙色的布局（如图 16.13、图 16.14 所示）后，图片产生一个等待时间的问题。公平地说，除了这个问题，Safari 浏览器能够近乎完美地呈现布局，而且这个 bug 也很容易解决。



图 16.11
可选择式（“橙色”）造成

图 16.12

橙色的布局和开始被隐藏的“外部链接”工具条展现出来（见 15 章关于怎样通过 DOM 显示和隐藏内容）



你会想起来，白色的布局在工具条底部用的是一张背景图片（如图 16.4、图 16.10 所示），橙色的布局没用这样的背景图片，如果你不上网查看 <http://www.zeldman.com/color.css>，你就不可能知道出现了这种情况，当 Safari 浏览器转换成橙色布局时，错误地保留了白色布局的背景图片。

图 16.13

你可以在苹果的 Safari 浏览器上看到背景图片 bug



修复方法很简单：在橙色布局的修正版本中，明确地告诉所有浏览器不要在工具条里使用背景图片。图 16.15 是 CSS 修改后在 Safari 浏览器上的显示。



图 16.14

当工具条单独显示的时候，这个 bug 特别明显。本来工具条的底部应该是 一个褐色条，现在出现了两个，与图 16.2 和图 16.11 中的对比一下。



图 16.15

调整 CSS 解决 Safari 浏览器上的问题（注意：从图 16.13 到图 16.15，可以看到早期版本的 CSS 样式转换——悬在三个按钮和搜索区的上方）

下面是我们为橙色布局书写的初始规则：

```
background: #c60;
```

这是使 Safari 正确显示的替换规则：

```
background-color: #c60;  
background-image: none;
```

除了 Safari，没有浏览器要求做这样的改变。但是这样的 bug 很容易解决，规则中的变更也是合法的。Safari 的最终版本将不会有任何小故障，不过 background-image: none; 可以修复问题又没什么危害，所以 Safari 还是可以用的。其他的网络设计都把排除程序错误作为最后一个步骤，但在这个例子中，排除错误和其他步骤一样是在读者阅读的过程中实时发生的。

不使用外观标记或给每个浏览器发送单独的 CSS 文件，一个无表格 CSS 布局和用户选择的可选版本就完成了。站点下载得很快，在所有现代浏览器上的外观和功能都相同，对于任何因特网设备都是可访问的。这一章的内容就结束了，但是网站的重构还会继续，例如我们计划创建附加的可选布局，提供大型的或更多的可调整大小的文档。不过这又是另外一回事了，学了这么多，动手做自己的网站吧。

Part III

第3部分 结尾

现代浏览器，品质优良的、低劣的和非常低劣的。

现代浏览器：品质优良的、 低劣的和非常低劣的

在第 1 章中就解释过，当我们说“现代”或“兼容标准”浏览器时，指的是那些能够理解并支持 HTML 和 XHTML、CSS、ECMAScript 和 W3C 文档对象模型（DOM）的浏览器。这些基本标准能够帮助设计师和开发者超越以前守旧的方法（表现标记和不兼容的脚本语言）及其造成的使网站荒废的陈旧方法。

现在还没有浏览器是完全支持标准的，而且也不太可能达到。但是 2000 年开始，就涌现了一大批几乎全面支持基本核心标准的浏览器。当这些浏览器的升级版本投放市场时，它们已经更具兼容性，bug 也更少了。这些浏览器所占的市场份额在不断地增加。

到写这本书时，几乎所有的 Web 用户都已经更新成下面列出的浏览器其中一种，或者它们的后续的改良版本。这张列表只列出了用得最多的一些浏览器及其主要特性。

兼容标准的浏览器：第一批

Opera 7

推出时间：2002 年

支持 HTML/XHTML 吗？：支持。

支持 CSS 吗？：几乎全面支持所有 CSS1 和大部分 CSS2。

支持 ECMAScript/DOM 吗？：支持，第一个 Opera 版本就已经支持了。

特性：Opera 的第一个版本支持 W3C DOM，是第一个真正“兼容标准”的 Opera 版本。其公司的产品对最早期的标准如 HTML 和 CSS 一直都提供良好的

支持。如 Opera 的以前的版本一样，Opera 7 加入了页面缩放功能，可以帮助增强网络文档和网络图片对于视力受损用户的可访问性。

MSIE 5+/Macintosh

推出时间：2001 年

支持 HTML/XHTML 吗？：支持

支持 CSS 吗？：全面支持所有 CSS1，部分支持一些 CSS2。

支持 ECMAScript/DOM 吗？：支持

特性：2001 年 3 月第一个“兼容标准”浏览器投放市场，它是第一个正确支持 JavaScript/ECMAScript 的 IE/Mac 版本，第一个在任何平台上都正确支持 CSS 盒模型的浏览器。文本的缩放功能帮助增强网络文档对视力受损用户的可访问性。支持用户样式表。

浏览器的用于提高对标准的支持的 Tasman 表现引擎面世，IE5/Macintosh 对 DOM 的支持非常良好，但是并不全面。运行速度很慢，有时候出现的奇怪行为令动态文档的制作者们抓狂。浏览器支持基本的 DOM 功能，从标准的角度来说，整体品质还是很不错的。

Netscape 6+

推出时间：2001 年

支持 HTML/XHTML 吗？：支持

支持 CSS 吗？：支持全部 CSS1，大部分 CSS2。

支持 ECMAScript/DOM 吗？：基本上支持，虽然有些位数是奇数，相对于 IE/Windows，它的动态色彩刷新的速度很慢。

特性：基于 Gecko 的浏览器，完全支持网络标准 CSS，XML，XHTML，DOM 和 ECMAScript（如 Tasman，Gecko 是支持核心网络标准的一个表现引擎。Tasman 只适合 Macintosh，而 Gecko 适合所有的平台）。

早期的 Netscape 6.0 版本有一些 bug，后来的版本要好些，7.0 和更高的版本都很出色。

为增强可访问性而加入了文本缩放功能，支持用户样式表和可选样式表，从 Netscape 7.01 开始，支持自动弹出广告屏挡。

Mozilla 1.0

推出时间：2002 年，Mozilla 1.0 是 2002 年 5 月推出的，最早的 Mozilla 要追溯

到 1996 年左右。

支持 HTML/XHTML 吗？：支持

支持 CSS 吗？：支持全部 CSS1，大部分 CSS2。

支持 ECMAScript/DOM 吗？：见 DOM 在 Netscape 6+ 里的说明。

特性：开放源代码的基于 Gecko 浏览器完全支持网络标准。为增强可访问性而加入了文本缩放功能，支持用户样式表和可选样式表。基于 Mozilla 浏览器还包括 Chimera/Camino 和 Phoenix，但是 Mozilla 不仅仅用于浏览。你可以用 Gecko 和开放源代码的 Mozilla 基本代码创建超越传统台式浏览器的新应用程序（例如，用 Mozilla 和 Java 制作的电视置顶盒）。

Safari

推出时间：2002 年下半年

支持 HTML/XHTML 吗？：支持

支持 CSS 吗？：似乎支持大部分 CSS1，一部分 CSS2，有时候支持的方式很奇怪。

支持 ECMAScript/OOM 吗？：基本上支持。

特性：它是苹果电脑为 OS X 用户创建的，基于开放源码 KHTML 引擎。在大部分站点上，轻便、高速、精确。虽然现在还属于二流浏览器，但是已经被上百万的 Macintosh 用户使用。加入了文本缩放功能以提高可访问性。还收入了快捷 Bug 报告按钮，可以快速修复 CSS、XHTML 或脚本里的错误。

MSIE 6/Windows

推出时间：2001 年

支持 HTML/XHTML 吗？：支持。

支持 CSS 吗？：支持大部分 CSS1，部分 CSS2。

支持 ECMAScript/DOM 吗？：基本上支持，但是需要一些专门的辅助程序。这些辅助程序只为 IE6/Windows 编写一些代码。一般没有必要用辅助程序编写代码，除非你要创建一个只用 IE 的内部网站，而且即使这样，也最好用标准 DOM，以防将来把网站的特性和部分转移到公共网络空间而产生问题。

特性：IE/Windows 版本适应范围最广的浏览器，也是目前网络使用最多的浏览器，部分原因是它是惟一的操作系统内置浏览器，当与 Windows XP Clear Type 配合使用时，显示文档的效果非常好。

不包含文本缩放或页面缩放，不支持可选样式表。视力受损用户可以通过设

置“辅助功能”里的“忽略文档字号”来提高可访问性。但是这种“要么全有，要么全无”的选择没有其他标准兼容浏览器提供的文本缩放和页面缩放方便好用。

IE/Windows 用户可以通过给浏览器添加一个窗口小部件来调整文档字号，但是不能调整某些字号设置方式设置的字体。（例如，用像素设置的文档在 IE/Windows 上就不能调整大小），而在 IE5+/Macintosh、Mozilla、Netscape 6+、Opera 和 Chimera 上，无论是用特殊方式设置的文档，其字号都是可以调整的。

MSIE5.5/Windows

推出时间：2001 年

支持 HTML/XHTML 吗？：支持。

支持 CSS 吗？：大部分支持（但是有一些大 bug）。

支持 ECMAScript/DOM 吗？：不太支持。

特性：理解标准，过去是很不错的浏览器，但是它的 CSS 的 bug 和脚本漏洞使它的兼容性没有其他所列的浏览器高。本书的第一部分内容“设计和建造”里介绍了关于怎样解决 CSS 的 bug 问题。

MSIE5/Windows

推出时间：1999 年

支持 HTML/XHTML 吗？：支持，虽然有些漏洞。

支持 CSS 吗？：一点点，但是有大的 bug 和漏洞。

支持 ECMAScript/DOM 吗？：一点点。

特性：见 IE5.5 的说明。

Netscape 4

推出时间：1997 年

支持 HTML/XHTML 吗？：只是部分支持。

支持 CSS 吗？：几乎不支持。

支持 ECMAScript/DOM 吗？：不支持。

特性：它是在浏览器竞争激烈的时候推出的，曾经是功能强大的浏览器，只支持专门的代码和标记，不支持标准，几乎不支持 CSS，所以它对基本 HTML 的支持也不出色。不支持 DOM 因为 DOM 那时候还没被写出来……而且就算那个时候 DOM 已经出现了，它也可能不支持，因为在浏览器竞争强烈的时期，

Netscape 和微软都坚信只有牺牲标准，发明新技术才能“取胜”。虽然大部分用户都已经升级为 Netscape 6+ 或是其他的，如 MSIE、Opera 浏览器，还是有一些用户出于这样那样的原因没有升级。因为还有用户坚持用它们，还因为它们对标准不令人满意的支待，很多设计师和开发者觉得要继续采取过时的方法来“支持”不断减少的这部分用户。但是本书证明，可以在使用标准的同时支持 Netscape 4 和任何用户。

MSIE 4

推出时间：1997 年

支持 HTML/XHTML 吗？：Netscape 4 支持得多一些，但不是很多。

支持 CSS 吗？：Netscape 4 支持得多一些。

支持 ECMAScript/DOM 吗？：不支持

特性：它在浏览器竞争激烈的时期推出的，支持专门的代码和标记，不支持标准。几乎所有的 IE 4 用户后来都升级为最新的版本，部分原因是微软把浏览器和操作系统捆绑在一起。例如，从 Windows 95 升级到 Windows XP，你就需要不断地从 IE 4 转换到 IE 6。虽然从标准的角度来说，IE 4 没有 IE6 “好”，但是对于开发者来说，IE 4 产生的问题更少，因为 IE 4 用得更少。

术 语 表

Web standard

Web 标准，一个贯穿全书的术语。这里 Web 的含义不仅仅是指网站，也包含通过网络方式进行交互的各种应用程序和设备，因此我们保留“Web”英语单词，以避免翻译得不准确而误导读者。另一方面本书论述的技术被称为“Web 标准”，这个称法是由 Web 标准组织（见术语：Web standard project）定义的，W3C 组织定义这些技术为“推荐的”。

Web Standard Project (WaSP)

Web 标准组织。本书作者于 1998 年创建的一个网站设计师和开发人员的联盟组织 (www.Webstandards.org)，目的是帮助终止 Microsoft 与 Netscape 之间的浏览器之争，并且劝说他们在新版本浏览器中支持相同技术。Web standard project 又缩写为 WaSP。

W3C

万维网联盟组织。W3C (World Wide Web Consortium, <http://www.w3.org/>) 创建于 1994 年，主要研究 Web 技术标准和指导方针，致力于推动 Web 发展及保证各种 Web 技术能很好地协同工作。大约 500 名会员和组织加入这个团体。W3C 推行的主要规范有 HTML, CSS, XML, XHTML 和 DOM (Document Object Model)。

A List Apart

一本专门为网站设计师和开发人员开办的杂志 (www.alistapart.com)。最早是作者和 Brian Platz 在 1998 年创办的电子邮件列表，1999 年变为正式发行的周刊。缩写为 ALA。

ECMA (European Computer Manufacturers Association)

欧洲计算机制造商协会 (www.ecma-international.org)。成立于 1961 年，总部位于日内瓦，主要建立信息及通信系统方面的标准。

Section 508

508 条款。美国 1988 年社会福利法案的第 508 节“伤残资源法案”中规定：身心障碍者有无障碍地使用电子资讯科技的权利。为网站和 Web 应用程序的残疾人可访问性制定了 16 条准则。

RDF (Resource Description Framework)

资源描述框架，又称做 Web 数据的语义描述模型。因为 XML 不具备语义描述能力，所以 W3C 推荐以 RDF (<http://www.w3.org/RDF/>) 标准来解决 XML 的语义局限。

WAI(Web Accessibility Initiative)

Web 可访问性组织。1990 年 W3C 建立的组织。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/>)。该组织提供了一种可访问性标准等级。

Gecko

Gecko 是 Mozilla 使用的设计与分析引擎，它被认为是与 W3C 标准最接近的。

Box Model

CSS 把 HTML 中以<some>…</some>的部分称为 BOX(容器)，BOX 有三类属性：padding，margin 和 border。

Quirks Model

怪癖模式，是指 Netscape 和 IE 浏览器忽略标准，自行其事的做法。这些做法包括对 HTML 1 标记的私有扩展，以及错误地、部分地执行 CSS，还有它们自己制定的脚本语言。

URI

统一资源定位符简称 URI。

DOM (Document Object Model)

文档对象模型。

DTD

Document Type Definition 是一个包含了 XML 语言描述的文件。它实际上是一份由所有可能标签、可能属性及可能组合的列表。DTD 描述了在 XML 语言中，什么是可能的，什么是不可能的。所以，当我们谈论 XML 语言时，我们实际上是在谈论一个特定的 DTD。

PalmPilot

PalmPilot 的始祖开发商是美国 3Com 公司，后来开发部门独立成为现在的 Palm Computing 公司，并且上市。简单而言，PalmPilot 是一部 PDA Personal Digital Assistant，即个人数字助理。

Pocket PC

Pocket PC (简称 PPC) 都是 PDA (个人数字助理) 的一种，但是它和 PALM 是很不相同的。PPC 所使用的是 Microsoft 开发的系统 Pocket PC 2002 或 2003 (2003 又称 MOBILE 2003)。

XHTML

eXtensible HyperText Markup Language 的简写，XHTML 是一种为适应 XML 而重新改造的 HTML。

XML

eXtensible Markup Language 的简写，可扩展标记语言，它是标准通用标记语言 (Standard Generic Markup Language, SGML) 的一个子集。其目的在于使得在 Web 上能以现有超文本标记语言 (Hypertext Markup Language, HTML) 的使用方式提供，接收和处理通用的 SGML 成为可能。

RFP (Request for Proposal)

需求。

Bobby

Bobby 是 Watchfire 提供网页可访问性的线上检测服务。目前 BOBBY 可分别检测 WAI 制定的 Web Content Accessibility Guidelines 1.0 及美国政府制定的法规 U.S. Section 508 Guidelines。

Workaround

Workaround 简单地说是一个 Web 网页元素的支持环境，比如各种浏览器对 HTML 和 CSS 的支持不一样，为了能让设计好的页面在各种浏览器上显示（或者尽量）一样，设计者会想出各种各样的 Workaround 来实现。

LIFT

LIFT 是由 UsableNet 公司 (www.usablenet.com) 提供的一种可以分析网页潜在的可用性问题的服务。同时 UsableNet 公司的 LIFT 为 Macromedia Dreamweaver Pro Suite 提供了所有必要的工具来迅速发现并解决网站存在的影响残疾人访问的问题。